CAMBRIDGE SEMANTICS

Anzo® 5.4 User Guide

Last Updated: 1/29/2024

Online documentation is available at docs.cambridgesemantics.com

© 2024 Cambridge Semantics, Inc.

Table of Contents

About This Doc **19**

Onboard & Virtualization **20**

Onboard Structured Data 21

Adding Data Sources 22

Connecting to a Database 23

Defining a Database Schema 45

Adding an HTTP or SPARQL Data Source 58

Adding a CSV Data Source 60

Adding a JSON Data Source 68

Adding an XML Data Source 75

Adding a SAS Data Source 81

Adding a Parquet Data Source 88

Configuring a CSV or Parquet Source for Incremental Processing 92

Assigning Primary Keys in a Schema 97

Adding Foreign Keys 100

Onboarding Data with the Automated Workflow 105

Creating a Graphmart from a Data Source 105

Adding a Data Source to an Existing Graphmart 108

Direct Load Advanced Settings Reference 112

Onboarding or Virtualizing Data with SPARQL Queries 117

Introduction to the GDI 117

GDI Concepts and Basic Usage 120

Options for Data Types, Data Linking, and Models 189

Advanced Usage by Data Source Type 205

GDI Property Reference 339
Onboard Unstructured Data 355
Unstructured Onboarding Process Overview 356
Creating an Unstructured Pipeline 358
Running an Unstructured Pipeline 379
Pipeline Settings Reference 381
Annotator Settings Reference 387
External Service Annotator 387
Keyword and Phrase Annotator 390
Knowledgebase Annotator 393
Regex Annotator 397
Model 401
Model Concepts and Vocabulary 402
Managed Model Concepts 405
Model Requirements 407
Uploading a Model 412
Creating a Model 415
Editing a Custom Model 418
Opening Models in the Editor 419
Changing Model Components 421
Class Editor Reference 424
Property Editor Reference 426
Editing a Managed Model 428
Downloading a Model 434
Defining Resource Templates 438

Blend 441

Working with Datasets 442

Adding an Empty Dataset for an Export Step 443

Importing an Existing Dataset (FLDS) 446

Creating a Dataset from RDF Files 451

Managing Dataset Editions 456

Introduction to Editions 456

Creating an Edition 457

Modifying an Edition 460

Deleting a Saved Edition 462

Limiting the Number of Editions in a Dataset 464

Creating a Graphmart from a Dataset 466

Adding a Dataset to a Graphmart 471

Dataset FAQ 475

Working with Graphmarts 481

Creating a Graphmart 482

Copying a Graphmart 483

Graphmart Settings Reference 485

Creating an Elasticsearch Index from a Graphmart 489

Adding Data Layers to Graphmarts 497

Creating a New Layer 497

Cloning a Layer 500

Using Query Contexts 503

Defining Execution Conditions 507

Advanced Data Access Settings 509

Adding Steps to Layers 512

Directly Load a Data Source (Direct Load Step) 512
Create an Elasticsearch Index (Elasticsearch Indexing Step) 517
Take a Snapshot of an Index (Elasticsearch Snapshot Step) 521
Export Data to an FLDS (Export Step) 524
Load a Dataset from the Catalog (Load Dataset Step) 531
Pre-Compile a Query (Pre-Compile Query Step) 535
Create a Reusable Query Template 538
Run a Transformation Query (Query Step) 548
Infer New Data (RDFS+ Inference Step) 551
Validate the Data (Validation Step) 562
Construct a View of the Data (View Step) 567
Creating Data on Demand Endpoints 574
Creating an Auto-Generated Endpoint 574
Creating a Custom Endpoint 581
Sharing Access to Graphmarts 592
Sharing Concepts 592
Graphmart FAQ 601
Profiling Datasets and Graphmarts 604
Generating a Dataset Data Profile 605
Generating a Graphmart Data Profile 610
Data Profiling Metrics 614
Access & Analyze 627
Access Data with Hi-Res Analytics Dashboards 628
Introduction to Hi-Res Analytics 629
Getting Started: Explore and Visualize Your Data 637

Create a New Dashboard 637
Explore the Data 639
Create Visualizations of the Data 643
Working with Dashboards 648
Creating a Graphmart Dashboard 648
Creating a Network Navigator Dashboard 651
Configuring a Dashboard to Update in Batch Reporting vs. Interactive Mode 676
Capturing User-Defined Values in Dashboards 682
Working with Lenses 697
Creating a Lens 697
Cloning a Lens 821
Exporting a Lens 822
Deleting a Lens 825
Working with Filters 827
Adding a Cloud Filter 827
Adding a Date Range Filter 832
Adding a Hierarchy Filter 837
Adding a Limit Filter 841
Adding a List Filter 845
Adding a Numeric Range Filter 849
Adding a Presence Filter 854
Adding a Quartile Filter 858
Adding a Range Slider Filter 861
Adding a Relative Time Filter 864
Adding a Search Filter 868
Adding a Single Select List Filter 872

Adding a Types Filter ... 876
Calculating Values in Lenses and Filters ... 881
Combining Data from Multiple Classes ... 887
Searching for Text in Unstructured Documents ... 892
Supported Search Syntax ... 897
Sharing Access to Dashboards and Lenses ... 899
Configure Dashboard or Lens Permissions ... 902
Access Data with the Query Builder ... 906
Running SPARQL Queries in the Query Builder ... 907
Searching for Quads in the Query Builder ... 915
Access Data on Demand Endpoints ... 921
Accessing an Endpoint Programmatically ... 922
Accessing an Endpoint from an Application ... 926
Accessing Data via the OData API ... 928
Downloading the Anzo ODBC and JDBC Drivers ... 930
JDBC Driver Documentation ... 936
OData Reference ... 944
OData URL Conventions ... 944
Supported Query Operators ... 945
Access the SPARQL Endpoint ... 953
Access the HTTP Client Interface ... 965
Share Access to Artifacts ... 973
Version and Migrate Artifacts ... 980
Creating and Restoring Versions of Artifacts ... 981
Create a Backup Version ... 981

Restore a Backup Version ... 983
Exporting an Artifact ... 986
Making Properties Replaceable on Export ... 991
Importing Exported Versions of Artifacts ... 992
SPARQL Best Practices and Query Templates ... 996
SPARQL Best Practices ... 997
SPARQL Query Templates ... 1004
Function and Formula Reference ... 1010
String Functions ... 1011
BUSINESS_ENTITY_EXCLUDER ... 1013
CONCATENATE ... 1014
CONCATURL ... 1014
CONTAINS ... 1015
ENCODE_FOR_URI ... 1015
ESCAPEHTML ... 1016
FIND ... 1016
FINDREVERSE ... 1017
GROUP_CONCAT ... 1018
GROUPCONCAT ... 1018
LANG ... 1019
LANGMATCHES ... 1020
LCASE ... 1021
LEFT ... 1021
LEN ... 1022
LEVENSHTEIN_DIST ... 1022
LOWER ... 1023

MD5 1023
MID 1024
REGEX 1024
REGEXP_SUBSTR 1025
REPLACE 1026
RIGHT 1027
SEARCH 1028
SHA1 1028
SHA224 1029
SHA256 1029
SHA384 1030
SHA512 1030
STRAFTER 1031
STRBEFORE 1031
STRDT 1032
STRENDS 1033
STRLANG 1033
STRLEN 1034
STRSTARTS 1034
STRUUID 1035
SUBSTITUTE 1035
SUBSTR 1036
TOURI 1037
TRIM 1037
UCASE 1038
UPPER 1038

Math Functions 1040
ABS 1042
ADD 1043
AVERAGEIF 1043
AVERAGEIFS 1044
AVG 1045
CEILING 1045
COS 1046
DIVIDE 1046
EQUAL 1047
EXP 1047
FACT 1048
FLOOR 1048
GE 1049
GT 1050
HAMMING_DIST 1050
HAVERSINE_DIST 1051
LE 1052
LN 1052
LOG 1053
LOG2 1053
LT 1054
MAXVAL 1055
MINVAL 1055
MOD 1056
MULTIPLY 1056

NOT_EQUAL 1057
NPV 1057
NUMERIC-ADD 1058
NUMERIC-SUBTRACT 1059
PI 1059
POWER 1060
PRODUCT 1060
QUOTIENT 1061
RAD 1061
RAND 1062
RANDBETWEEN 1062
ROUND 1063
ROUNDDOWN 1063
ROUNDUP 1064
SIN 1064
SQRT 1065
SUBTRACT 1066
SUM 1066
SUMIF 1067
SUMIFS 1068
SUMPRODUCT 1068
SUMSQ 1069
TAN 1069
Aggregate Functions 1071
AVERAGEIF 1072
AVERAGEIFS 1073

AVG 1074
CHOOSE_BY_MAX 1075
CHOOSE_BY_MIN 1075
COUNT 1076
COUNT_DISTINCT 1077
COUNTIF 1077
COUNTIFS 1078
GROUP_CONCAT 1079
GROUPCONCAT 1079
MAX 1080
MEDIAN 1081
MIN 1081
MODE 1082
MODEPERCENT 1082
PERCENTILE_CONT 1083
PERCENTILE_DISC 1084
PRODUCT 1084
SAMPLE 1085
STDEV 1085
STDEVP 1086
SUM 1086
SUMIF 1087
SUMIFS 1088
SUMPRODUCT 1088
SUMSQ 1089
VAR 1089

VARP 1090
WEIGHTEDAVERAGE 1090
Date and Time Functions 1092
DATE 1094
DATEPART 1094
DATETIME (or xsd:dateTime) 1095
DAY 1095
DAYSFROMDURATION 1096
DUR_TO_MILLIS 1096
DURATION 1097
DURATIONFORMAT 1097
DURATIONPERIODFORMAT 1098
FORMATDATE 1100
HOUR 1100
MASKEDDATETIME 1101
MILLIS 1102
MINUTE 1102
MONTH 1103
NOW 1103
NOWMILLIS 1104
PARSEDATETIME 1104
SECOND 1105
TIME 1106
TIMEPART 1107
TIMEVALUE 1107
TIMEZONE 1108

TODAY 1108
TZ 1108
WEEKDAY 1109
WEEKNUM 1110
xsd:date 1110
YEAR 1111
YEARMONTH 1111
Casting Functions 1113
BNODE 1114
BOOLEAN 1115
BYTE 1115
CONCATURL 1116
DATATYPE 1116
DATETIME (or xsd:dateTime) 1117
DATEVALUE 1117
DECIMAL 1118
DOUBLE 1118
DURATION 1119
DURATIONFORMAT 1119
ENCODE_FOR_URI 1120
FLOAT 1121
FORMATDATE 1121
FORMATFRACTION 1122
FORMATNUMBER 1123
INT 1123
INTEGER 1124

LONG 1124
PARSEDATETIME 1125
RAD 1126
SERIALIZE 1127
SHORT 1127
STR 1128
TEXT 1128
TIMEVALUE 1129
TOURI 1129
UUID 1130
xsd:date 1130
Logical Functions 1131
AND 1132
BOUND 1132
CASE 1133
COALESCE 1134
EQUAL 1134
IF 1135
IFERROR 1136
IN 1136
NOT 1136
NOT_EQUAL 1137
NOT_IN 1138
OR 1138
PARTITIONINDEX 1139
SAMETERM 1140

UNBOUND 1140
Informational or Testing Functions 1142
CONTAINS 1143
GE 1143
GT 1144
ISBLANK 1145
ISDATATYPE 1145
ISERROR 1146
ISIRI 1147
ISLITERAL 1147
ISNUMERIC 1148
ISURI 1148
LANG 1149
LANGMATCHES 1149
LE 1150
LOCALNAME 1150
LT 1151
METADATAGRAPHURI 1152
NAMESPACE 1152
SAMETERM 1153
Hash Functions 1154
MD5 1154
SHA1 1155
SHA224 1155
SHA256 1156
SHA384 1156

SHA512 1157
Window Aggregate and Ranking Functions 1158
WINDOW_AVG 1159
WINDOW_COUNT 1161
WINDOW_MAX 1162
WINDOW_MIN 1164
WINDOW_NTILE 1166
WINDOW_PERCENTILE 1167
WINDOW_PERCENTILE_CONT 1167
WINDOW_PERCENTILE_DISC 1168
WINDOW_PRODUCT 1169
WINDOW_QUARTILE 1171
WINDOW_SUM 1172
Develop 1175
Anzo Rest API 1176
Introduction to the API 1177
Viewing the API Documentation 1180
Enabling Cross-Origin Resource Sharing 1181
Step Type Schemas 1183
Direct Load Step 1183
Elasticsearch Indexing Step 1187
Elasticsearch Snapshot Step 1191
Export Step 1195
Load Dataset Step 1202
Pre-Compile Query Step 1207
Query-Driven Templated Step 1211

Query Step 1215
RDFS+ Inference Step 1219
Templated Step 1222
Validation Step 1227
Anzo Java SDK 1233

About This Doc

This document provides guidance on onboarding, modeling, blending, and accessing data in Anzo.

> **Tip**
> You can view the contents of this guide as well as release notes, getting started, deployment, and administration documentation online at docs.cambridgesemantics.com. You can also find PDF versions of the getting started, deployment, and administration documentation here.

The following list introduces the sections in this guide.

- Onboard & Virtualization: Provides instructions for connecting to data sources and onboarding or virtualizing structured and semi-structured data as well as details about creating pipelines to onboard unstructured data.
- Model: Provides conceptual information about data models, describes model requirements, and includes instructions for creating, editing, uploading, and downloading models.
- Blend: Includes information about the Datasets catalog and instructions on creating and managing datasets and editions. Also provides instructions on creating graphmarts, data layers, steps, Data on Demand endpoints, and managing graphmart permissions, as well as creating data profiles for datasets and graphmarts.
- Access & Analyze: Discusses the options for accessing and analyzing your data, including creating Hi-Res Analytics dashboards, using the Query Builder to run SPARQL queries, and accessing the SPARQL endpoint or Data on Demand endpoints. Also includes information about sharing, versioning, and migrating artifacts.
- Develop: Includes information for developers about the Anzo REST API and Java SDK.

Onboard & Virtualization

The topics in this section provide instructions for onboarding unstructured data and onboarding or virtualizing structured or semi-structured data. For instructions on importing files that are in RDF format (Turtle or N-Triple), see Creating a Dataset from RDF Files.

In this section:

Onboard Structured Data 21

Onboard Unstructured Data 355

Onboard Structured Data

For structured and semi-structured data sources—databases, HTTP REST endpoints, CSV, JSON, XML, Parquet, and SAS files, and raw data—there are two ways to onboard and/or virtualize the data:

Automated Direct Data Load

Data from databases and CSV, JSON, XML, Parquet, and SAS files can be onboarded to Anzo via the automated direct data load workflow. This workflow follows an extract, load, and transform (ELT) process to ingest data. In the ELT workflow, data sources are extracted and loaded to graphmarts. Data layers and Direct Load Steps with SPARQL queries are automatically generated to transform and blend the data to analytics-ready knowledge graphs. The AnzoGraph Graph Data Interface (GDI) Java plugin is used to connect to the sources, create a model, and generate the data layer queries. For more information about this workflow, see Onboarding Data with the Automated Workflow.

Load or Virtualization with SPARQL Queries

Raw data and data from databases, HTTP REST endpoints, and CSV, JSON, XML, Parquet, and SAS files can be onboarded or virtualized by invoking the Graph Data Interface (GDI) with manually written SPARQL queries. The GDI is extremely flexible, allowing you to connect directly to sources and control all aspects of the extract, load, and transform process. You can onboard data into Anzo by adding a Direct Load Step query. Or you can create a virtual graph by adding a View step query. By creating views, you can access the source data exactly when it is needed, without requiring you to ingest all of the data into Anzo up front. For more information about this workflow, see Onboarding or Virtualizing Data with SPARQL Queries.

> **Note**
> Whether you plan to use the automated workflow or manually invoke the GDI, the first step in the process is to connect your data sources to Anzo and onboard the schemas. Start with Adding Data Sources.

Adding Data Sources

This topics in this section provide instructions for connecting to structured and semi-structured data sources and working with schemas.

> **Tip**
> For instructions on onboarding RDF files (Turtle or N-Triple files) to Anzo, see Creating a Dataset from RDF Files.

In this section:

Connecting to a Database 23

Defining a Database Schema 45

Adding an HTTP or SPARQL Data Source 58

Adding a CSV Data Source 60

Adding a JSON Data Source 68

Adding an XML Data Source 75

Adding a SAS Data Source 81

Adding a Parquet Data Source 88

Configuring a CSV or Parquet Source for Incremental Processing 92

Assigning Primary Keys in a Schema 97

Adding Foreign Keys 100

Connecting to a Database

The topics in this section provide instructions on configuring connections to the database types that Anzo and AnzoGraph support by default.

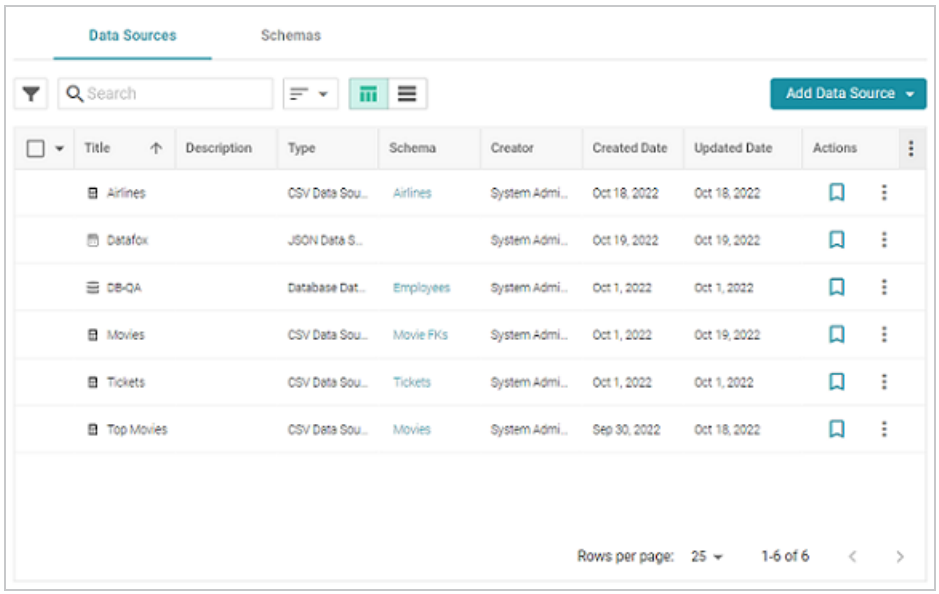
> **Tip**
> To connect to other types of databases, first add JDBC drivers to Anzo (see Uploading a Plugin in the Administration Guide) and AnzoGraph (see Deploy Drivers for Custom Database Sources in the Deployment Guide). Then you can follow any of the instructions in this section to configure the connection. All database connections are similar, but settings may vary based on the JDBC driver.

- Connecting to a Databricks Source
- Connecting to a DB2 Source
- Connecting to an MSSQL Source
- Connecting to an Oracle Source
- Connecting to a PostgreSQL Source
- Connecting to a Snowflake Source
- Connecting to a Sybase Source

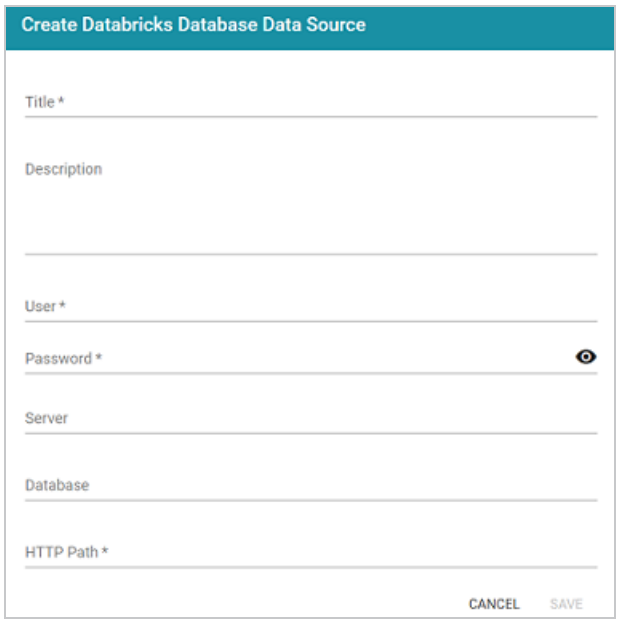
Connecting to a Databricks Source

Follow the instructions below to connect to a Databricks database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



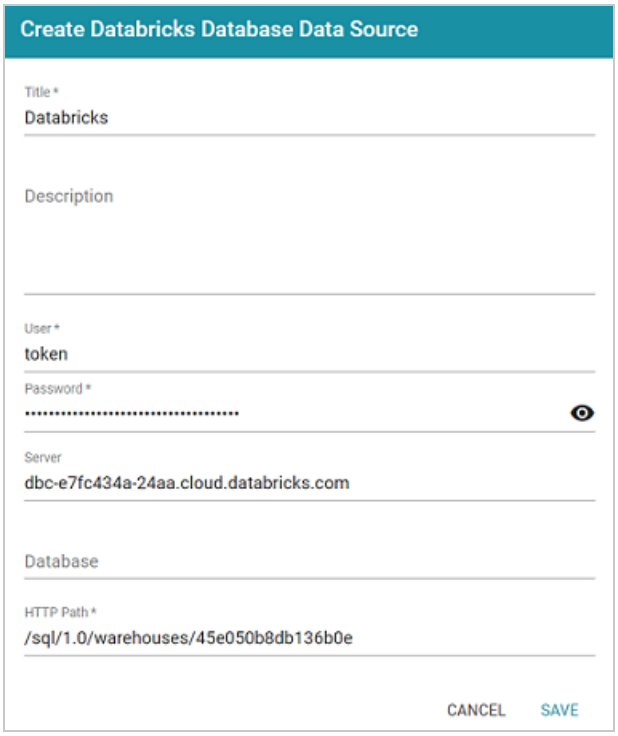
2. Click the **Add Data Source** button, select **Database**, and then select **Databricks Database Data Source**. The Create screen is displayed:



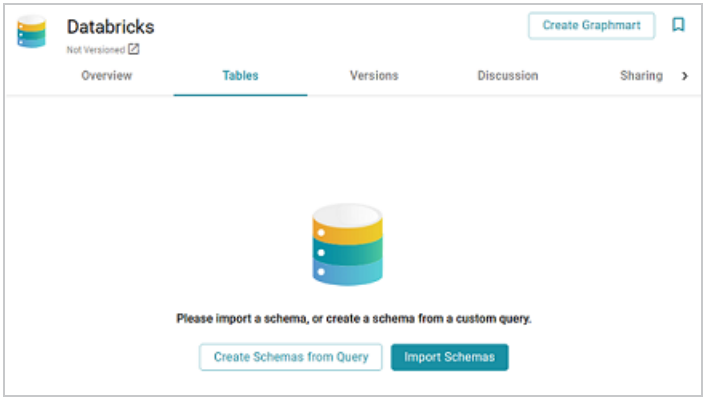
3. Complete the required fields and configure any optional settings. The list below describes each setting.
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.

- **User**: The user name to use for logging in to the database.
- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.
- **HTTP Path**: The Databricks compute resources URL.

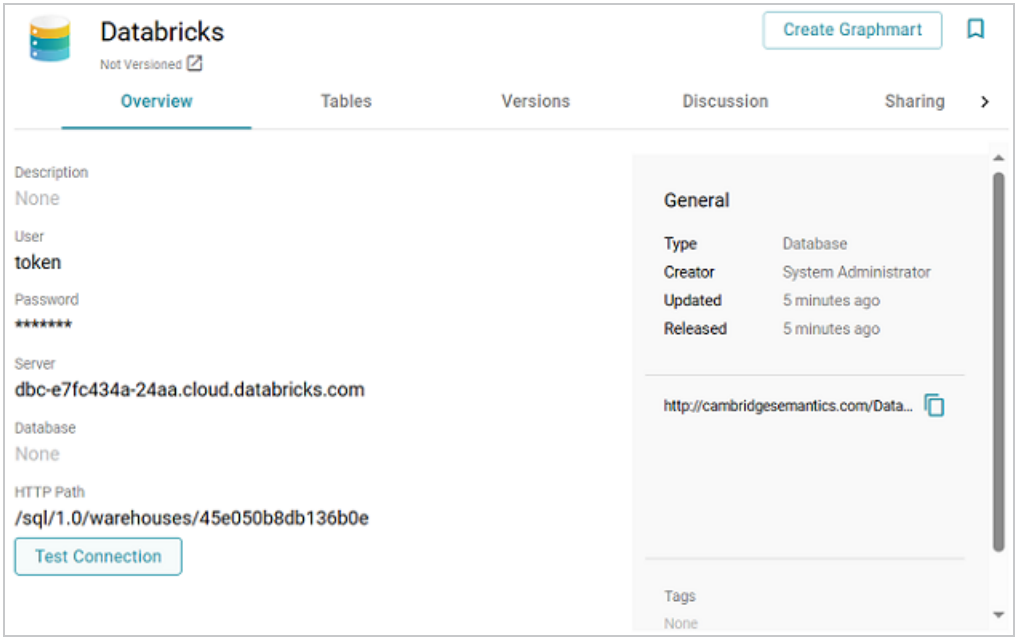
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

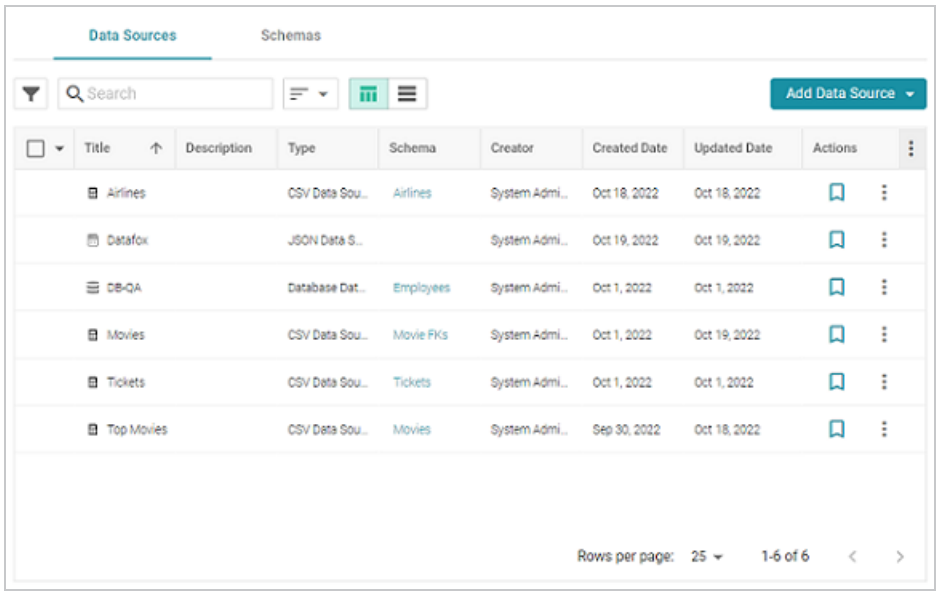
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

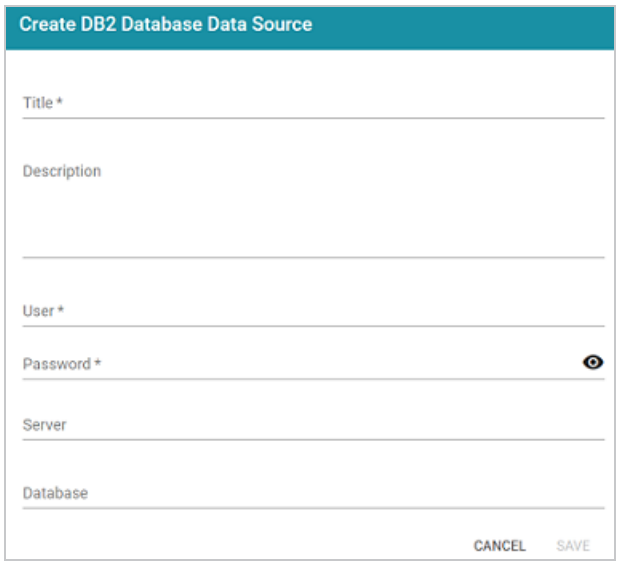
Connecting to a DB2 Source

Follow the instructions below to connect to a DB2 database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



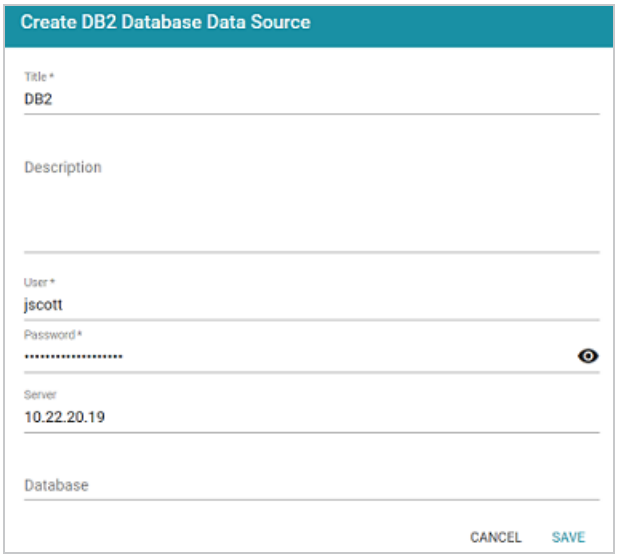
2. Click the **Add Data Source** button, select **Database**, and then select **DB2 Database Data Source**. The Create screen is displayed:



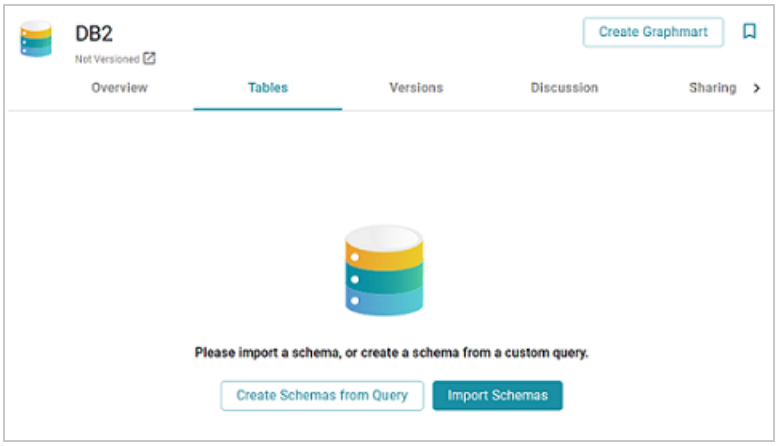
3. Complete the required fields and configure any optional settings. The list below describes each setting.
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.
   - **User**: The user name to use for logging in to the database.

- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.

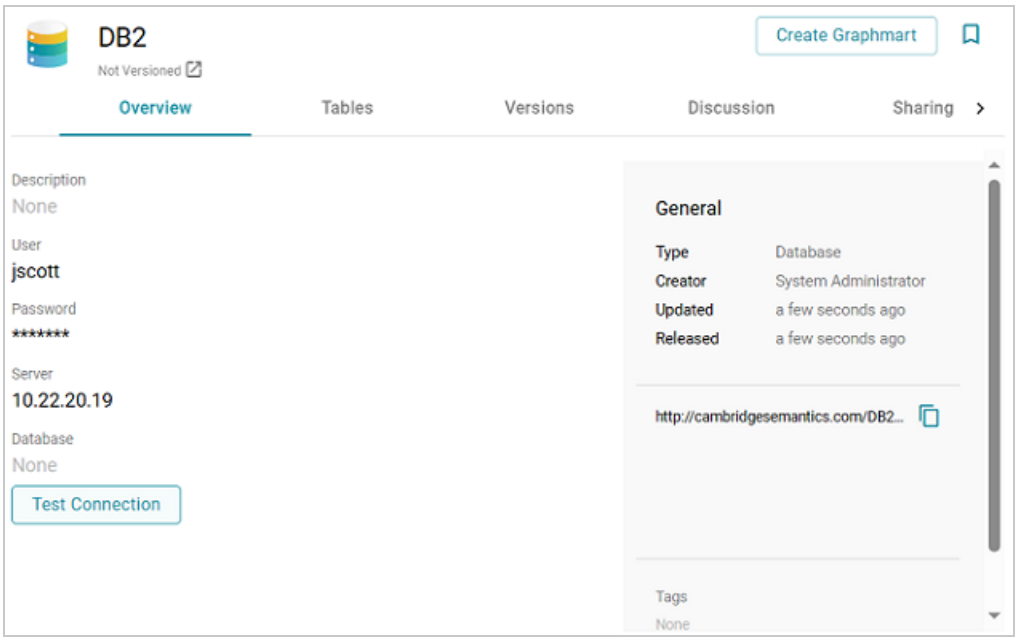
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

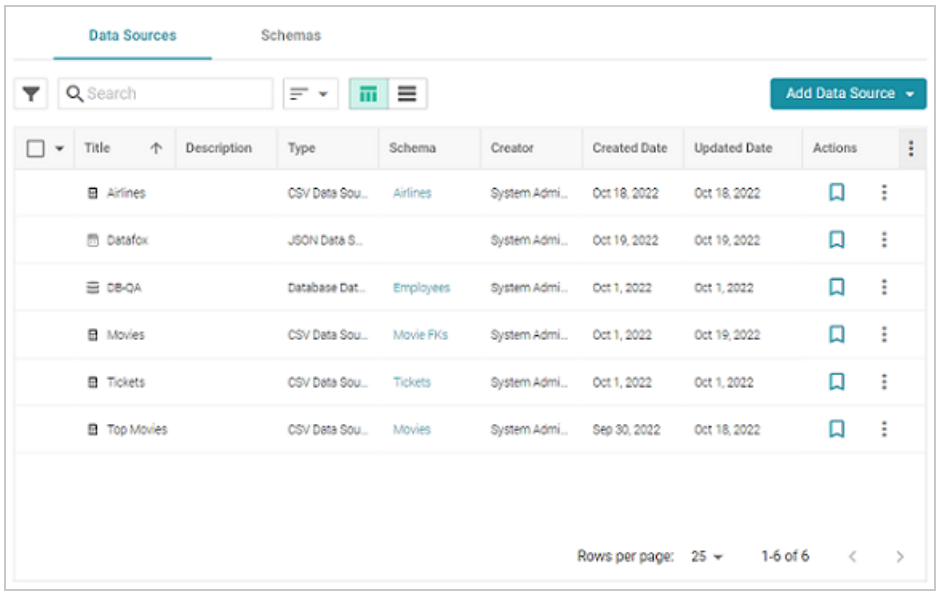
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

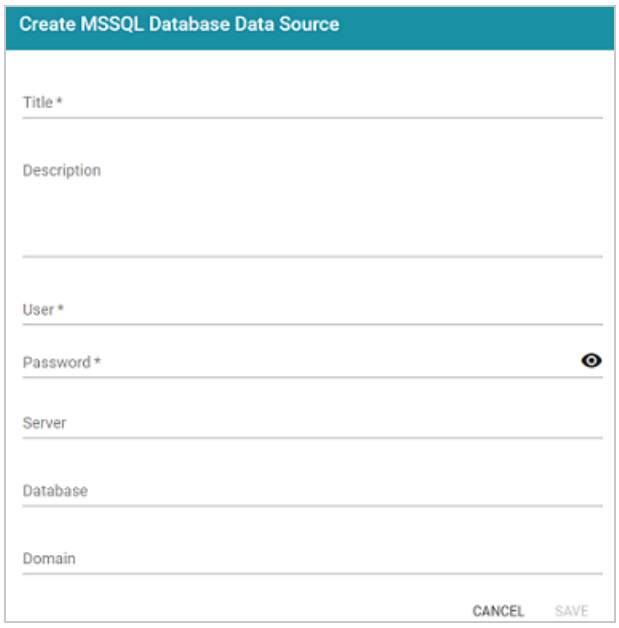
Connecting to an MSSQL Source

Follow the instructions below to connect to a SQL Server or MSSQL database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the **Add Data Source** button, select **Database**, and then select **MSSQL Database Data Source**. The Create screen is displayed:

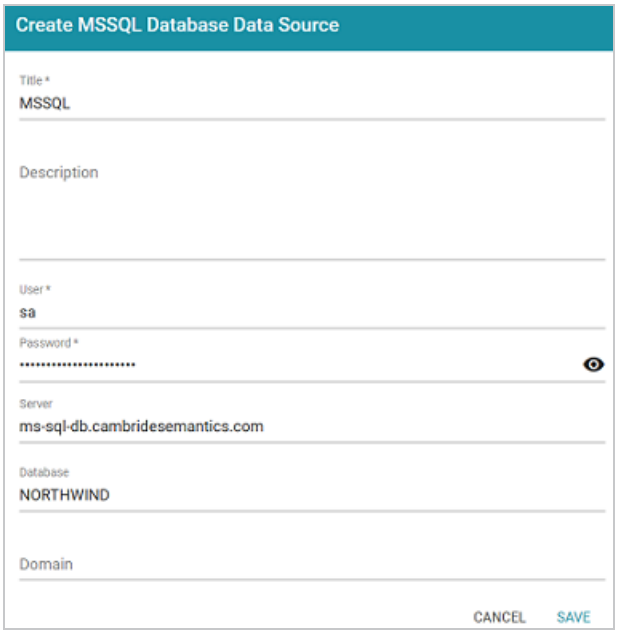


3. Complete the required fields and configure any optional settings. The list below describes each setting.

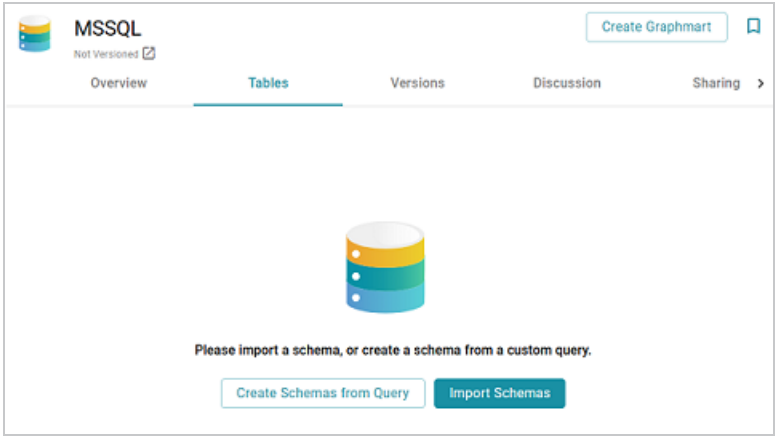
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.

- **User**: The user name to use for logging in to the database.
- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.
- **Domain**: When using NTLM authentication, this is the domain to authenticate against.

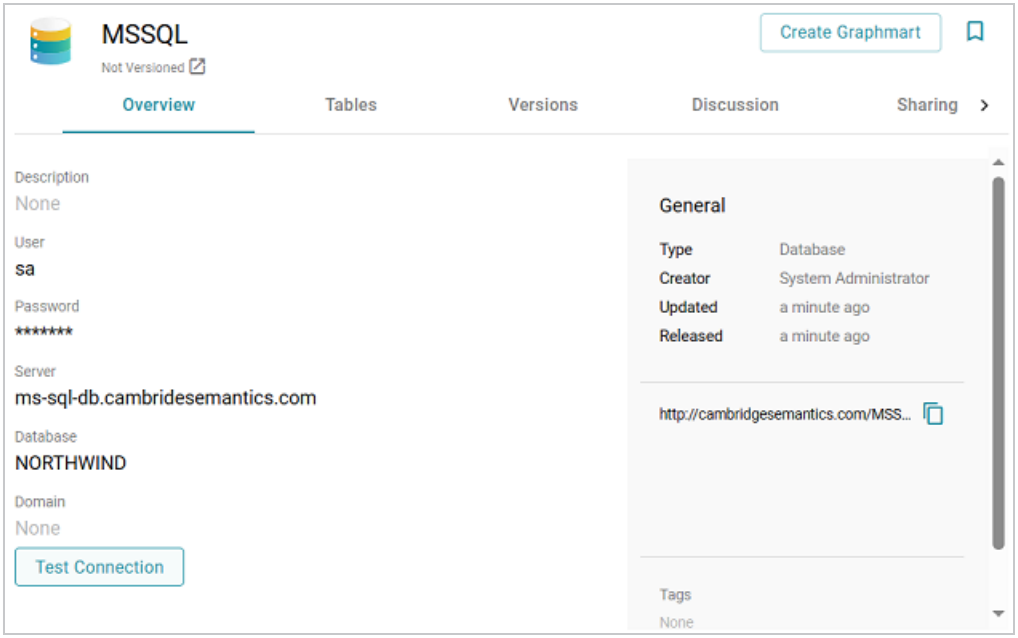
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

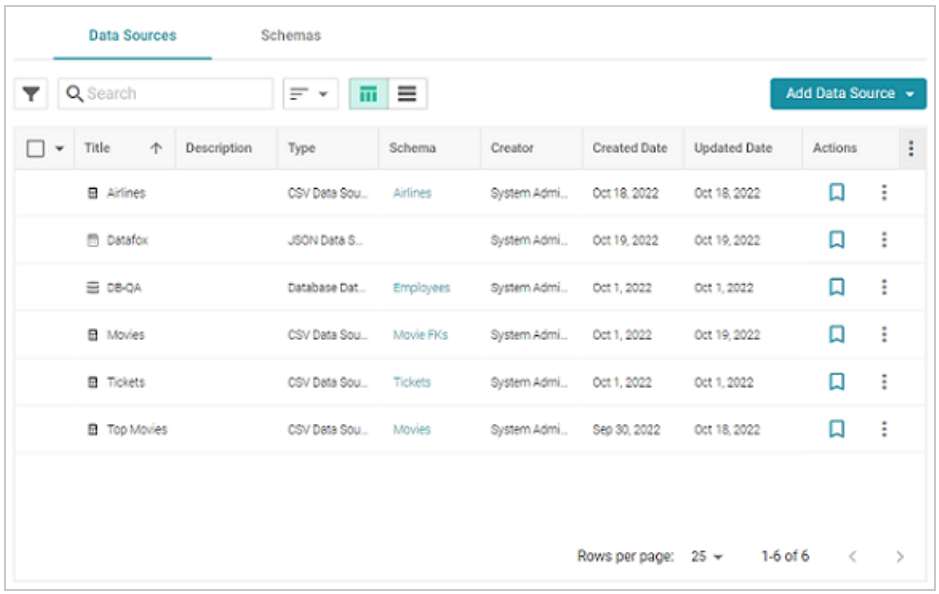
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

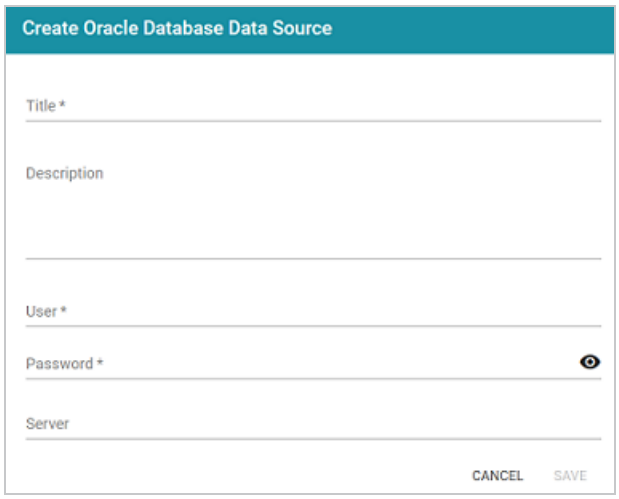
Connecting to an Oracle Source

Follow the instructions below to connect to an Oracle database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



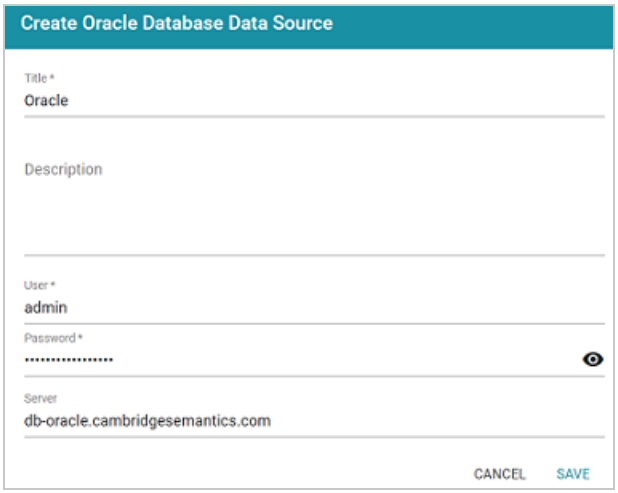
2. Click the **Add Data Source** button, select **Database**, and then select **Oracle Database Data Source**. The Create screen is displayed:



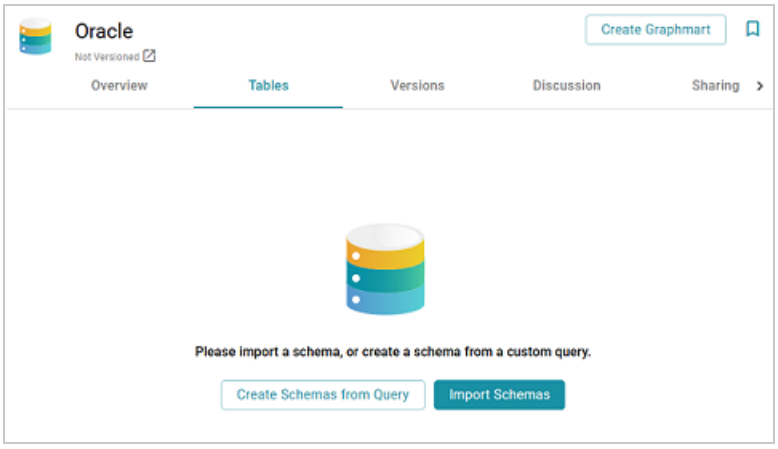
3. Complete the required fields and configure any optional settings. The list below describes each setting.
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.
   - **User**: The user name to use for logging in to the database.

- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.

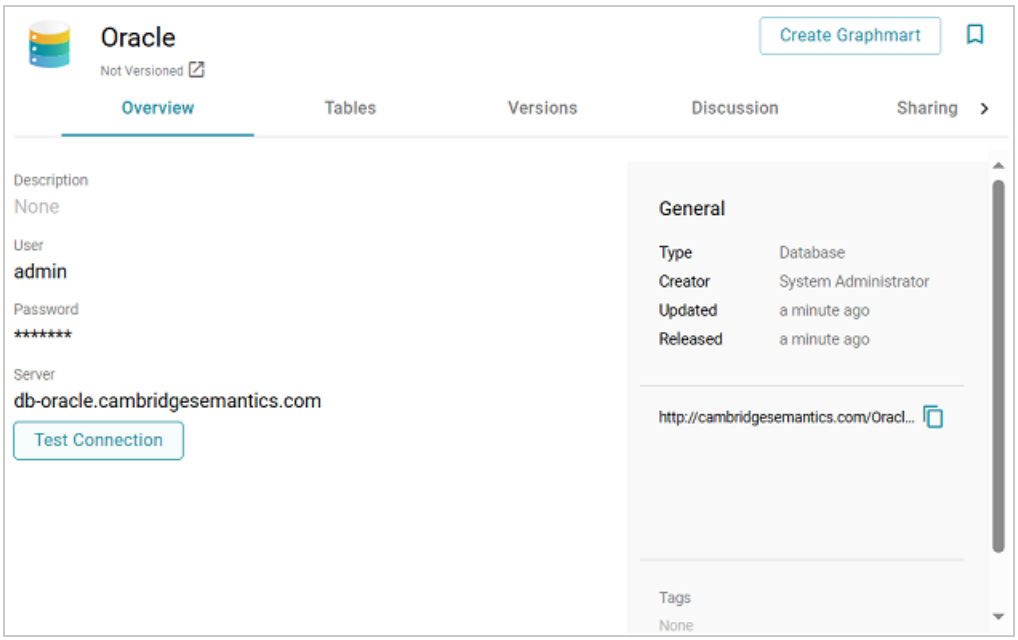
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

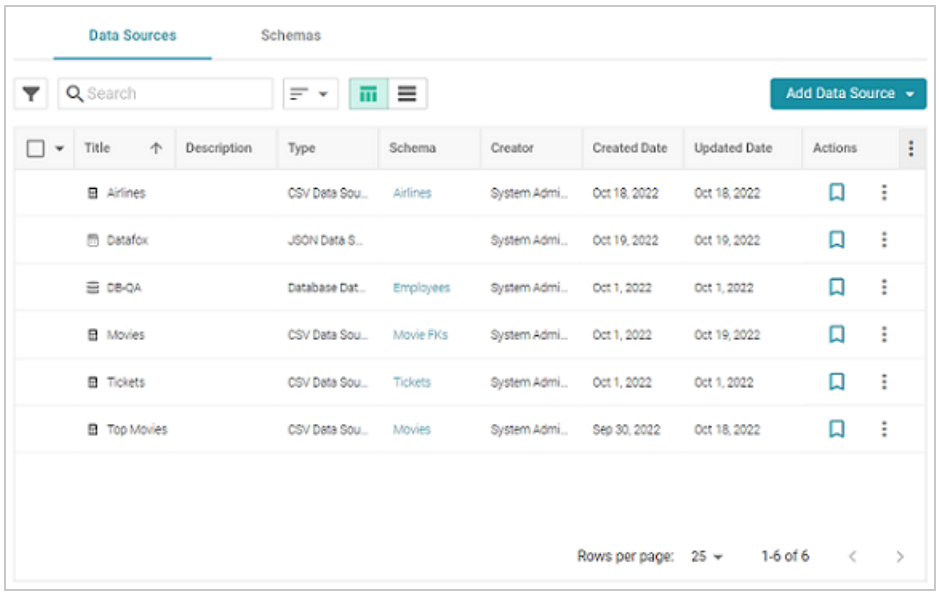
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

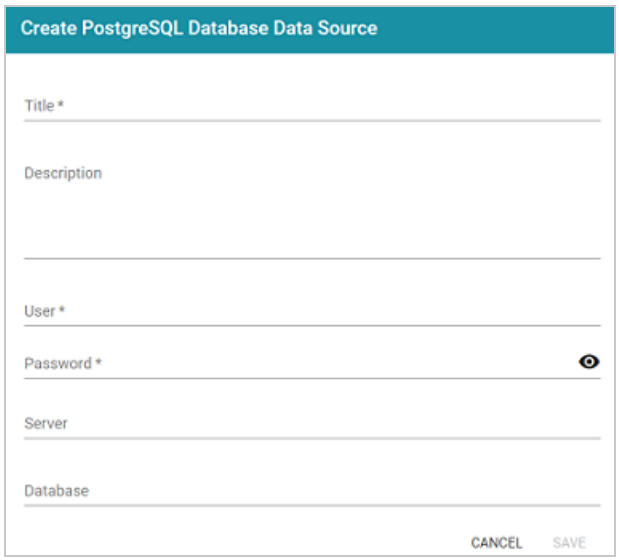
Connecting to a PostgreSQL Source

Follow the instructions below to connect to a PostgreSQL database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the **Add Data Source** button, select **Database**, and then select **PostgreSQL Database Data Source**. The Create screen is displayed:

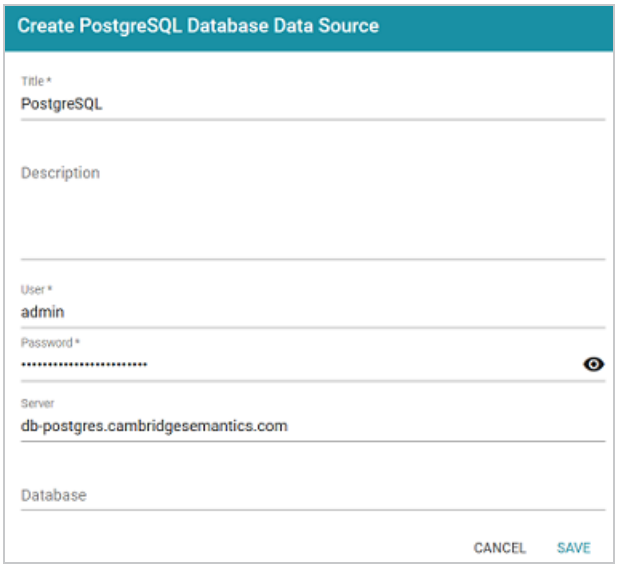


3. Complete the required fields and configure any optional settings. The list below describes each setting.

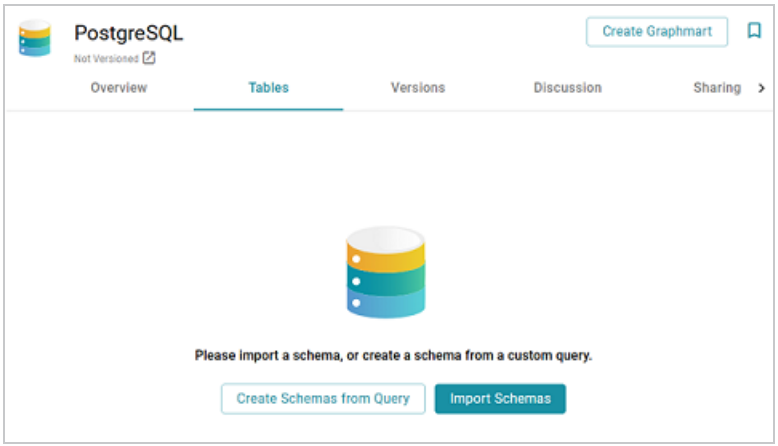
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.
   - **User**: The user name to use for logging in to the database.

- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.

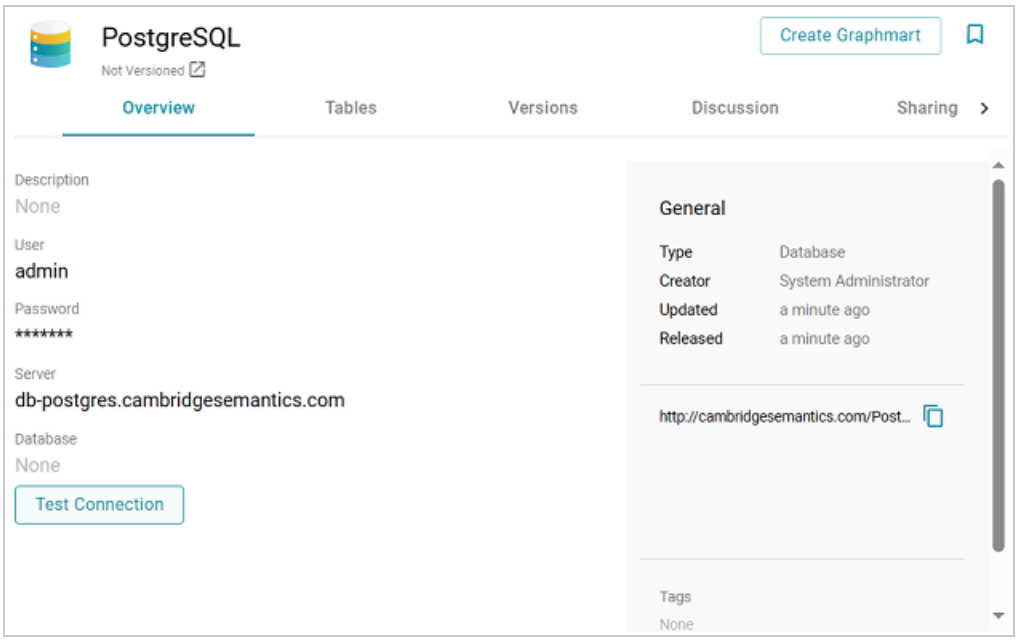
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

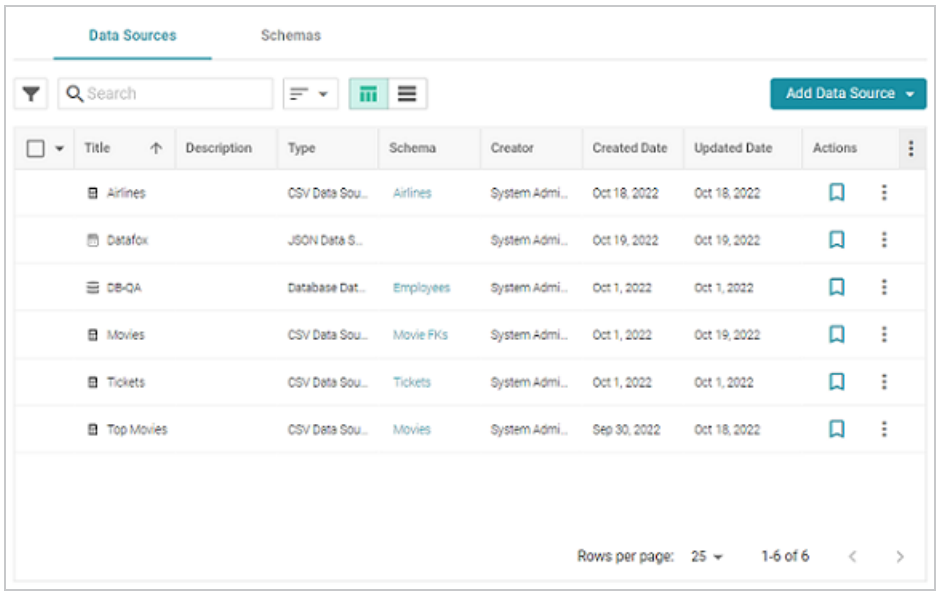
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

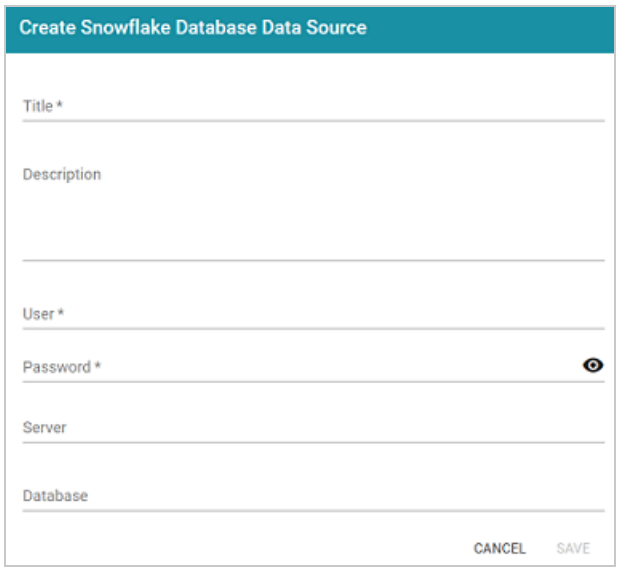
Connecting to a Snowflake Source

Follow the instructions below to connect to a Snowflake database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



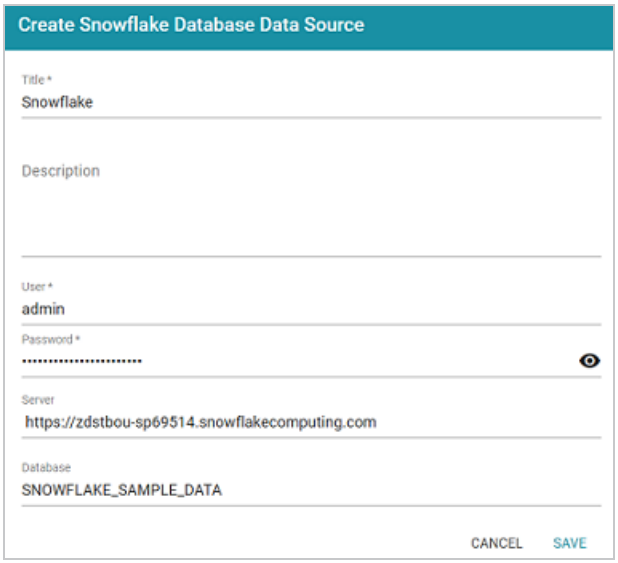
2. Click the **Add Data Source** button, select **Database**, and then select **Snowflake Database Data Source**. The Create screen is displayed:



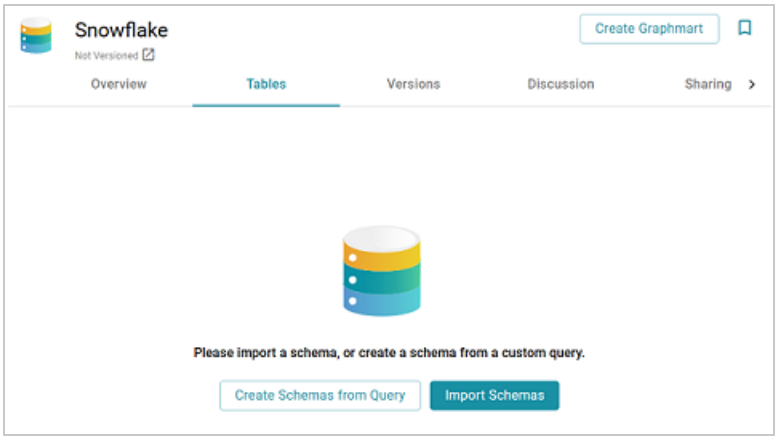
3. Complete the required fields and configure any optional settings. The list below describes each setting.
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.
   - **User**: The user name to use for logging in to the database.

- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.

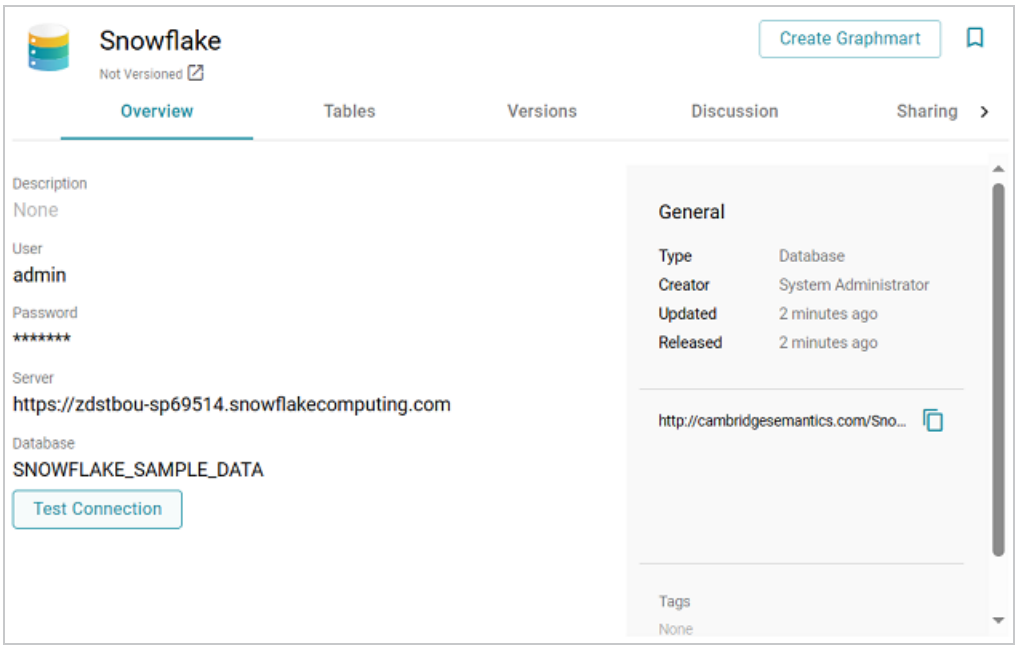
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source details until the connection is successful.

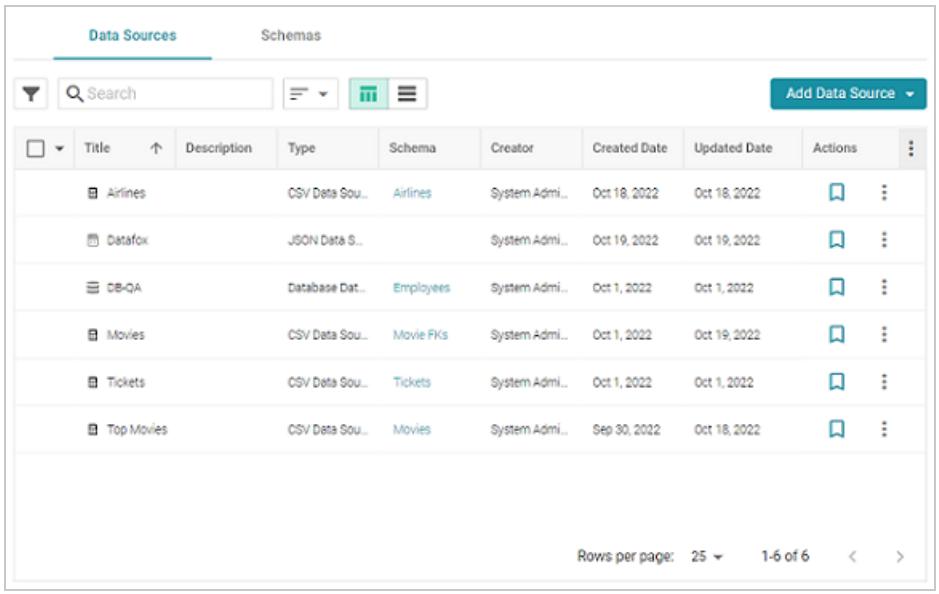
> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

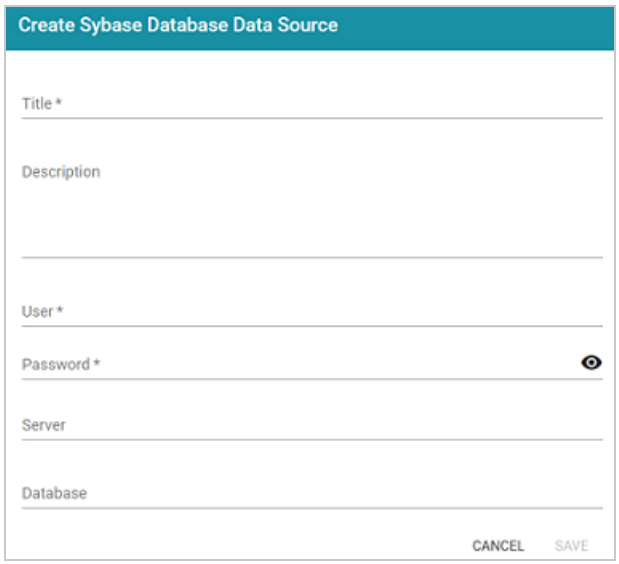
Connecting to a Sybase Source

Follow the instructions below to connect to a Sybase database.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



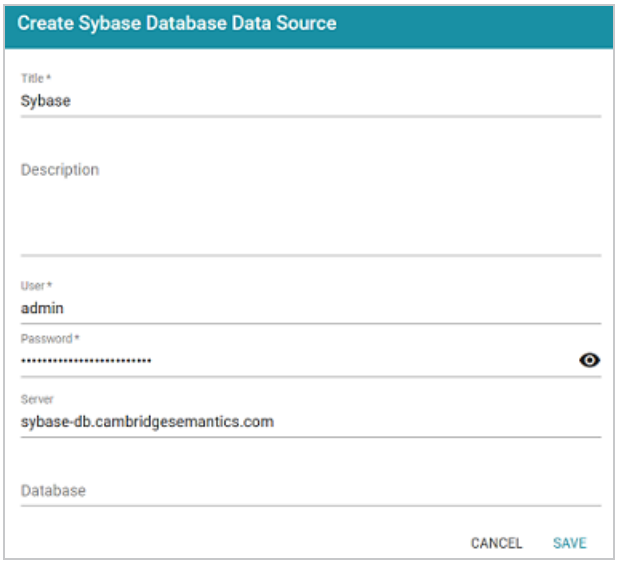
2. Click the **Add Data Source** button, select **Database**, and then select **Sybase Database Data Source**. The Create screen is displayed:



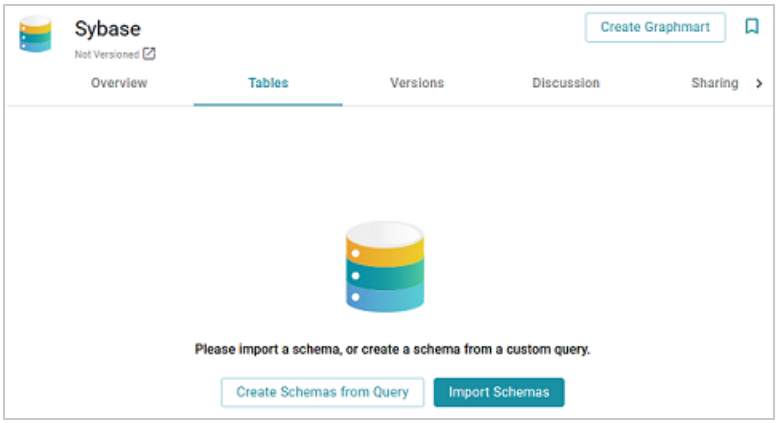
3. Complete the required fields and configure any optional settings. The list below describes each setting.
   - **Title**: The name to give to this data source connection.
   - **Description**: An optional description of the connection.
   - **User**: The user name to use for logging in to the database.

- **Password**: The password for the user name.
- **Server**: The host name or IP address for the source.
- **Database**: An optional partition that contains the data to onboard.

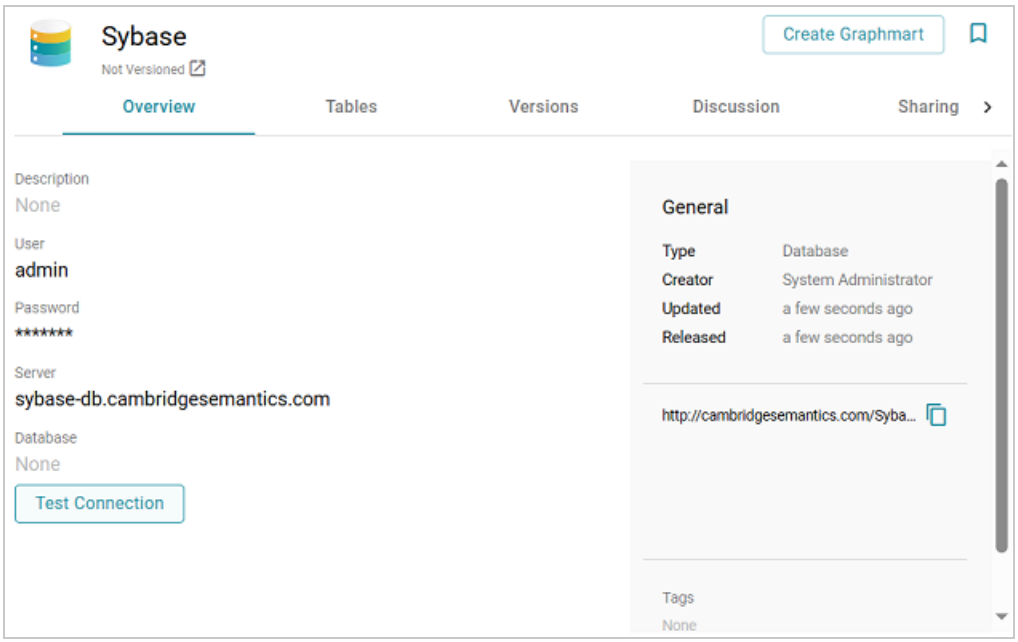
The image below shows an example of a completed configuration.



4. Click **Save** to save the data source connection. The **Tables** tab is displayed.



5. Before proceeding to select or create a schema, you may want to test connectivity to the database. To do so, click the **Overview** tab and then click the **Test Connection** button at the bottom of the screen.



If the connection fails, adjust the data source connection details until the connection is successful.

> **Note**
> Only the connection between Anzo and the database is tested. Connectivity is not tested between AnzoGraph and the database. Cambridge Semantics recommends that you perform a manual test to ensure that AnzoGraph can also connect to the database.

After connecting to the source, the next step is to add one or more schemas. See Defining a Database Schema for instructions.

Defining a Database Schema

The schema defines the source data to onboard. There are multiple options available for defining a database schema. You can import a predefined schema from the database, you can write a static SQL query that defines the data to onboard, or, if you want to import data incrementally, you can write an incremental SQL query that ingests a subset of the data.

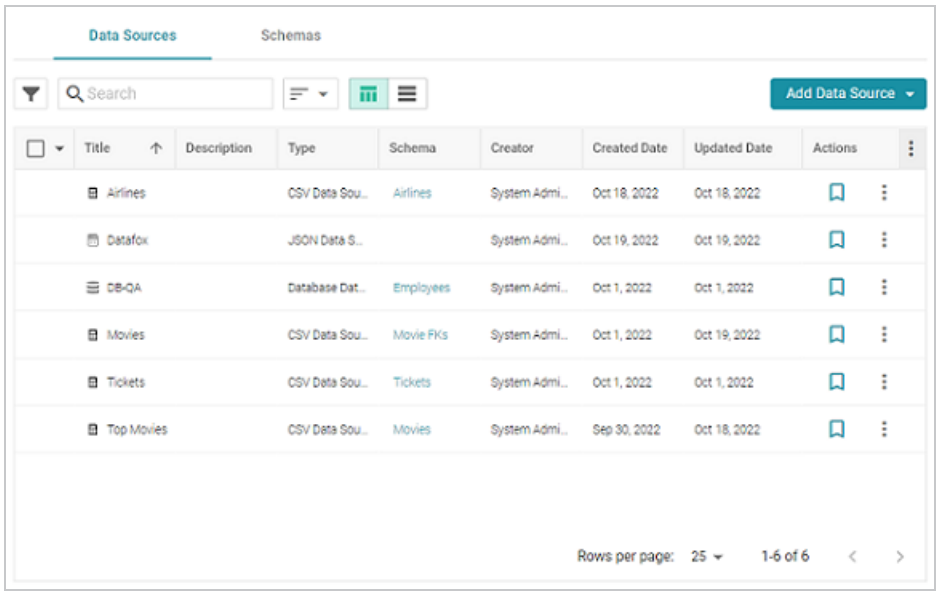
> **Note**
> You can import or create up to 5 schemas per database data source. To include more than 5 schemas, create another data source for the additional schemas.

> **Tip**
> By default, Anzo is configured to exclude Views from the list of available Schemas to import. For information about including Views as tables that can be imported, see Including Views as Schemas for Database Data Sources in the Administration Guide.

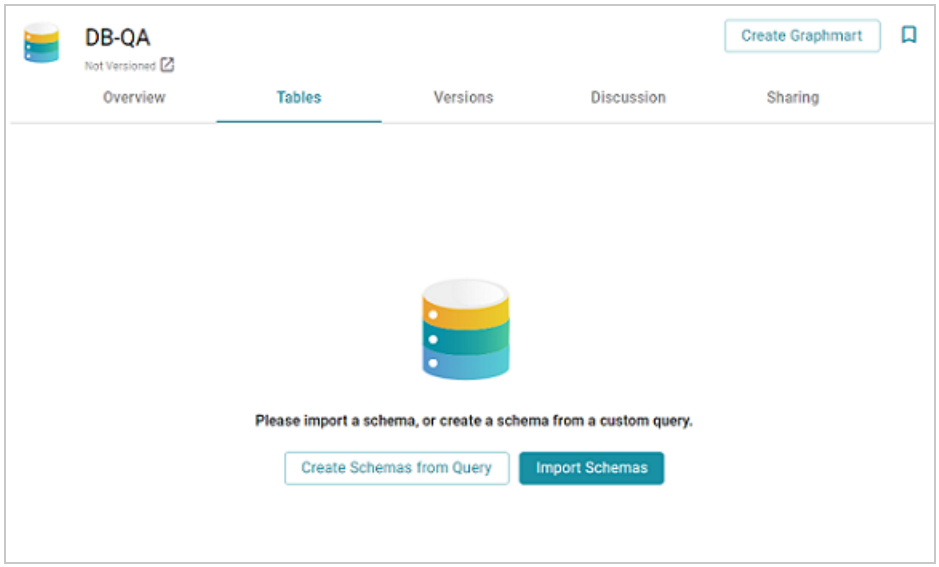
- Import a Predefined Schema
- Create a Schema from an SQL Query
- Create an Incremental Schema

Import a Predefined Schema

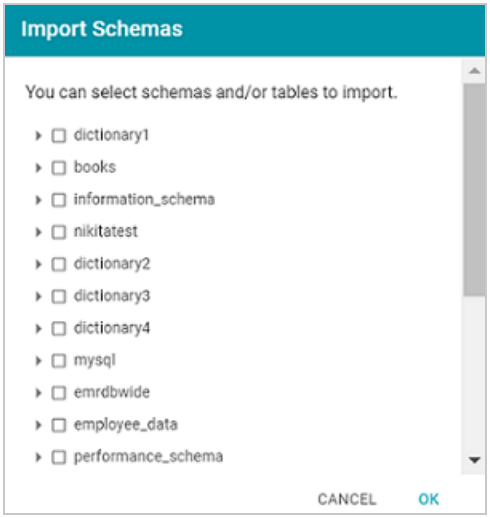
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the database for which you want to import a schema. Anzo displays the Tables tab for the source. For example, the image below shows the Tables tab before any schemas have been added:

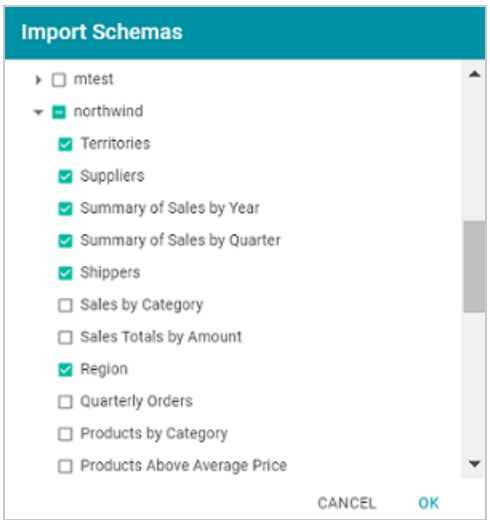


3. Click the **Import Schemas** button. Anzo displays the Import Schemas dialog box, which lists any predefined schemas in the database. For example:



> **Note**
> If you do not see a schema that you expect to see, make sure that you have the appropriate permissions to access to the data source.

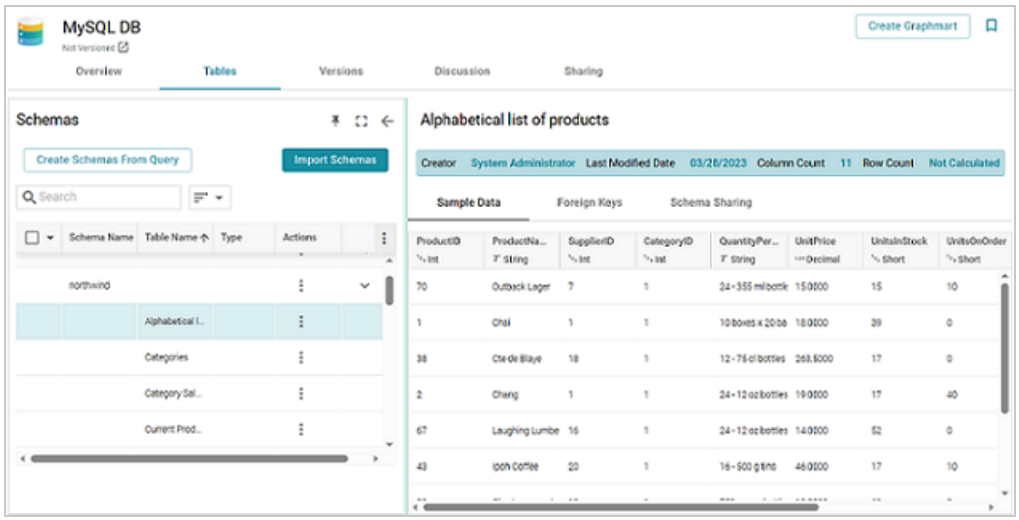
4. To import entire schemas, select the checkbox next to each schema name that you want to import. If you want to import a subset of the tables in a schema, expand the schema and select the checkbox next to each table that you want to include. For example:



5. When you have finished selecting schemas, click **OK**. Anzo imports the selected schemas and lists them on the Tables tab. You can expand a schema to view its tables. Selecting a row in the schema displays the sample data on the right side of the screen.

Important

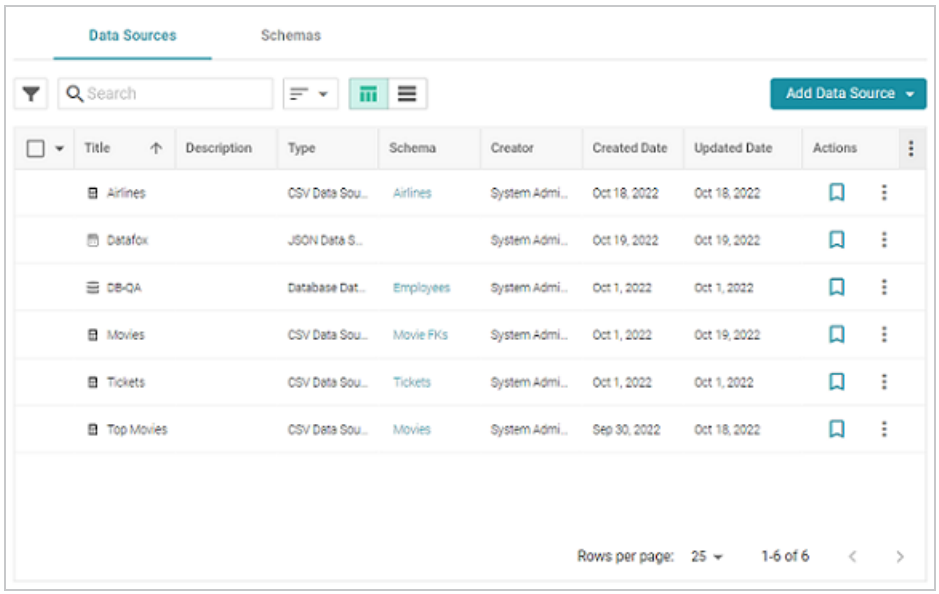
The automated data load workflow ignores all changes that are made to the schema on the Tables screen—except for changes to primary and foreign keys. For example, if you edit a column heading to change its semantic type, that change is disregarded when the graphmart is created. Only the original type from the data source is considered. If you add or change primary and foreign keys on the Tables screen, however, the automated data load workflow will retain those changes.



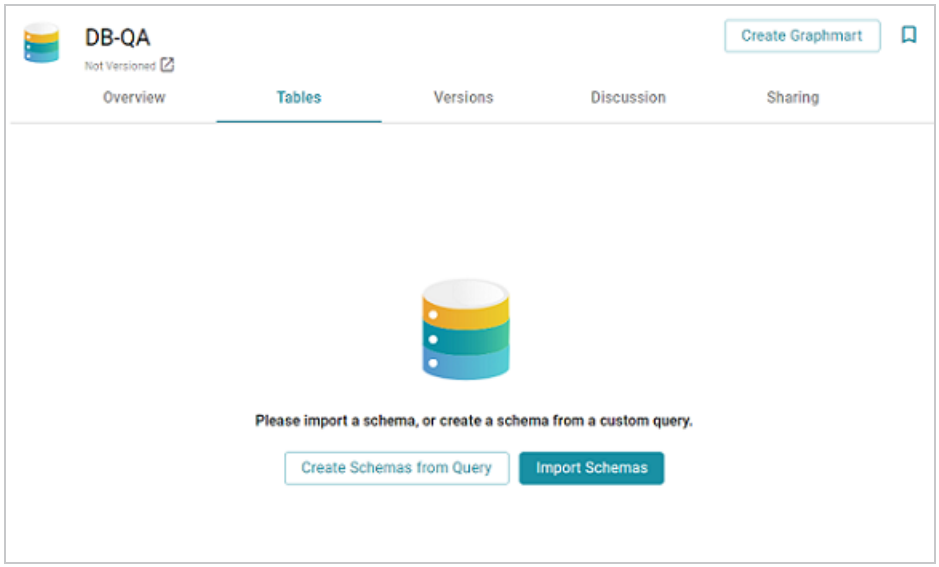
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

Create a Schema from an SQL Query

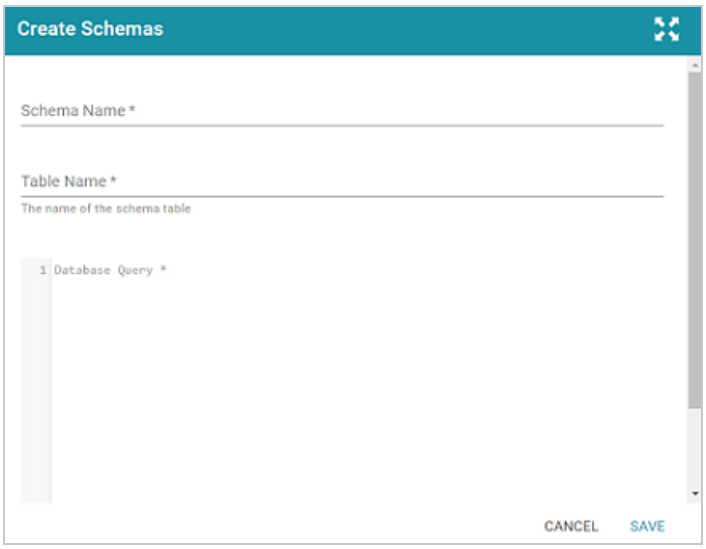
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the database for which you want to create a schema. Anzo displays the Tables tab for the source. For example, the image below shows the Tables tab before any schemas have been created:



3. Click the **Create Schemas From Query** button. Anzo displays the Create Schemas dialog box:

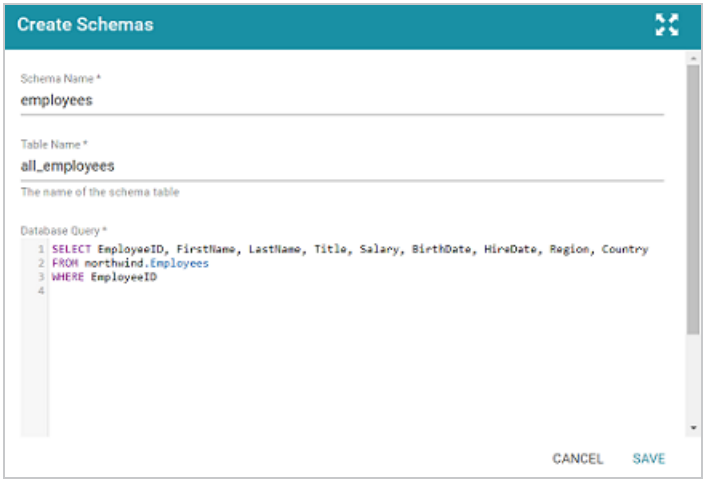


4. In the Create Schemas dialog box, specify a name for this schema in the **Schema Name** field.
5. In the **Table Name** field, specify a name for the table in the schema that the query will create.
6. Type the SQL statement in the text box. The statement can include any functionality that the source database supports. Anzo does not validate the SQL.

> **Note**
> If the SQL query requires quotes around values, such as '2010-01-01' or 'TestValue', make sure that you use single quotes ('). For example:
>
> ```
> SELECT * FROM Movies WHERE production_day='2021-08-01'
> ```
>
> Including double quotes (") in a schema query results in an error when the query is run.

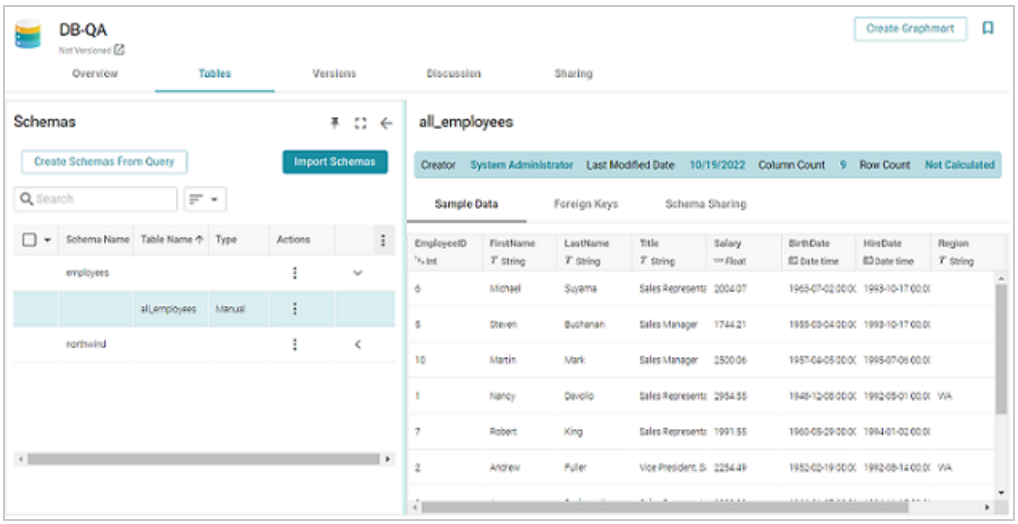
The following example creates a schema named employees. A table named all_employees will be created in the schema, and the table is created from the following SQL query:

```
SELECT EmployeeID, FirstName, LastName, Title, Salary, BirthDate, HireDate,
Region, Country
FROM northwind.Employees
WHERE EmployeeID
```



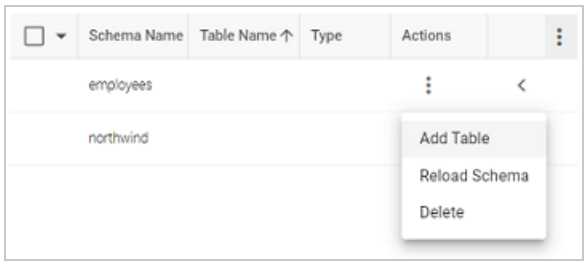
7. Click **Save** to save the query. Anzo creates the new schema and adds it to the list of schemas on the Tables screen. Selecting the schema displays sample data on the right side of the screen.

> **Important**
>
> The automated data load workflow ignores all changes that are made to the schema on the Tables screen—except for changes to primary and foreign keys. For example, if you edit a column heading to change its semantic type, that change is disregarded when the graphmart is created. Only the original type from the data source is considered. If you add or change primary and foreign keys on the Tables screen, however, the automated data load workflow will retain those changes.



8. If you want to create additional tables in the schema, follow these steps:

   a. Click the menu icon (⋮) in the Actions column for the schema name and select **Add Table**. For example:



   The Create New Table dialog box is displayed.



   b. In the Create New Table dialog box, specify a name for the new table in the **Table Name** field.

   c. In the **Schema Query** field, write the SQL query that defines the data for the new table.

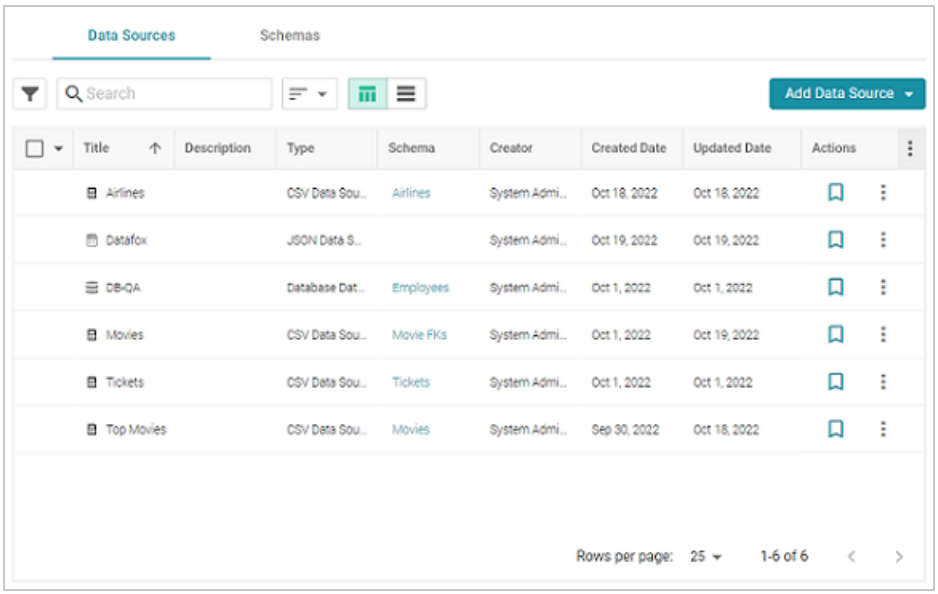
   d. Click **Save** to add the table to the schema and return to the Tables screen.

For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

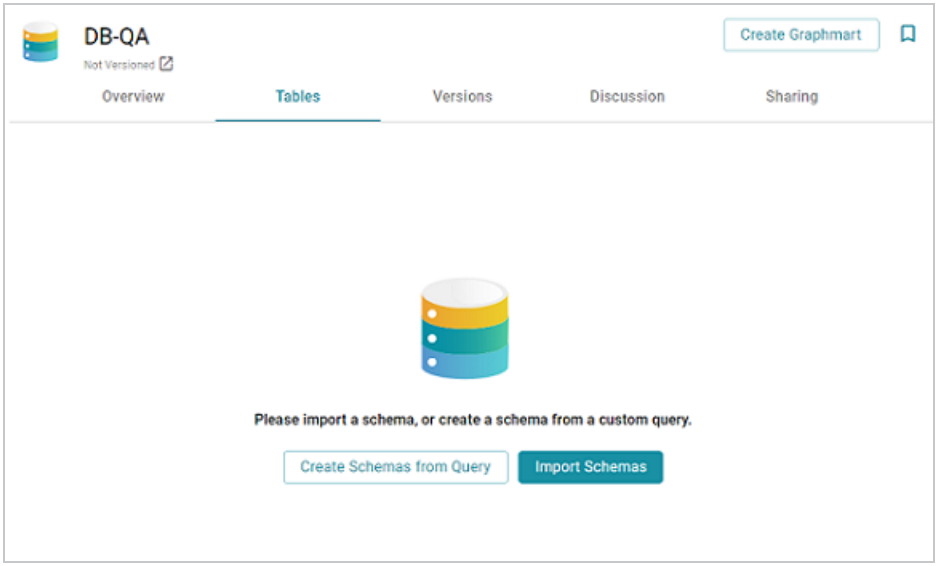
Create an Incremental Schema

Follow the instructions below to create a schema by writing an SQL query that defines a subset of the data to onboard in increments.

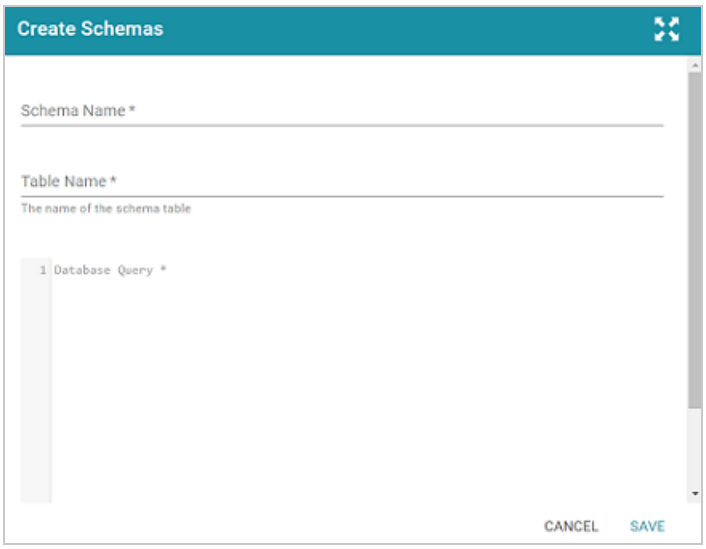
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



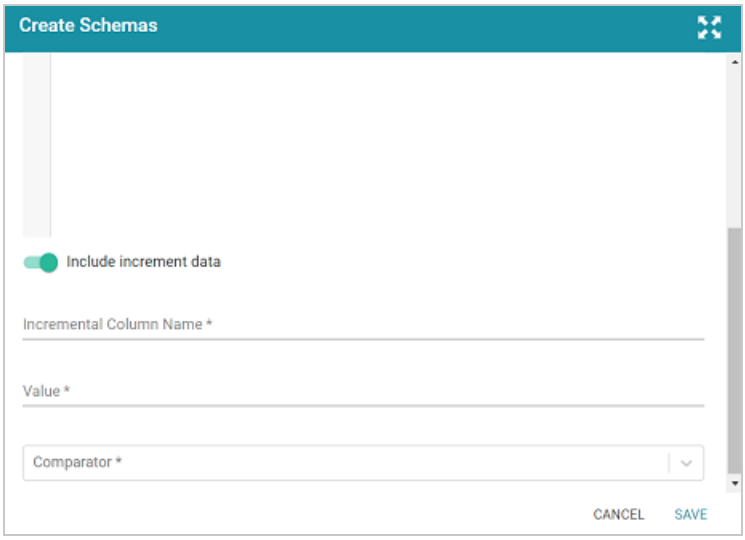
2. Click the database for which you want to create an incremental schema. Anzo displays the Tables tab for the source. For example, the image below shows the Tables tab before any schemas have been created:



3. Click the **Create Schemas From Query** button. Anzo displays the Create Schemas dialog box:



4. In the Create Schemas dialog box, specify a name for this schema in the **Schema Name** field.
5. In the **Table Name** field, specify a name for the table in the schema that the query will create.
6. At the bottom of the screen, enable the **Include increment data** option by sliding the slider to the right. Anzo displays additional settings:



7. Populate the following fields so that you can use the values as a guide for writing the schema query:
   - **Incremental Column Name**: The source column whose value will be used to increment the data.

- **Value**: The value in the column to use as the stopping point for the first import process and the starting point for the next import.

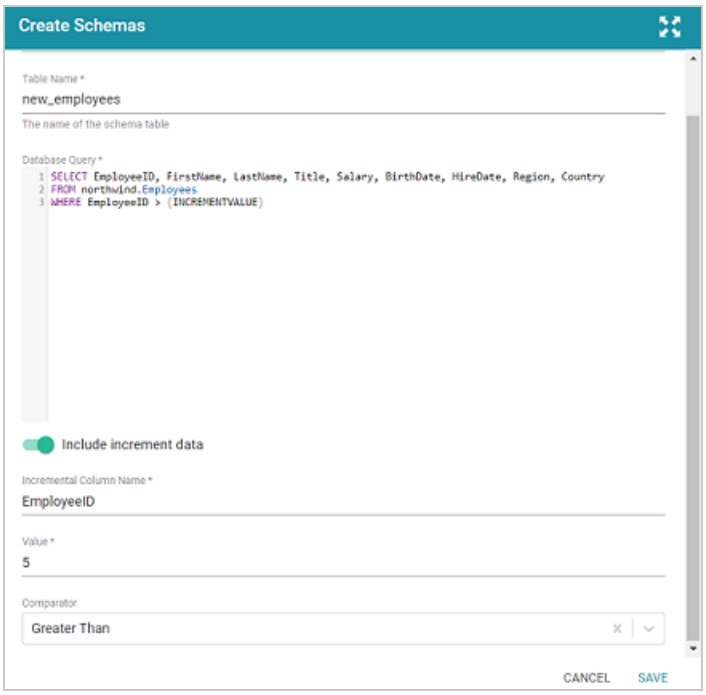
  > **Note**
  > Do not include quote characters in the **Value** field. If the SQL query requires quotes around values, such as '2010-01-01' or 'TestValue', include the quotes around the {INCREMENTVALUE} parameter in the query and not in the Value field. For example, if the value to increment on is '2010-01-01', specify **2010-01-01** in the Value field and add the quotes to the query like the following example:
  >
  > ```
  > SELECT * FROM Orders WHERE OrderData > '{INCREMENTVALUE}'
  > ```
  >
  > In addition, make sure that you use single quotes (') in schema queries. Including double quotes (") in a schema query results in an error when the query is run.

- **Comparator**: The operator to use for comparing source values against the value above.

8. In the query text field, type the SQL statement that will target the appropriate source data. The WHERE clause must include the incremental column name, the comparison operator, and an INCREMENTVALUE parameter. This parameter is substituted with the **Value** at runtime. For example, in the query below, the incremental column name is **EmployeeID**, the comparator is **>** (greater than), and the `{INCREMENTVALUE}` parameter is specified after the comparator. {INCREMENTVALUE} is replaced with the value in the **Value** field at runtime:

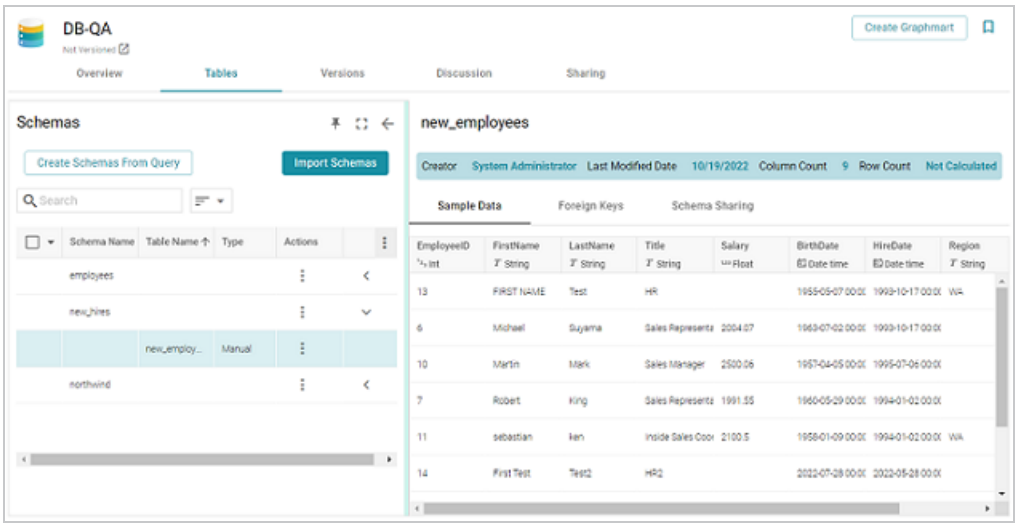
   ```
   SELECT EmployeeID, FirstName, LastName, Title, Salary, BirthDate, HireDate,
   Region, Country
   FROM northwind.Employees
   WHERE EmployeeID > {INCREMENTVALUE}
   ```

   Make sure that the query includes the INCREMENTVALUE parameter and uses the same Incremental Column Name and Comparator values as the fields below the query. For example:



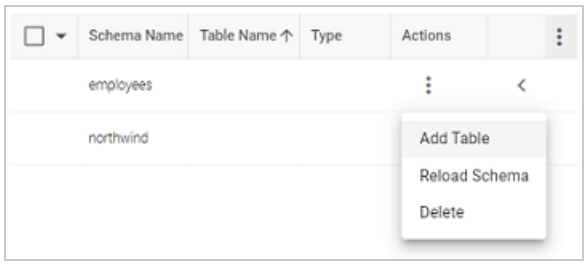
9. Click **Save** to save the query. Anzo creates the new schema and adds it to the list of schemas on the Tables screen. Selecting the schema displays sample data on the right side of the screen.

> **Important**
>
> The automated data load workflow ignores all changes that are made to the schema on the Tables screen—except for changes to primary and foreign keys. For example, if you edit a column heading to change its semantic type, that change is disregarded when the graphmart is created. Only the original type from the data source is considered. If you add or change primary and foreign keys on the Tables screen, however, the automated data load workflow will retain those changes.



10. If you want to create additional tables in the schema, follow these steps:

    a. Click the menu icon (⋮) in the Actions column for the schema name and select **Add Table**. For example:



    The Create New Table dialog box is displayed.



    b. In the Create New Table dialog box, specify a name for the new table in the **Table Name** field.

c. In the **Schema Query** field, write the SQL query that defines the data for the new table.

d. Click **Save** to add the table to the schema and return to the Tables screen.

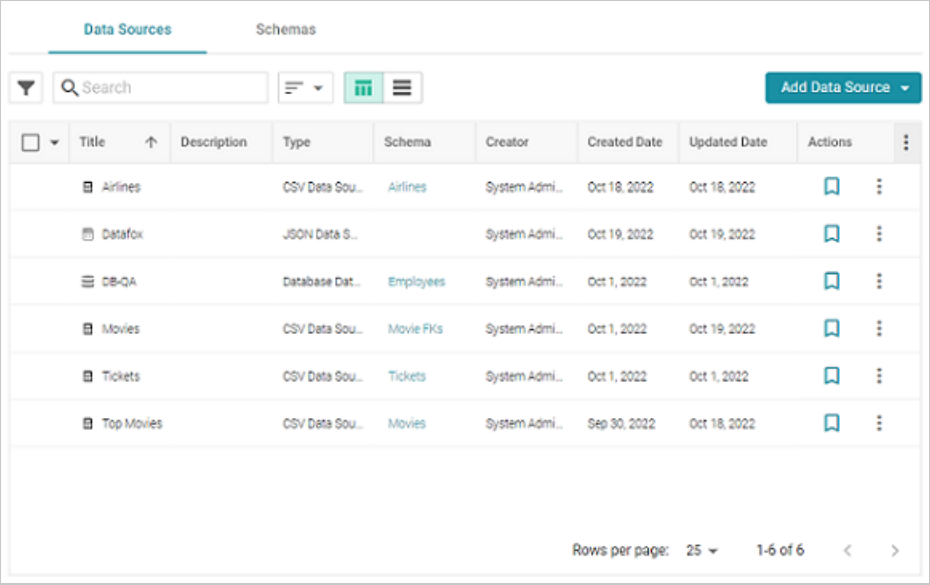
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

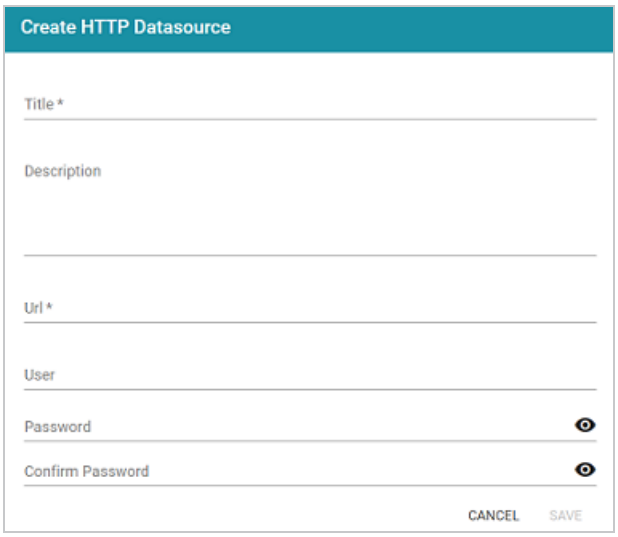
Adding an HTTP or SPARQL Data Source

Follow the instructions below to add a connection to a SPARQL or HTTP endpoint. Though you cannot use the automated workflow to ingest data to Anzo from an endpoint, as you can from file-based and database sources, configuring the connection to endpoints adds those sources as Context Providers so that you can reference context keys in Graph Data Interface (GDI) queries against the endpoints.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the **Add Data Source** button, select **HTTP**, and then choose the type of endpoint to connect to. Note that HTTP Datasource and Sparql Datasource can be used interchangeably at this time. There are no differences in the configuration options. Anzo opens the Create Data Source screen. For example:



3. At the top of the screen, specify a **Title** for the source and enter an optional **Description**.
4. Enter the endpoint connection details:
   - **Url**: The URL for the endpoint. When connecting to an Anzo SPARQL endpoint, make sure that any URIs, such as a graphmart URI, are URL-encoded. For example:

     ```
     https://10.0.10.10/sparql/graphmart/http%3A%2F%2Fcambridgesemantics.com%2FGraphmart%2F1686168b-3eaf-4fdc-9730-1903717b9e62
     ```

   - **User**: The user name to use for authentication.
   - **Password**: The password for the user name.
   - **Confirm Password**: Confirm the password for the user.
5. Click **Save** to save the data source connection. Anzo displays the Overview tab where you can view the connection details.

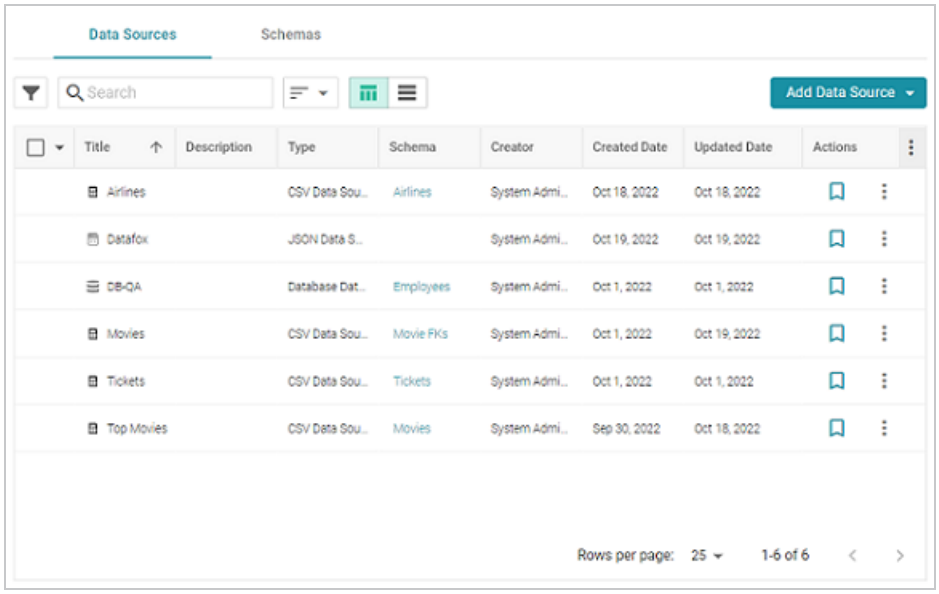
Now that the connection to the endpoint is configured, you can write Graph Data Interface (GDI) queries to virtualize or ingest data from the source. For information about using the GDI, see Onboarding or Virtualizing Data with SPARQL Queries.

Adding a CSV Data Source

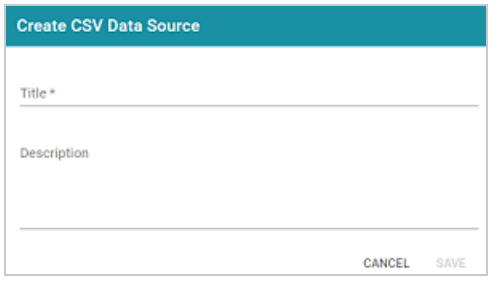
Follow the instructions below to add a CSV data source and import the data from the files.

> **Tip**
> If your CSV data source is consistently updated with new or changed files, you can configure the source to process the data incrementally. For details, see Configuring a CSV or Parquet Source for Incremental Processing.

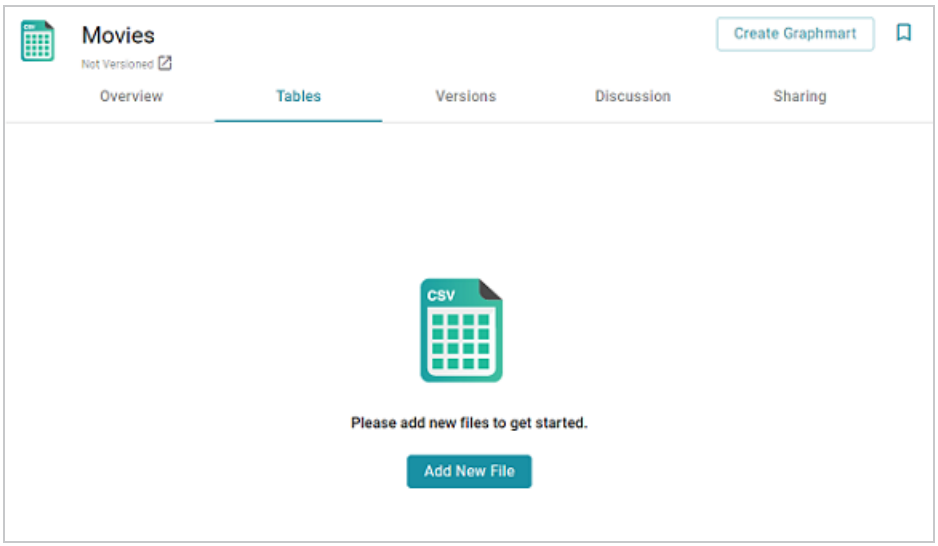
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



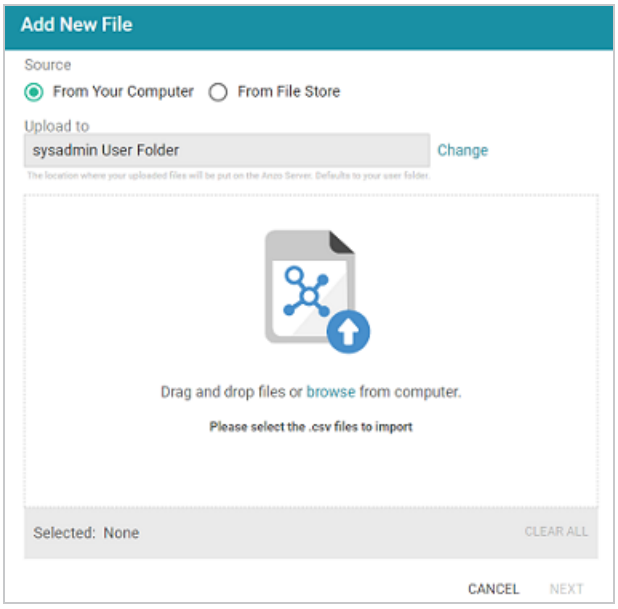
2. Click the **Add Data Source** button and select **File** > **CSV Data Source**. Anzo opens the Create CSV Data Source screen.



3. Specify a name for the Data Source in the **Title** field, and type an optional description in the **Description** field. Then click **Save**. Anzo saves the source and displays the Tables tab.



4. Click the **Add New File** button. Anzo displays the Add New File dialog box.



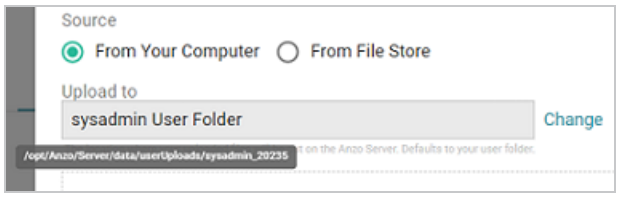
5. Follow the appropriate steps below depending on whether the CSV files are on your computer or the shared File Store:

   If the files are on your computer:

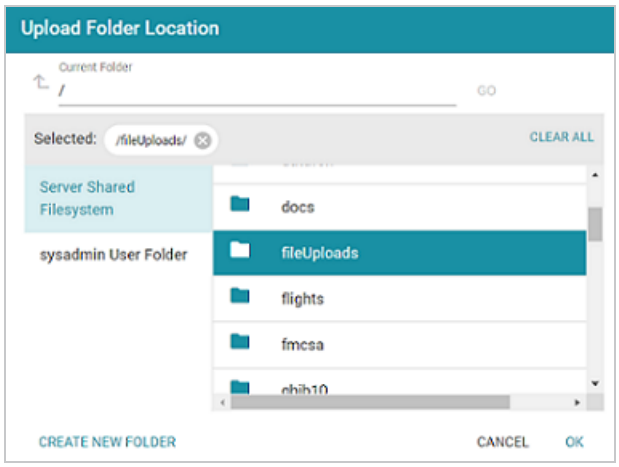
> **Note**
>
> The **From Your Computer** option is a convenient way to do a one-time ingestion so you can quickly get started with your data. It should not be relied upon as part of a regular onboarding workflow unless the server is configured to store uploaded files on the shared file store as described in Setting the Default Base File Store Path for File Uploads in the Administration Guide. Data source files that are routinely updated and re-ingested should be hosted on a shared file store.

a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is a directory on the shared file store and not in the server installation path. If the file is not uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they typically do not have access to the file location.

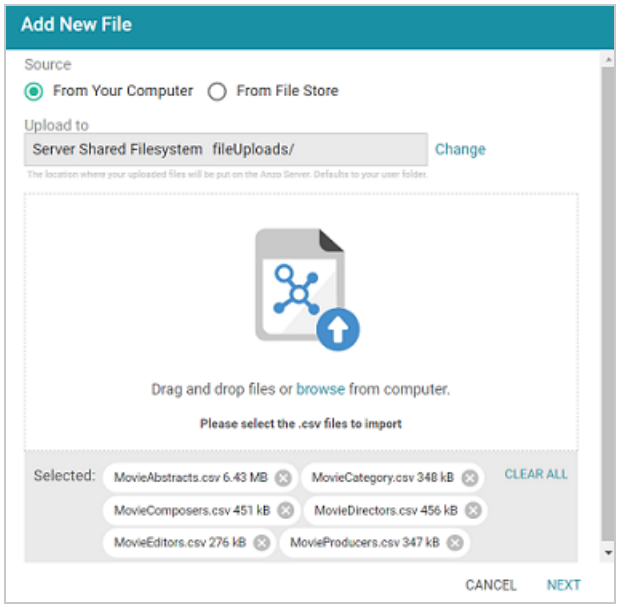
   For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`



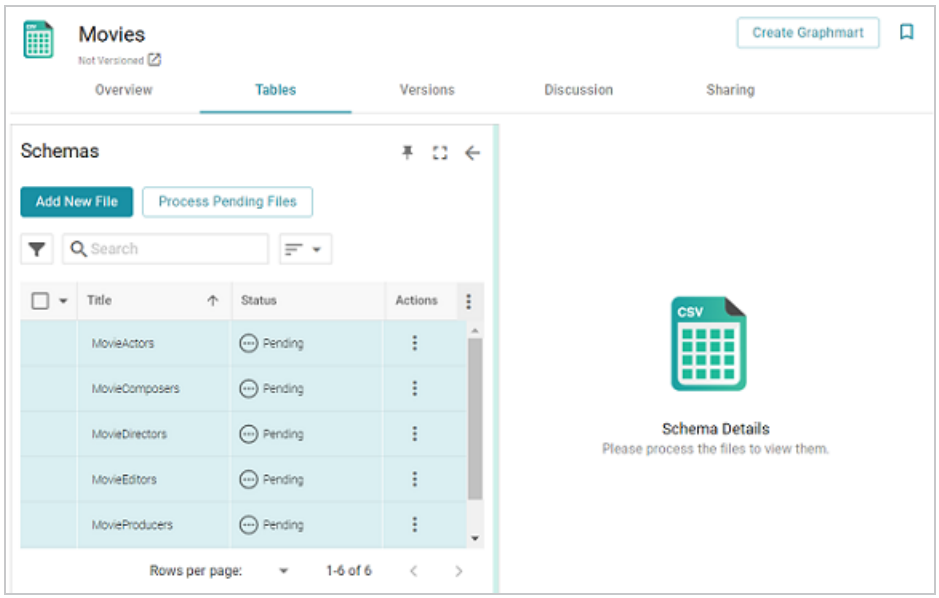
   If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.



b. Drag and drop the files onto the screen or click **browse** to navigate to the files and select them. Anzo attaches the files and the **Next** button becomes active. For example:

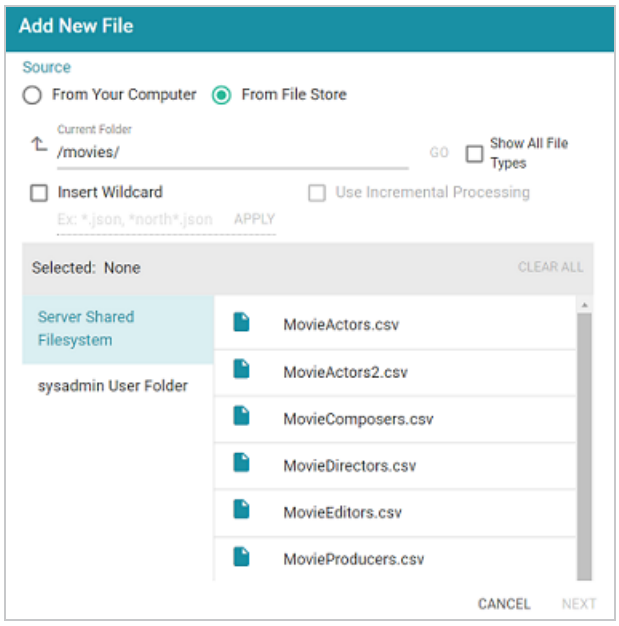


c. Click **Next**. Anzo lists the uploaded files on the left side of the screen with a status of Pending. For example:



If the files are on the File Store:

a. Click the **From File Store** radio button. Anzo displays the file selection dialog box. For example:

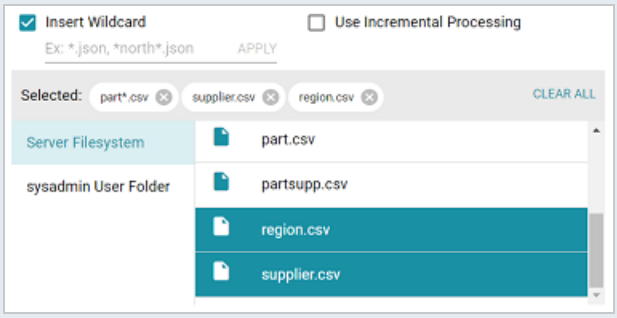


b. On the left side of the screen, select the file store that hosts the CSV files. On the right side of the screen, navigate to the directory that contains the files to import. The screen displays the list of files in the directory.

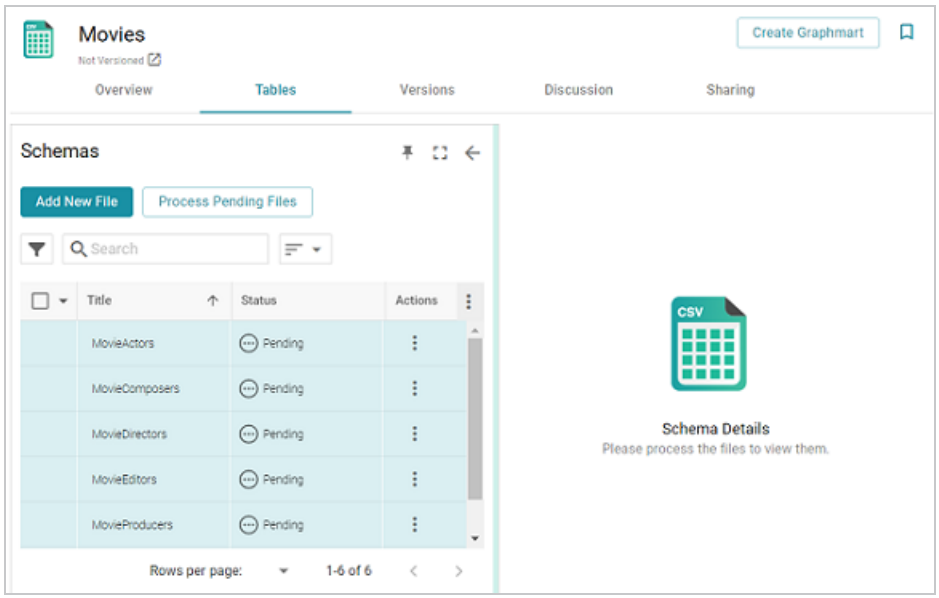
c. Select each file that you want to import. If you have multiple files with the same schema— the files contain the same columns listed in the same order—and you want the files to be imported as if they are a **single file**, you can select the **Insert Wildcard** option. Then type a string using asterisks as wildcard characters to find the files with similar names. Files that match the specified string will be imported as **one file**. After typing a string, click **Apply** to include that string in the Selected list.

Example

The image below shows a directory with several CSV files. For this example, **part.csv** and **partsupp.csv** have the same schema and can be imported as one file. The **Insert Wildcard** option is selected, and **part*.csv** is specified to identify the two files.

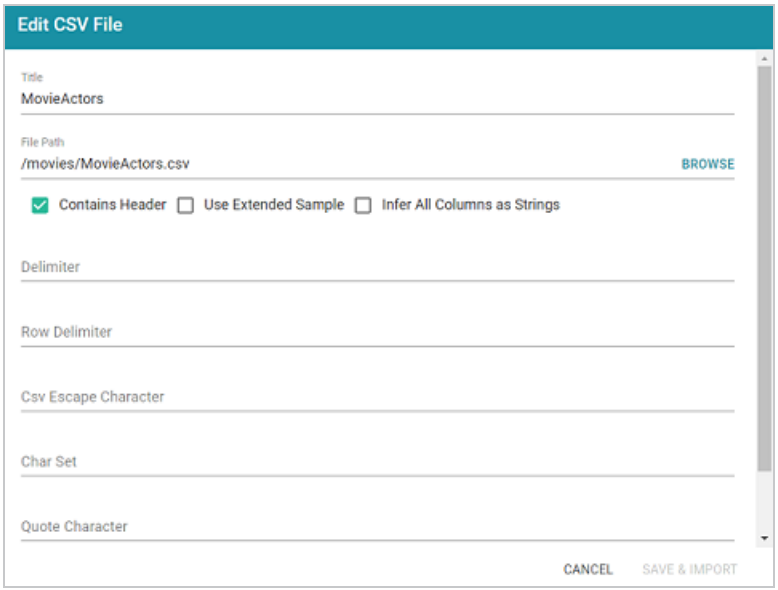


d. When you have finished selecting files, click **Next** to close the dialog box. Anzo lists the uploaded files on the left side of the screen with a status of Pending. For example:



6. If you do not need to change CSV file options, click the **Process Pending Files** button to import all of the pending files. Anzo imports the data and updates the status to Processed.

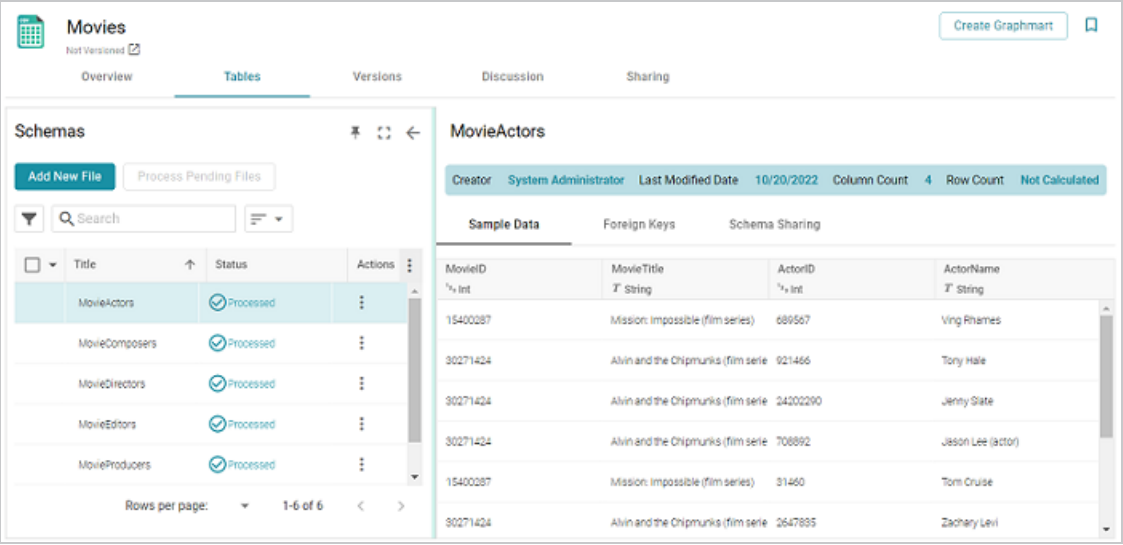
   If you do need to change CSV file options, click the menu icon (⋮) in the Actions column for that file and select **Edit**. To change the options for multiple files, select the checkbox next to each file, and then click the **Edit** button at the bottom of the table. Anzo displays the Edit CSV File screen. For example, the image below shows the Edit screen for a single file:



Change the options as needed and then click **Save & Import** to import the file or files. Anzo imports the data and updates the status to Processed.

7. Once the files are processed, you can click a table row on the left side of the screen to display the schema on the right side of the screen.

> **Important**
> The automated data load workflow ignores all changes that are made to the schema on the Tables screen—except for changes to primary and foreign keys. For example, if you edit a column heading to change its semantic type, that change is disregarded when the graphmart is created. Only the original type from the data source is considered. If you add or change primary and foreign keys on the Tables screen, however, the automated data load workflow will retain those changes.



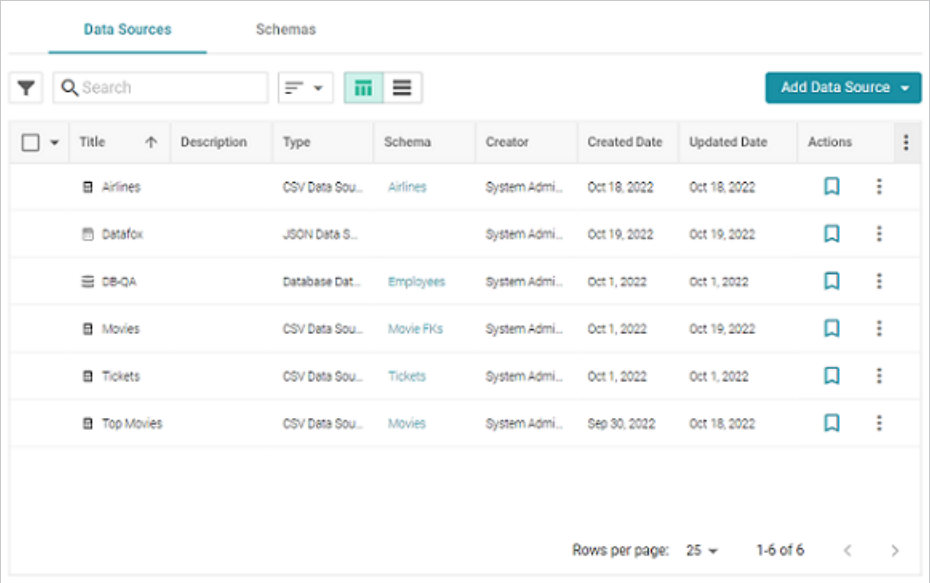
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

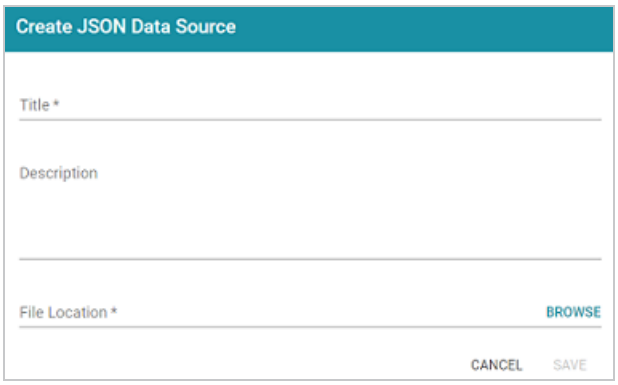
Adding a JSON Data Source

This topic provides instructions for adding a JSON data source.

1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:

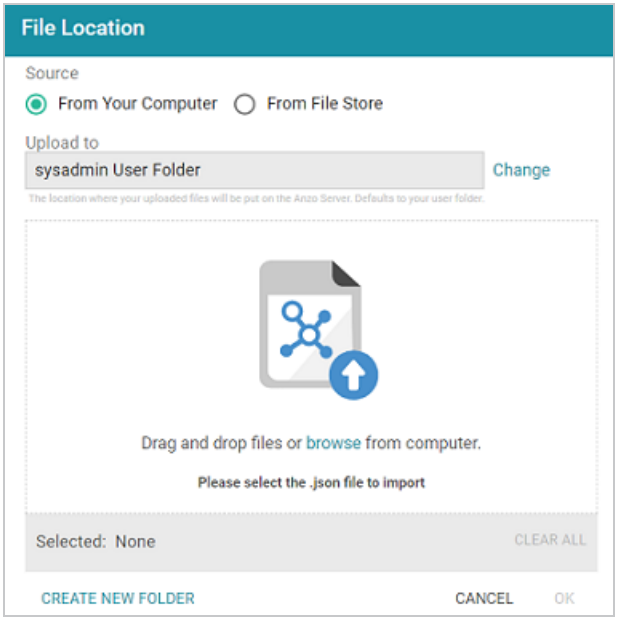


2. Click the **Add Data Source** button and select **File** > **JSON Data Source**. Anzo opens the Create JSON Data Source screen.



3. Specify a name for the data source in the **Title** field, and type an optional description in the **Description** field.

4. Click the **JSON File Location** field to open the File Location dialog box.



5. Follow the appropriate steps below depending on whether the file is on your computer or the shared File Store:

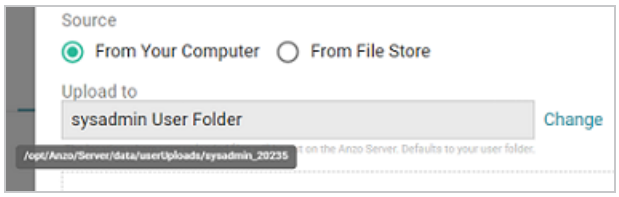
   If the file is on your computer:

   > **Note**
   > The **From Your Computer** option is a convenient way to do a one-time ingestion so you can quickly get started with your data. It should not be relied upon as part of a regular onboarding workflow unless the server is configured to store uploaded files on the shared file store as described in Setting the Default Base File Store Path for File Uploads in the Administration Guide. Data source files that are routinely updated and re-ingested should be hosted on a shared file store.

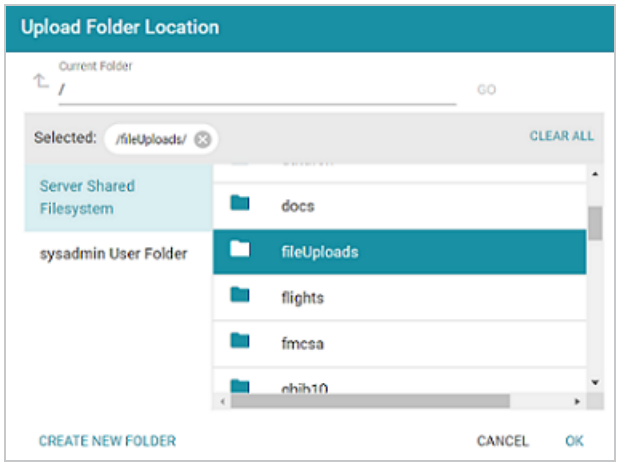
   a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is a directory on the shared file store and not in the server installation path. If the file is not uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they

typically do not have access to the file location.

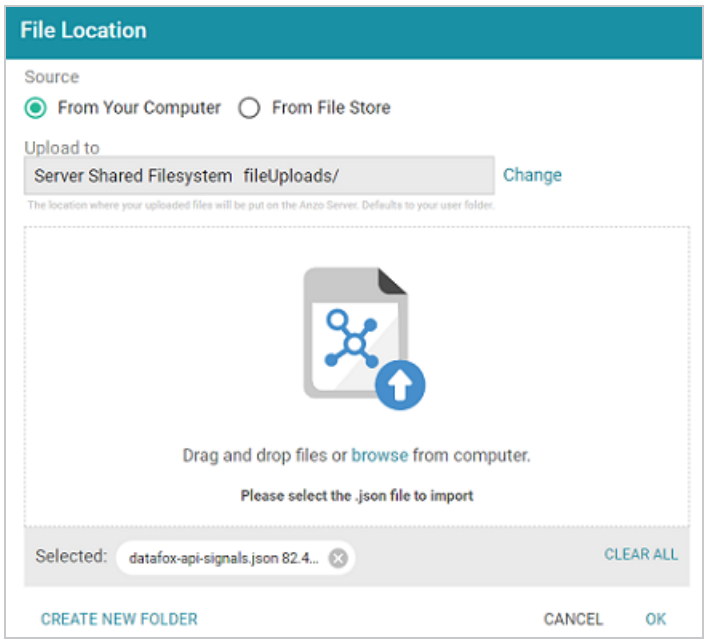
For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`



If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.



b. Drag and drop the file onto the screen or click **browse** to navigate to the file and select it. Anzo attaches the file and the **OK** button becomes active. For example:

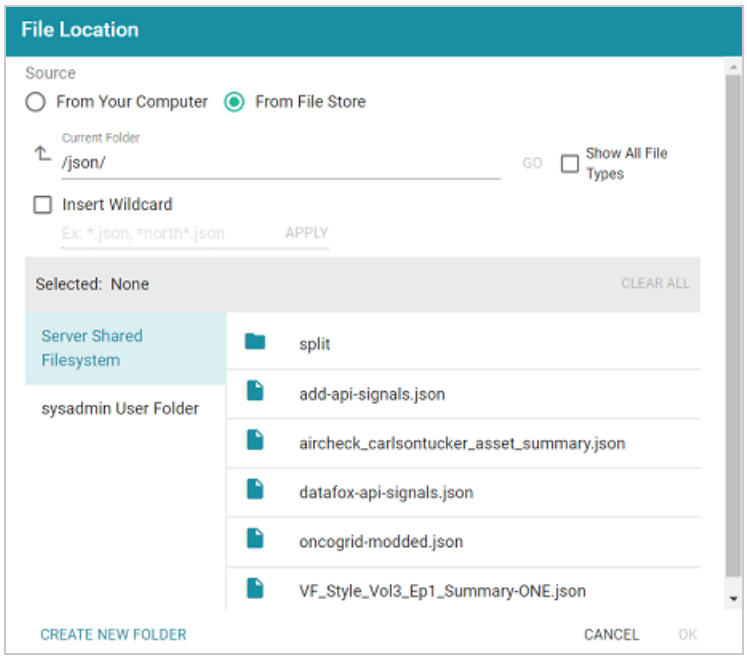


c. Click **OK**. Anzo lists the path to the file in the JSON File Location field.

If the file is on the File Store:

a. Click the **From File Store** radio button.

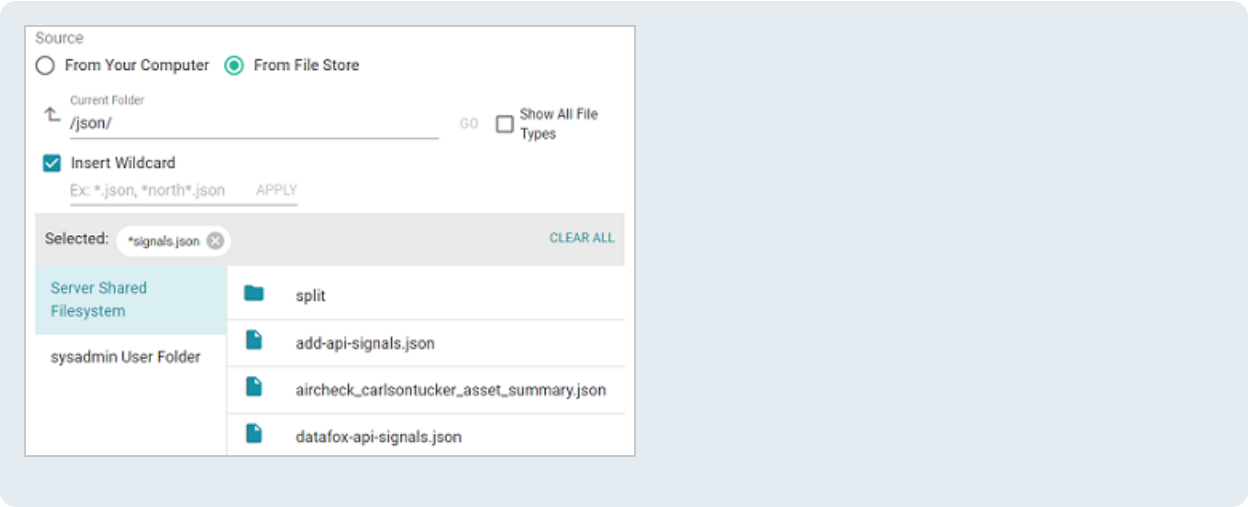
b. In the File Location dialog box, on the left side of the screen, select the shared file store that hosts the file. On the right side of the screen, navigate to the directory that contains the file to import. The screen displays the list of files in the directory. For example:



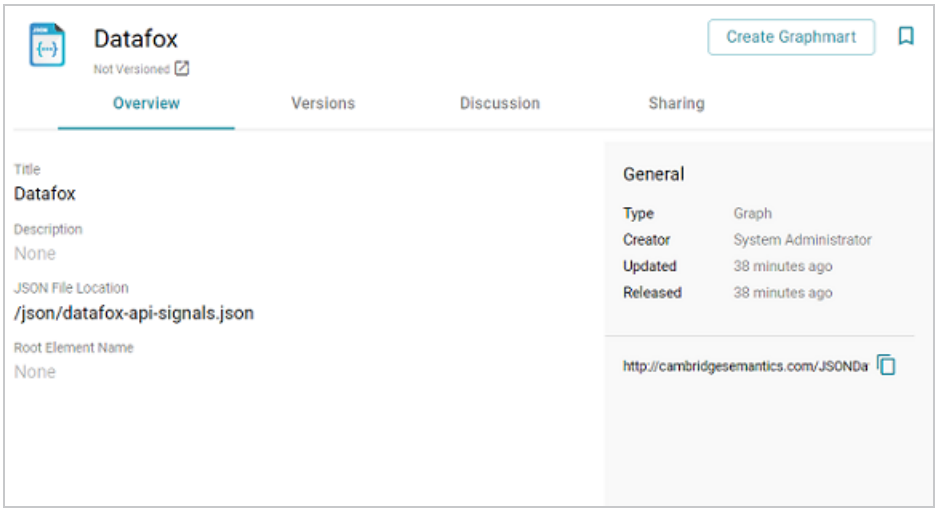
c. Select the file that you want to import and then click **OK** to close the dialog box. If you have multiple files with the same schema— the files contain the same arrays in the same order—you can select the **Insert Wildcard** option. Then type a string using asterisks as wildcard characters to find the files with similar names. Files that match the specified string will be imported as **one file**. You can specify up to 16,000 files using a wildcard. After typing a string, click **Apply** to include that string in the Selected list.

Example

The image below shows a directory with multiple JSON files. For this example, **add-api-signals.json** and **datafox-api-signals.json** have the same schema and can be imported as one file. The **Insert Wildcard** option is selected, and ***signals.json** is specified to identify the two files.



6. Click **Save** to create the data source. Anzo adds the source and displays the Overview screen. For example:



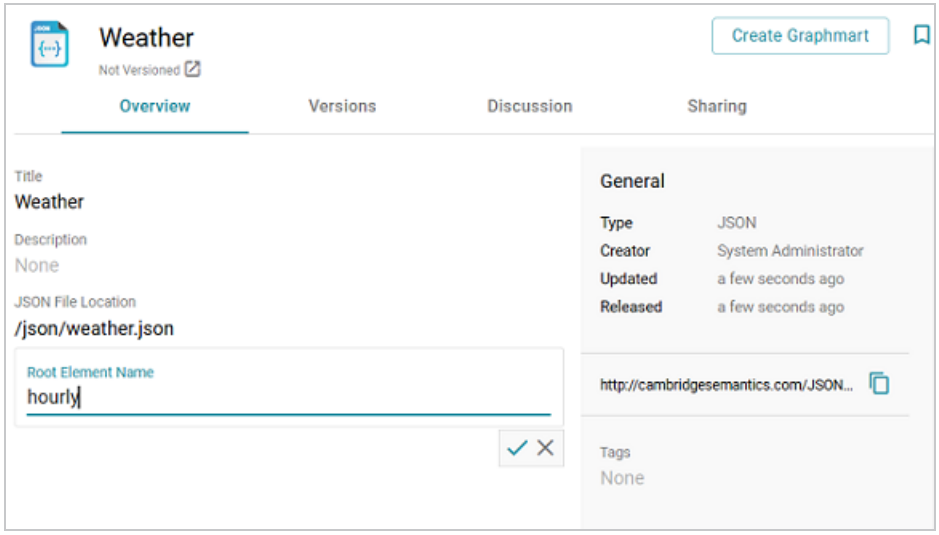
7. By default, when the data from this source is ingested, the entire root node is captured; the node at the root of the hierarchy is loaded to AnzoGraph in its entirety. Building the hierarchical record for a large file is extremely memory intensive. To increase load performance and decrease memory usage when onboarding a large file with many repeating elements, Cambridge Semantics recommends that you configure the **Root Element Name** field on the Overview tab. This field designates the element in the hierarchy that should be treated as the root node. Specifying the desired root node tells the Graph Data Interface to scan into memory only the data that you are interested in and not the entire file. To set the root element, follow these steps:

a. Click in the **Root Element Name** field to make it editable.

b. Add the name of the element to designate as the root element. Type the name the same way it appears in the file. Data is captured from whichever node in the hierarchy matches the Root Element Name value in its entirety.

> **Tip**
> It is not necessary to express the path to an element if it is low in the hierarchy. For specificity, however, you can use dot notation to supply the path. For example, specifying "city" captures all city elements anywhere in the file. But specifying "country.state.city" captures only the city elements that are under state and city in the hierarchy. You can also include the dollar sign ($) character to anchor the selector at the root of the file. For example, "data" captures all data elements anywhere in the file. But "$.data" captures only the data elements that are at the root of the hierarchy.

As an example, for a file that contains weather data in daily, hourly, minutely, and currently hierarchies, "hourly" is specified to target only the data under the hourly hierarchy:



c. Click the checkmark icon (✓) to save the change.

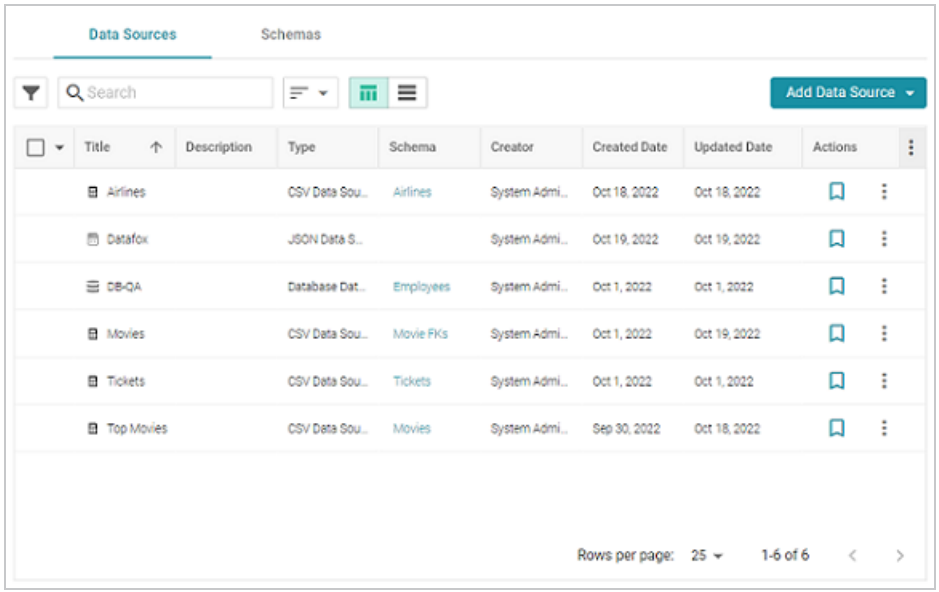
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

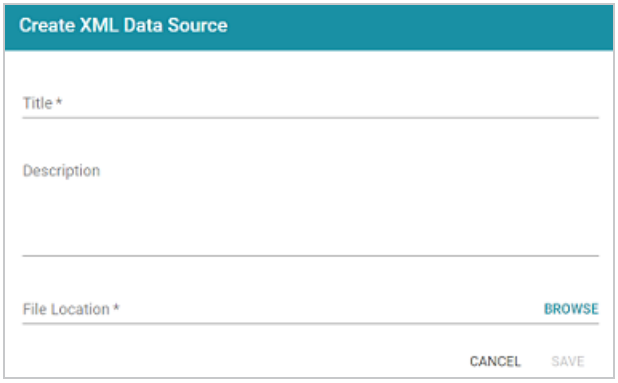
Adding an XML Data Source

This topic provides instructions for adding an XML data source.

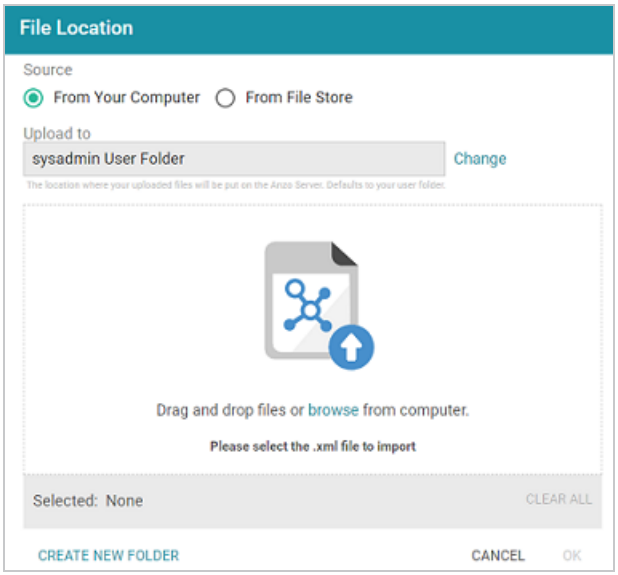
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the **Add Data Source** button and select **File** > **XML Data Source**. Anzo opens the Create XML Data Source screen.



3. Specify a name for the data source in the **Title** field, and type an optional description in the **Description** field.
4. Click the **File Location** field to open the File Location dialog box.



5. Follow the appropriate steps below depending on whether the file is on your computer or the shared File Store:

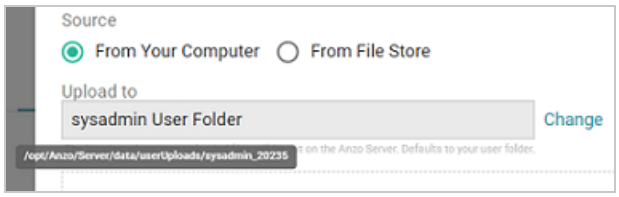
   If the file is on your computer:

Note

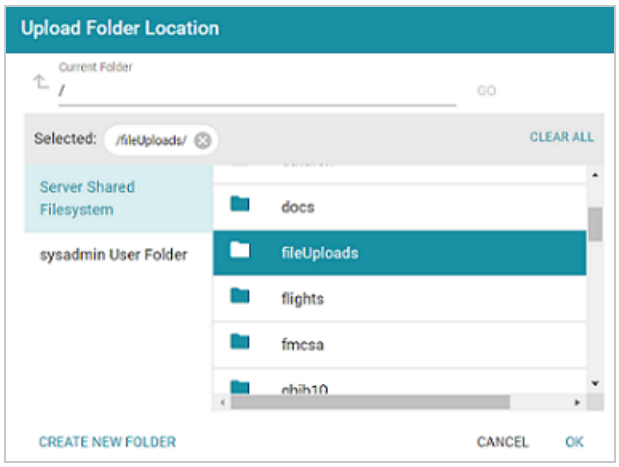
The **From Your Computer** option is a convenient way to do a one-time ingestion so you can quickly get started with your data. It should not be relied upon as part of a regular onboarding workflow unless the server is configured to store uploaded files on the shared file store as described in Setting the Default Base File Store Path for File Uploads in the Administration Guide. Data source files that are routinely updated and re-ingested should be hosted on a shared file store.

a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is a directory on the shared file store and not in the server installation path. If the file is not uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they typically do not have access to the file location.

   For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`



   If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.



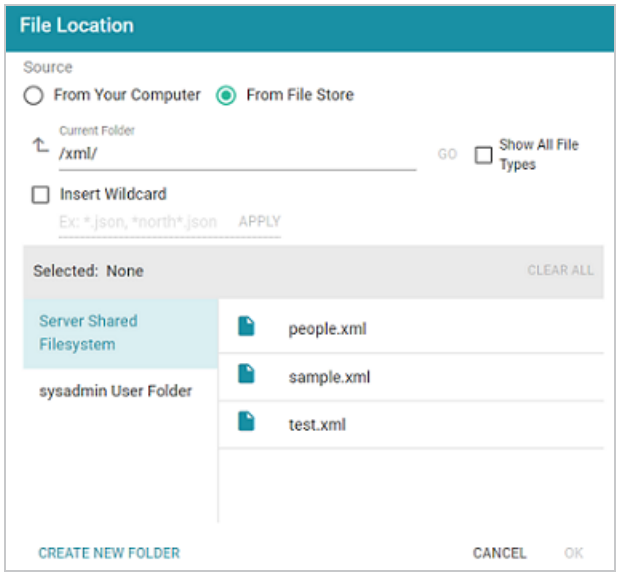
b. Drag and drop the file onto the screen or click **browse** to navigate to the file and select it. Anzo attaches the file and the **OK** button becomes active.

c. Click **OK**. Anzo lists the path to the file in the XML File Location field.

If the file is on the File Store:

a. Click the **From File Store** radio button.

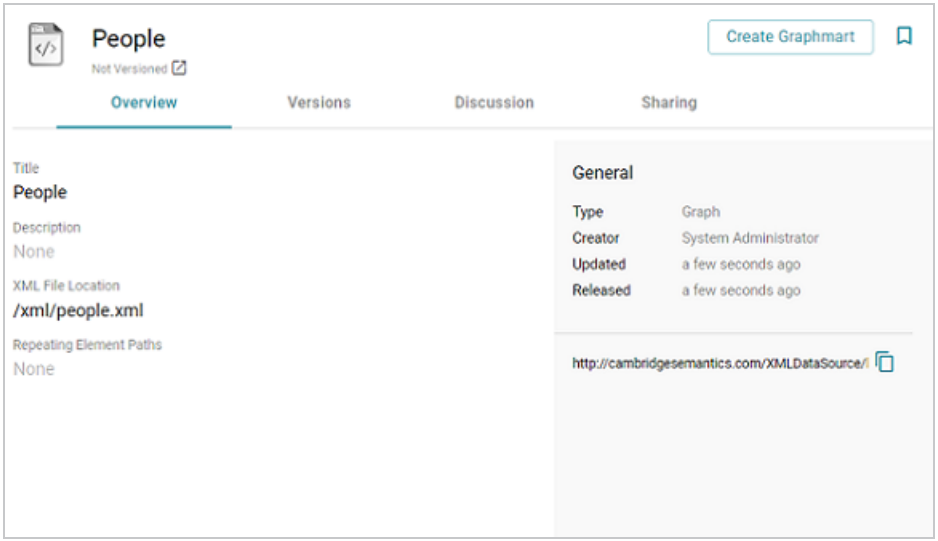
b. In the File Location dialog box, on the left side of the screen, select the appropriate File Store. On the right side of the screen, navigate to the directory that contains the file to import. The screen displays the list of files in the directory. For example:



c. Select the file that you want to import and then click **OK** to close the dialog box. Anzo lists the path to the file in the XML File Location field.

> **Note**
> If you have multiple files with the same schema— the files contain the same elements in the same order—and you want the files to be imported as if they are a **single file**, you can select the **Insert Wildcard** option. Then type a string using asterisks as wildcard characters to find the files with similar names. Files that match the specified string will be imported as **one file** and will result in one job being created in the pipeline to ingest all of the files that are selected by the specified string. After typing a string, click **Apply** to include that string in the Selected list.

6. Click **Save** to create the data source. Anzo adds the source and displays the Overview screen. For example:



7. By default, when the data from this source is ingested, the entire root node is captured; the node at the root of the hierarchy is loaded to AnzoGraph in its entirety. Building the hierarchical record for a large file is extremely memory intensive. To increase load performance and decrease memory usage when onboarding a large file with many repeating elements, Cambridge Semantics recommends that you configure the **Root Element Name** field on the Overview tab. This field designates the element in the hierarchy that should be

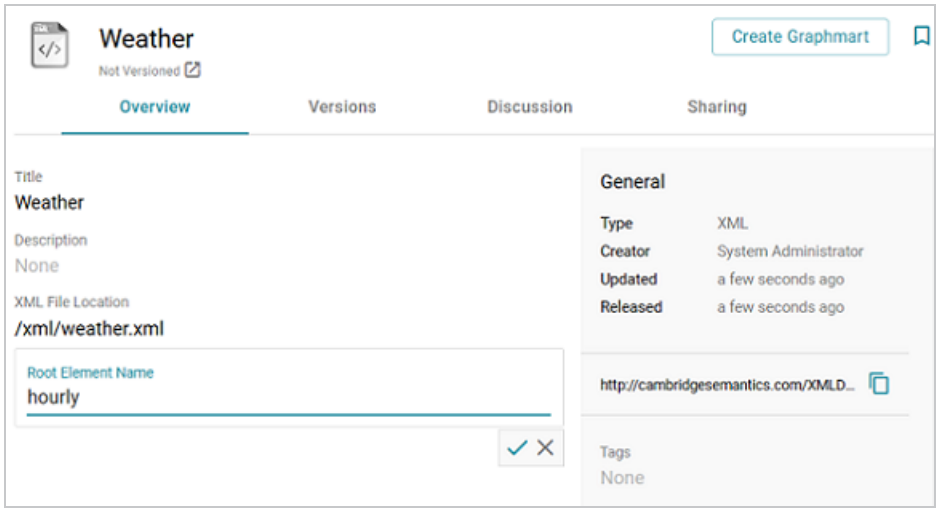
treated as the root node. Specifying the desired root node tells the Graph Data Interface to scan into memory only the data that you are interested in and not the entire file. To set the root element, follow these steps:

a. Click in the **Root Element Name** field to make it editable.

b. Add the name of the element to designate as the root element. Type the name the same way it appears in the file. Data is captured from whichever node in the hierarchy matches the Root Element Name value in its entirety.

> **Tip**
> It is not necessary to express the path to an element if it is low in the hierarchy. For specificity, however, you can use dot notation to supply the path. For example, specifying "city" captures all city elements anywhere in the file. But specifying "country.state.city" captures only the city elements that are under state and city in the hierarchy. You can also include the dollar sign ($) character to anchor the selector at the root of the file. For example, "data" captures all data elements anywhere in the file. But "$.data" captures only the data elements that are at the root of the hierarchy.

As an example, for a file that contains weather data in daily, hourly, minutely, and currently hierarchies, "hourly" is specified to target only the data under the hourly hierarchy:



c. Click the checkmark icon (✓) to save the change.

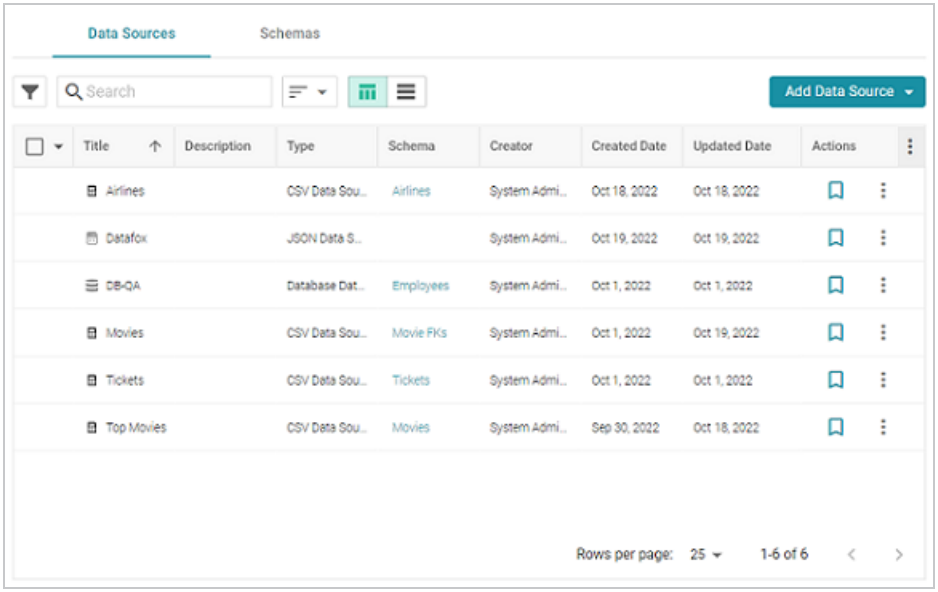
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

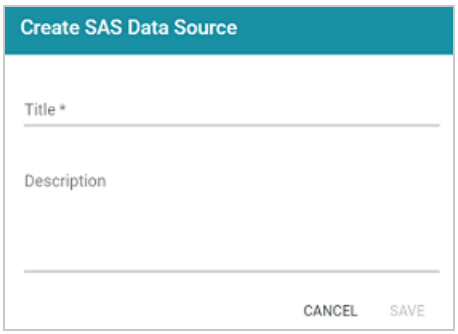
Adding a SAS Data Source

Follow the instructions below to add a SAS data source and import data from SAS7BDAT files.

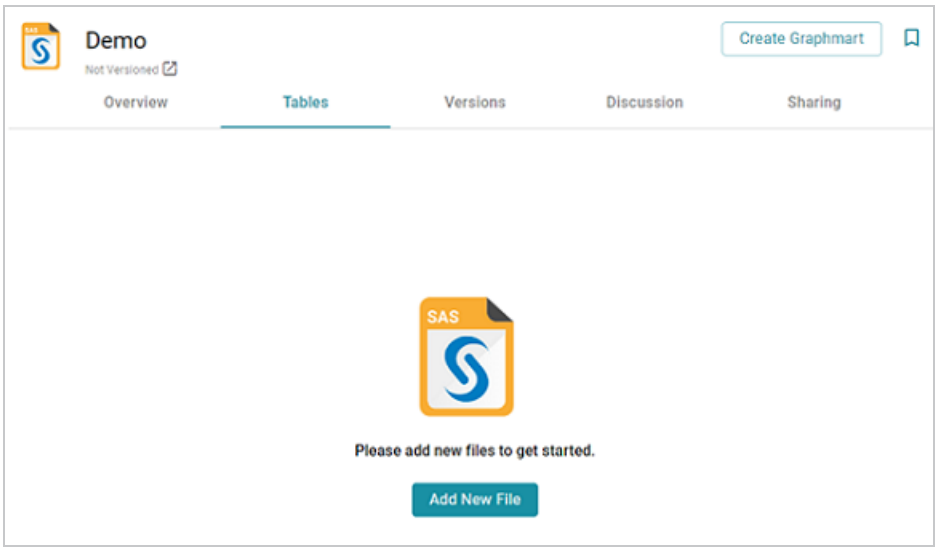
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



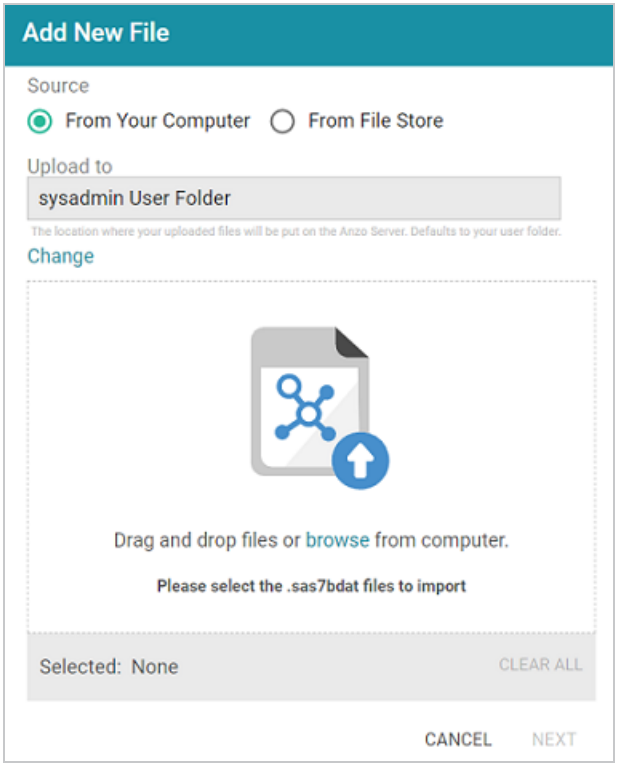
2. Click the **Add Data Source** button and select **File** > **SAS Data Source**. Anzo opens the Create SAS Data Source screen.



3. Specify a name for the source in the **Title** field, and type an optional description in the **Description** field. Then click **Save**. Anzo saves the source and displays the Tables tab.



4. Click the **Add New File** button. Anzo displays the Add New File dialog box.



5. Follow the appropriate steps below depending on whether the SAS files are on your computer or the shared File Store:

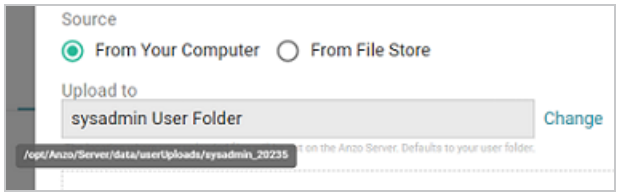
   If the files are on your computer:

   > **Note**
   > The **From Your Computer** option is a convenient way to do a one-time ingestion so you can quickly get started with your data. It should not be relied upon as part of a regular onboarding workflow unless the server is configured to store uploaded files on the shared file store as described in Setting the Default Base File Store Path for File Uploads in the Administration Guide. Data source files that are routinely updated and re-ingested should be hosted on a shared file store.

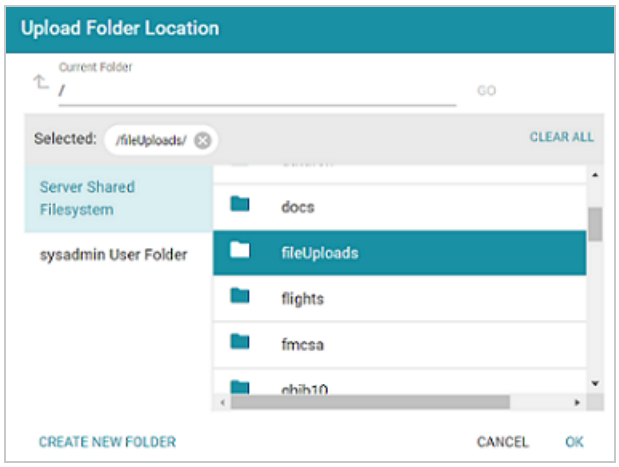
   a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is

a directory on the shared file store and not in the server installation path. If the file is not uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they typically do not have access to the file location.

For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`

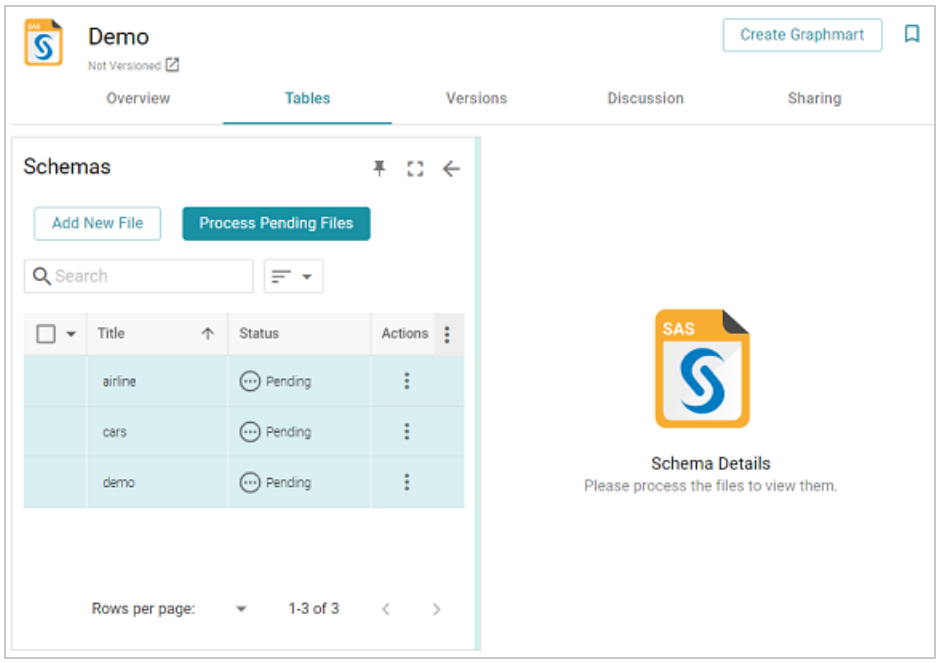


If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.



b. Drag and drop the files onto the screen or click **browse** to navigate to the files and select them. Anzo attaches the files and the Next button becomes active.

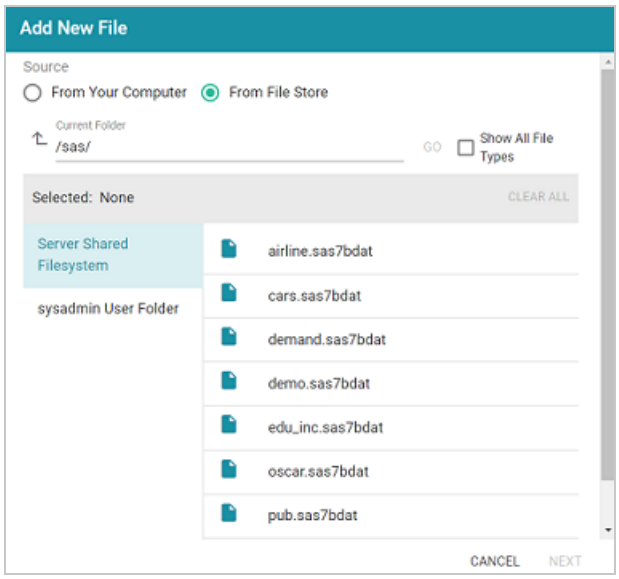
c. Click **Next**. Anzo lists the uploaded files on the left side of the screen with a status of Pending. For example:



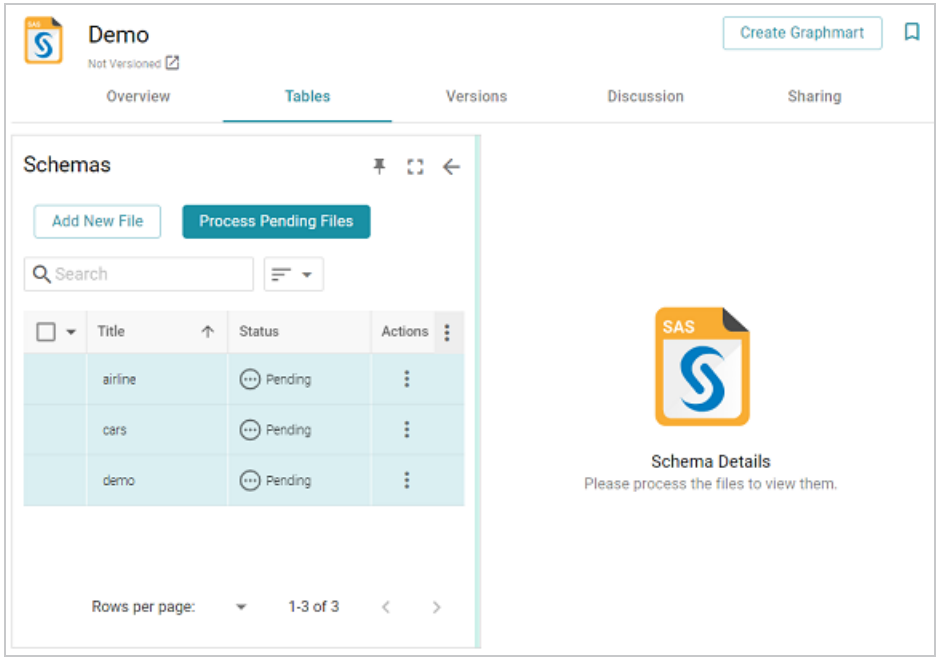
If the files are on the File Store:

a. Click the **From File Store** radio button.

b. In the File Location dialog box, on the left side of the screen, select the appropriate File Store. On the right side of the screen, navigate to the directory that contains the file to import. The screen displays the list of files in the directory. For example:

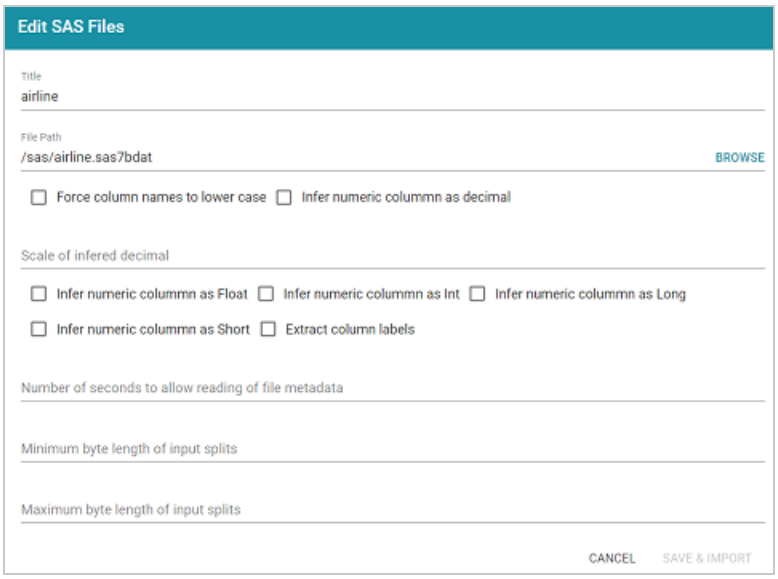


c. Select each file that you want to import. When you finish selecting files, click **Next** to close the dialog box. Anzo lists the uploaded files on the left side of the screen with a status of Pending. For example:



6. If you do not need to change SAS file options, click the **Process Pending Files** button to import all of the pending files. Anzo imports the data and updates the status to Processed.

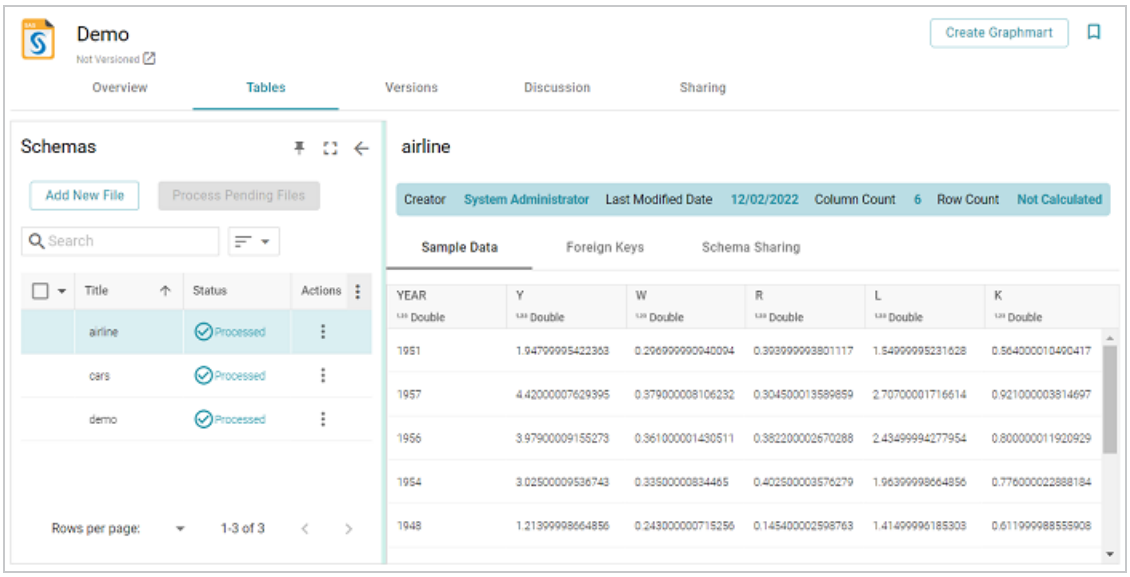
   If you do need to change SAS file options, click the menu icon (⋮) for that file and select **Edit**. To change the options for multiple files, select the checkbox next to each of the files, and then click the **Edit** button at the bottom of the table. Anzo displays the Edit SAS File screen. For example, the image below shows the Edit screen for a single file:



Change the options as needed and then click **Save & Import** to import the SAS file or files. Anzo imports the data and updates the status to Processed.

7. Once the files are processed, you can click a table row on the left side of the screen to display the schema on the right side of the screen.

> **Important**
>
> The automated data load workflow ignores all changes that are made to the schema on the Tables screen—except for changes to primary and foreign keys. For example, if you edit a column heading to change its semantic type, that change is disregarded when the graphmart is created. Only the original type from the data source is considered. If you add or change primary and foreign keys on the Tables screen, however, the automated data load workflow will retain those changes.



For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

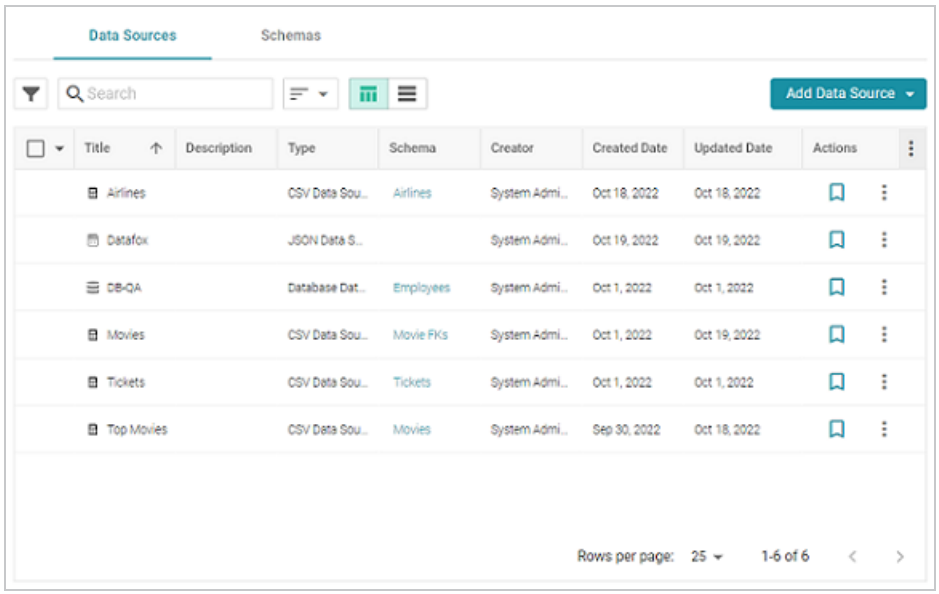
When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

Adding a Parquet Data Source

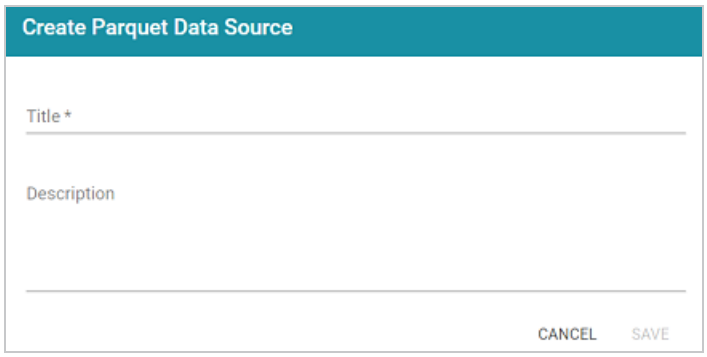
Follow the instructions below to create a Parquet data source. You can onboard one file or multiple files with the identical format (schema) per data source.

> **Tip**
> If your Parquet source is consistently updated with new or changed files, you can configure the source to process the data incrementally. For details, see Configuring a CSV or Parquet Source for Incremental Processing.

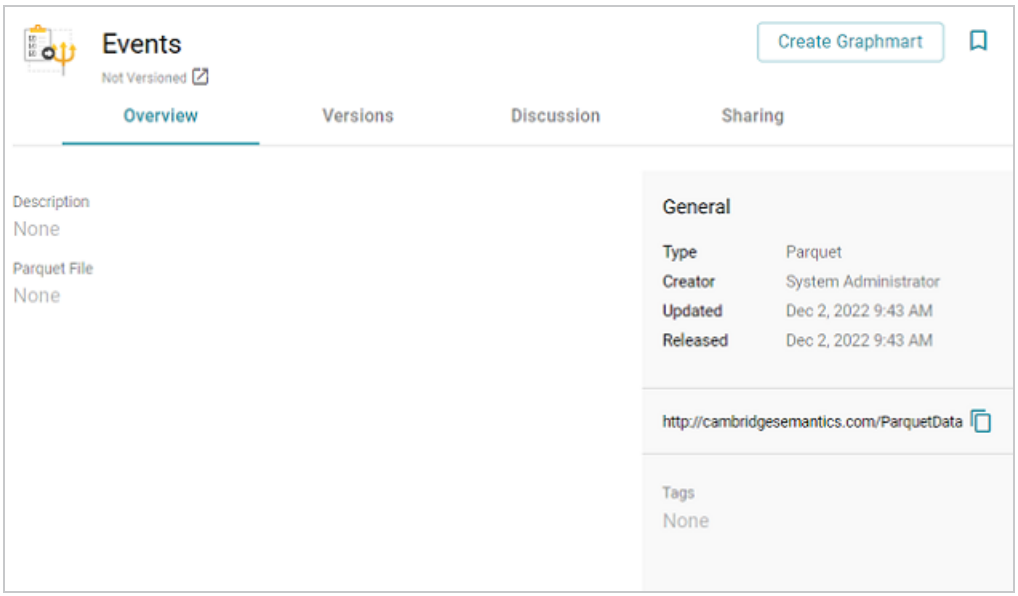
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



2. Click the **Add Data Source** button and select **File** > **Parquet Data Source**. Anzo opens the Create Parquet Data Source screen.

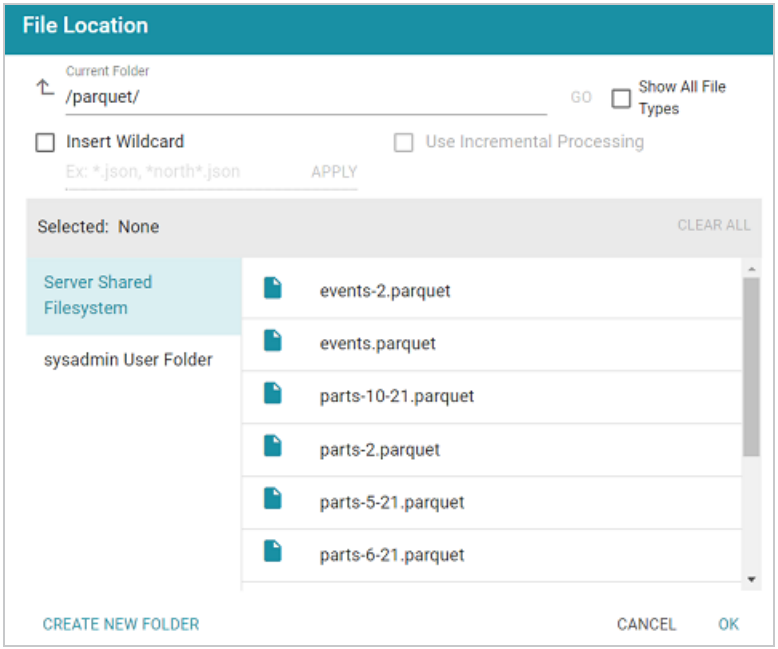


3. Specify a name for the source in the **Title** field, and type an optional description in the **Description** field. Then click **Save**. Anzo saves the source and displays the Overview tab. For example:



4. On the Overview tab, click in the **Parquet File** field to make the value editable. Then click **Browse** to open the File Location dialog box and select the file to import.

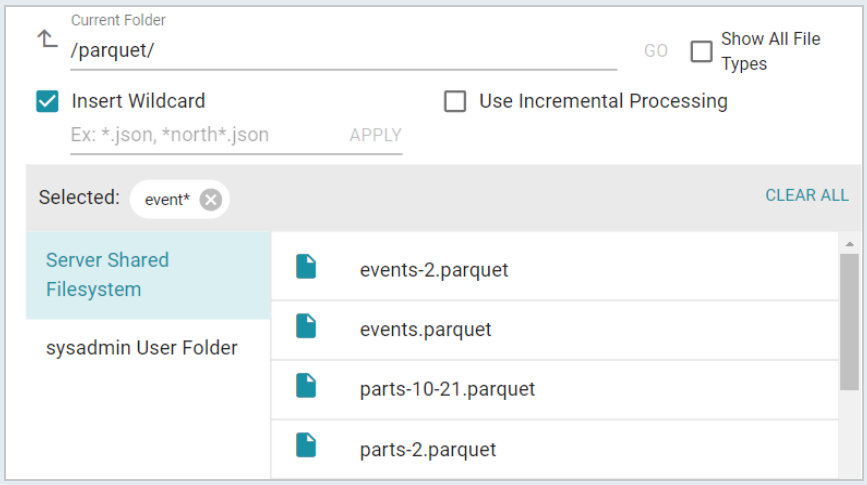
5. In the File Location dialog box on the left side of the screen, select the file store for the Parquet file. On the right side of the screen, navigate to the directory that contains the file to import. The screen displays the list of files in the directory. For example:



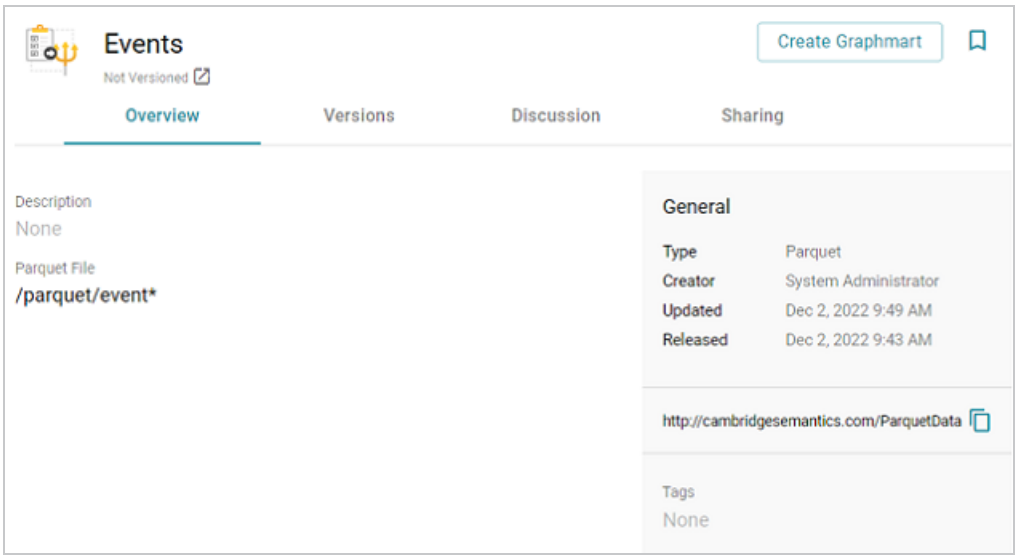
6. Select the file that you want to import. If you have multiple files with the identical format you can select the **Insert Wildcard** option. Then type a string using asterisks as wildcard characters to find the files with similar names. Files that match the specified string will be imported as **one file** and will result in one job being created in the pipeline to ingest all of the files that are selected by the specified string. You can specify up to 16,000 files using a wildcard. After typing a string, click **Apply** to include that string in the Selected list.

Example

The image below shows a directory with multiple parquet files. The **events.parquet** and **events-2.parquet** file have the identical format and can be imported as one file. The **Insert Wildcard** option is selected, and **event*** is specified to identify the two files.



7. After selecting the file, click **OK** to close the File Location dialog box. Then click the checkmark icon (✓) to save the change to the Parquet File field. Anzo imports the file and generates a data model.



For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

Configuring a CSV or Parquet Source for Incremental Processing

If you have a CSV or parquet data source that is consistently updated with new or changed files, you can configure the source to process the data incrementally. When a source is set up to onboard data incrementally and new files are added to the data source directory, those new files are ingested and added to the existing dataset. This topic describes how incremental processing works, gives recommendations for organizing the source files, and provides instructions for configuring incremental processing.

- Incremental Processing Overview
- Organizing the Data Source Directory
- Configuring Incremental Processing

Incremental Processing Overview

When you configure file-based incremental processing, there are two methods to choose from:

File Name Strategy

The first method is called the **File Name Strategy**. When this strategy is chosen, Anzo saves the names of the files that are onboarded. Subsequent workflow runs ingest any files whose names are not saved. This method is useful when you know that new files will be added to the data source directory but the existing files will not be edited. If the contents of a previously ingested file changes but the file name does not, that file will not be reprocessed during the next run.

Last Modified Strategy

The second method is called the **Last Modified Strategy**. When this strategy is chosen, Anzo saves the last modified timestamps for the files that are ingested when the workflow is run. Subsequent runs ingest any files whose last modified timestamps are greater than the saved timestamps. Each time an incremental workflow run, the job-specific last modified date will be updated to the latest modified date of the files processed. This method is useful when you know that the contents of previously ingested files may change in addition to adding new files. Since the last modified date is updated when a file is changed, the changes will be processed during the next run.

Organizing the Data Source Directory

Regardless of the strategy you use to process data incrementally, it is important to consider the schema when organizing the data source files on the file store. In order to enable incremental processing, the source files to import must be specified using wildcard characters (*). That means the list of files that are targeted by the wildcard need to have the same schema.

CSV Data Sources

For CSV data sources, you have two options. You can create one subdirectory per schema and then add files multiple times, once for each schema. With this structure, you would import the files in each directory separately and could specify the wildcard like `*.csv` to import all of the

files in a directory. You can also place all of the files into a single directory and use more detailed text when specifying wildcards. For example, when you add files you apply multiple wildcard values such as `patients_*.csv` and `medication_*.csv`.

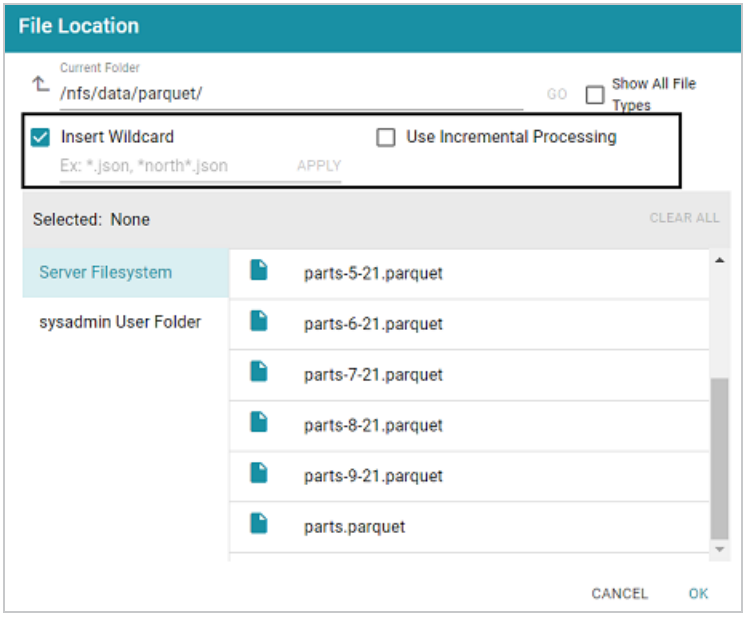
Parquet Data Sources

For Parquet data sources, you can only choose one schema per source. You must create a separate data source for each schema type. You may want to create one directory per schema. Then each Parquet source can target one directory and specify a wildcard value such as `*.parquet`.

Configuring Incremental Processing

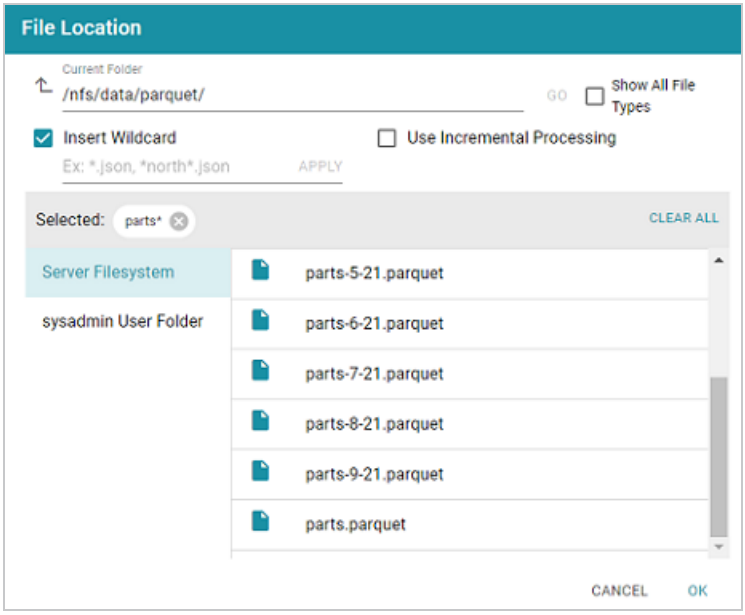
When adding or modifying a CSV or Parquet data source, you configure incremental processing when you are adding files to the source from the file store. This section provides instructions for adding files to a source and configuring incremental processing. For instructions on adding a new data source, see Adding a CSV Data Source or Adding a Parquet Data Source.

1. When selecting the source files on the file store, select the **Insert Wildcard** checkbox. Enabling the wildcard option activates the Use Incremental Processing option.

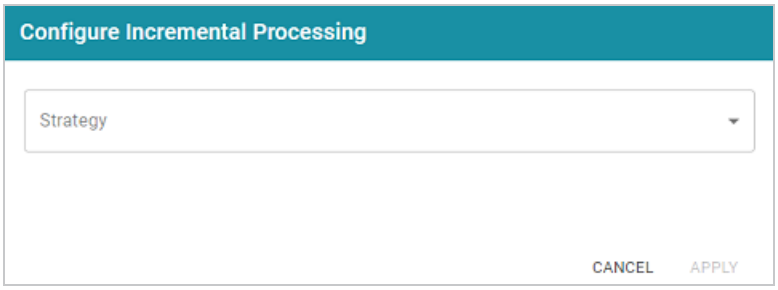


2. Below the checkbox, type a string using asterisks (*) as wildcard characters to find the files to be processed. Then click **Apply** to apply the string. If you are configuring a CSV data source,

you can apply multiple wildcard strings to target files with different schemas. The image below shows an example for a Parquet source. The string `parts*` is applied to select all of the files with names that start with "parts."



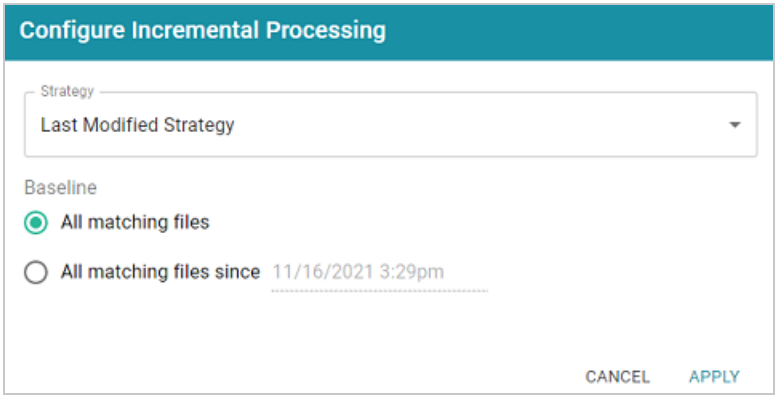
3. Next, select the **Use Incremental Processing** checkbox. The Configure Incremental Processing dialog box is displayed:



4. Click the **Strategy** drop-down list and select the strategy to use for incremental processing. The following list describes the options. For more details about the strategies, see Incremental Processing Overview above.
    - **File Name Strategy**: Select this option if file names should be used to target the new source data to process each time the graphmart that contains this source is reloaded or refreshed.

- **Last Modified Strategy**: Select this option if the last modified date should be used to target the new source data to process each time the graphmart that contains this source is reloaded or refreshed.

5. If you chose **File Name Strategy**, click **Apply** and then click **OK** or **Next** to proceed. The source is now configured to process data incrementally. If you chose **Last Modified Strategy**, proceed to the next step.
6. If you chose **Last Modified Strategy**, the Baseline options are displayed. The Baseline determines when the last modified date begins.



7. By default **All matching files** is selected as the Baseline. This means all files in the directory that are matched by the wildcard string will be ingested when the graphmart is reloaded or refreshed. If you have older files that you do not want to be ingested, you can select **All matching files since**. Then click the timestamp field and specify the date and time to use as the Baseline.
8. When you have finished configuring the Strategy, click **Apply**. Then click **OK** or **Next** to proceed and finish configuring the data source if necessary.

When the graphmart is created, all of the data that matches the wildcard string and meets the baseline requirements will be onboarded. When new files are added to the data source directory, reloading or refreshing the graphmart will update the data.

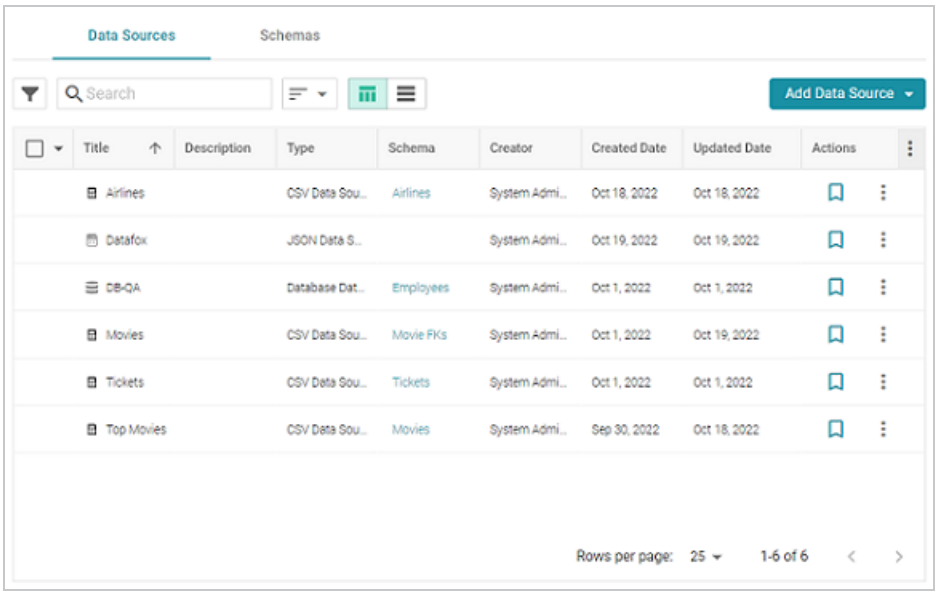
For information about creating or changing primary keys and foreign keys, see Assigning Primary Keys in a Schema and Adding Foreign Keys.

When you are ready to onboard the data to Anzo, see Onboarding Data with the Automated Workflow for next steps. Or, if you want to onboard or virtualize the source by manually writing SPARQL queries against the Graph Data Interface service, see Onboarding or Virtualizing Data with SPARQL Queries.

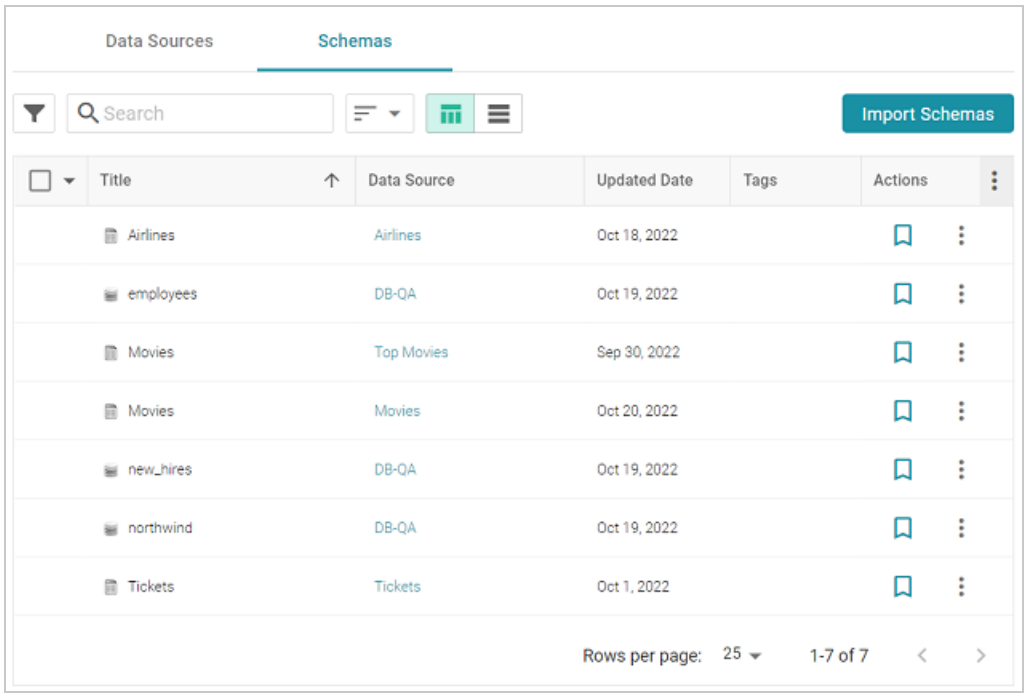
Assigning Primary Keys in a Schema

If you have a data source without primary keys and you want to be able to create relationships, you can assign primary keys in the schema. Follow the instructions below to edit a schema and assign primary keys. For instructions on creating foreign keys, see Adding Foreign Keys.

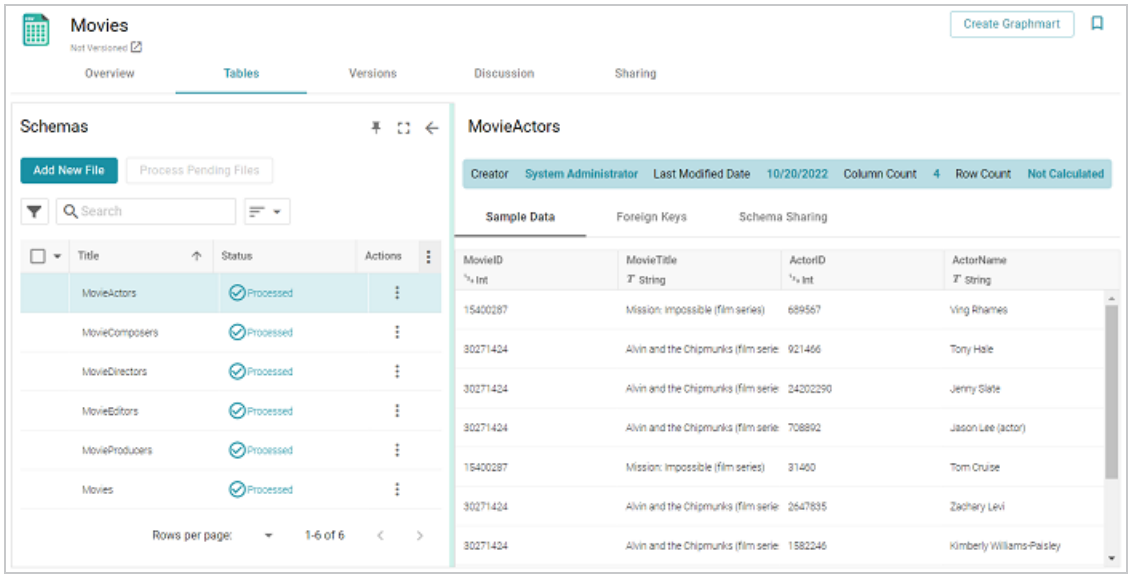
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



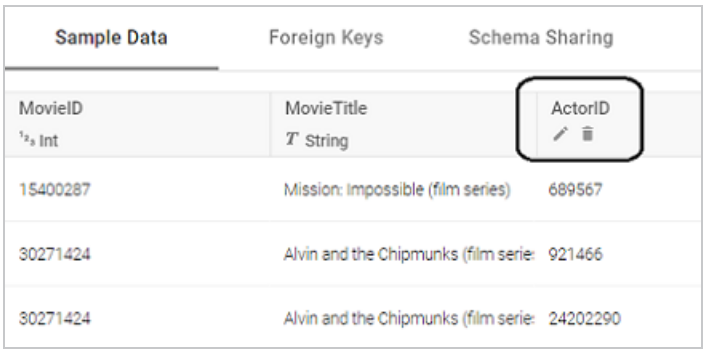
2. Click the **Schemas** tab to view the list of all schemas. (You can also access a schema by selecting the data source that contains the schema.) The image below shows a view of the Schemas tab:



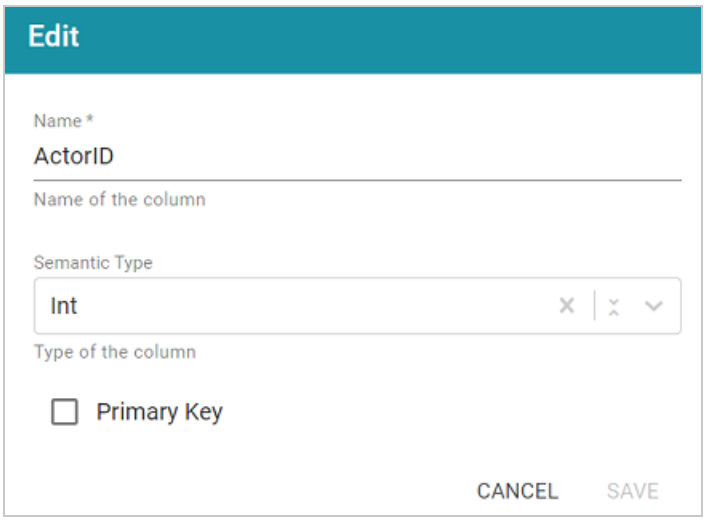
3. Click the schema for which you want to assign primary keys. Anzo displays the Tables tab for the data source that contains the schema. For example:



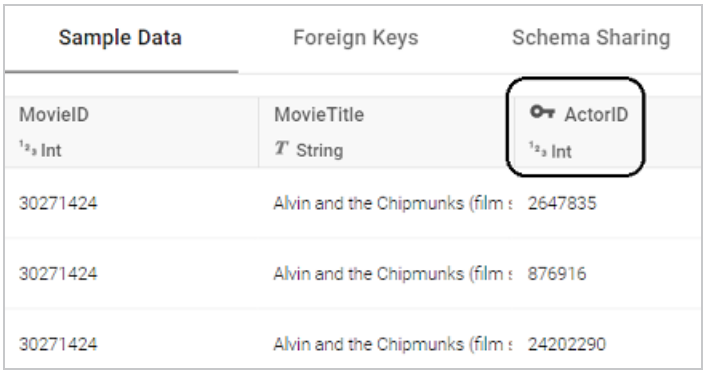
4. On the left side of the screen select the table for which you want to assign a primary key.
5. On the right side of the screen, find the column that you want to label as the primary key. Hover your pointer over the column name to display additional icons. Edit and Delete icons replace the data type under the column name. For example:



6. Click the edit icon ( ). The Edit dialog box is displayed. For example:



7. On the Edit screen, select the **Primary Key** checkbox. Then click **Save** to save the change. The column is now the primary key for the table, and a key icon is displayed next to the column name. For example:

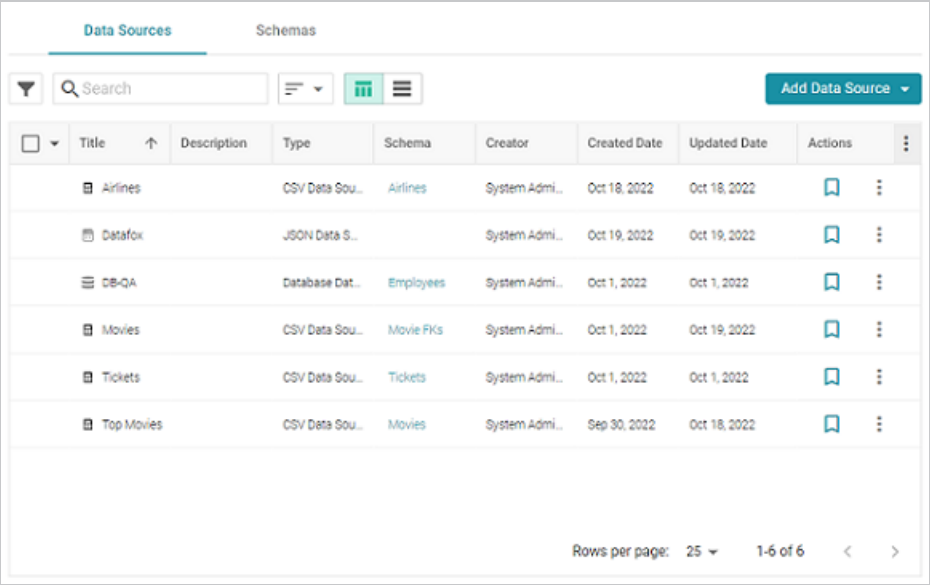


Repeat the steps above to assign primary keys for additional tables. For instructions on creating foreign keys, see Adding Foreign Keys.

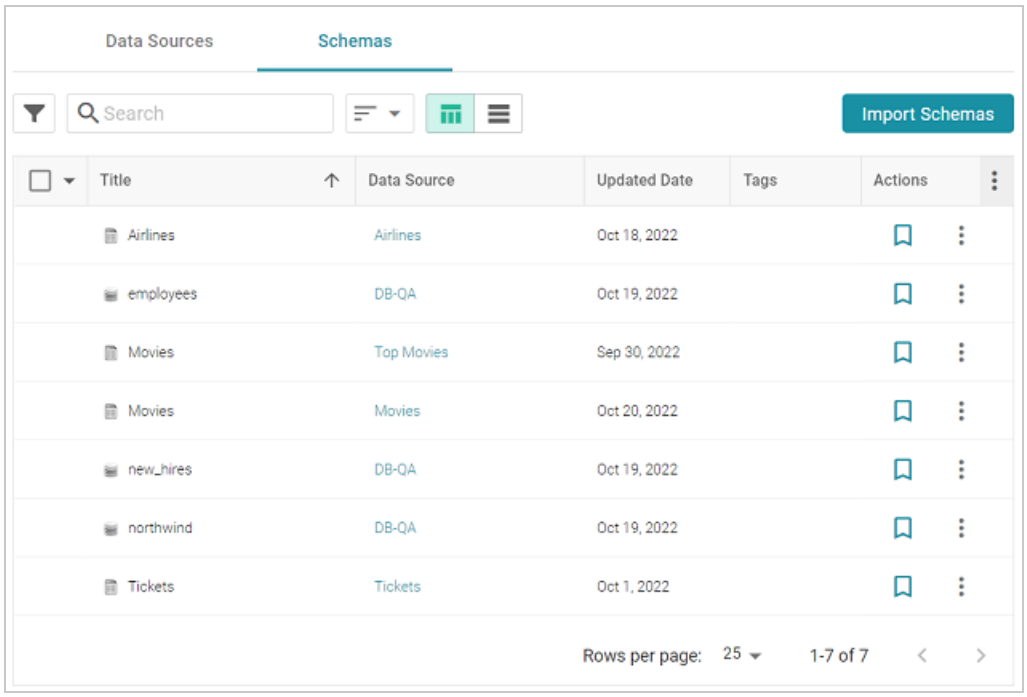
Adding Foreign Keys

Follow the instructions below to edit a schema to add foreign keys. For instructions on designating primary keys, see Assigning Primary Keys in a Schema.

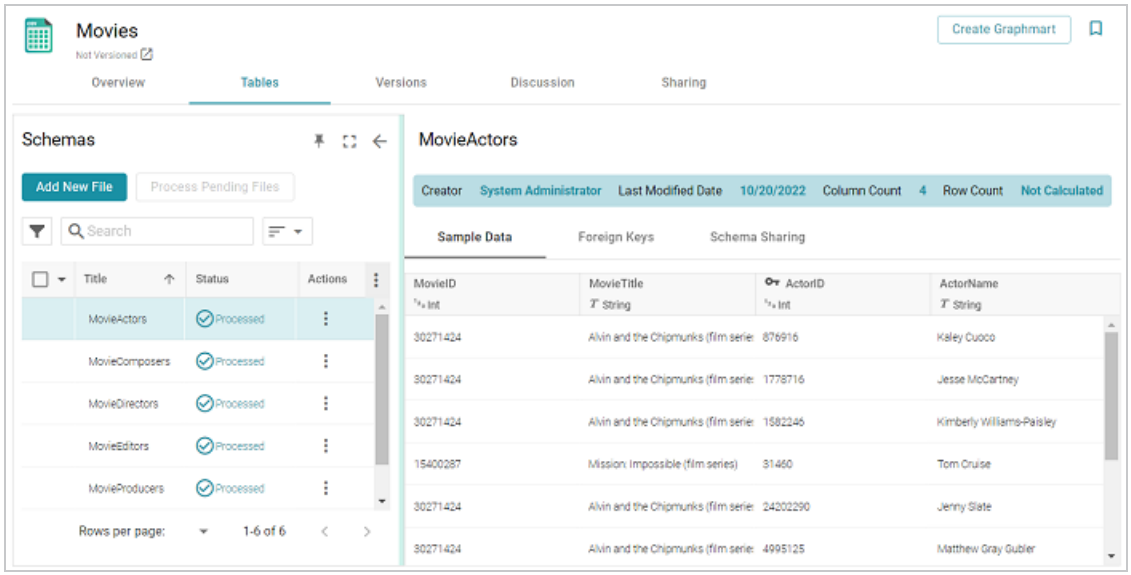
1. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing data sources. For example:



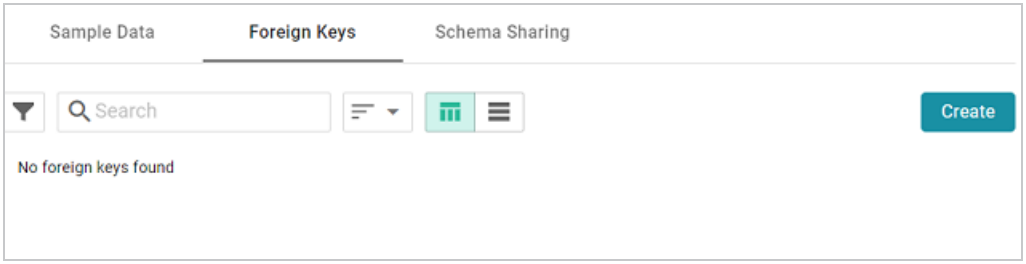
2. Click the **Schemas** tab to view the list of all schemas. (You can also access a schema by selecting the data source that contains the schema.) The image below shows a view of the Schemas tab:



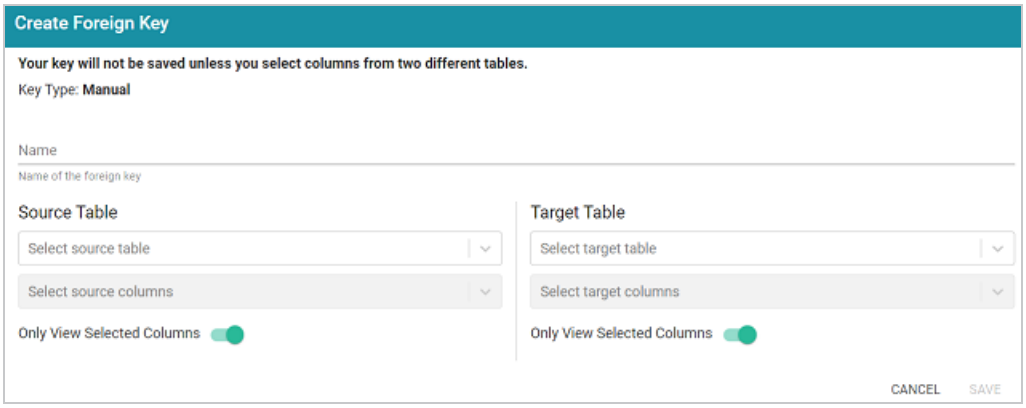
3. Click the schema for which you want to assign foreign keys. Anzo displays the Tables tab for the data source that contains the schema. For example:



4. Click the **Foreign Keys** tab on the right side of the screen. The tab lists any existing keys. For example, the image below shows a schema that does not have keys defined:



5. To change an existing key, click a row in the table to open the Edit Foreign Key dialog box. To create a new key, click the **Create** button to open the Create Foreign Key dialog box.



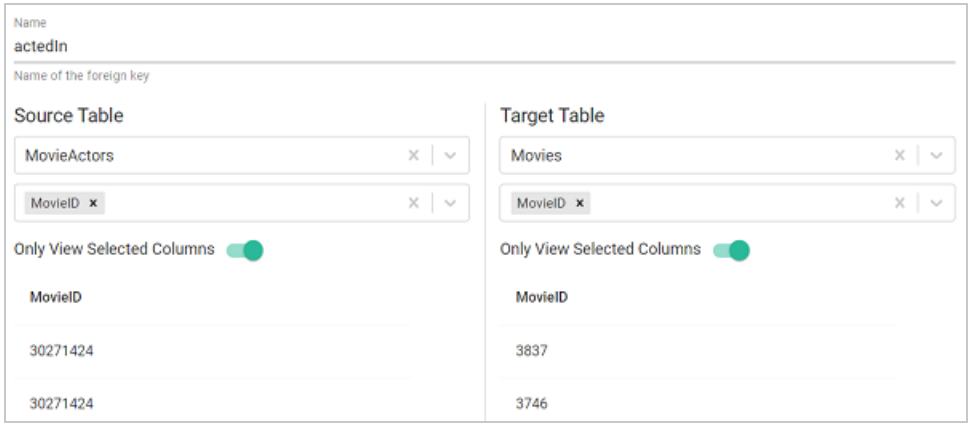
6. On the Create Foreign Key screen, specify a name for the key in the **Name** field.
7. Then specify the source and target tables for this key.
   - **Source Table**: The source table is the table where the new foreign key is created. This table refers to the primary key from the Target Table. Click the **Source Table** drop-down list and select the schema table where the foreign key should be created.
   - **Target Table**: The target table is the table that contains the primary key to be referenced by the Source Table. Click the **Target Table** drop-down list and select the table that will pass values to the source table.
8. Next, specify the source and target columns for this key:
   - **Source Columns**: The source column is the column that becomes the foreign key to the target table's primary key. Click the **Select Source Columns** drop-down list and select the source column. To create a composite key by selecting an additional column, click the Select Source Columns drop-down list again and select a column.

> **Tip**
> By default the screen shows sample values from the selected source column. If you want to view sample values from all columns in the source table, you can disable the **Only View Selected Columns** option by sliding the slider to the left.

- **Target Columns**: The target column is the primary key column in the target table. Click the **Select Target Columns** drop-down list and select the target column. To create a composite key by selecting an additional column, click the Select Target Columns drop-down list again and select a column.

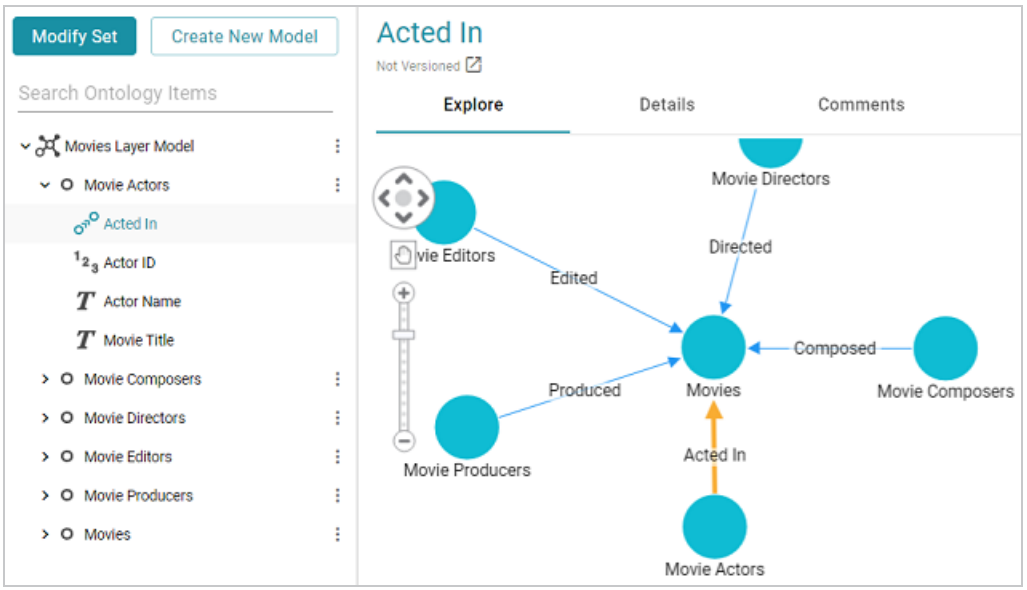
  > **Tip**
  > By default the screen shows sample values from the selected target column. If you want to view sample values from all columns in the target table, you can disable the **Only View Selected Columns** option by sliding the slider to the left.

For example, the image below creates a relationship called **actedIn** where the **MovieID** column in the **MovieActors** table becomes the foreign key and references the values from the primary key column, **MovieID**, in the **Movies** table.



9. When you have finished supplying values, click **Save** to create the new key and return to the Foreign Key list. Repeat this process to create additional keys.

When you onboard the data using this schema, the foreign keys become RDF OWL object properties in the data model. For example, the image below shows a portion of the model that was generated after ingesting the schema that has the foreign key in the example above. In the model, **Acted In** is an object property in the **Movie Actors** class:



Onboarding Data with the Automated Workflow

There are two ways to load a data source with the automated direct data load workflow. You can build a graphmart from a selected data source or you can add a data source to an existing, activated graphmart. Both procedures automatically generate data layers to extract, load, and transform the data to a knowledge graph.

> **Tip**
> When you build a new graphmart from a data source, advanced graphmart options are made available that are not presented when you add a data source to an existing graphmart. See Graphmart Options for information about the additional options.

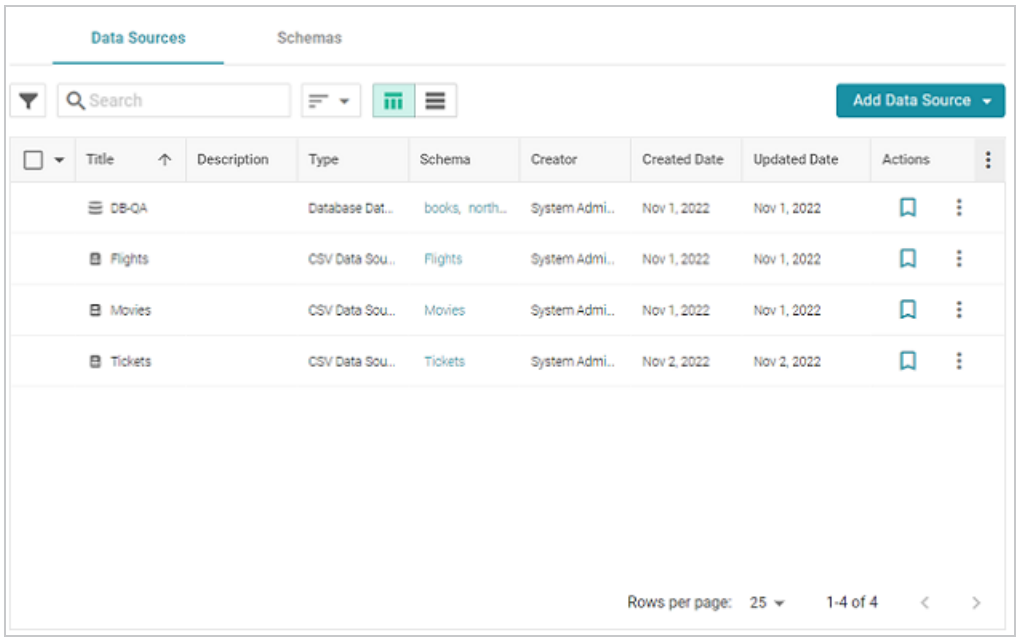
- Creating a Graphmart from a Data Source
- Adding a Data Source to an Existing Graphmart
- Direct Load Advanced Settings Reference

Creating a Graphmart from a Data Source

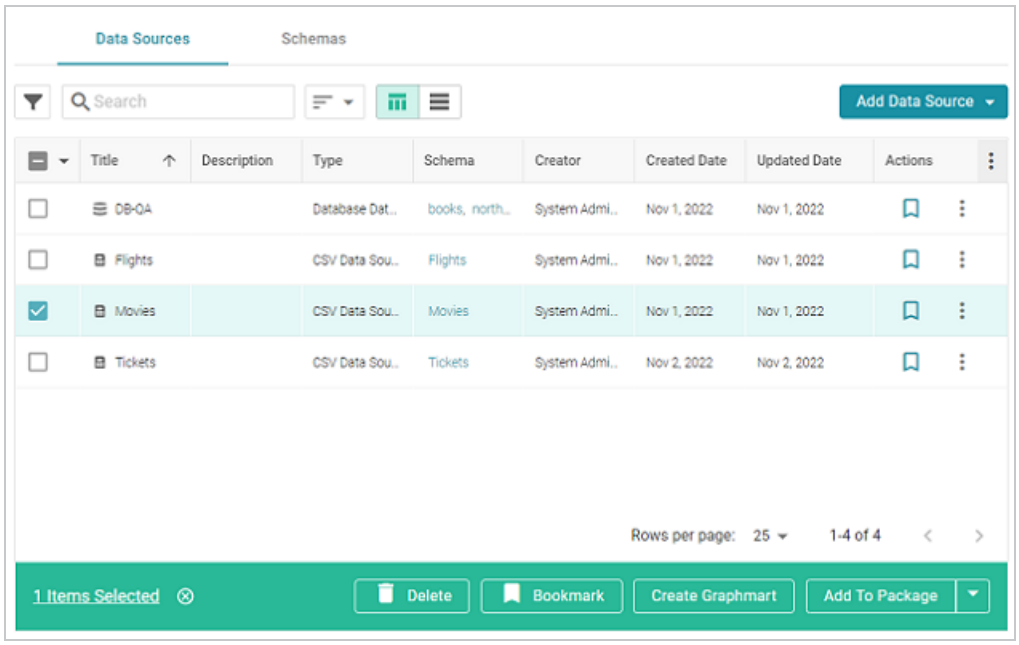
Follow the steps below if you want to create a new graphmart from a data source.

> **Note**
> AnzoGraph uses the Graph Data Interface (GDI) Java plugin to connect directly to sources. For file-based sources, make sure the source files are available to AnzoGraph on the shared file store. For databases, if you have configured custom drivers to access those sources in Anzo, the same drivers need to be added to AnzoGraph. For instructions, see Deploy Optional Drivers for Accessing Custom Database Sources in the Deployment Guide.

1. If necessary, add the data source. See Adding Data Sources for instructions.
2. In the Anzo application, expand the **Onboard** menu and click **Structured Data**. Anzo displays the Data Sources screen, which lists any existing sources. For example:

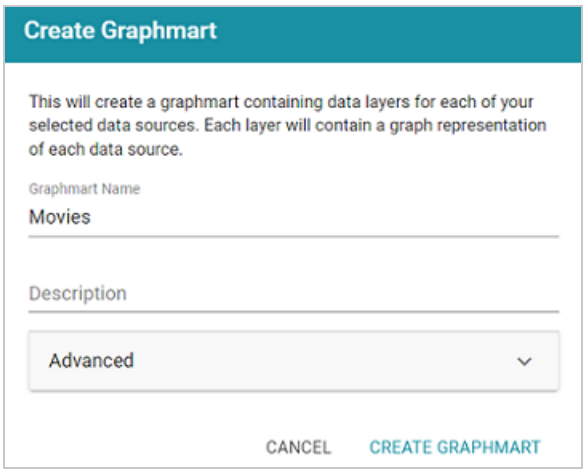


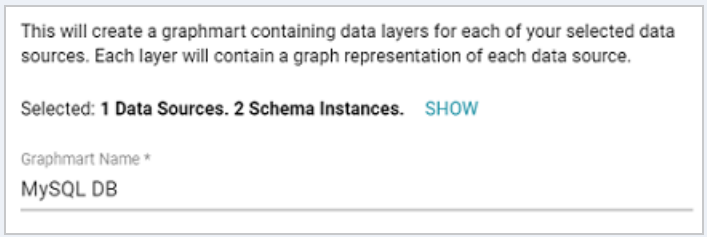
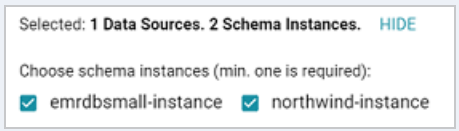
3. Select the checkbox next to the data source that you want to ingest. Options are enabled at the bottom of the screen. For example:



> **Tip**
> Users with the **Batch Direct Data Loading** permission can select multiple data sources to ingest.

4. Click the **Create Graphmart** button. The Create Graphmart dialog box is displayed. For example:



> **Note**
>
> If the selected data source includes more than one schema, the number of schema instances is shown at the top of the screen. For example:
>
> 
>
>
> Clicking **Show** displays the selected schemas. For example:
>
> 
>
>
> If you would like to exclude one or more schemas, clear the checkbox for each schema that you want to exclude.

5. On the Create Graphmart dialog box, the **Graphmart Name** is populated with the name of the selected source. If multiple sources were selected, the Graphmart Name is blank. Edit the **Graphmart Name** if necessary and add an optional **Description** for the new graphmart.

6. If you would like to configure any of the advanced settings, click **Advanced** to view the options. For details about the each of the Advanced settings, see Direct Load Advanced Settings Reference.
7. When you have finished configuring the workflow, click **Create Graphmart**. The new graphmart is created and activated and the data layers and steps are generated according to the chosen strategy. A managed model is also generated. See Managed Model Concepts for information. If you chose to export a dataset, the new dataset is also added to the Datasets catalog.

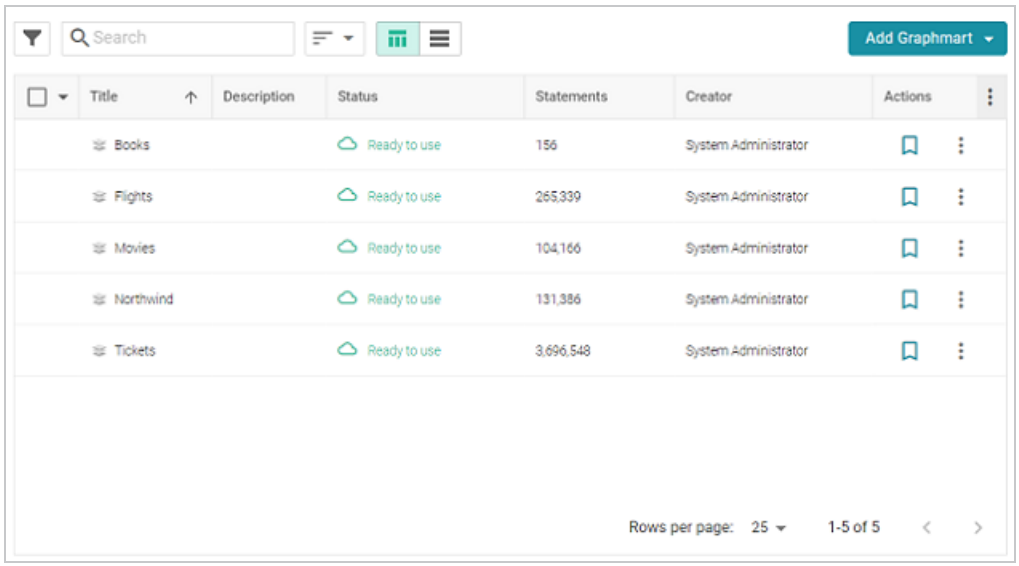
Once the graphmart is online, the data can be analyzed. See Access & Analyze for next steps. Or see Working with Graphmarts for information about managing graphmarts.

Adding a Data Source to an Existing Graphmart

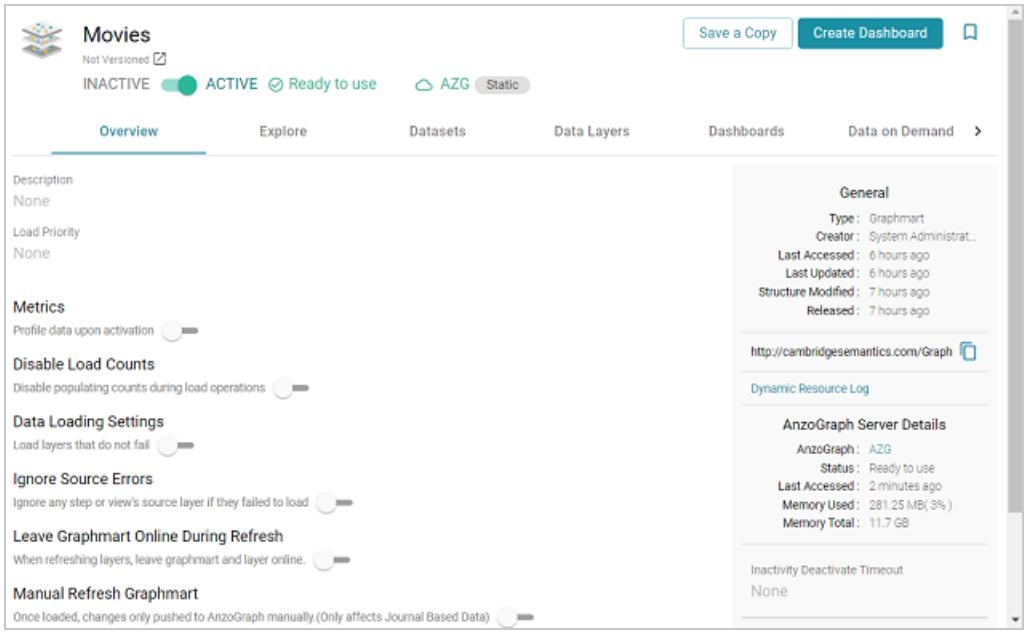
Follow the steps below if you want to add a data source to an existing graphmart.

> **Note**
> AnzoGraph uses the Graph Data Interface (GDI) Java plugin to connect directly to sources. For file-based sources, make sure the source files are available to AnzoGraph on the shared file store. For databases, if you have configured custom drivers to access those sources in Anzo, the same drivers need to be added to AnzoGraph. For instructions, see Deploy Optional Drivers for Accessing Custom Database Sources in the Deployment Guide.

1. If necessary, add the data source. See Adding Data Sources for instructions.
2. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:

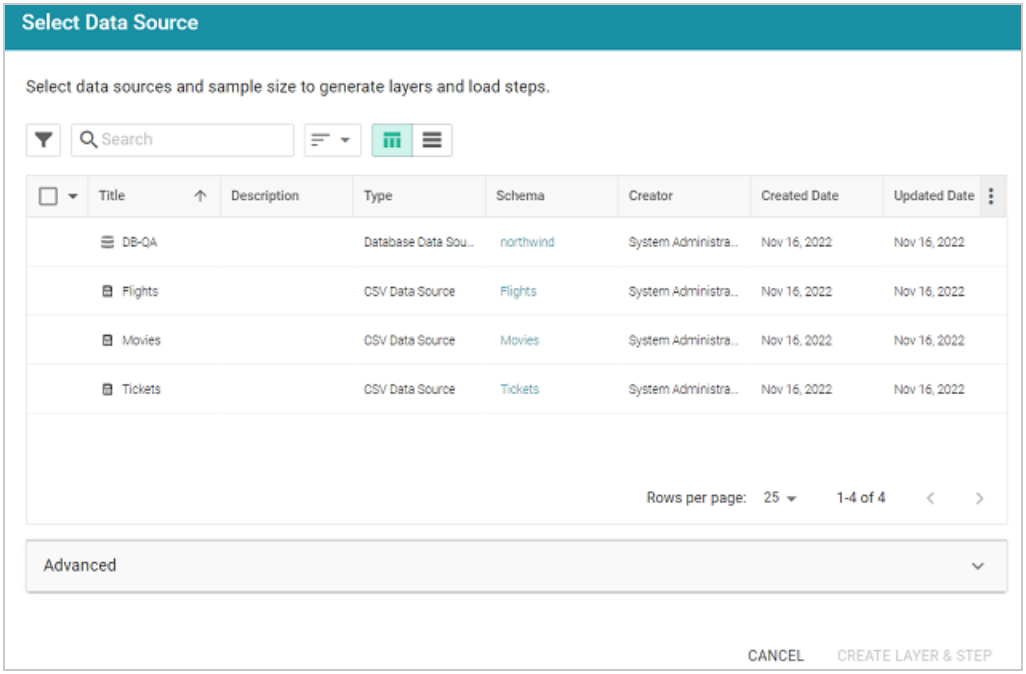


3. Click the name of the graphmart that you want to add the data source to. The Overview is displayed. For example:

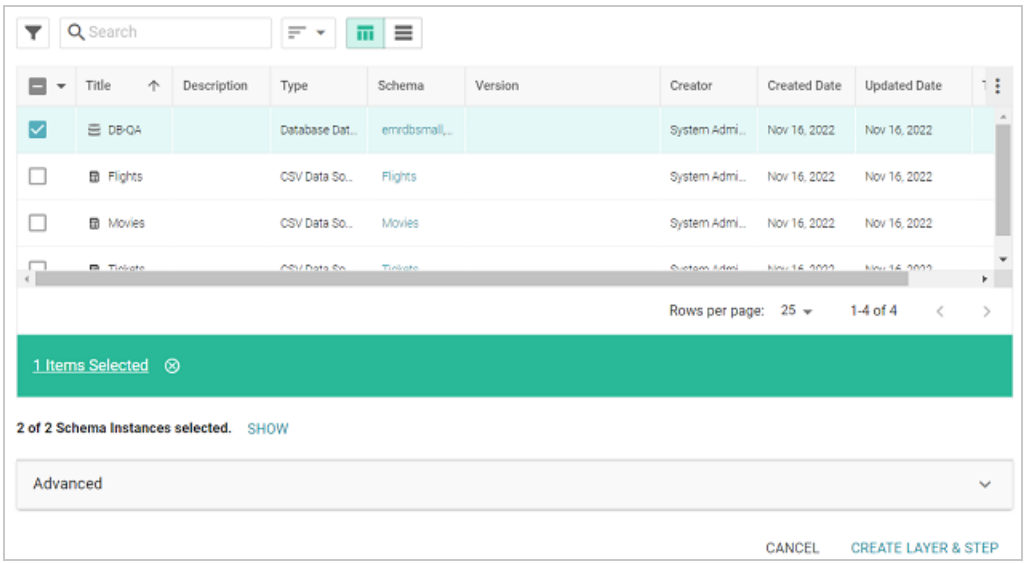


4. If necessary, activate the graphmart. Graphmarts must be online to be able to add data sources to them.

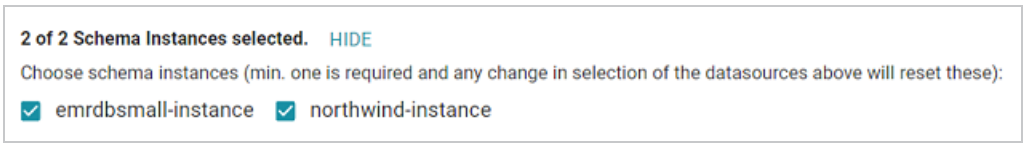
5. Click the **Data Layers** tab. On the Data Layers tab, click **Add** on the right side of the screen and select **Data Source**. The Select Data Source dialog box is displayed. For example:



6. Select the checkbox next to the data source that you want to add to the graphmart. (Users with the **Batch Direct Data Loading** permission can select multiple sources.) The selected schema or schemas are shown at the bottom of the screen. For example, in the image a below, a source with two schema instances is selected.



Clicking **Show** displays the selected schemas. For example:



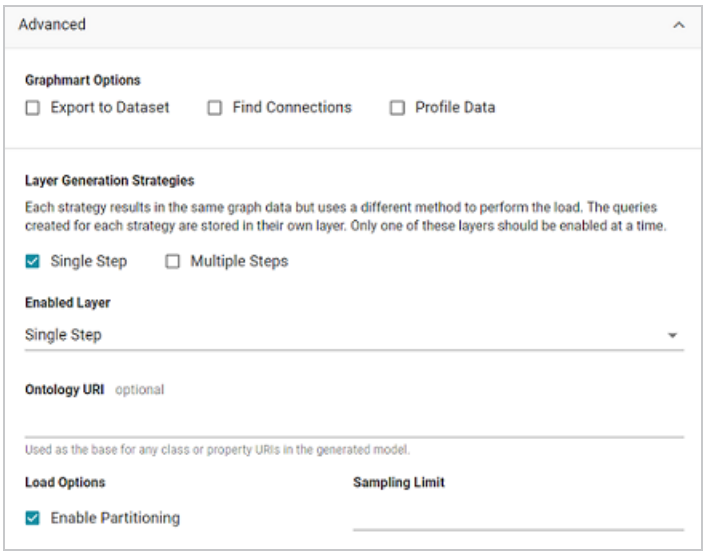
If you would like to exclude one or more schemas, clear the checkbox for each schema that you want to exclude.

7. If you want to configure any of the advanced settings, click **Advanced** to view the options. For details about the each of the Advanced settings, see Direct Load Advanced Settings Reference.
8. When you have finished configuring the workflow, click **Create Layer & Step**. The new layer is created and the steps are generated according to the chosen strategy. A managed model is also generated for the layer. See Managed Model Concepts for information.

Once the layer is online, the data can be analyzed. See Access & Analyze for next steps. Or see Working with Graphmarts for information about managing graphmarts.

Direct Load Advanced Settings Reference

This topic describes the Advanced options that are available when you create or configure the Direct Data Load workflow to load a data source via auto-generated data layers.



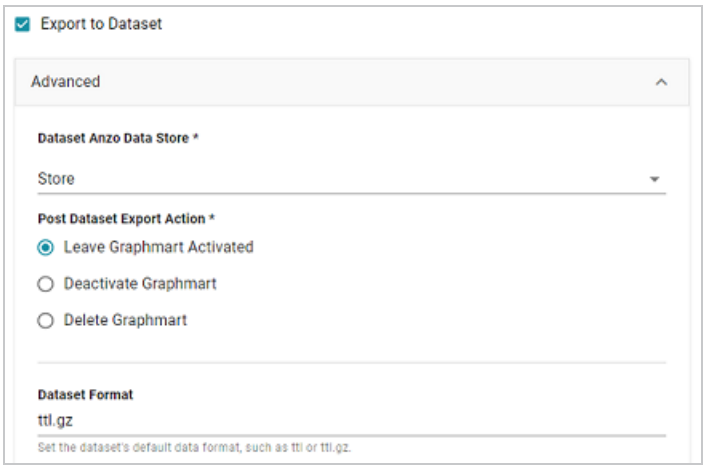
- Graphmart Options
  - Export to Dataset
  - Find Connections
  - Profile Data
- Layer Generation Strategies
  - Single Step
  - Multiple Steps
  - Enabled Layer
- Ontology URI
- Enable Partitioning
- Sampling Limit

Graphmart Options

The Graphmart Options are available when you create a graphmart from a data source (as described in Creating a Graphmart from a Data Source). They are not available when you add a data source to an existing graphmart (as described in Adding a Data Source to an Existing Graphmart).

Export to Dataset

This setting controls whether the automated workflow generates a dataset by exporting the graphmart. This option does not create an Export Step in the graphmart but it does generate a dataset in the selected data store and adds the dataset to the Datasets catalog. If you leave Export to Dataset disabled, a dataset is not automatically generated. For information about creating an Export Step to export a graphmart to a dataset at a later time, see Export Data to an FLDS (Export Step). If you enable Export to Dataset, the following settings are displayed. The list below the image describes the settings.



- **Dataset Anzo Data Store**: This required setting specifies the Anzo Data Store to export the dataset to. The data store must be a location on the shared file store that AnzoGraph has access to.
- **Post Dataset Export Action**: This required setting specifies how to treat the graphmart after the Export Step is processed. To leave the graphmart online, select **Leave Graphmart Activated** (the default value). To retain the graphmart but disable it and remove the data from

AnzoGraph, select **Deactivate Graphmart**. And to designate the graphmart as temporary and remove it after the dataset is exported, select **Delete Graphmart**.

- **Dataset Format**: This setting specifies the file format for the RDF TTL files that are generated, i.e., whether they are compressed or not. The valid options are **ttl** for uncompressed and **ttl.gz** for compressed.

Find Connections

This optional setting specifies whether to find relationships between tables in the schema (or between data sources if multiple sources are selected). Finding connections is useful if the schema does not define primary and foreign key relationships and you want the Graph Data Interface to create the connections.

> **Note**
> When **Find Connections** is enabled, two models are created, one that contains the classes and properties and one that contains only the connections. To view the complete model, both models must be added to the Working Set in the Model viewer.

Profile Data

This optional setting specifies whether to generate a Data Profile after the graphmart is activated. For information about the metrics that are run when a profile is generated, see Data Profiling Metrics.

Layer Generation Strategies

These settings control the strategy to use for auto-generating the data layer queries in the graphmart. Each strategy produces the same graph data but uses a different method for structuring the queries that produce the data. You can select both options if you want and review the resulting layers and steps. Each option results in a separate layer. However, only one of the resulting layers can be enabled by default.

Single Step

This is the default strategy. When **Single Step** is selected, a layer with a single Direct Load Step is created. The single query loads all tables from the selected schema or schemas. The generated query is an RDF and Ontology Generator query. See Onboarding Data with a Direct Load Step to learn more about the GDI RDF and Ontology Generator.

Multiple Steps

When Multiple Steps is selected, the layer has a separate Direct Load Step for each table in the selected schema or schemas. The generated query in each step is also an RDF and Ontology Generator query. With this strategy, you can enable and disable certain steps to control which tables are included in the graphmart.

Enabled Layer

This required setting specifies the data layer that should be enabled by default when the graphmart is activated.

Ontology URI

This optional setting specifies the custom URI to use for the model that is automatically generated. The value must be a valid URI without a hash (#) or slash (\) character at the end. If you do not specify a custom URI, the Graph Data Interface generates a URI in the following format:

```
http://cambridgesemantics.com/Layer/<layer_ID>/Model
```

Enable Partitioning

This option specifies whether to enable file partitions for file-based data sources. When file partitions are enabled, files will be partitioned and ingested in parallel for increased performance.

> **Note**
> Multiline CSV files may fail to load when this option is enabled. If the following type of error message is returned when onboarding files, disable the Enable File Partitions setting:
>
> ```
> File uses multiline records and cannot be segmented.
> Please disable segmenting for this file.
> ```

Sampling Limit

This optional setting specifies the number of rows to scan before inferring the data types for each column.

Onboarding or Virtualizing Data with SPARQL Queries

The topics in this section provide information about exploring, analyzing, virtualizing, and ingesting data by writing federated SPARQL queries that invoke the Graph Data Interface (GDI), an AnzoGraph plugin that enables you to connect directly to sources and control all aspects of the extract, load, and transform process. Depending on the type of query you write, you can ingest data into Anzo or create a virtual graph that accesses the source only when it is needed without ingesting the data.

Tip

This section focuses on writing your own SPARQL queries to read, ingest, or virtualize data from various sources. The GDI can also be used to automatically generate models, data layers, and steps with ingestion queries that you can edit. For more information about the automated workflow, see Onboarding Data with the Automated Workflow.

In this section:

Introduction to the GDI 117

GDI Concepts and Basic Usage 120

Options for Data Types, Data Linking, and Models 189

Advanced Usage by Data Source Type 205

GDI Property Reference 339

Introduction to the GDI

The Graph Data Interface (GDI) (sometimes called the Data Toolkit) is an extremely flexible and configurable AnzoGraph plugin that enables users to access a variety of data sources via federated SPARQL queries. Depending on the type of query you write, i.e., whether it is an INSERT query against the GDI service or a CONSTRUCT query against the view or virtualized service, you can ingest source data into Anzo or create a virtual graph that accesses the source only when it is needed without ingesting the data into Anzo.

The GDI has built-in, native support for various file format types, HTTP/REST endpoints, and common database types. Internally, the GDI API has a records-oriented view of data. This view enables the GDI to bridge graph operations to operations for data in other formats. Though the GDI views the source as rows in a table, ultimately it has the capability to convert the records to graph format, enabling the data to be incorporated into data layers to augment existing data.

Tip
When you query a source such as a database, the GDI service leverages that source to retrieve only the data that it needs for the query. Unlike a JDBC driver, the GDI service does not need to retrieve all values and then complete an often time-consuming step to filter the results.

Supported Data Sources

This table below lists the data sources, file systems, and applications that the GDI supports.

Source	Description
HTTP/REST Endpoints	The GDI natively supports reading or ingesting data from HTTP/REST endpoints.
Databases	Cambridge Semantics supplies JDBC drivers for the following databases: • Databricks • H2 • IBM DB2 • Microsoft SQL Server • MariaDB • Oracle • PostgreSQL • SAP Sybase (jTDS)

Source	Description
	• Snowflake To extend the service to access other databases, additional JDBC drivers can be added to AnzoGraph. For information about acquiring additional JDBC drivers, contact your Cambridge Semantics Customer Success manager. For instructions on deploying drivers, see Deploy Optional Drivers for Accessing Custom Database Sources in the Deployment Guide.
File Formats	The following file types are supported: • CSV and TSV • JSON and NDJSON • Parquet • SAS (SAS Transport XPT and SAS7BDAT formats) • XML • Raw text format
File Systems	The following types of file storage systems are supported: • Amazon S3 • FTP & FTPS • Google Cloud Storage • HDFS (Kerberized HDFS is not supported at this time.) • NFS • SFTP • WebDAV
Applications	Queries against Elasticsearch and Kafka applications are supported.

GDI Concepts and Basic Usage

The topics in this section help you get to know the Graph Data Interface (GDI) by introducing you to the main concepts and giving a general overview of the query syntax, available properties, and functionality that is applicable across query and data source types.

- Getting Started with GDI Queries
- Onboarding Data with a Direct Load Step
- Reading a Data Source's Metadata
- Paginating Requests
- Binding and Hierarchy Concepts
- Incremental Onboarding Concepts

Getting Started with GDI Queries

This topic provides details about the structure to use when writing GDI queries. It focuses on the properties that are common to all types of data sources. It also includes example queries that demonstrate the data integration capabilities for different types of sources.

> **Tip**
> Rather than manually writing complex queries, you can use the GDI to automatically generate graphs and ontologies by including a few key statements in a relatively simple query. For information, see Onboarding Data with a Direct Load Step.

- GDI Query Syntax
- GDI Query Examples

GDI Query Syntax

The following query syntax shows the structure of a GDI query. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
```

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:source_type ;
# Based on the source_type, additional connection and input parameters are
# available. The options below are valid for all sources. For source-related
# options, see GDI Property Reference.
        s:url "string" ;
        [ s:username "string" ; ]
        [ s:password "string" ; ]
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
```

```
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ]
        # Mapping variables
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.

Option	Type	Description
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed,

Option	Type	Description
		i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
source_type	object	The `?data a s:source_type` triple pattern specifies the type of data source that the query will run against. For example, `?data a s:DbSource`, specifies that the source type is a database. The list below describes the available types: • **DbSource** to connect to any type of database. • **FileSource** for flat files. The supported file types are CSV and TSV, JSON, NDJSON, XML, Parquet, and SAS (SAS Transport XPT and SAS7BDAT formats). The GDI automatically determines the file type from the file extensions. When querying file sources, make sure that the files are accessible to both Anzo and AnzoGraph. • **HttpSource** to connect to HTTP endpoints. • **ElasticSource** to connect to Elasticsearch indexes on an Elasticsearch server. • **KafkaSource** to connect to Kafka streaming sources. • **MetadataSource** for metadata discovery. **Tip** Certain connection and input parameters are available based on the specified source type. For details about the options for your source, see GDI Property Reference.
url	string	This property specifies the URL for the data source, such as the database URL, Elasticsearch URL, or HTTP endpoint URL. For file-based sources, the `url` property specifies the file system

Option	Type	Description

		location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the `pattern` and/or `maxDepth` properties described in FileSource Properties.
		
		Important
		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example, the triple patterns below reference keys from a Query Context:
		`?data a s:DbSource ;`
		`s:url "{{@db.eca4bfa8...ff9a.url}}" ;`
		`s:username "{{@db.eca4bfa8...ff9a.user}}" ;`
		`s:password "{{@db.eca4bfa8...ff9a.password}}" ;`

username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout`

Option	Type	Description
		`5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

Option	Type	Description

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader

Option	Type	Description
		table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.

Option	Type	Description
normalize	boolean and/or RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_ variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Note** The parentheses around the binding, data type, and format specifications are not required but are included in this document for readability.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a

Option	Type	Description
		source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column. For example for a flat source like CSV, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline ("AIRLINE")`. For a database source, this example binds the ?subject variable by navigating to the SUBJECT column in the FILM table in the dbo schema: `?subject ("dbo.FILM.SUBJECT")`. And for an HTTP source, this example binds the ?time variable to the time object under the minutely data path: `?time ("minutely.data.time")`. **Note** For **FileSource** and **HttpSource**, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern could be written as `?averagePerDay ("average\\/day")`. For **DbSource**, database, schema, and table names in bindings are parsed according to the specific rules for that database type. You do not need to escape characters in database names. However, database names with

Option	Type	Description
		characters that do not match `(_|A-Z|a-z)(_|A-Z|a-z|0-9)*` should be quoted, such as `("'Adventure.Works'.Sales.'Daily.Totals'").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date`

Option	Type	Description
		"dd-MMM-yy^1990". When one of those values is specified, 02-04-99 is parsed to 02-04-1999.

GDI Query Examples

The query below reads data from a sample HTTP source that compiles worldwide weather statistics. The source has several models available for retrieving data that is current, daily, historical, etc. To target current data, the query includes `s:selector "currently"` as an input parameter. In addition, the query demonstrates the use of the "topdown" functionality, where the query sends values to the source to narrow the results. The query includes the TOPDOWN keyword in the GDI service call, and the VALUES clause specifies the latitude and longitude values for the cities to return data for. In addition, since this sample source requires parameters to be specified in the connection URL, the `s:url` value includes `?lat` and `?long` as parameters as part of the value.

```
PREFIX s:      <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
PREFIX anzo:   <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:   <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>
PREFIX ex:     <http://example.org/ontologies/City#>

SELECT
       ?city ?state ?summary ?temp ?rainChance
       ?humidity ?pressure ?windSpeed
WHERE
{
       SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit>
    {
         ?data a s:HttpSource ;
         s:url "https://sampleEndpoint.com/forecast/{{?lat}},{{?long}}" ;
         s:selector "currently" ;
         ?lat ("latitude") ;
         ?long ("longitude") ;
```

```
        ?temp ("temperature") ;
        ?rainChance ( "precipProbability" ) ;
        ?humidity () ;
        ?pressure () ;
        ?windSpeed () .
    }
    VALUES( ?city ?state ?lat ?long )
    {
        ( "Lakeway" "TX" 30.374563 -97.975892 )
        ( "Boston" "MA" 42.358043 -71.060415 )
        ( "Seattle" "WA" 47.590720 -122.307053 )
        ( "Chicago" "IL" 41.837741 -87.823296 )
        ( "Hilo" "HI" 19.702040 -155.090312 )
    }
}
ORDER BY ?city
```

The query returns the following results:

```
city    | state | summary       | temp  | rainChance | humidity | pressure | windSpeed
--------+-------+---------------+-------+------------+----------+----------+-----------
Boston  | MA    | Overcast      | 79.81 |          0 |      0.6 |   1018.7 |      7.71
Chicago | IL    | Clear         |  81.7 |          0 |     0.52 |   1021.1 |      5.13
Hilo    | HI    | Partly Cloudy |  72.6 |       0.13 |     0.79 |   1018.6 |      4.86
Lakeway | TX    | Partly Cloudy | 92.43 |          0 |     0.48 |   1013.3 |     10.85
Seattle | WA    | Mostly Cloudy | 61.82 |          0 |     0.76 |   1018.2 |      4.57
5 rows
```

The example below ingests data into a data layer from a database source using an INSERT query in a Direct Load Step.

```
PREFIX s:       <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:     <http://www.w3.org/2002/07/owl#>
PREFIX anzo:    <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:    <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
PREFIX :        <http://example.com/ontologies/kl_hosp#>

INSERT
{
```

```
        GRAPH ${targetGraph}
    {
        ?InputEvent_cv a :InputEvent_cv ;
            :row_id ?row_id ;
            :subject_id ?subject_id ;
            :hadm_id ?hadm_id ;
            :icustay_id ?icustay_id ;
            :charttime ?charttime ;
            :itemid ?itemid ;
            :amount ?amount ;
            :amountuom ?amountuom ;
            :rate ?rate ;
            :rateuom ?rateuom ;
            :storetime ?storetime ;
            :cgid ?cgid ;
            :orderid ?orderid ;
            :linkorderid ?linkorderid ;
            :stopped ?stopped ;
            :newbottle ?newbottle ;
            :originalamount ?originalamount ;
            :originalamountuom ?originalamountuom ;
            :originalroute ?originalroute ;
            :originalrate ?originalrate ;
            :originalrateuom ?originalrateuom ;
            :originalsite ?originalsite .
    }
}
WHERE
{
        SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
        ?data a s:DbSource ;
            s:url "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.url}}" ;
             s:username "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.user}}"
             s:password "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.password}}"
            s:selector "kl_hosp_schema.inputevents_cv" ;
            ?row_id (xsd:int) ;
            ?subject_id (xsd:int) ;
            ?hadm_id (xsd:int) ;
            ?icustay_id (xsd:int) ;
            ?charttime (xsd:dateTime) ;
            ?itemid (xsd:int) ;
            ?amount (xsd:float) ;
```

```
            ?amountuom (xsd:string) ;
            ?rate (xsd:float) ;
            ?rateuom (xsd:string) ;
            ?storetime (xsd:dateTime) ;
            ?cgid (xsd:int) ;
            ?orderid (xsd:int) ;
            ?linkorderid (xsd:int) ;
            ?stopped (xsd:string) ;
            ?newbottle (xsd:int) ;
            ?originalamount (xsd:float) ;
            ?originalamountuom (xsd:string) ;
            ?originalroute (xsd:string) ;
            ?originalrate (xsd:float) ;
            ?originalrateuom (xsd:string) ;
            ?originalsite (xsd:string) ;
       BIND(IRI("http://example.com/inputevent_cv/{{?row_id}}") AS ?InputEvent_cv)
       BIND(IRI("http://example.com/patients/{{?subject_id}}") AS ?patient)
       BIND(IRI("http://example.com/admissions/{{?hadm_id}}") AS ?admission)
    }
}
```

The following query ingests airport-related data from a CSV file.

```
PREFIX s:       <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:     <http://www.w3.org/2002/07/owl#>
PREFIX anzo:    <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:    <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:      <http://purl.org/dc/elements/1.1/>

INSERT
{
  GRAPH ${targetGraph}
  {
   ?code a <http://anzograph.com/airport> ;
      <http://anzograph.com/airport/name> ?name ;
      <http://anzograph.com/airport/city> ?city ;
      <http://anzograph.com/airport/state> ?state ;
      <http://anzograph.com/airport/latitude> ?lat;
      <http://anzograph.com/airport/longitude> ?long.
  }
```

```
}
WHERE
{
        SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
        ?data a s:FileSource ;
           s:url "/opt/shared-files/airports.csv" ;
           ?iata_code ("IATA_CODE" xsd:string) ;
           ?name ("AIRPORT" xsd:string) ;
           ?city ("CITY" xsd:string) ;
           ?state ("STATE" xsd:string) ;
           ?lat ("LATITUDE" xsd:double) ;
           ?long ("LONGITUDE" xsd:double).
        BIND(IRI("http://anzograph.com/airport/{{?iata_code}}") as ?code)
    }
}
```

The query below creates a view of a database source.

```
PREFIX s:    <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX anzo: <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl: <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX ont:  <http://cambridgesemantics.com/ont/autogen/Rh/MIMIC-III-Data_Source/mimic_
iii_schema#>

CONSTRUCT
{
   ?caregiversURI a ont:caregivers ;
      ont:caregivers_cgid ?cgid ;
      ont:caregivers_description ?description ;
      ont:caregivers_label ?label .
}
WHERE
{
   GRAPH ?g
   {
      SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})
{
```

```
    {
        ?data a s:DbSource ;
          s:url "{{@db.eca4bfa83481f3638b93ab5fdf93dd9a.url}}" ;
          s:username "{{@db.eca4bfa83481f3638b93ab5fdf93dd9a.user}}"
          s:password "{{@db.eca4bfa83481f3638b93ab5fdf93dd9a.password}}"
          s:selector "mimic_iii_schema.caregivers" ;
          ?row_id (xsd:int) ;
          ?cgid (xsd:int) ;
          ?label (xsd:string) ;
          ?description (xsd:string) .
      BIND(IRI("http://cambridgesemantics.com/class/caregivers/{{?row_id}}") AS
?caregiversURI) }
    }
  }
}
```

Onboarding Data with a Direct Load Step

With no mapping required, Anzo's Direct Load Step functionality automatically generates a graph and an ontology (model) for a data source. Using a relatively simple SPARQL query, the direct load option invokes the Graph Data Interface (GDI) RDF and Ontology Generators. The GDI Generators recognize the structure of a data source and automatically generate the necessary statements.

Invoking the Generators is preferable to producing a hand-written query, especially when the structure of the data is very complex, such as a JSON data source with many inner repeating structures or a database with many tables and keys. When the source contains complex structures, the GDI will generate only the required statements and avoid cross-products, optimizing query execution and memory usage. In addition, the GDI Generator parallelizes the load across the AnzoGraph cluster so that a data source (such as a database) can be ingested with a single query.

This topic provides details about invoking the GDI RDF and Ontology Generators. The Generators can be used with all of the supported data source types.

- How to Use the GDI Generator
- GDI Generator Query Syntax
- GDI Generator Example Queries

How to Use the GDI Generator

To invoke the GDI Generator in a data layer, you add a **Direct Load Step** to the layer. In the Direct Load Step, you compose a SPARQL query that incorporates the GDI Generator parameters as detailed below in GDI Generator Query Syntax.

> **Tip**
> For instructions on adding steps to layers, see Adding Steps to Layers.

Why Use a Direct Load Step?

It is important to use a Direct Load Step with the RDF and Ontology Generators because it is the only type of step with the ability to manage the generated ontologies (models). An ontology that is generated in a Direct Load Step is automatically registered in Anzo. The registered model is linked to and managed by the data layer that contains the step. If an Ontology Generator query is changed, additional Direct Load Steps are added to the same layer, or the underlying source schema changes, the managed model is automatically updated when the graphmart is reloaded or refreshed. See Managed Model Details below for important details about layer-managed models.

Managed Model Details

Though an ontology that is generated in a Direct Load Step is registered in Anzo and is available for viewing in the Model editor, **the model is owned and managed by the data layer that contains the Direct Load Step**. That means any manual changes made to the model outside of the step, such as from the Model editor, will be overwritten any time the graphmart or layer is refreshed or reloaded. **Do not modify generated managed models except by editing (or adding) Direct Load Step queries**. For information on updating managed models, see Editing a Managed Model.

There is only one managed model per layer. If you include multiple Direct Load Steps in the same layer, they will all update the same ontology. This functionality can be useful if you want to align the data and generated model across multiple steps. If you have multiple sources that are not intended to align or update the same model, create separate layers.

If you delete a layer that includes a managed model, the model is also deleted. Use caution when referencing a managed model outside of a graphmart. For example, if you create a dataset and reference a managed model when you select the ontology, the reference will break if the data layer that manages the model is deleted.

GDI Generator Query Syntax

The following query syntax shows the structure of a GDI Generator query. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:    <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

#Result Clause
INSERT {
   GRAPH ${targetGraph} {
     ?s ?p ?o .
  }
}
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:source_type ;
# Based on the source_type, additional connection and input parameters are
# available. The options below are valid for all sources. For source-related
# options, see GDI Property Reference.
        s:url "string" ;
        [ s:model "class_name_for_this_source" ; ]
        [ s:username "string" ; ]
        [ s:password "string" ; ]
        [ s:timeout int ; ]
        [ s:maxConnections int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
```

```
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ source_normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ] .

      # Multiple data sources can be merged if they project a similar set
      # of output variables. Make sure each source has a unique subject variable.

        [ ?unique_variable a s:source_type ;
          ...
        . ]

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
        s:as (?s ?p ?o);
        s:ontology ontology_uri ;
        s:base base_uri ;
        [ s:normalize boolean | [ global_normalization_rules ] ; ]
        .
      # Additional clauses such as BIND, FILTER
   }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.

Option	Type	Description
Result Clause	N/A	The result clause for Direct Load Steps is typically an INSERT query with the graph pattern in the template above. **Note** It is important to include the **GRAPH** keyword and target graph parameter **${targetGraph}** when you are writing an INSERT query. Anzo automatically replaces the ${targetGraph) parameter with the appropriate target URI(s) when the query runs.
source_type	object	The `?data a s:source_type` triple pattern specifies the type of data source that the query will run against. For example, `?data a s:DbSource`, specifies that the source type is a database. The list below describes the available types: • **DbSource** to connect to any type of database. • **FileSource** for flat files. The supported file types are CSV and TSV, JSON, NDJSON, XML, Parquet, and SAS (SAS Transport XPT and SAS7BDAT formats). The GDI automatically determines the file type from the file extensions. When querying file sources, make sure that the files are accessible to both Anzo and AnzoGraph. • **HttpSource** to connect to HTTP endpoints. • **ElasticSource** to connect to Elasticsearch indexes on an Elasticsearch server.

Option	Type	Description
		• **KafkaSource** to connect to Kafka streaming sources. **Tip** Certain connection and input parameters are available based on the specified source type. For details about the options for your source, see GDI Property Reference.
url	string	This property specifies the URL for the data source, such as the database URL, Elasticsearch URL, or HTTP endpoint URL. For file-based sources, the `url` property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the `pattern` and/or `maxDepth` properties described in FileSource Properties. **Important** For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example, the triple patterns below reference keys from a Query Context:

Option	Type	Description
		``` ?data a s:DbSource ; s:url "{{@db.eca4bfa8...ff9a.url}}" ; s:username " {{@db.eca4bfa8...ff9a.user}}" ; s:password " {{@db.eca4bfa8...ff9a.password}}" ; ```
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
maxConnections	int	For database sources, this property can be used to set a limit on the maximum number of active connections to the source. For example, `s:maxConnections 16` sets the

Option	Type	Description
		limit to 16 connections. The default value is `10`.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`

Option	Type	Description
		`s:executorsPerNode 8 ;` `] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.

Option	Type	Description
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time,

Option	Type	Description
		and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
RdfGenerator	object	Include this property to invoke the RDF Generator. If you only want to generate a model without RDF, you can exclude `RdfGenerator`.
OntologyGenerator	object	Include this property to invoke the Ontology Generator. If you only want to generate RDF without a model, you can exclude `OntologyGenerator`.
as	N/A	This property provides the variable bindings for the RDF Generator's projection to RDF. Typically the value is `s:as (?s ?p ?o)` to match the variables in the result clause.

Option	Type	Description
ontology	URI	This property specifies the URI to use as the base URI for any generated ontology artifacts. For example, `s:ontology <http://abc.com/ontologies/MyOntology>`. **Note** In the graphmart, the data layer ID is appended to the ontology URI that is generated. The complete URI is based on the layer and cannot be customized.
base	URI	This property specifies the base URI for instance data. The base value should NOT end in **#**. The Generator will add a trailing slash (/) if one does not exist. For example, `s:base <http://abc.com/>`.

GDI Generator Example Queries

This section includes sample queries that may be useful as a starting point for writing your own RDF and Ontology Generator queries.

- Basic Query that Generates RDF and Ontology for a JSON File
- Basic Query that Generates an Ontology for a Directory of CSV Files
- Query that Normalizes and Generates RDF and Ontology for a Database
- Query with Query Context that Normalizes and Generates RDF and Ontology for a Database
- Query for Multiple Sources that Generates RDF and Ontology with Resource Templates and Object Properties

Basic Query that Generates RDF and Ontology for a JSON File

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
     ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:FileSource ;
         s:model "People" ;
         s:url "/opt/shared-files/json/people.json" .

      ?rdf a s:RdfGenerator , s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/ontologies/People> ;
         s:base <http://cambridgesemantics.com/data/> .
  }
}
```

Basic Query that Generates an Ontology for a Directory of CSV Files

```
PREFIX s:  <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
     ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:FileSource ;
         s:model "Sales" ;
         s:url "/opt/shared-files/csv/sales" ;
         s:format [
            s:delimiter "," ;
            s:headers true ;
            s:comment "#" ;
```

```
            s:quote "\"" ;
            s:maxColumns 22 ;
         ] .

      ?rdf a s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/ontologies/Sales> ;
         s:base <http://cambridgesemantics.com/data/> .
  }
}
```

Query that Normalizes and Generates RDF and Ontology for a Database

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:DbSource ;
      s:url "jdbc:mysql://10.11.12.9/emrdbbig" ;
      s:username "root" ;
      s:password "sql1@#" ;
      s:normalize [
         s:model [
            s:removeStart "emr_" ;
            s:words "activity 'patient complaint' medication observation patient
specialty study" ;
         ] ;
         s:field [
            s:removePartialPrefix true ;
            s:words "provider description start end drug complaint date medication
normal code
                     observation product active dose generic route admin strength
collection
                     activity home first last status first year birth death directed
complex
```

```
                    period age flag gender language" ;
         ] ;
      ] .

      ?rdf a s:RdfGenerator , s:OntologyGenerator ;
      s:as (?s ?p ?o) ;
      s:ontology <http://cambridgesemantics.com/ontologies/EMR> ;
      s:base <http://cambridgesemantics.com/EMR> .
  }
}
```

Query with Query Context that Normalizes and Generates RDF and Ontology for a Database

The query below references a Query Context to supply the username and password for the database connection.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:DbSource ;
         s:url "jdbc:sqlserver://localhost;databaseName=AdventureWorks2012" ;
         s:username "{{@db.username}}" ;
         s:password "{{@db.password}}" ;
         s:schema "Production", "HumanResources", "Person", "Sales", "Purchasing" ;
         s:normalize [
            s:model [
               s:localNamePrefix "C_" ;
               s:localNameSeparator "_" ;
               s:match [ s:pattern "(.+)Enlarged" ; s:replace "$1" ] ;
            ] ;
            s:field [
               s:localNamePrefix "P_" ;
               s:localNameSeparator "_" ;
               s:ignore "rowguid ModifiedDate" ;
```

```
               s:match (
                  [ s:pattern "(.+)GUID$" ; s:replace "$1" ]
                  [ s:pattern "(.+)ID$" ; s:replace "$1" ]
               ) ;
            ] ;
         ] .

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/ontologies/AdventureWorks> ;
         s:base <http://cambridgesemantics.com/AdventureWorks> .
  }
}
```

Query for Multiple Sources that Generates RDF and Ontology with Resource Templates and Object Properties

This query also includes global normalization rules for normalizing the data across all Data Sources.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?event a s:FileSource ;
         s:model "event" ;
         s:url  "/opt/shared-files/csv/events.csv" ;
         s:key ("EVENT_ID") .

      ?listing a s:FileSource ;
         s:model "listing" ;
         s:url " /opt/shared-files/csv/listings.csv" ;
         s:key ("LIST_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?date a s:FileSource ;
         s:model "date" ;
```

```
         s:url  "/opt/shared-files/csv/event_dates.csv" ;
         s:key ("DATE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?venue a s:FileSource ;
         s:model "venue" ;
         s:url " /opt/shared-files/csv/venues.csv" ;
         s:key ("VENUE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?sale a s:FileSource ;
         s:model "sale" ;
         s:url " /opt/shared-files/csv/sales.csv" ;
         s:key ("SALE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] ;
         s:reference [ s:model "listing" ; s:using ("LIST_ID") ; s:key ("LIST_ID") ] .

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/tickets> ;
         s:base <http://cambridgesemantics.com/data> ;
         s:normalize [
            s:all [
               s:casing s:UPPER ;
               s:localNameSeparator "_" ;
            ] ;
         ] .
   }
}
```

Reading a Data Source's Metadata

If you want to retrieve instance data from a source but are unsure about the data model, schema, or the exact names of columns and their data types, you can use the Graph Data Interface (GDI) to explore the source's metadata. The GDI can be used to return a list of the catalogs (schemas), models, columns, data types, and other data source information. This topic describes the metadata query syntax and provides several example queries.

- Metadata Query Syntax
- Metadata Query Examples

Metadata Query Syntax

The following query syntax shows the structure of a metadata query. The clauses, patterns, and placeholders in blue are described below.

```
# PREFIX Clause
PREFIX s:      <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
PREFIX anzo:   <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:   <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>

# Result Clause
SELECT *
WHERE
{
      # SERVICE Clause: Include the following service call
        SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
        [] s:select ?metadata .

        ?data a s:source_type ;
            s:url "string" ;
            [ s:username "string" ; ]
            [ s:password "string" ; ]

        ?metadata a s:MetadataSource ;
            s:from ?data ;

      # The metadata selector below specifies the type of metadata to return.
        ?catalogs | ?fields | ?models [
            ?metadata_type datatype ;
            ... ;
        ] .
    }
}
```

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the results to return. For metadata queries, the result clause is typically `SELECT *`.
SERVICE Clause		Include the required GDI SERVICE call in the WHERE clause. The rest of the WHERE clause defines the patterns to look for in the source.
[] s:select ?metadata	N/A	Include this required triple pattern in metadata queries. The select property specifies the source that should be used to return data.
source_type	object	The `?data a s:source_type` triple pattern specifies the type of data source that the query will run against. For example, `?data a s:DbSource`, specifies that the source type is a database. The list below describes the available types: • **DbSource** to connect to any type of database. • **FileSource** for flat files. The supported file types are CSV and TSV, JSON, NDJSON, XML, Parquet, and SAS (SAS Transport XPT and SAS7BDAT formats). The GDI automatically determines the file type from the file extensions. When querying file sources, make sure that the files are accessible to both Anzo and AnzoGraph. • **HttpSource** to connect to HTTP endpoints. • **ElasticSource** to connect to Elasticsearch indexes on an Elasticsearch server.

Option	Type	Description
		• **KafkaSource** to connect to Kafka streaming sources. **Tip** Certain connection and input parameters are available based on the specified source type. For details about the options for your source, see GDI Property Reference.

url	string	This property specifies the URL for the data source, such as the database URL, Elasticsearch URL, or HTTP endpoint URL. For file-based sources, the `url` property specifies the file system location of the source file or directory of files.
		
		Important
		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example, the triple patterns below reference keys from a Query Context:
		`?data a s:DbSource ;`
		`s:url "{{@db.eca4...93ff9a.url}}" ;`
		`s:username "{{@db.eca4...93ff9a.user}}" ;`
		`s:password "`
		`{{@db.eca4...93ff9a.password}}" ;`

username	string	If authentication is required to access the source, include this property to specify the user name.

Option	Type	Description
password	string	This property lists the password for the given username.
catalogs	variable	This selector narrows the results to schema-related metadata such as the schema names. Even when additional metadata types (metadata_type datatype) are specified as objects, only catalog (schema) information is returned.
fields	variable	This selector is the broadest and most flexible option. Using the `fields` selector enables users to return any and all of the source metadata information, depending on the specified metadata types (metadata_type datatype).
models	variable	This selector narrows the results to model-related metadata such as the model names. Even when additional metadata types (metadata_type datatype) are specified as objects, only model information is returned.
metadata_type datatype	N/A	The triple patterns in the array for the metadata selector specify the type of metadata to return as well as the data type for the return value. The following list shows all of the valid options. You can include any combination of properties. The results that are returned depend on the type of data source and whether the information exists in the source. The parentheses around the data type are not required but are included in this document for readability. • **?model (xsd:string)**: Returns model names in string format. For file sources, this property returns file names. • **?field (xsd:string)**: Returns column names. • **?catalog (xsd:string)**: Returns schema names. • **?datatype (owl:Thing)**: Returns the data types of the columns.

Option	Type	Description
		• **?keys (xsd:string)**: Returns primary and foreign key columns. For compound keys, the GDI returns a comma-separated list of columns comprising the key. • **?format (xsd:string)**: Returns the format of the source. • **?cardinality (xsd:string)**: Returns the cardinality of relationships between tables: optional, many, or required. • **?count (xsd:int)**: Returns the number of times the field appears in the source. • **?order (xsd:int)**: Returns the order in which the field was encountered.

Metadata Query Examples

This section includes sample metadata queries that run against different types of data sources.

- List Database Schemas
- Explore a Database Schema
- Explore a Directory of SAS Files
- Explore an HTTP Endpoint
- Explore a Directory of CSV Files

List Database Schemas

The query below sends a metadata query to a MySQL database to return a list of the schemas that are available:

```
PREFIX s:      <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
```

```
PREFIX anzo:   <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:   <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
          [] s:select ?metadata .

          ?data a s:DbSource ;
                s:url "jdbc:mysql://10.100.2.9:5555/?user=root&password=Mysql1@#" .

          ?metadata a s:MetadataSource ;
                s:from ?data ;
          ?catalogs [
                ?catalog (xsd:string) ;
                ?order (xsd:int) ;
          ] .
   }
}
ORDER BY ?catalog
```

The query returns the following results:

```
catalog               | order
----------------------+-------
BANKTEST_DB           |     1
EMR                   |     4
GOLFCLUB_DB           |     8
NORTHWIND             |    10
SPORTDB               |    13
SQLPOCKET_DB          |    14
WORDPRESS_DB          |    16
classicmodels         |     2
crm_national_patients |     3
emrdbbig              |     5
emrdbsmall            |     6
emrnational_schema    |     7
mysql                 |     9
optum                 |    11
performance_schema    |    12
```

```
sys                    |     15
16 rows
```

Explore a Database Schema

Using the list of schemas that were returned in the example above (List Database Schemas), the query below returns metadata about the columns in one of the schemas. To narrow the results to a schema, the schema name (NORTHWIND) is added to the connection URL.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
       [] s:select ?metadata .

       ?data a s:DbSource ;
              s:url "jdbc:mysql://10.100.2.9:5555/NORTHWIND?user=root&password=Mysql1@#" .

       ?metadata a s:MetadataSource ;
              s:from ?data ;
       ?fields [
              ?model (xsd:string) ;
              ?field (xsd:string) ;
              ?datatype (owl:Thing) ;
       ] .
   }
}
ORDER BY ?model
```

The query returns the following results:

```
model                          | field                | datatype
-------------------------------+----------------------+---------------------------------
-------------
Alphabetical list of products  | CategoryID           |
http://www.w3.org/2001/XMLSchema#int
Alphabetical list of products  | Discontinued         |
http://www.w3.org/2001/XMLSchema#boolean
Alphabetical list of products  | SupplierID           |
http://www.w3.org/2001/XMLSchema#int
Alphabetical list of products  | UnitPrice            |
http://www.w3.org/2001/XMLSchema#decimal
Alphabetical list of products  | ProductName          |
http://www.w3.org/2001/XMLSchema#string
Alphabetical list of products  | QuantityPerUnit      |
http://www.w3.org/2001/XMLSchema#string
Alphabetical list of products  | UnitsOnOrder         |
http://www.w3.org/2001/XMLSchema#short
Alphabetical list of products  | CategoryName         |
http://www.w3.org/2001/XMLSchema#string
Alphabetical list of products  | ProductID            |
http://www.w3.org/2001/XMLSchema#int
Alphabetical list of products  | ReorderLevel         |
http://www.w3.org/2001/XMLSchema#short
Alphabetical list of products  | UnitsInStock         |
http://www.w3.org/2001/XMLSchema#short
Categories                     | CategoryID           |
http://www.w3.org/2001/XMLSchema#int
Categories                     | Description          |
http://www.w3.org/2001/XMLSchema#string
Categories                     | Picture              |
http://www.w3.org/2001/XMLSchema#base64Binary
Categories                     | CategoryName         |
http://www.w3.org/2001/XMLSchema#string
Categories                     | categoryid           |
Category Sales for 1997        | CategoryName         |
http://www.w3.org/2001/XMLSchema#string
Category Sales for 1997        | CategorySales        |
http://www.w3.org/2001/XMLSchema#double
Current Product List           | ProductName          |
http://www.w3.org/2001/XMLSchema#string
Current Product List           | ProductID            |
http://www.w3.org/2001/XMLSchema#int
...
```

```
201 rows
```

Explore a Directory of SAS Files

The query below explores a directory of SAS files to return the model, catalog (schema), field, data type, and cardinality information. The query also orders the results by model name, which is the file name for file sources of a data model does not exist.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE
{
        SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
        [] s:select ?metadata .

        ?data a s:FileSource ;
                s:url "/opt/shared-files/sas" .

        ?metadata a s:MetadataSource ;
                s:from ?data ;

        ?fields [
                ?model (xsd:string) ;
                ?field (xsd:string) ;
                ?catalog (xsd:string) ;
                ?datatype (owl:Thing) ;
                ?cardinality (xsd:string) ;
        ] .
   }
}
ORDER BY ?model
```

The query returns the following results:

```
model   | field       | catalog | datatype                                  | cardinality
--------+-------------+---------+-------------------------------------------+------------
-
demand  | P1          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | P2          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | P3          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | Y           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | Q1          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | Q2          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demand  | Q3          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demo    | YEAR        | les/sas | http://www.w3.org/2001/XMLSchema#long   | REQUIRED
demo    | QTR         | les/sas | http://www.w3.org/2001/XMLSchema#long   | REQUIRED
demo    | GDP         | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demo    | PR          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demo    | M1          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
demo    | RS          | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
airline | YEAR        | les/sas | http://www.w3.org/2001/XMLSchema#long   | REQUIRED
airline | Y           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
airline | W           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
airline | R           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
airline | L           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
airline | K           | les/sas | http://www.w3.org/2001/XMLSchema#double | REQUIRED
cars    | MPG         | les/sas | http://www.w3.org/2001/XMLSchema#long   | REQUIRED
cars    | CYL         | les/sas | http://www.w3.org/2001/XMLSchema#long   | REQUIRED
...
50 rows
```

Explore an HTTP Endpoint

The query below explores the metadata for a sample HTTP source that compiles worldwide weather statistics.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

SELECT *
```

```
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
       [] s:select ?metadata .

       ?data a s:HttpSource ;
              s:url "https://sampleEndpoint.com/forecast/30.374563,-97.975892" .

       ?metadata a s:MetadataSource ;
              s:from ?data ;

       ?fields [
              ?model (xsd:string) ;
              ?field (xsd:string) ;
              ?datatype (owl:Thing) ;
              ?cardinality (xsd:string) ;
              ?order (xsd:int) ;
       ] .
   }
}
ORDER BY ?model ?order
```

The query returns the following results:

```
model     | field                       | datatype                                  |
cardinality | order
----------+-----------------------------+-------------------------------------------+----
---------+-------
currently | time                        | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |     6
currently | summary                     | http://www.w3.org/2001/XMLSchema#string |
REQUIRED    |     7
currently | icon                        | http://www.w3.org/2001/XMLSchema#string |
REQUIRED    |     8
currently | nearestStormDistance        | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |     9
currently | nearestStormBearing         | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    10
currently | precipIntensity             | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    11
currently | precipProbability           | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    12
```

```
currently | temperature                | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    13
currently | apparentTemperature        | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    14
currently | dewPoint                   | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    15
currently | humidity                   | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    16
currently | pressure                   | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    17
currently | windSpeed                  | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    18
currently | windGust                   | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    19
currently | windBearing                | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    20
currently | cloudCover                 | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    21
currently | uvIndex                    | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    22
currently | visibility                 | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    23
currently | ozone                      | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    24
daily     | summary                    | http://www.w3.org/2001/XMLSchema#string |
REQUIRED    |    75
daily     | icon                       | http://www.w3.org/2001/XMLSchema#string |
REQUIRED    |    76
daily     | data                       |                                         |
MANY        |    77
data      | time                       | http://www.w3.org/2001/XMLSchema#int    |
REQUIRED    |    29
data      | precipIntensity            | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    30
data      | precipProbability          | http://www.w3.org/2001/XMLSchema#float  |
REQUIRED    |    31
data      | summary                    | http://www.w3.org/2001/XMLSchema#string |
OPTIONAL    |    32
...
81 rows
```

The following query retrieves the model, field, and data type metadata for the United States from the publicly available Data API Covid Tracking Project.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
       [] s:select ?metadata .

       ?data a s:HttpSource ;
              s:url "https://covidtracking.com/api/v1/us/current.csv" .

       ?metadata a s:MetadataSource ;
              s:from ?data ;

       ?fields [
              ?model (xsd:string) ;
              ?field (xsd:string) ;
              ?datatype (owl:Thing) ;
       ] .
    }
}
```

The query returns the following results:

```
model | field                    | datatype
------+--------------------------+-----------------------------------------
us    | date                     | http://www.w3.org/2001/XMLSchema#string
us    | states                   | http://www.w3.org/2001/XMLSchema#string
us    | positive                 | http://www.w3.org/2001/XMLSchema#string
us    | negative                 | http://www.w3.org/2001/XMLSchema#string
us    | pending                  | http://www.w3.org/2001/XMLSchema#string
us    | hospitalizedCurrently    | http://www.w3.org/2001/XMLSchema#string
us    | hospitalizedCumulative   | http://www.w3.org/2001/XMLSchema#string
us    | inIcuCurrently           | http://www.w3.org/2001/XMLSchema#string
us    | inIcuCumulative          | http://www.w3.org/2001/XMLSchema#string
```

```
us    | onVentilatorCurrently    | http://www.w3.org/2001/XMLSchema#string
us    | onVentilatorCumulative   | http://www.w3.org/2001/XMLSchema#string
us    | recovered                | http://www.w3.org/2001/XMLSchema#string
us    | dateChecked              | http://www.w3.org/2001/XMLSchema#string
us    | death                    | http://www.w3.org/2001/XMLSchema#string
us    | hospitalized             | http://www.w3.org/2001/XMLSchema#string
us    | lastModified             | http://www.w3.org/2001/XMLSchema#string
us    | total                    | http://www.w3.org/2001/XMLSchema#string
us    | totalTestResults         | http://www.w3.org/2001/XMLSchema#string
us    | posNeg                   | http://www.w3.org/2001/XMLSchema#string
us    | deathIncrease            | http://www.w3.org/2001/XMLSchema#string
us    | hospitalizedIncrease     | http://www.w3.org/2001/XMLSchema#string
us    | negativeIncrease         | http://www.w3.org/2001/XMLSchema#string
us    | positiveIncrease         | http://www.w3.org/2001/XMLSchema#string
us    | totalTestResultsIncrease | http://www.w3.org/2001/XMLSchema#string
us    | hash                     | http://www.w3.org/2001/XMLSchema#string
25 rows
```

Explore a Directory of CSV Files

The query below explores a directory of CSV files to return the model, field, and data type. The query also orders the results by model name, which is the file name for file sources of a data model does not exist. In addition, the query includes `s:sampling true`, which means the GDI will scan the entire file or files before returning results.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE
{
       SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
       [] s:select ?metadata .

       ?data a s:FileSource ;
```

```
                s:url "/opt/shared-files/movie-csv" .

        ?metadata a s:MetadataSource ;
                s:from ?data ;

        # Sample the whole file
        s:sampling true ;

        # Sample the first N records #
        # s:sampling 1000 ;

        ?fields [
                ?model (xsd:string) ;
                ?field (xsd:string) ;
                ?datatype (owl:Thing) ;
        ] .
    }
}
ORDER BY ?model
```

The query returns the following results:

```
model                   | field                       | datatype
------------------------+-----------------------------+-----------------------------------
------
MovieActors1            | MovieID                     |
http://www.w3.org/2001/XMLSchema#int
MovieActors1            | MovieTitle                  |
http://www.w3.org/2001/XMLSchema#string
MovieActors1            | ActorID                     |
http://www.w3.org/2001/XMLSchema#int
MovieActors1            | ActorName                   |
http://www.w3.org/2001/XMLSchema#string
MovieActors2            | MovieID                     |
http://www.w3.org/2001/XMLSchema#int
MovieActors2            | MovieTitle                  |
http://www.w3.org/2001/XMLSchema#string
MovieActors2            | ActorID                     |
http://www.w3.org/2001/XMLSchema#int
MovieActors2            | ActorName                   |
http://www.w3.org/2001/XMLSchema#string
MovieActors2            | ActorCategory               |
http://www.w3.org/2001/XMLSchema#string
```

```
MovieCategory          | MovieID                   |
http://www.w3.org/2001/XMLSchema#int
MovieCategory          | MovieTitle                |
http://www.w3.org/2001/XMLSchema#string
MovieCategory          | MoveCategoryID            |
http://www.w3.org/2001/XMLSchema#int
MovieCategory          | MovieCategory             |
http://www.w3.org/2001/XMLSchema#string
MovieCinematographers  | MovieID                   |
http://www.w3.org/2001/XMLSchema#int
MovieCinematographers  | MovieTitle                |
http://www.w3.org/2001/XMLSchema#string
MovieCinematographers  | MovieCinematographerID    |
http://www.w3.org/2001/XMLSchema#int
MovieCinematographers  | MovieCinematographerName  |
http://www.w3.org/2001/XMLSchema#string
MovieComposers         | MovieID                   |
http://www.w3.org/2001/XMLSchema#int
MovieComposers         | MovieTitle                |
http://www.w3.org/2001/XMLSchema#string
MovieComposers         | MovieComposerID           |
http://www.w3.org/2001/XMLSchema#int
MovieComposers         | MovieComposerName         |
http://www.w3.org/2001/XMLSchema#string
MovieDirectors         | MovieID                   |
http://www.w3.org/2001/XMLSchema#int
MovieDirectors         | MovieTitle                |
http://www.w3.org/2001/XMLSchema#string
...
79 rows
```

The following example shows a query that returns metadata for an Elasticsearch source.

```
PREFIX s:    <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX anzo: <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl: <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX ex:   <http://example.org/ontologies/City#>
PREFIX es:   <http://elastic.co/search/>
```

```
PREFIX :     <http://example.org/cities/>

SELECT *
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
      [] s:select ?_fields .

      ?data a es:ElasticSource ;
         es:url "http://localhost:9200/" ;
         es:index "account" ;
         ?account_number xsd:long ;
         ?age (xsd:long) ;
         ?balance (xsd:long) ;
         ?address (xsd:string) ;
         ?city (xsd:string) ;
         ?state (xsd:string) ;
         ?email (xsd:string) ;
         ?employer (xsd:string) ;
         ?firstname (xsd:string) ;
         ?lastname (xsd:string) ;
         ?gender (xsd:string) .

      ?_fields a s:MetadataSource ;
         s:from ?data ;
         ?fields [
            ?catalog () ;
            ?model () ;
            ?field () ;
            ?cardinality () ;
            ?datatype () ;
            ?type () ;
            ?object () ;
         ] .
  }
}
ORDER BY ?catalog ?model ?field
```

For instructions on querying the instance data based on the data source metadata, see Getting Started with GDI Queries.

Paginating Requests

The GDI exposes paging models that enable you to access large amounts of data across a number of smaller requests. Paging is configured by including the **paging** property in a query and configuring a combination of the pagination options described below. The GDI supports keyset-based, page-based, cursor-based, and offset-based pagination. Paging is supported for all data source types.

- Paging Syntax
- Paging Examples

Paging Syntax

```
s:paging [
  s:key (?variable) ;
  s:page ?variable ;
  s:cursor ?variable ;
  s:offset ?variable ;
  s:size int ;
  s:limit ?variable ;
] ;
```

Option	Type	Description
key	variable	Include this property if you want to configure keyset-based pagination where a key is specified to act as a delimiter of the page. The `s:key` value is a variable that is bound to an expression that defines how to delimit the data. It is usually calculated by an aggregate expression and/or filter that can be pushed to the source. The aggregate expression is typically MAX, but MIN can also be used to page through data in reverse order, such as when working with temporal data. See Key-Based Examples below for examples that configure paging using the `s:key` property.
page	variable	Include this property if you want to configure page-based pagination where the set is divided into pages. The `s:page` property value is a variable that the GDI can use to track the current page across requests.

Option	Type	Description
		See Page-Based Example below for an example that configures paging using the `s:page` property.
cursor	variable	Include this property if you want to configure cursor-based pagination. The `s:cursor` property value is a variable that is bound against the source to capture the "cursor" value. The GDI uses this value as input to the source to deliver the next page of data. See Cursor-Based Example below for an example that configures paging with the `s:cursor` property.
offset	variable	Include this property along with the limit property if you want to configure offset-based pagination. The `s:offset` property value is a variable that the GDI can use to track the current offset across requests. See Offset-Based Example below for an example that configures paging using the `s:offset` property.
size	int	This property can be included with any of the paging models to configure the maximum size of each page. For example, `s:size 5000` limits the page size to 5,000 rows.
limit	variable	This property can be included to define the variable that the GDI should use to push the page size back to the source.

Paging Examples

- Key-Based Examples
- Page-Based Example
- Cursor-Based Example
- Offset-Based Example

Key-Based Examples

The example SERVICE clause below pages data based on the `?LastID` key, which is calculated by finding the maximum value of `SalesOrderID` and binding it to `?LastID`. A FILTER is used to filter for data where the `SalesOrderID` is greater than `?LastID`.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
  BIND(MAX(?SalesOrderID) AS ?LastID)
  FILTER(?SalesOrderID > ?LastID)

  ?SalesOrderHeaderEnlarged a s:DbSource ;
    s:url "jdbc:sqlserver://..." ;
    s:table "Sales.SalesOrderHeaderEnlarged" ;
    s:paging [
      s:key (?LastID) ;
      s:size 5000 ;
    ] ;
    ?SalesOrderID (xsd:int) ;
    ?RevisionNumber (xsd:int) ;
    ?OrderDate ("OrderDate" xsd:dateTime) ;
    ?DueDate (xsd:dateTime) .
}
```

The SERVICE clause below shows an example where key-based paging is configured to page through temporal data in reverse order. The `s:limit` property is configured on the `s:HttpSource` to limit the overall number of results returned across all pages. This query retrieves at most 1000 records (`s:limit 1000`), 100 rows (`s:size 100`) at a time.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
  BIND(MIN(?Timestamp) AS ?LastTimestamp)

  ?api a s:HttpSource ;
    s:url "http://slack.com/api/messages/latest" ;
    s:parameter [ s:name "before" ; s:value ?LastTimestamp ] ;
    s:parameter [ s:name "limit" ; s:value ?limit ] ;
    s:limit 1000 ;
    s:paging [
      s:key (?LastTimestamp) ;
      s:limit ?limit ;
```

```
      s:size 100 ;
    ] ;
    ?Message (xsd:string) ;
    ?Author (xsd:string) ;
    ?Timestamp (xsd:dateTime) .
}
```

Page-Based Example

The SERVICE clause below shows an example that uses the `s:page` property to configure page-based paging where the page size is 100 rows. This query retrieves at most 1000 records (`s:limit 1000`), 100 rows (`s:size 100`) at a time.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
  ?api a s:HttpSource ;
    s:url "http://slack.com/api/messages" ;
    s:parameter [ s:name "page" ; s:value ?page ] ;
    s:parameter [ s:name "size" ; s:value ?limit ] ;
    s:limit 1000 ;
    s:paging [
      s:page ?page ;
      s:limit ?limit ;
      s:size 100 ;
    ] ;
    ?Message (xsd:string) ;
    ?Author (xsd:string) ;
    ?Timestamp (xsd:dateTime) .
}
```

Cursor-Based Example

The SERVICE clause below shows an example that uses the `s:cursor` property to configure cursor-based paging.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
  ?api a s:HttpSource ;
    s:url "http://slack.com/api/messages" ;
    s:parameter [ s:name "cursor" ; s:value ?cursor ] ;
    s:parameter [ s:name "limit" ; s:value ?limit ] ;
```

```
    s:limit 1000 ;
    s:paging [
      s:cursor ?cursor ;
      s:limit ?limit ;
      s:size 100 ;
    ] ;
    ?Message (xsd:string) ;
    ?Author (xsd:string) ;
    ?Timestamp (xsd:dateTime) ;
    ?cursor ("next_cursor" xsd:string) .
}
```

Offset-Based Example

The SERVICE clause below shows an example that uses the `s:offset` property to configure offset-based paging.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
  ?api a s:HttpSource ;
    s:url "http://slack.com/api/messages" ;
    s:parameter [ s:name "offset" ; s:value ?offset ] ;
    s:parameter [ s:name "limit" ; s:value ?limit ] ;
    s:limit 1000 ;
    s:paging [
      s:offset ?offset ;
      s:limit ?limit ;
      s:size 100 ;
    ] ;
    ?Message (xsd:string) ;
    ?Author (xsd:string) ;
    ?Timestamp (xsd:dateTime) .
}
```

Binding and Hierarchy Concepts

As part of the Graph Data Interface's (GDI) flexibility, there are multiple ways to express binding hierarchies in queries. This topic describes the options for expressing hierarchies.

- Using Binding Trees and Selector Paths
- Unpacking JSON with Bindings and Arrays

- Returning Hierarchies as JSON Strings

Using Binding Trees and Selector Paths

One way to express hierarchies in queries is to use brackets ([ ]) to group objects into binding trees. For example, the WHERE clause snippet below organizes mapping variable objects into an `hourly/data` hierarchy by nesting the `?data` patterns inside the `?hourly [ ]` tree:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:HttpSource;
        s:url "https://sampleEndpoint.com/forecast/" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        ?hourly
[
          ?data
[
            ?time (xsd:long) ;
            ?summary (xsd:string) ;
            ?rainIntensity ("precipIntensity" xsd:double) ;
            ?rainProbability ("precipProbability" xsd:double) ;
            ?temperature (xsd:double) ;
            ?feelsLike ("apparentTemperature" xsd:double) ;
            ?humidity (xsd:double) ;
            ?pressure (xsd:double) ;
            ?windSpeed (xsd:double) ;
        ] ;
    ] .
  }
}
```

When constructing object binding trees, if you choose to introduce the hierarchy with a variable name that is not an exact match to the source label, include a **selector** property to list the value from the source. For example, in the WHERE clause snippet below, `s:selector` is included to select `eventHeader` in the source as `?event` in the query and `statLocation` as `?location`.

```
WHERE
{
```

```
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?data a s:FileSource ;
      s:url "/mnt/data/json/part_1.json" ;
      ?event
[
         s:selector "eventHeader" ;
           ?eventId (xsd:string) ;
           ?eventName (xsd:string) ;
           ?eventVersion (xsd:string) ;
           ?eventTime (xsd:dateTime) ;
      ] ;
      ?location
[
         s:selector "statLocation" ;
           ?locationId (xsd:string) ;
           ?lineNo (xsd:int) ;
           ?statNo (xsd:int) ;
           ?statId (xsd:int) ;
      ] .
  }
}
```

As an alternative to grouping objects in binding trees, the **selector** property also supports using dot notation to specify paths. For example, the WHERE clause snippet below rewrites the first example query to express the same `hourly/data` hierarchy as a path in the `s:selector` value:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:HttpSource;
         s:url "https://sampleEndpoint.com/forecast/" ;
         ?latitude (xsd:double) ;
         ?longitude (xsd:double) ;
         ?timezone (xsd:string) ;
         s:selector: "hourly.data" ;
         ?time (xsd:long) ;
         ?summary (xsd:string) ;
         ?rainIntensity ("precipIntensity" xsd:double) ;
         ?rainProbability ("precipProbability" xsd:double) ;
         ?temperature (xsd:double) ;
         ?feelsLike ("apparentTemperature" xsd:double) ;
```

```
          ?humidity (xsd:double) ;
          ?pressure (xsd:double) ;
          ?windSpeed (xsd:double) .
   }
}
```

You can also include the `$` character to anchor the selector at the root of the file. For example, `s:selector "data"` captures all `data` elements anywhere in the file. But `s:selector "$.data"` captures only the `data` elements that are at the root of the hierarchy.

Unpacking JSON with Bindings and Arrays

In addition to object binding trees and selectors, the GDI offers additional syntax for reading or ingesting JSON sources with nested objects and arrays. For example, following the JSON sample file below is a query that captures each value in the arrays:

```
{
   "payload" :
   {
      "IBP_IndEvent_MSR" :
      {
         "unit" : "ms",
         "value" : [ 0, 1 ]
      },
      "IBP_IndEvent_RMF" :
      {
         "unit" : "-",
         "value" : [ 0.012, 1.398, 3.1415 ]
      }
   }
}
```

To read the JSON file above, the following query uses an object binding (`?values [ ]`) to drill down to the `value` arrays in the source. An `@` selector is specified in the `?value` variable binding (`?value ("@" xsd:double)`) to retrieve each of the array values. For an array of primitive values, the @ selector captures each value in the array. If the source `value` was an array of objects, the @ selector would retrieve a JSON representation for each object in the array. In addition to creating a new binding context for the primitive array values, the `?values` object binding also includes `?index ("!array::index")` to capture the index array with the primitive value.

```
PREFIX s:   <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:FileSource ;
      s:url "/mnt/data/json/array-index.json" ;
      s:selector "payload.*" ;
      ?unit (xsd:string) ;
      ?values [
         s:selector "value" ;
         ?value ("@" xsd:double) ;
         ?index ("!array::index") ;
      ] .
  }
}
```

The results of the query are shown below:

```
unit | value  | index
-----+--------+-------
ms   |      0 |     0
ms   |      1 |     1
-    |  0.012 |     0
-    |  1.398 |     1
-    | 3.1415 |     2
```

If you do not want to retrieve all of the values in an array, you can include the specific index number to retrieve instead of using the @ symbol. In the variable binding, the index number is appended in brackets ([ ]) to the binding column name. For example, the following variable binding retrieves the second index value (the third value in the array) from a "projects" array: `?project ("projects[2]")`. The next example uses the following JSON file:

```
{
   "field1" : "value1" ,
   "arrayfield" : [
        "arrayvalue1",
        "arrayvalue2"
   ]
}
```

To retrieve only the second value in the array, the following query appends the index value `1` to the array column name, `arrayfield`:

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
SELECT *
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
       ?json a s:FileSource ;
       s:url "/mnt/data/json/array-index-2.json" ;
       ?field1 (xsd:string) ;
       ?arrayval ("arrayfield[1]" xsd:string) .
  }
}
```

The results of the query are shown below:

```
field1   | arrayval
---------+----------
value1   |arrayvalue2
```

Returning Hierarchies as JSON Strings

When working with schema-less sources, you can also capture a tree of data as a JSON string. For example, the query snippet below targets an HTTP endpoint. In this case, the properties under the **hourly** class of data are unknown. So the query binds all of the data below hourly to the **?hourly** variable by including empty parentheses. As a result, the GDI returns a JSON string representation of all of the properties and instance data under hourly:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:HttpSource;
        s:url "https://sampleEndpoint.com/forecast/" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        ?hourly () .
   }
}
```

For example, the results look like this:

```
 latitude  | longitude  | timezone        | hourly
-----------+------------+-----------------+---------------------------
30.374563  | -97.975892 | America/Chicago | {"summary":"\"Humid and partly cloudy
throughout the day.\"","icon":"\"partly-cloudy-day\"","data":[{"time":"1595559600",
summary":"\"Clear\"","icon":"\"clear-night\"","precipIntensity":"0",
"precipProbability":"0","temperature":"88.39","apparentTemperature":"91.72",
"dewPoint":"67.42","humidity":"0.5","pressure":"1011.7","windSpeed":"7.48",
"windGust":"16.71","windBearing":"109","cloudCover":"0.06","uvIndex":"0",
"visibility":"10","ozone":"285.2"},{"time":"1595563200","summary":"\"Clear\"",
"icon":"\"clear-night\"","precipIntensity":"2.0E-4","precipProbability":"0.01",
"precipType":"\"rain\"","temperature":"86.69","apparentTemperature":"90.1",
"dewPoint":"67.84","humidity":"0.54","pressure":"1012","windSpeed":"7.05",
"windGust":"17.56","windBearing":"110","cloudCover":"0.12","uvIndex":"0",
"visibility":"10","ozone":"284.9"},...
```

Similar to the example above, you can write a query that specifically captures some of the properties in a hierarchy and then returns the rest of the properties and their values as a JSON string representation. To do so, use **"@"** as the binding path. For example:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:HttpSource;
      s:url
"https://api.darksky.net/forecast/bdbe3f638eb908c9b94919537dad5945/30.374563,-
97.975892" ;
      ?latitude (xsd:double) ;
      ?longitude (xsd:double) ;
      ?timezone (xsd:string) ;
      ?hourly [
        s:selector "hourly.data" ;
        ?time (xsd:long) ;
        ?summary (xsd:string) ;
        ?hourly_data ("@") ;
      ] .
   }
}
```

Sample results are shown below:

```
 latitude  | longitude  | timezone        | time       | summary          | hourly_data
```

```
----------+-----------+----------------+-----------+-----------------+-----------
---
 30.374563 | -97.975892 | America/Chicago | 1595559600 | Clear            |
{"time":"1595559600","summary":"\"Clear\"",
"icon":"\"clear-
night\"","precipIntensity":"0","precipProbability":"0","temperature":"88.39",

"apparentTemperature":"91.72","dewPoint":"67.42","humidity":"0.5","pressure":"1011.7","
windSpeed":"7.48",

"windGust":"16.71","windBearing":"109","cloudCover":"0.06","uvIndex":"0","visibility":"
10","ozone":"285.2"}

 30.374563 | -97.975892 | America/Chicago | 1595563200 | Clear            |
{"time":"1595563200","summary":"\"Clear\"",
"icon":"\"clear-night\"","precipIntensity":"2.0E-
4","precipProbability":"0.01","precipType":"\"rain\"","temperature":"86.69",

"apparentTemperature":"90.1","dewPoint":"67.84","humidity":"0.54","pressure":"1012","wi
ndSpeed":"7.05","windGust":"17.56",

"windBearing":"110","cloudCover":"0.12","uvIndex":"0","visibility":"10","ozone":"284.
9"}

 30.374563 | -97.975892 | America/Chicago | 1595566800 | Partly Cloudy    |
{"time":"1595566800","summary":"\"Partly Cloudy\"",
"icon":"\"partly-cloudy-night\"","precipIntensity":"3.0E-4","precipProbability":"0.01",
"precipType":"\"rain\"","temperature":"85.63","apparentTemperature":"89.21",

"dewPoint":"68.33","humidity":"0.56","pressure":"1012.6","windSpeed":"6.48","windGust":
"17.92","windBearing":"110",
"cloudCover":"0.34","uvIndex":"0","visibility":"10","ozone":"284.5"}
...
```

Incremental Onboarding Concepts

When loading data from a database or file-based data source with a Graph Data Interface (GDI) query, you can add a few statements to the query to load a portion of the data incrementally rather than all of the data at once. As data is added or changed in the source, new data can be ingested without having to reload all of the previously ingested data. Because incremental ingestion is

configured as a filter in a SPARQL query, it is extremely flexible, allowing for various conditions to be defined for diverse data sources. When the data is ingested, the GDI evaluates the current state of the data and then loads only the data that meets the conditions defined in the query.

This topic provides example incremental queries to get you started and includes instructions for configuring a Data Layer as an incremental ingestion workflow.

- Incremental DbSource Example
- Incremental FileSource Example
- Setting Up a Data Layer to Ingest Data Incrementally

Incremental DbSource Example

The following query from a Direct Load Step ingests data from a database. All of the values for the requested columns in the ORDER_DETAILS table will be loaded.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
${usingSources}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:DbSource ;
         s:url "jdbc:oracle:thin:@10.10.10.10:1111/XE" ;
         s:username "northwind" ;
         s:password "NORTHWIND123" ;
         s:schema "NORTHWIND" ;
         s:table "ORDER_DETAILS" ;
         ?database ("!") ;
         ?schema ("!") ;
         ?table ("!") ;
         ?OrderID (xsd:int) ;
```

```
         ?ProductID (xsd:int) ;
         ?UnitPrice (xsd:double) ;
         ?Quantity (xsd:short) ;
         ?Discount xsd:double .


      BIND(IRI("http://cambridgesemantics.com/orders/{{?OrderID}}") AS ?resource)

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
      s:as (?s ?p ?o) ;
      s:ontology <http://cambridgesemantics.com/ontologies/northwind> ;
      s:base <http://cambridgesemantics.com/data> .
  }
}
```

The query below adds statements that configure the same Direct Load Step to ingest data incrementally. It captures the maximum order ID as the incremental value. When the source is updated with records that increase the order ID, only the records with larger order IDs than the previous maximum value will be ingested when the Graphmart is refreshed or reloaded. In the query:

- A ?MaxID variable is bound to the result of MAX(?OrderID): `BIND (MAX(?OrderID) AS ?MaxID).`
- The ?MaxID variable is defined as the incremental value: `?MaxID a s:IncrementalValue.`
- A filter clause is added to create a condition that ingests only the records where the order ID is greater than the previously ingested maximum ID: `FILTER (?OrderID > ?MaxID).`

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
${usingSources}
```

```
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?MaxID a s:IncrementalValue .
      FILTER (?OrderID > ?MaxID)
      BIND (MAX(?OrderID) AS ?MaxID)

      ?data a s:DbSource ;
         s:url "jdbc:oracle:thin:@10.10.10.10:1111/XE" ;
         s:username "northwind" ;
         s:password "NORTHWIND123" ;
         s:schema "NORTHWIND" ;
         s:table "ORDER_DETAILS" ;
         ?database ("!") ;
         ?schema ("!") ;
         ?table ("!") ;
         ?OrderID (xsd:int) ;
         ?ProductID (xsd:int) ;
         ?UnitPrice (xsd:double) ;
         ?Quantity (xsd:short) ;
         ?Discount xsd:double .


      BIND(IRI("http://cambridgesemantics.com/orders/{{?OrderID}}") AS ?resource)

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
      s:as (?s ?p ?o) ;
      s:ontology <http://cambridgesemantics.com/ontologies/northwind> ;
      s:base <http://cambridgesemantics.com/data> .
  }
}
```

Incremental FileSource Example

The following query from a Direct Load Step ingests data from all of the CSV files in the `/nfs/data/fmcsa` directory:

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>

DELETE {
```

```
   GRAPH ${targetGraph} {
  }
}
INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?data a s:FileSource ;
         s:model "fmcsa" ;
         s:url "/nfs/data/fmcsa" ;
         s:pattern "*.csv" .

      ?rdf a s:RdfGenerator , s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/ontologies/fmcsa> ;
         s:base <http://cambridgesemantics.com/data/> .
  }
}
```

The query below adds statements that configure the same Direct Load Step to ingest data incrementally. It uses a "last modified" strategy to determine what files are new or modified and should be ingested the next time the Graphmart is refreshed or reloaded. In the query:

- The modified timestamp metadata on the files is captured with `?Modified ("!")`.
- The ?LastRun variable is bound to the result of the NOW() function: `BIND (NOW() AS ?LastRun)`.
- A filter clause is added to check whether the modified timestamp is later than the timestamp from the last time the query was run: `FILTER (?Modified > ?LastRun)`.
- ?LastRun is defined as the incremental value: `?LastRun a s:IncrementalValue`.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
```

```
DELETE {
   GRAPH ${targetGraph} {
  }
}
INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?LastRun a s:IncrementalValue .
      FILTER (?Modified > ?LastRun)
      BIND (NOW() AS ?LastRun)

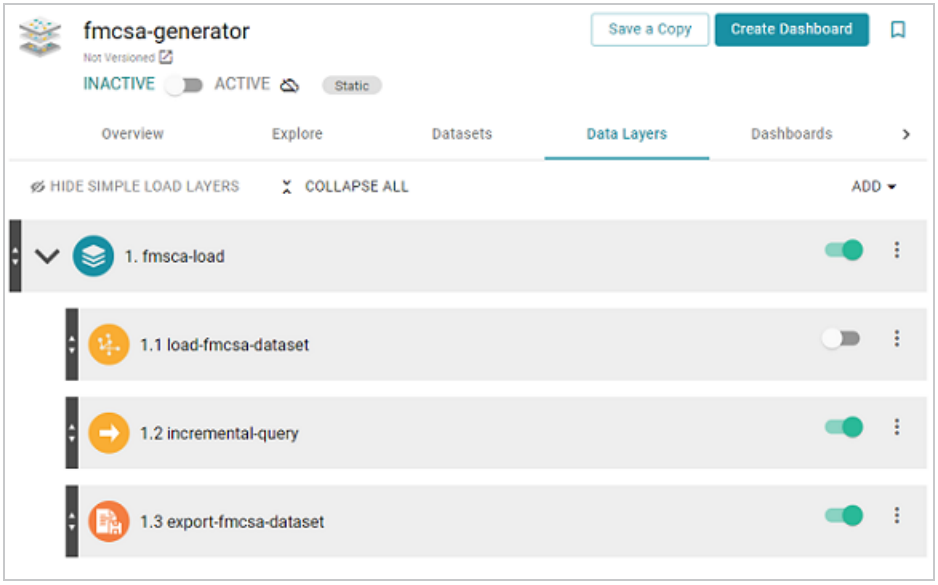
      ?data a s:FileSource ;
         s:model "fmcsa" ;
         s:url "/nfs/data/fmcsa" ;
         s:pattern "*.csv" ;
         ?Modified ("!") .

      ?rdf a s:RdfGenerator , s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/ontologies/fmcsa> ;
         s:base <http://cambridgesemantics.com/data/> .
  }
}
```

Setting Up a Data Layer to Ingest Data Incrementally

1. Create a new empty dataset in the Anzo Data Store. For instructions, see Adding an Empty Dataset for an Export Step.
2. In the graphmart where you want to add a GDI query that ingests data incrementally, add a new layer.
3. In the new layer, add a Load Dataset Step as the first step. The Linked Dataset for this step should be the empty dataset that you created in the first step.
4. Now, add a Direct Load Step as the next step in the layer. Edit the query template in the step to compose the incremental query.

5. As the last step in the layer, add an Export Step. The Target FLDS for the step should also be the empty dataset that you created in the first step. For example, the image below shows a graphmart with a layer that is set up to ingest data incrementally.



6. Activate the graphmart to ingest the data and export it to an FLDS. Once the graphmart is activated, enable the Load Dataset Step.

Options for Data Types, Data Linking, and Models

The topics in this section describe the options that are available across data source types for controlling the way that strings are coerced to other data types, the relationships that define connections across multiple sources, and the way label and URI values are generated in the data model.

- Data Type Formatting Options
- Data Linking Options
- Model Normalization Options

Data Type Formatting Options

To give you control over the data types that are used when coercing strings to other types, the **formats** property can be included in GDI queries to define the desired types. In addition, formats can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. You can also use the formats property to suppress the conversion so that the generated values are typed the same way as the source.

> **Tip**
> The GDI takes locale into account when formatting the generated date and time values.

For sources that do not include data type specifications and natively treat values as strings, the GDI Generator automatically converts the values to the appropriate type. For example, if a CSV file includes the value "Feb-18-2022," the GDI parses the string to an `xsd:date` with the format "2022-02-18". A column with numbers is converted to an `xsd:int` type and a column with a decimal value is converted to `xsd:float`. The formats property usage is described below.

- Formats Syntax
- Formats Examples

Formats Syntax

```
s:formats [
   s:strict boolean ; [
       xsd:data_type "format"
     | xsd:data_type boolean ;
       [ ... ; ]
   ]
] ;
```

Option	Type	Description
strict	boolean	This property enables or disables the automatic type conversion feature. By default, strict is set to false (`s:strict false`), meaning the GDI's automatic type conversion feature is enabled. When strict is false, any formats specified in `s:formats [ ]` augment the GDI's built-in date and time formats. You can selectively disable certain type conversions, however, by including `xsd:data_type false`. For example, `xsd:dateTime false` disables the parsing of strings to dateTime. When strict is true (`s:strict true`), the auto conversion logic is essentially disabled and the generated data will be represented the same way it is in the source. When strict is true, you can selectively enable certain conversions by including `xsd:data_type true` or by defining `xsd:data_type "format"`. In this case, values that do not match any of the formats provided will be typed as `xsd:string`.
xsd:data_type "format"	N/A	Include `xsd:data_type "format"` when you want to describe the formats of date and time values in the source. The GDI supports Java date and time format notation. For example, if dates in the source are formatted like "yyyy-MM-dd," include the statement `xsd:date "yyyy-MM-dd"`. If the source uses multiple formats for dates, e.g., 18-MAR-1978 and 03/18/1978, you can list multiple formats for xsd:date, such as `xsd:date "dd-MMM-yyyy",`

Option	Type	Description
		"MM/dd/yyyy". **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.
xsd:data_type boolean	N/A	When strict is `false` or not set, you can disable specific type conversions by listing data types and setting their values to `false`.For example, if you want the GDI to convert strings to integers or floats when possible but you want the dates in the source to be preserved as strings, you can include `xsd:date false` to disable the conversion of strings to dates. When strict is `true`, you can enable specific type conversions by listing data types and setting their values to `true`. For example, if you want the GDI to preserve the strings in the source except for when the string is a number, you can include `xsd:int true` to enable the conversion of strings to integers.

Formats Examples

The example below sets strict to `true` and forces the GDI to parse values only to the data types that are enabled with `true`. It also defines the format to look for when converting strings to dateTime:

```
s:formats [
   s:strict true ;
   xsd:int true ;
   xsd:dateTime true ;
   xsd:dateTime "yyyy-MM-dd-HH-mm-ss" ;
] ;
```

The example below does not set strict, so the default value of `false` is used. The data type definitions specify the formats of the values to parse as date, time, and dateTime values. The example also disables the conversion from string to long:

```
s:formats [
   xsd:date "MM/dd/yyyy", "MMM dd", "MMM dd yyyy" ;
   xsd:time "HH[:mm][:ss][ ]a" ;
   xsd:dateTime "M/d/yyyy HH:mm:ss a", "yyyy-MM-dd-HH-mm-ss" ;
   xsd:long false ;
] ;
```

Data Linking Options

When a data source does not define keys (such as a CSV or JSON source), the GDI provides properties that enable you to create a connected knowledge graph by defining relationships, resource templates (primary keys) and object properties (foreign keys), when you are loading data from multiple sources. The properties that are available are described below.

- Data Linking Syntax
- Data Linking Examples

Data Linking Syntax

```
s:key ("column_name") ;
s:reference [
    s:model "table_to_reference" ;
    s:using ("foreign_key_column")
]
```

Option	Type	Description
key	string	Include this property when you want to define the primary key column for the source file or table. This column is leveraged in a resource

Option	Type	Description
		template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. If none of the columns contain unique values, you can specify a combination of columns that would create a unique value. For example, `s:key ("FlightNumber", "TailNumber")`.

reference	RDF list	Include this property when you want to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column in the source table.
		`s:reference [`
		`s:model "table_to_reference" ;`
		`s:using ("foreign_key_column")`
		`]`
		Tip
		You can also include an optional `key` property within the `s:reference` list that defines the key column in the target table and can be used as a way to expose additional metadata that helps inform the GDI how to name the object property. For example:
		`s:reference [ s:model "Employees" ; s:using ("EMPLOYEE_ID") ; s:key ("EMPLOYEE_ID") ]`

Data Linking Examples

For example, the query snippet below defines two data sources. The `s:model` property defines the table/class for each source, and the `s:key` defines the primary key for each table/class. The `s:reference` property for the "venue" table defines a foreign key relationship from `venue.EVENT_ID` to `event.EVENT_ID`.

```
?event a s:FileSource ;
   s:model "event" ;
   s:url  "/opt/shared-files/csv/events.csv" ;
   s:key ("EVENT_ID") .

?venue a s:FileSource ;
   s:model "venue" ;
   s:url " /opt/shared-files/csv/venues.csv" ;
   s:key ("VENUE_ID") ;
   s:reference [ s:model "event" ; s:using ("EVENT_ID") ] .
```

The following query for multiple file sources generates RDF and an ontology with resource templates and object properties. The query also includes global normalization rules for normalizing the data across all sources (see Model Normalization Options for information about normalization).

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {

      ?event a s:FileSource ;
         s:model "event" ;
         s:url  "/opt/shared-files/csv/events.csv" ;
         s:key ("EVENT_ID") .

      ?listing a s:FileSource ;
         s:model "listing" ;
         s:url " /opt/shared-files/csv/listings.csv" ;
         s:key ("LIST_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?date a s:FileSource ;
         s:model "date" ;
         s:url  "/opt/shared-files/csv/event_dates.csv" ;
         s:key ("DATE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?venue a s:FileSource ;
```

```
         s:model "venue" ;
         s:url " /opt/shared-files/csv/venues.csv" ;
         s:key ("VENUE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] .

      ?sale a s:FileSource ;
         s:model "sale" ;
         s:url " /opt/shared-files/csv/sales.csv" ;
         s:key ("SALE_ID") ;
         s:reference [ s:model "event" ; s:using ("EVENT_ID") ; s:key ("EVENT_ID") ] ;
         s:reference [ s:model "listing" ; s:using ("LIST_ID") ; s:key ("LIST_ID") ] .

      ?rdf a s:RdfGenerator, s:OntologyGenerator ;
         s:as (?s ?p ?o) ;
         s:ontology <http://cambridgesemantics.com/tickets> ;
         s:base <http://cambridgesemantics.com/data> ;
         s:normalize [
            s:all [
               s:casing s:UPPER ;
               s:localNameSeparator "_" ;
            ] ;
         ] .
   }
}
```

Model Normalization Options

To give users control over the labels and URIs that are generated in the data model, the GDI offers several options for normalizing the class and property fields that are created from the specified data source(s). Normalization rules can be specified at the source level to normalize the data from each source independently, or they can be used at the RDF Generator level to apply global rules across all specified data sources.

> **Note**
> Normalization rules are applied only at the model level. The rules to do affect the instance data values that are ingested.

Including the **normalize** parameter is optional. If you include it, you can specify any combination of rules. See Default Normalization Behavior below for details about the Generator's default behavior when normalization rules are not specified in your query.

- Default Normalization Behavior
- Normalize Syntax
- Normalize Examples

Default Normalization Behavior

The GDI Generator normalizes data according to the following rules by default. If you do not include the `s:normalize` parameter in your query, these are the rules that are applied:

```
s:normalize [
  s:all [
      s:removePrefix true ;
      s:removePartialPrefix false ;
      s:allowWhiteSpace false ;
      s:allowPunctuation false ;
      s:allowSymbols false ;
      s:separator " " ;
      s:singularize false ;
      s:casing s:UpperCamel ;
      s:localNameSeparator "." ;
  ]
]
```

Normalize Syntax

```
s:normalize boolean | [
  s:model | s:field | s:all
  [
    s:removeStart "string" ;
    s:removeEnd "string" ;
    s:removePrefix boolean ;
    s:removePartialPrefix boolean ;
    s:match [ s:pattern "java_regex" ; s:replace "java_regex" ] ;
    s:disambiguationLevel int ;
    s:ignore "string" ;
    s:words "string" ;
    s:preserve "string" ;
```

```
    s:split "string" ;
    s:allowWhiteSpace boolean ;
    s:allowPunctuation boolean ;
    s:allowSymbols boolean ;
    s:singularize boolean ;
    s:casing property ;
    s:separator "string" ;
    s:localNamePrefix "string" ;
    s:localNameSeparator "string" ;
  ] ;
] ;
```

Property	Type	Description
boolean	N/A	Normalize is enabled by default for all GDI Generator queries. If you want to disable normalization, you can include `s:normalize false`. If normalization is disabled, the names in the source will be used verbatim both for labeling and in generating the local names for property and class URIs. However, when normalization is disabled, the labels in the data source are used verbatim. In addition, the Generator creates hard-to-read, URL-encoded local names for property and class URIs.
s:model | s:field | s:all	N/A	This property defines whether the specified normalization rules should be applied across the model or only to the classes or properties. The list below describes each option: • **s:model**: Applies the rules to the file/table/class names only. • **s:fields**: Applies the rules to the column/property/field names only. • **s:all**: (Default) Applies the rules to both the class and property names. This is the default

Property	Type	Description
		value if not specified.
removeStart	string	If you want to remove text from the beginning of identifiers, include the **removeStart** rule to specify the string to remove. For example, `s:removeStart "temp_"`.
removeEnd	string	If you want to remove text from the end of identifiers, include the **removeEnd** rule to specify the string to remove. For example, `s:removeEnd "NEW"`.
removePrefix	boolean	If there are property identifiers that share a prefix with the class, the RDF Generator automatically removes the shared prefix from the property name; the **removePrefix** rule is set to `true` by default. For example, if there is an EMPLOYEE class with an EMPLOYEE_ID column, the shared prefix "EMPLOYEE" is removed from the generated property so that it becomes "ID." If you do not want the Generator to remove prefixes, you can include `s:removePrefix false`.
removePartialPrefix	boolean	If there are property identifiers that share a partial prefix with the class, you can enable **removePartialPrefix** to remove the partial prefix from the property name. The **removePartialPrefix** rule is set to `false` by default. If you want the Generator to remove partial prefixes, you can include `s:removePrefix true`.
match	RDF list	This rule provides a way to use regular expressions (REGEX) to match a **pattern** against source identifiers and **replace** the matched text in the normalized name.

Property	Type	Description

		The `s:pattern` property defines the Java REGEX pattern to match against, and `s:replace` defines the Java REGEX replacement pattern. As shown in the example below, the match rule can also be configured with an `rdf:List` of objects to perform match evaluation in a certain order:
		`s:match (`
		`[ s:pattern "(.+)GUID$" ; s:replace "$1" ; ]`
		`[ s:pattern "(.+)ID$" ; s:replace "$1" ; ]`
		`)`

disambiguationLevel	int	This rule specifies the number of levels to use to resolve ambiguities between similarly named elements in a hierarchical source. For example, an element named "Data" appears in two contexts: "Currently" and "Hourly." By default, the Generator retains all levels, meaning two classes are generated: "Currently Data" and "Hourly Data." If `s:disambiguationLevel` is set to `0`, a single class named "Data" is generated and both the Currently and Hourly classes have a "Data" property. The disambiguationLevel value is also used to determine the number of hierarchy levels to use when encoding the local name of the generated URI.
ignore	string	This rule can be used to list identifiers that should be ignored. Properties and classes will not be generated for identifiers that match the specified string(s). The ignore rule is a multi-valued property. For simplicity,

Property	Type	Description
		you can enter a list by separating words with a space, rather than quoting each term and separating them with a comma. For multi-word identifiers, use single quotes. For example, `s:ignore "sample example 'test column' old"`.
words	string	Since many sources do not encode word boundaries very well, the **words** rule can be used to list the set of words that should be separate identifiers. This rule tells the Generator which words may be encountered. The words rule is a multi-valued property. For simplicity, you can enter a list by separating words with a space, rather than quoting each term and separating them with a comma. For multi-word identifiers, use single quotes. For example: `s:words "activity 'patient complaint' medication observation patient signal specialty study" ;`
preserve	string	This rule can be used to identify any words whose casing should be preserved in the input identifiers. For example, if casing is set to `lower` but you want preserve the original upper casing of certain words, you can specify the words to preserve. The preserve rule is a multi-valued property. For simplicity, you can enter a list by separating words with a space, rather than quoting each term and separating them with a comma. For multi-word identifiers, use single quotes. For example: `s:preserve "ABC 'Laundry List' TriG"`. The preserve rule is case-insensitive. You do not have to match the casing of the words to preserve.

Property	Type	Description
split	string	This rule specifies the string that should be used to split source identifiers into individual terms. If neither `split` nor `words` is specified, input identifiers are split on casing changes and character class changes.
allowWhiteSpace	boolean	This rule specifies whether or not white space should be preserved in identifiers after they have been split into individual terms. This rule is set to `false` by default, meaning white space is not preserved. You can specify `s:allowWhiteSpace true` to preserve spaces.
allowPunctuation	boolean	This rule specifies whether or not punctuation should be preserved in identifiers after they have been split into individual terms. This rule is set to `false` by default, meaning punctuation is not preserved. You can specify `s:allowPunctuation true` to preserve punctuation.
allowSymbols	boolean	This rule specifies whether or not symbols should be preserved in identifiers after they have been split into individual terms. This rule is set to `false` by default, meaning symbols are not preserved. You can specify `s:allowSymbols true` to preserve symbols.
singularize	boolean	This rule specifies whether or not to change any plural identifiers to singular. This rule is set to `false` by default, meaning plural identifiers are preserved. You can specify `s:singularize true` to change plural terms to the singular version of the term.
casing	object	This rule specifies how the generated labels should be

Property	Type	Description
		cased. By default, the Generator outputs labels in upper camel case (`s:casing s:UpperCamel`). To use a different casing, specify any of the following properties: • **default**: This object preserves the casing from the source. Labels will not be converted. • **UPPER**: This object converts all characters to uppercase. For example, "uppercase" becomes "UPPERCASE." • **lower**: This object converts all characters to lowercase. For example, "Lower Case" becomes "lower case". • **UpperCamel**: This is the default casing value and converts labels to upper camel case, where terms are concatenated and the first letter of each word is capitalized. For example, "upper camel case" becomes "UpperCamelCase." • **lowerCamel**: This object converts labels to lower camel case, where terms are concatenated and the first letter of the first word is lower case. The first letter of subsequent terms is capitalized. For example, "lower camel case" becomes "lowerCamelCase."
separator	string	This rule specifies the character or characters to use to separate terms in the generated label. The default separator is a space (`s:separator " "`).

Property	Type	Description
localNamePrefix	string	This rule specifies a string to use as the prefix for local names when generating a URI.
localNameSeparator	string	This rule specifies the string to use for separating local names when encoding hierarchies according to the specified disambiguationLevel. By default, localNameSeparator is a period (`s:localNameSeparator "."`). If localNameSeparator is empty, hierarchical context will not be encoded into the local name of any properties or child classes. The result would be an ontology where only the class or property name is used to determine the local name. For example, a property URI would look like `ont:employeeID` rather than `ont:Employee.employeeID`. The result could lead to "conflicts" in the generated ontology, but those "conflicts" may be desired as properties with same name are reused across the generated ontology.

Tip

You can specify normalization rules at both the source and global level in the same query. If you include multi-valued rules (such as `ignore`, `words`, or `preserve`) at both levels, the Generator combines all values from both instances of the rule. If you specify single value rules at both levels and the values are conflicting, the Generator applies the value at the source level.

Normalize Examples

The example below uses the normalize property to normalize data at both the model and field level.

```
s:normalize [
    s:model [
       s:localNamePrefix "C_" ;
```

```
      s:localNameSeparator "_" ;
      s:match [ s:pattern "(.+)Enlarged" ; s:replace "$1" ] ;
   ] ;
   s:field [
      s:localNamePrefix "P_" ;
      s:localNameSeparator "_" ;
      s:ignore "rowguid ModifiedDate" ;
      s:match (
        [ s:pattern "(.+)GUID$" ; s:replace "$1" ]
        [ s:pattern "(.+)ID$" ; s:replace "$1" ]
      ) ;
   ] ;
] ;
```

Advanced Usage by Data Source Type

The topics in this section provide more advanced GDI usage information by including descriptions for all of the query options for each type of supported data source.

> **Note**
> Although connections to data sources can be made directly in GDI queries. Creating the connection in Anzo prior to writing queries enables your organization to get an overview of the sources that are in use or available to use. Connecting to a source also gives a sample view of the data to be onboarded (except for JSON and XML sources). And, for sources that require input of sensitive connection and authorization information, connecting the sources to Anzo allows you to later hide that sensitive information (through the use of a Query Context) from any queries that are generated for that source. See Adding Data Sources for instructions on connecting to sources.

> **Tip**
> Rather than manually writing complex queries, you can use the GDI to automatically generate graphs and ontologies by including a few key statements in a relatively simple query. For information, see Onboarding Data with a Direct Load Step.

- Querying a Database Source
- Querying an HTTP Source
- Querying an Elasticsearch Source
- Querying a File Source

Querying a Database Source

This topic provides details about the structure to use when writing GDI queries to read or ingest data from database data sources. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Supported Databases
- Query Syntax
- Query Examples

Supported Databases

The GDI supports querying any database through a JDBC connection. AnzoGraph installations include JDBC drivers for the following databases:

- Databricks
- H2
- IBM DB2
- Microsoft SQL Server
- MariaDB
- Oracle
- PostgreSQL
- SAP Sybase (jTDS)
- Snowflake

To extend the service to access other databases, additional JDBC drivers can be added to AnzoGraph. For information about acquiring additional JDBC drivers, contact your Cambridge Semantics Customer Success manager. For instructions on deploying drivers, see Deploy Optional Drivers for Accessing Custom Database Sources in the Deployment Guide.

Query Syntax

The following query syntax shows the structure of a GDI query for database sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:DbSource ;
        s:url "string" ;
        s:username "string" ;
        s:password "string" ;
        [ s:token "string" ; ]
        [ s:driver "string" ; ]
        [ s:property [ s:name "string" ; s:value "string" ] ; ]
        [ s:timeout int ; ]
        [ s:maxConnections int ; ]
        [ s:batching boolean | int ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:partitionBy "string" | ?variable | s:auto ; ]
        [ s:paging [ pagination_options ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
```

```
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:query "string" ; ]
        [ s:database "string" ; ]
        [ s:schema "string" ; ]
        [ s:table "string" ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:orderBy "string" | ?variable ; ]
        [ s:limit int ; ]
        # Mapping variables
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH	N/A	Include the `GRAPH` keyword and target graph parameter

Option	Type	Description
${targetGraph}		`${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE`

Option	Type	Description
		`<http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.

url	string	This property specifies the URL to use to access the database.
		Important
		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example, the triple patterns below reference a Query Context and add a JDBC driver level connection property:
		`?data a s:DbSource ;`
		`s:url "{{@db.eca4bf...ff9a.url}}" ;`
		`s:username "{{@db.eca4bf...ff9a.user}}" ;`
		`s:password "{{@db.eca4bf...ff9a.password}}" ;`
		`s:property [ s:name "access" ; s:value "all" ] ;`

username	string	This property lists the user name to use for the connection to the database.

Option	Type	Description

		Tip
		If you want to group the username and password properties, you can wrap them with `s:credentials [ ]`. For example:
		`s:credentials [`
		`  s:username "username" ;`
		`  s:password "password" ;`
		`] ;`

password	string	This property lists the password for the given username.
token	string	For connections that require a bearer token, this property can be included to specify the token.
driver	string	This property can be included to specify the JDBC driver to use.

property	RDF list	This property can be included to list any JDBC driver-specific connection properties. To incorporate `property`, use the following syntax:
		`s:property [`
		`  s:name "custom_driver_property_name" ;`
		`  s:value "custom_value"`
		`]`

timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
maxConnection	int	This property can be used to set a limit on the maximum number

Option	Type	Description
s		of active connections to the source. For example, `s:maxConnections 16` sets the limit to 16 connections. The default value is `10`.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of

Option	Type	Description

		requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

partitionBy	string, variable, object	The GDI attempts to partition queries automatically across the available cores (slices) in AnzoGraph. To determine how to partition the query, the GDI uses metadata from the source database. It looks for any column in an index, preferring the primary key column if it is interpolable. However, it only considers the first column in any index on the table. After determining the partition column, the GDI does a MIN/MAX on the column as well as a basic sizing query. To specify which column or columns the GDI should partition on, you can include the `partitionBy` property in the query. The property supports a list of source field names, bound variables, or the object `s:auto`, which forces the

Option	Type	Description
		GDI to partition the data when the source does not define partitioning metadata.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. As an alternative to including the selector property for identifying the target data, you could use the database, schema, and/or table properties.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource

Option	Type	Description
		template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
query	string	If you want to access the source data by running an SQL query, you can include this property to specify the query string to run. The language does not have to be SQL if the source supports another language. However, some GDI features where the query is dynamically altered may not work with a non-SQL language. Including `{{?variable}}` substitutions is supported within `s:query` strings.

Option	Type	Description
		Important If you include `s:query`, you must also specify table and partitionBy. Specify the table name in `s:table` and the column to partition the table on in `s:partitionBy`. If the table and partition column are not specified, the GDI will not partition the query and query execution may fail or perform very poorly.
database	string	This property can be used to specify the database to target in the source if the database is not listed in the `s:url` or `s:selector` strings.
schema	string	This property can be included to specify the target schema to query. If you include `s:schema "schema_name"` without specifying `s:table` (described below) or `s:query`, all tables in the schema are queried.
table	string	This property can be included to specify the target table or tables for the query.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.
orderBy	string, variable, list	You can include this property to order the result set by a field name, a bound variable, or a list of names or bound variables.

Option	Type	Description
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_variable	variable	The mapping variables, in `?variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Note** The parentheses around the binding, data type, and format specifications are not required but are included in this document for readability.
binding	string	The **binding** is a literal value that binds a ?variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column that is not already identified by the selector, database, schema, table, or query properties, then the binding is needed to map the new variable to that source column. For example, `?subject ("dbo.FILM.SUBJECT")` binds the ?subject variable by navigating to the SUBJECT column in the FILM table in the dbo schema. **Note** Database, schema, and table names in bindings are

Option	Type	Description
		parsed according to the specific rules for that database type. You do not need to escape characters in database names. However, database names with characters that do not match `(_|A-Z|a-z)(_|A-Z|a-z|0-9)*` should be quoted, such as `("'Adventure.Works'.Sales.'Daily.Totals'").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation

Option	Type	Description
		`^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Query Examples

The example below selects data from the AdventureWorks2012 database. The `s:selector` property is used to specify the table (`salesOrderHeader` in the `Sales` schema) to target.

```
PREFIX s:   <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT (COUNT(*) as ?count)
WHERE
{
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
{
      ?SalesOrderHeader a s:DbSource ;
         s:url "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.url}}" ;
         s:username "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.user}}" ;
         s:password "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.password}}" ;
         s:selector "Sales.SalesOrderHeader" ;
         ?SalesOrderID (xsd:int) ;
         ?RevisionNumber (xsd:int) ;
         ?OrderDate (xsd:dateTime) ;
         ?DueDate (xsd:dateTime) ;
         ?TerritoryID (xsd:int) ;
         ?TotalDue (xsd:decimal) .
      FILTER(?TerritoryID IN (1, 2, 3))
      FILTER(?TotalDue < 11.0 || ?TotalDue > 250)
  }
}
```

The example below ingests data from a database. To define the data to target, the query includes the `s:query` property to run an SQL query. The `s:table` and `partitionBy` properties are also included to aid the GDI in partitioning the query.

```
PREFIX s:   <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont:
<http://cambridgesemantics.com/Layer/2f1e926b130a402db6fc10fa54199d49/Model#>

INSERT {
   GRAPH ${targetGraph} {
      ?resource a ont:EmrPatient ;
      ont:EmrPatient.patientid ?PATIENTID ;
      ont:EmrPatient.gender ?GENDER ;
      ont:EmrPatient.language ?LANGUAGE ;
      ont:EmrPatient.patientfirstdocactivitydate ?PATIENTFIRSTDOCACTIVITYDATE .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:DbSource ;
      s:url "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.url}}" ;
      s:username "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.user}}" ;
      s:password "{{@db.eca4bfa83481f3638b93ab5fdf93ff9a.password}}" ;
      s:query "select * from emrdbsmall.emr_patient where emr_patient.PATIENTID < 500"
;
      s:partitionBy "PATIENTID" ;
      s:table "emrdbsmall.emr_patient" ;
      ?PATIENTID (xsd:int) ;
      ?GENDER (xsd:string) ;
      ?LANGUAGE (xsd:string) ;
      ?PATIENTFIRSTDOCACTIVITYDATE (xsd:dateTime "M/d/yyyy HH:mm:ss") .

   BIND(IRI("http://cambridgesemantics.com/Layer/2f1e926b130a402db6fc10fa54199d49/
{{?PATIENTID}}") AS ?resource)
  }
}
```

Querying an HTTP Source

This topic provides details about the structure to use when writing GDI queries to read or ingest data from HTTP data sources. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Query Syntax
- Mapping the Content Property to JSON
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for HTTP sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})
```

```
{
  ?data a s:HttpSource ;
    s:url "string" ;
    [ s:authorization [
        a s:BearerToken ; s:token "string" ;
      | a s:AWSSignature ; s:accessKey "string" ; s:region "string" ;
              s:secretKey "string" ; s:serviceName "string" ;
              s:sessionToken "string" ;
      | a s:BasicAuth ; s:username "string" ; s:password "string" ;
    ] ; ]
    [ s:trust "string" ; ]
    [ s:proxy "string" | [ s:host "string" ; s:port int ] ]
    [ s:header [ s:name: "string" ; s:value "string" ] ; ]
    [ s:mimetype "string" ; ]
    [ s:contentType "string" ; ]
    [ s:content """string""" ; ]
    [ s:parameter [ s:name "string" ; s:value "string" ] ; ]
    [ s:method "string" ; ]
    [ s:encoding "string" ; ]
    [ s:form [ s:name: "string" ; s:value "string" ] ; ]
    [ s:format [ source_format_options ; ] ; ]
    [ s:timeout int ; ]
    [ s:batching boolean | int ; ]
    [ s:paging [ pagination_options ; ]
    [ s:concurrency int | [ list_of_properties ] ; ]
    [ s:rate int | "string" ; ]
    [ s:partitionBy "string" | ?variable ; ]
    [ s:locale "string" ; ]
    [ s:sampling int ; ]
    [ s:selector "string" | [ list ] ; ]
    [ s:model "string" ; ]
    [ s:key ("string") ; ]
    [ s:reference [ s:model "string" ; s:using ("string") ]
    [ s:formats [ datatype_formatting_options ] ; ]
    [ s:normalize boolean | [ normalization_rules ] ; ]
    [ s:count ?variable ; ]
    [ s:offset int ; ]
    [ s:orderBy "string" | ?variable ; ]
    [ s:limit int ; ]
    # Mapping variables
    ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
    ... ;
    .
```

```
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing

Option	Type	Description
		input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
url	string	This property specifies the URL to use to access the source. Query binding variables can be inserted into the url string by surrounding the variable name with double curly braces. For

Option	Type	Description

		example, `"{{?name}}"`.
		
		Important
		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example:
		`?data a s:HttpSource ;`
		`s:url "{{@db.eca...f93ff9a.url}}" ;`
		`s:authorization [ a s:BasicAuth ;`
		`s:username "{{@db.eca...f93ff9a.user}}" ;`
		`s:password "{{@db.eca...f93ff9a.password}}" ;`
		`]`

authorization	RDF list	This property specifies the type of authorization to use and the values for authentication. The options are BearerToken, AWSSignature, or BasicAuth. `s:authorization [ a s:BearerToken | s:AWSSignature | s:BasicAuth ]`

BearerToken	string	Specify this property when a bearer token is used for authentication, and include the token property.
		`s:authorization [ a s:BearerToken ;`
		`s:token "string"`

Option	Type	Description
		]

AWSSignature	RDF list	For authorization to AWS service endpoints, specify this property and include the appropriate authentication properties from the list below:
		• **accessKey**: Include this property to specify the AWS access key.
		• **region**: Include this property to specify the AWS region.
		• **secretKey**: Include this property to specify the AWS secret key.
		• **serviceName**: Include this property to specify the AWS service name.
		• **sessionToken**: Include this property to specify the AWS session token.
		`s:authorization [`
		`a s:AWSSignature ; s:accessKey "string" ;`
		`s:region "string" ; s:secretKey "string" ;`
		`s:serviceName "string" ; s:sessionToken "string" ;`
		`]`

BasicAuth	RDF list	Specify this property when basic authentication is used, and include the username and password properties.
		`s:authorization [ a s:BasicAuth ;`
		`s:username "string" ;`
		`s:password "string" ;`
		`]`

Option	Type	Description
trust	string	Include this property to set the level of trust for the source's SSL certificate. The value can be either `"system"` or `"all"`.
proxy	string or RDF list	Include this property to specify proxy information if a proxy is used. The value can be a string, such as `s:proxy "host_url:port_number"`, or an RDF list that includes `host` and `port` properties, such as `s:proxy [ s:host "host_url" ; s:port port_number ]`.
header	RDF list	You can use this property to specify name-value pairs to include as headers in the request. For example: `s:header [ s:name "Accept" ; s:value "application/json" ]` If you are creating a view, you can include variables in the `s:header` list. When another query is run against a view with variables, that query can map the variables through the view by including predicates in the CONSTRUCT clause.
mimetype	string	You can include this property to specify the MIME type of the source. For example, `s:mimetype "text/html"`.
contentType	string	Include this property to specify the content type of the body of the request. For example, `s:contentType "application/sparql-query"` or `s:contentType "application/json"`.
content	string or RDF list	This property can be included to send content to the source in the body of the request. For example, `content` can be a SPARQL query, JSON arrays, or a list of key-value pairs. Content can also be configured with an inline object (blank node) that gets translated to JSON. For more information, see Mapping the

Option	Type	Description
		Content Property to JSON below.

parameter	RDF list	You can include this property to list any URL parameters as name-value pairs. For example, the `s:parameter` property below adds `format` to return results in CSV format and the `named-graph-uri` parameter to target a specific layer in a graphmart.
		`s:parameter [ s:name "format" ; s:value "csv" ]`
		`,`
		`[ s:name "named-graph-uri" ;`
		`s:value`
		`"http://cambridgesemantics.com/Layer/d541..." ]`
		If you are creating a view, you can include variables in the `s:parameter` list. When another query is run against a view with variables, that query can map the variables through the view by including predicates in the CONSTRUCT clause.

method	string	You can include this property to specify the HTTP method. For example, `s:method "GET"` or `s:method "POST"`.
encoding	string	When targeting a file, you can include this property to specify the character encoding used by the file. The default value is `s:encoding "utf8"`.
form	RDF list	To send data to the HTTP endpoint, you can use this property to post the data. Form is a list of name-value pairs. When including `s:form`, you must also include `s:contentType "multipart/form-data"`. The GDI sends the form object as an `application/x-www-form-urlencoded` string that contains the specified parameters. See Example form Parameter Usage below for sample usage.

Option	Type	Description
format	RDF list	If the data is file-based, you can include the `format` property to add parameters that describe the source. See File Source Format Options for details about the supported parameters.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.
concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:

Option	Type	Description

		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

partitionBy	string, variable,	The GDI attempts to partition queries automatically across the available cores (slices) in AnzoGraph. To determine how to

Option	Type	Description
	list	partition the query, the GDI uses metadata from the source. It looks for any column in an index, preferring the primary key column if it is interpolable. However, it only considers the first column in any index on the table. After determining the partition column, the GDI does a MIN/MAX on the column as well as a basic sizing query. To specify which column or columns the GDI should partition on, you can include the `partitionBy` property in the query. The property supports a list of source field names, bound variables, or the object `s:auto`, which forces the GDI to partition the data when the source does not define partitioning metadata.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for

Option	Type	Description
		each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.

Option	Type	Description
offset	int	This property can be used to offset the data that is returned by a number of rows.
orderBy	string, variable, list	You can include this property to order the result set by a field name, a bound variable, or a list of names or bound variables.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_ variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Note** The parentheses around the binding, data type, and format specifications are not required but are included in this document for readability.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column.

Option	Type	Description
		For example for CSV, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline ("AIRLINE").` **Note** For FileSource, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern could be written as `?averagePerDay ("average\\/day").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19."

Option	Type	Description
		Note The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Mapping the Content Property to JSON

The `s:content` property can be configured with an inline object (blank node) that gets translated to JSON in the request body. This mapping allows for creation of embedded objects and arrays as well as a mechanism for iterating over all available input so that HTTP endpoints that support batching can be used more effectively.

Using Blank Nodes

Blank nodes are used to create an object in the output JSON. The local name of any predicate used within `content` becomes a key in the generated JSON object. Blank nodes can be embedded within each other, allowing the hierarchical nature of JSON to be represented. For example:

```
s:content [ ex:firstName "Mary" ; ex:lastName "Barry" ] ;
```

Or

```
s:content [ ex:person [ ex:firstName "Mary" ] ] ;
```

Using Variables

Variables can be also used in the object position to construct a request from input at runtime. For example:

```
s:content [ ex:firstName ?firstName ; ex:lastName ?lastName ] ;
```

The values for the variables can come from a TOPDOWN variable, a VALUES clause in the SERVICE block, or another data source. Any unbound variables in the input will not be added to the generated JSON object.

Using RDF Lists

An RDF list can also be used to create an array in the output JSON. For example:

```
s:content [ ex:allKnownNames ( ?firstName ?lastName ?nickName ) ]
```

An RDF list can also be embedded inside another list to create an array in the output JSON and populate it with items evaluated against a repeating pattern across all available input rows for a slice. That pattern can be a variable, which generates an array of primitive values, or a blank node, which generates an array of mapped JSON objects. For example:

```
s:content [ ex:documents ((?id)) ] ;
```

Or

```
s:content [ ex:documents (([ ex:id ?id ; ex:title ?title ])) ;
```

Example

The following example query demonstrates the use of `s:content` to generate JSON. The query also includes the `s:concurrency` property to restrict execution to a single slice. Without limiting execution when there are a small number of inputs (as in the VALUES clause), each input row gets executed on its own. As the inputs increase, each slice operates over a larger number of inputs until the default `s:batching 5000` is applied.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX api: <http://contoso.com/api/>
```

```
SELECT *
WHERE {
    SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit>
   {
      VALUES (?firstName ?lastName ?dob ?email)
      {
        ("Gray" "Hay" "1978-03-18"^^xsd:date "gray@abc.com")
        ("Ana" "Bana" "1974-10-20"^^xsd:date "ana@abc.com")
        ("George" "Forge" "1975-08-13"^^xsd:date "george@abc.com")
        ("Miles" "Giles" "1977-04-12"^^xsd:date "miles@abc.com")
      }

      ?data a s:HttpSource ;
        s:url "https://postman-echo.com/post" ;
        s:header [ s:name "Accept" ; s:value "application/json" ] ;
        s:concurrency 1 ;
        s:content
        (([
           api:dateOfBirth ?dob ;
           api:email ?email ;
           api:year 2020 ;
           api:person [ api:firstName ?firstName ; api:lastName ?lastName ] ;
        ])) ;
        s:selector "data" ;
        ?firstName ("person.firstName" xsd:string) ;
        ?lastName ("person.lastName" xsd:string) ;
        ?dob ("dateOfBirth" xsd:date) ;
        ?email ("email" xsd:string) ;
        ?year ("year" xsd:int) .
  }
}
```

The `content` portion of the request that the query generates is shown below:

```
[{
   "firstName": "Gray" ,
   "lastName": "Hay" ,
   "dateOfBirth": "1978-03-18" ,
   "email": "gray@abc.com" ,
   "year": 2020
  },
  {
```

```
   "firstName": "Ana" ,
   "lastName": "Bana" ,
   "dateOfBirth": "1974-10-20" ,
   "email": "ana@abc.com" ,
   "year": 2020
  },
  {
    ...
}]
```

Query Examples

- Topdown Query with URL Parameters
- Generator Query against an Anzo SPARQL Endpoint
- API Queries
- Example form Parameter Usage

Topdown Query with URL Parameters

The query below reads data from a sample HTTP source that compiles worldwide weather statistics. The source has several models available for retrieving data that is current, daily, historical, etc. To target current data, the query includes `s:selector "currently"`. In addition, the query demonstrates the use of the "topdown" functionality, where the query sends values to the source to narrow the results. The VALUES clause specifies the latitude and longitude values for the cities to return data for. In addition, since this sample source requires parameters to be specified in the connection URL, the `s:url` value includes `?lat` and `?long` as parameters as part of the value.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ex: <http://example.org/ontologies/City#>

SELECT
       ?city ?state ?temp ?rainChance
       ?humidity ?pressure ?windSpeed
WHERE
{
```

```
       SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit>
    {
         ?data a s:HttpSource ;
         s:url "https://sampleEndpoint.com/forecast/{{?lat}},{{?long}}" ;
         s:selector "currently" ;
         ?lat ("latitude") ;
         ?long ("longitude") ;
         ?temp ("temperature") ;
         ?rainChance ( "precipProbability" ) ;
         ?humidity () ;
         ?pressure () ;
         ?windSpeed () .
    }
    VALUES( ?city ?state ?lat ?long )
    {
         ( "Lakeway" "TX" 30.374563 -97.975892 )
         ( "Boston" "MA" 42.358043 -71.060415 )
         ( "Seattle" "WA" 47.590720 -122.307053 )
         ( "Chicago" "IL" 41.837741 -87.823296 )
         ( "Hilo" "HI" 19.702040 -155.090312 )
    }
}
ORDER BY ?city
```

Generator Query against an Anzo SPARQL Endpoint

The example below is a GDI Generator query that retrieves graphmart data from a remote Anzo SPARQL endpoint.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

INSERT {
  GRAPH ${targetGraph} {
    ?s ?p ?o }
}
${usingSources}
WHERE {
    SERVICE <http://cambridgesemantics.com/services/DataToolkit>
```

```
  {
        ?data a s:HttpSource ;
            s:url
"https://10.10.0.10/sparql/graphmart/http%3A%2F%2Fcambridgesemantics.com%2FGraphmart%2F
1686168b-3eaf-4fdc-9730-1903717b9e62";
            s:trust "all" ;
            s:username "user";
            s:password "pass";
            s:contentType "application/sparql-query" ;
            s:header [ s:name "Accept" ; s:value "text/csv" ] ;
        s:content """
          PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          SELECT ?s ?p ?o
          WHERE {
            ?s ?p ?o .
            FILTER(ISLITERAL(?o))
          }
        """ .
        ?rdf a s:RdfGenerator, s:OntologyGenerator ;
            s:as (?s ?p ?o) ;
            s:ontology <http://cambridgesemantics.com/ontologies/TopMovies> ;
            s:base <http://cambridgesemantics.com/data> .
  }
}
```

API Queries

The following example queries the Google Recognize API to request transcriptions for voice recordings that are stored in a Google bucket.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX anzo: <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl: <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

INSERT {
GRAPH ${targetGraph}{
    ?record <http://google.com/transcript> ?transcript .
```

```
    ?record <http://google.com/confidence> ?confidence .
    ?record <http://google.com/file> ?file .
  }
}
${usingSources}
WHERE {
    BIND(<gs://csi-se/demo/emergency-test.mp3> as ?file)
    BIND(UUID() as ?record)
{
        SERVICE <http://cambridgesemantics.com/services/DataToolkit>
        {
            ?data a s:HttpSource ;
              s:selector "results.alternatives" ;
              s:url "https://speech.googleapis.com/v1p1beta1/speech:recognize" ;
              s:authorization [ a s:BearerToken ; s:token """ya29..."""" ] ;
              s:content """
    {
      "config": {
        "encoding":"MP3",
        "sampleRateHertz": 16000,
        "languageCode": "en-US",
        "enableWordTimeOffsets": false
    },
      "audio": {
        "uri":"gs://csi-se/demo/emergency-test.mp3"}
    }
    """ ;
            ?confidence ("confidence") ;
            ?transcript ("transcript") .
      }
   }
}
```

The example below includes the `header` and `content` properties to send a request that contains small text snippets for sentiment analysis.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://cambridgesemantics.com/ontologies/Sentiment_Analysis#>

INSERT {
```

```
  GRAPH ${targetGraph} {
    ?requirement a ont:Sentiment ;
      ont:p_Sentiment_Type ?sentiment ;
      ont:p_Sentiment_Score ?polarity .
  }
}
${usingSources}
WHERE {
  ?requirement a
<http://cambridgesemantics.com/Layer/3b4163e7f53149d5a815627be5d409bd/Model#Requirement
s> ;

<http://cambridgesemantics.com/Layer/3b4163e7f53149d5a815627be5d409bd/Model#Requirement
s.reqText> ?requirement_text .

  SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:HttpSource ;
        s:url "https://text-analysis12.p.rapidapi.com/sentiment-analysis/api/v1.1" ;
        s:method "POST" ;
        s:header [ s:name "Accept" ; s:value "application/json" ] ,
                 [ s:name "X-RapidAPI-Key" ; s:value "key" ] ,
                 [ s:name "X-RapidAPI-Host" ; s:value "text-analysis12.p.rapidapi.com"
] ;
        s:contentType "application/json" ;
        s:content """{ "text": "{{?requirement_text}}" , "language": "english" }""" ;
        ?polarity ("aggregate_sentiment/compound" xsd:double);
        ?sentiment () .
  }
}
```

Example form Parameter Usage

The query snippet below shows an example that incorporates the `s:form` parameter.

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
  VALUES (?reviews_text) {
  ("Horrible, terrible, will never use again.")
  ("Wonderful, magnificent, will recommend to everyone!")
  ("Simply OK. I might purchase this product again.") }

  ?data a s:HttpSource ;
    s:url "https://api.meaningcloud.com/sentiment-2.1" ;
    s:contentType "multipart/form-data" ;
```

```
    s:form [ s:name "key" ; s:value "9eab751142..." ],
           [ s:name "lang" ; s:value "auto" ],
           [ s:name "txt" ; s:value ?reviews_text ] ;
    ?confidence ();
    ?score_tag ();
    ?subjectivity () ;
    ?irony () ;
    ?agreement () .
}
```

Querying an Elasticsearch Source

This topic provides details about the structure to use when writing GDI queries to read or ingest data from Elasticsearch data sources. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Query Syntax
- Query DSL and Filter Mapping
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for Elasticsearch sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX es:    <http://elastic.co/search/>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
```

```
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:ElasticSource ;
        s:url "string" ;
        [ s:username "string" ; ]
        [ s:password "string" ; ]
        [ s:property [ s:name "string" ; s:value "string" ; ]
        [ es:aggregations [ rdf_list ] ; ]
        [ es:config "string" ; ]
        [ es:document "string" ; ]
        [ es:field "string" | ?variable ; ]
        [ es:highlight [ rdf_list ] ; ]
        [ es:html boolean ; ]
        [ es:index "string" ; ]
        [ es:minScore float ; ]
        [ es:query "string" | [ rdf_list ] ; ]
        [ es:routing "string" ; ]
        [ es:searchAfter [ rdf_list ] ; ]
        [ es:size int ; ]
        [ es:source boolean | [ rdf_list ] ; ]
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:partitionBy "string" | ?variable ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
```

```
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:orderBy "string" | ?variable ; ]
        [ s:limit int ; ]
        # Mapping variables
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URIs `<http://cambridgesemantics.com/ontologies/DataToolkit#>` and `<http://elastic.co/search/>` as well as the `s:` and `es:` prefixes. As shown in the examples, however, the prefixes or full property URIs do need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.

Option	Type	Description
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It

Option	Type	Description
		also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.

url	string	This property specifies the URL to use to access the source.
		
		Important
		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example:
		`?data a s:ElasticSource ;`
		`s:url "{{@es.hostname}}:{{@es.port}}" ;`
		`s:username "{{@es.username}}" ;`
		`s:password "{{@es.password}}" ;`

username	string	This property lists the user name to use for the connection to the Elasticsearch.
		
		Tip
		If you want to group the username and password properties, you can wrap them with `s:credentials [ ]`. For example:
		`s:credentials [`
		`s:username “username” ;`
		`s:password “password” ;`

Option	Type	Description
		`]`
password	string	This property lists the password for the given username.
property	RDF list	This property can be included to list any source-specific configuration values. `s:property [ s:name "custom_property_name" ; s:value "custom_value" ]`
aggregations	RDF list	You can include this property to calculate aggregations over the specified bindings. For information about aggregations, see Aggregations in the Elasticsearch documentation.
config	string	To enable you to use explicit mappings, you can include this property to specify the URL to the index configuration file to employ. For example, `es:config "/opt/shared/elastic/mapping.json"`.
document	string	This property lists the document(s) to search.
field	string or variable	This property defines the field to operate on. The value can be a string or bound variable.
highlight	RDF list	You can include this property to define how results are highlighted. For information about the available properties, see Highlighting Elasticsearch Results.
html	boolean	This property controls whether to output HTML for highlighted results. Defaults to `true`.

Option	Type	Description
index	string	This property can be included to specify the indexes to search. Specify multiple indexes in a comma-separated list. For example, `es:index "projectA_mar", "projectA_apr" ;.`
minScore	float	This property defines the minimum score for matching documents. Documents with a lower score are not included in the search results.
query	string or RDF list	This property defines the query to execute. The value can be a string or a query object that maps to the Elasticsearch Query DSL. To generate the final query, the GDI combines `es:query` with any filters it can push to the Elasticsearch DSL. For more information about the `query` property and mapping Elasticsearch filters to SPARQL FILTER clauses, see Query DSL and Filter Mapping below.
routing	string	This property can be included to route a document to a specific shard or to limit the search to a particular shard.
searchAfter	RDF list	You can include this property to define the key values to start searching from.
size	int	This property maps to the `size` parameter in the Elasticsearch Search API and configures the batch size or maximum number of hits to return in a single call. Defaults to `10` and typically does not need to be changed.
source	boolean or RDF list	This property can be included to specify the source data to include in results. The value can be a boolean, list of fields, or a list of variable bindings. When `true`, all source data is returned. When `false`, no source data is returned.

Option	Type	Description
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`

Option	Type	Description
		`s:executorsPerNode 8 ;` `] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

partitionBy	string, variable, list	The GDI attempts to partition queries automatically across the available cores (slices) in AnzoGraph. To determine how to partition the query, the GDI uses metadata from the source database. It looks for any column in an index, preferring the primary key column if it is interpolable. However, it only considers

Option	Type	Description
		the first column in any index on the table. After determining the partition column, the GDI does a MIN/MAX on the column as well as a basic sizing query. To specify which column or columns the GDI should partition on, you can include the `partitionBy` property in the query. The property supports a list of source field names, bound variables, or the object `s:auto`, which forces the GDI to partition the data when the source does not define partitioning metadata.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the

Option	Type	Description
		source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`. The GDI runs an Elasticsearch value count aggregation.
offset	int	This property can be used to offset the data that is returned by a number of rows.

Option	Type	Description
orderBy	string, variable, list	You can include this property to order the result set by a field name, a bound variable, or a list of names or bound variables.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Note** The parentheses around the binding, data type, and format specifications are not required but are included in this document for readability.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column.
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports

Option	Type	Description
		the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Query DSL and Filter Mapping

The vocabulary used in GDI queries against an ElasticSource closely mimics the Elasticsearch Query DSL. The table below shows a side-by-side view of a DSL query that is mapped to SPARQL using the `es:query` property:

DSL	SPARQL
{ "query": { "bool" : { "must" : { "term" : { "user.id" : "kimchy" } }, "filter": { "term" : { "tags" : "production" } }, "must_not" : { "range" : { "age" : { "gte" : 10, "lte" : 20 } } }, "should" : [{ "term" : { "tags" : "env1" } }, { "term" : { "tags" : "deployed" } }], "minimum_should_match" : 1, "boost" : 1.0 } } }	es:query [a es:BoolQuery ; es:must [a es:TermQuery ; es:field "user.id" ; es:value "kimchy" ;] ; es:filter [a es:TermQuery ; es:field "tags" ; es:value "production" ;] ; es:mustNot [a es:RangeQuery ; es:field "age" ; es:gte 10 ; es:lte 20 ;] ; es:should ([a es:TermQuery ; es:field "tags" ; es:value "env1"] [a es:TermQuery ; es:field "tags" ; es:value "deployed"]) ; es:minimumShouldMatch 1 ; es:boost 1.0 ;]

DSL	SPARQL
}	

The following example SERVICE clause with comments provides details about how the GDI `es:query` property can be mapped to DSL:

```
SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
  ?data a es:ElasticSource ;
    s:url "http://localhost:9200/" ;
# When the value of es:query is a simple literal,
# it is mapped to an Elastic query string query.
    es:query "literal"

# When the value of es:query is an RDF list,
# you can specify other query types,
# such as a match query:
    es:query [
      a es:MatchQuery ;
      es:field "title" | ?title ; # field can be a literal or bound variable
      es:query "moby dick" ;
    ] ;
# or a boolean query:
    es:query [
      a es:BoolQuery ;
      es:should ([
        a es:RangeQuery ;
        es:field ?amount ;
        es:gt 500 ;
        es:lt 1000 ;
      ] [
        a es:TermQuery ;
        es:field ?status ;
        es:value 'late' ;
      ]) ;
    ] ;
}
```

Filter Mapping

Filtering can be performed inside the `es:query` list or you can add a FILTER clause to the query. For example, the table below shows the SPARQL snippet above expressed as a FILTER clause.

SPARQL Query	FILTER Clause

es:query [
  a es:BoolQuery ;
  es:must [
  a es:TermQuery ;
  es:field "user.id" ;
  es:value "kimchy" ;
  ] ;
  es:filter [
  a es:TermQuery ;
  es:field "tags" ;
  es:value "production" ;
  ] ;
  es:mustNot [
  a es:RangeQuery ;
  es:field "age" ;
  es:gte 10 ;
  es:lte 20 ;
  ] ;
  es:should (
  [a es:TermQuery ; es:field "tags" ; es:value "env1"]
  [a es:TermQuery ; es:field "tags" ; es:value "deployed"]
) ;
  es:minimumShouldMatch 1 ;
  es:boost 1.0 ;
]
	FILTER(?user_id = "kimchy" &&
	  ?tags = "production" &&
	  !(?age >= 10 && ?age <= 20) &&
	  (?tags == "env1" || ?tags == "deployed"))

The table below shows each of the supported ElasticSource FILTER translations. Only expressions matching the list below will be translated by the GDI. If the expression is of the form `value <= ?field`, the inequality is flipped to `?field > value` before translating.

es:query Expression	FILTER Clause Expression
es:query [a es:BoolQuery ; es:mustNot expr]	!expr
es:query [a es:BoolQuery ; es:must (left right)]	left && right
es:query [a es:BoolQuery ; es:should (left right)]	left || right
es:query [a es:RangeQuery ; es:field ?field ; es:lt value]	?field < value
es:query [a es:RangeQuery ; es:field ?field ; es:lte value]	?field <= value
es:query [a es:TermQuery ; es:field ?field ; es:value value]	?field = value
es:query [a es:BoolQuery ; es:mustNot [a es:TermQuery ; es:field ?field ; es:value value]]	?field != value
es:query [a es:RangeQuery ; es:field ?field ; es:gte value]	?field >= value
es:query [a es:RangeQuery ; es:field ?field ; es:gt value]	?field > value
es:query [a es:QueryStringQuery ; es:field ?field ; es:query pattern ; es:defaultOperator "AND"]	REGEX(?field, pattern, "q")
es:query [a es:TermsQuery ; es:field ?field ; es:value value, ...]	IN(?field, value, ...)
es:query [a es:BoolQuery ; es:mustNot [a es:TermsQuery ; es:field ?field ; es:value value, ...]]	NOT IN(?field, value, ...)

es:query Expression	FILTER Clause Expression
`es:query [ a es:MatchQuery ; es:field ?field ; es:query value ; es:lenient true ]`	`CONTAINS (?field, value)`
`es:query [ a es:PrefixQuery ; es:field ?field ; es:value value ]`	`STRSTARTS (?field, value)`
`es:query [ a es:ExistsQuery ; es:field ?field ]`	`BOUND(?field)`

Query Examples

- General Query
- Aggregations
- Highlighting

General Query

The following example queries any Elasticsearch indexes that are loaded in the graphmart for which you run the query. No configuration is needed because Anzo manages the indexes that it loads and uses predictable naming conventions and aliases.

```
PREFIX docm: <http://cambridgesemantics.com/ontologies/2011/07/DocumentMetadata#>
PREFIX es: <http://elastic.co/search/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX services: <http://cambridgesemantics.com/services/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?title
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    ?data a es:ElasticSource;
      s:selector "unstructuredfile" ;
      es:query "string" ; # input the text search string
      es:fields "fullText" ;
      ?title (xsd:string) .
```

```
  }
}
```

Aggregations

The following example query performs terms aggregations.

```
PREFIX es: <http://elastic.co/search/>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT *
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    ?data a es:ElasticSource;
    s:url "https://{{@es.hostname}}:{{@es.port}}/" ;
    s:username "{{@es.username}}" ;
    s:password "{{@es.password}}" ;
    es:index "templated_consumption_es" ;
    es:query "*ELM*" ;
    ?instance () ;
    es:aggregations [
      ?artifactTypes [
        a es:TermsAggregation ;
        es:field ?artifactType ;
        es:meta [
          ?label "artifactType" ;
        ] ;
        ?value () ;
        ?count () ;
      ] ;
      ?fileTypes [
        a es:TermsAggregation ;
        es:field ?fileType ;
        es:meta [
          ?label "fileType" ;
        ] ;
        ?value () ;
        ?count () ;
      ] ;
      ?managedBys [
        a es:TermsAggregation ;
        es:field ?managedBy ;
```

```
        es:meta [
          ?label "managedBy" ;
        ] ;
        ?value () ;
        ?count () ;
      ] ;
    ] .
  }
}
```

Highlighting

The following example configures highlighting for fragments from the actor field.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX es: <http://elastic.co/search/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
WHERE {
  SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit>
  {
    ?data a es:ElasticSource ;
      es:url "http://localhost:9200/" ;
      es:index "films" ;
      es:html false ;
      es:query "Clint" ;
      es:field ?actor, ?director ;
      es:highlight [
        es:field ?actor ;
        es:type "plain" ;
        es:fragmentSize 200 ;
        es:numberOfFragments 10 ;
        es:preTags "<mark hit='true'>" ;
        es:postTags "</mark>" ;
      ] ;
      s:selector "film" ;
      ?actor (xsd:string) ;
      ?awards (xsd:string) ;
      ?director (xsd:string) ;
      ?image (xsd:string) ;
      ?length (xsd:long) ;
      ?popularity (xsd:long) ;
      ?subject (xsd:string) ;
```

```
      ?title (xsd:string) ;
      ?year (xsd:long) ;
      ?score () ;
      ?id () ;
      ?highlights [
        ?field () ;
        ?fragment () ;
      ] .
  FILTER(?year = 1990 || ?length > 103)
  FILTER(REGEX(?title, "Manhattan", "q") || REGEX(?subject, "Comedy", "q") || REGEX
(?subject, "Drama", "q"))
  }
}
```

Highlighting Elasticsearch Results

By including the **highlight** property in ElasticSource GDI queries, you can configure the response to include highlights for search results. For general information about highlighting Elasticsearch responses, see Highlighting in the Elasticsearch documentation. Highlight property usage is described below.

- Highlight Syntax
- Highlight Examples

Highlight Syntax

```
es:highlight [
   es:boundaryChars "string" ;
   es:boundaryMaxScan int ;
   es:boundaryScannerLocale "string" ;
   es:boundaryScannerType "string" ;
   es:field "string" ;
   es:forceSource boolean ;
   es:fragmentSize int ;
   es:fragmenter "string" ;
   es:highlightFilter boolean ;
   es:highlightQuery "string" | [ rdf_list ] ;
   es:highlighterType "string" ;
   es:noMatchSize int ;
   es:numberOfFragments int ;
```

```
   es:order "string" ;
   es:phraseLimit int ;
   es:postTags "string" ;
   es:preTags "string" ;
   es:requireFieldMatch boolean ;
] ;
```

Option	Type	Description
boundaryChars	string	This property can be used to define the boundary characters to look for. Defaults to `.,!? \t\n`.
boundaryMaxScan	int	This property can be used to place a limit on the number of characters to scan when looking for boundary characters. Defaults to `20`.
boundaryScannerLocale	string	This property defines the language tag (such as "en-US" or "fr-FR") to apply when searching for sentence and word boundaries.
boundaryScannerType	string	If highlighterType is `unified` or `fvh`, this property can be used to specify how to break the highlighted fragments. This property is ignored when the highlighter type is `plain`. The list below describes the valid values: • **chars**: Valid when the highlighter type is fast vector highlighter (fvh) (`es:highlighterType "fvh"`). Specifies that the highlighting boundaries are the characters specified by boundaryChars. The boundaryMaxScan value controls how far to scan for boundary characters. This is the default value for fvh.

Option	Type	Description
		• **sentence**: This is the default value for the unified highlighter. It configures highlighted fragments to break at the next sentence boundary. You can specify the locale to use with boundaryScannerLocale. When used with the unified highlighter, the sentence scanner splits sentences bigger than fragmentSize at the first word boundary next to fragmentSize. You can set fragmentSize to `0` to avoid splitting sentences. • **word**: Configures highlighted fragments to break at the next word boundary. You can specify the locale to use with boundaryScannerLocale.

field	string or variable	This property specifies the field to retrieve highlights for. It can include a ?variable (which the GDI maps to the full path of the field in the Elasticsearch document), a field name, or a field name pattern. For example:
		`es:highlight [`
		`es:field ?actor ;`
		`es:field "film.actor" ;`
		`es:field "film.*" ;`
		`es:field "*" ;`
		`]`

forceSource	boolean	This property controls whether to highlight based

Option	Type	Description
		on the source even if the field is stored separately. Defaults to `false`.
fragmentSize	int	This property specifies the number of characters to include in highlighted fragments. Defaults to `100`.
fragmenter	string	If highlighterType is `plain`, this property can be used to specify how to break up text in highlight snippets. The list below describes the valid values: • **simple**: Breaks text into fragments that are the same size (as specified by fragmentSize). • **span**: The default value. Breaks text into fragments that are the same size but tries to avoid breaking text between highlighted terms.
highlightFilter	boolean	This property controls whether to highlight filter results.
highlightQuery	string or object	This property specifies the highlight query. The value can be a string or a query object that maps to the Elasticsearch query DSL.
highlighterType	string	This property defines the type of highlighter to use, `"plain"`, `"unified"`, or `"fvh"`.
noMatchSize	int	This property specifies the number of characters to return from the beginning of the field if there are no matching fragments to highlight. Defaults to `0`

Option	Type	Description
		(nothing is returned).
numberOfFragments	int	This property can be used to set the maximum number of fragments to generate. If this property is set to `0`, no fragments are returned. Instead, the entire field contents are highlighted and returned, which can be useful if you want to highlight short text (such as a title or address) for which fragmentation is not required. Defaults to `5`. If the number of fragments is `0`, fragmentSize is ignored.
order	string	This property can be included to sort highlighted fragments by score. When `es:order "score"`, the most relevant fragments are output first. Defaults to `"none"`; fragments are output in the order they appear in the field.
phraseLimit	int	If highlighterType is `fvh`, this property can be used to limit the number of matching phrases to consider. Limiting the number of phrases prevents the fvh highlighter from analyzing too many phrases and consuming too much memory. Defaults to `256`.
postTags	string	This property is used in conjunction with preTags to define the HTML tags to use for the highlighted elements. This property defines the closing tag to use after the highlighted text. Defaults to `</em>`.
preTags	string	This property is used in conjunction with postTags to define the HTML tags to use for the highlighted elements. This property defines the opening tag to use before the highlighted text. Defaults to `<em>`.

Option	Type	Description
requireFieldMatch	boolean	This property controls whether to highlight only the fields that contain a query match. Defaults to `true`. If `false`, all fields are highlighted.

Highlight Examples

The following example configures highlighting for fragments from the actor field.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX es: <http://elastic.co/search/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
WHERE {
  SERVICE TOPDOWN <http://cambridgesemantics.com/services/DataToolkit>
  {
    ?data a es:ElasticSource ;
      es:url "http://localhost:9200/" ;
      es:index "films" ;
      es:html false ;
      es:query "Clint" ;
      es:field ?actor, ?director ;
      es:highlight [
        es:field ?actor ;
        es:type "plain" ;
        es:fragmentSize 200 ;
        es:numberOfFragments 10 ;
        es:preTags "<mark hit='true'>" ;
        es:postTags "</mark>" ;
      ] ;
      s:selector "film" ;
      ?actor (xsd:string) ;
      ?awards (xsd:string) ;
      ?director (xsd:string) ;
      ?image (xsd:string) ;
      ?length (xsd:long) ;
      ?popularity (xsd:long) ;
      ?subject (xsd:string) ;
      ?title (xsd:string) ;
```

```
      ?year (xsd:long) ;
      ?score () ;
      ?id () ;
      ?highlights [
        ?field () ;
        ?fragment () ;
      ] .
  FILTER(?year = 1990 || ?length > 103)
  FILTER(REGEX(?title, "Manhattan", "q") || REGEX(?subject, "Comedy", "q") || REGEX
(?subject, "Drama", "q"))
  }
}
```

Querying a File Source

The Graph Data Interface (GDI) uses the Apache Commons VFS library to work with file systems. Many of the properties specified in queries against file sources reflect the requirements of the VFS library for that source. The topics in this section provide guidance on writing GDI queries for each of the supported file types.

- Querying CSV and TSV Files
- Querying JSON and NDJSON Files
- Querying XML Files
- Querying Parquet and SAS Files
- File Source Format Options
- File Storage Connection Options

Querying CSV and TSV Files

This topic provides details about the structure to use when writing GDI queries to read or ingest data from CSV or TSV files. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Query Syntax
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for CSV and TSV sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:FileSource ;
        s:url "string" ;
        [ s:options [ file_storage_connection_options ] ; ]
        [ s:pattern "string" ; ]
        [ s:maxDepth int ; ]
        [ s:format [ source_format_options ; ] ; ]
        [ s:mimetype "string" ; ]
        [ s:username "string" ; ]
        [ s:password "string" ; ]
```

```
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ]
        # Mapping variables
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the

Option	Type	Description
		set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.

Option	Type	Description
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
url	string	This property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the pattern and/or maxDepth properties.
options	RDF list	If additional connection information needs to be provided to access the file storage system, include the `options` property to list any storage-specific connection parameters. See File Storage Connection Options for information about the supported properties for each storage type.
pattern	string	This property can be used to specify a wildcard pattern for matching file names. For example, `s:pattern "common_prefix*.csv"`. You can include one `s:pattern` property per FileSource. The GDI supports Unix file globbing syntax outside of parentheses. Within parentheses, full Java regular expression language is supported. For example, including `s:pattern "data/**/customer_*.csv"` tells the GDI to load all files that match the pattern "customer_*.csv" from any number of

Option	Type	Description
		subdirectories under the `data` directory. Similarly `s:pattern "(\d+)/transaction_*.csv"` tells the GDI to load all files that match the pattern "transaction_*.csv" in all subdirectories.
maxDepth	int	This property can be used to limit the directory traversal depth. By default, when `s:url` specifies a directory (and a `s:pattern` that limits that traversal depth is not specified), all subdirectories are processed. To process only the files in the top level directory, set maxDepth to 0 (`s:maxDepth 0`). To process the files in the top level directory plus the first-level subdirectories, set maxDepth to 1 (`s:maxDepth 1`), and so on.
format	RDF list	You can include the `format` property to add parameters that describe the source files. See File Source Format Options for details about the supported parameters.
mimetype	string	This property can be included to specify the MIME type of the data. If you are querying TSV files that do not have a .tsv file extension, include the `mimetype` property with a value of `text/tsv` (`s:mimetype "text/tsv"`).
username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to

Option	Type	Description
		5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the

Option	Type	Description

		rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.

Option	Type	Description
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization

Option	Type	Description
		Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_ variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Note** The parentheses around the binding, data type, and format specifications are not required but are included in this document for readability.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column,

Option	Type	Description
		then the binding is needed to map the new variable to that source column. For example for CSV, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline ("AIRLINE").` **Note** For FileSource, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern could be written as `?averagePerDay ("average\\/day").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example,

Option	Type	Description
		`"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Query Examples

The example below ingests a directory of CSV files into a Graphmart. The pattern property (`s:pattern "post_[0-9]*_[0-9]*.csv"`) is used to narrow down the set of files to load. Since the files use a pipe (|) as the delimiter rather than a comma (,), the delimiter property is also included to specify the | character.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX snvoc: <http://www.ldbc.eu/ldbc_socialnet/1.0/vocabulary/>
PREFIX sntag: <http://www.ldbc.eu/ldbc_socialnet/1.0/tag/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

INSERT {
   GRAPH ${targetGraph}
  {
      ?postIRI a snvoc:Post, snvoc:Message ;
      snvoc:creationDate ?creationDate ;
```

```
      snvoc:id ?id ;
      snvoc:imageFile ?imageFile ;
      snvoc:locationIP ?locationIP ;
      snvoc:browserUsed ?browserUsed ;
      snvoc:language ?language ;
      snvoc:content ?content ;
      snvoc:length ?length ;
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?data a s:FileSource ;
      s:url "/opt/shared-files/data/csv/post_6_0/" ;
      s:pattern "post_[0-9]*_[0-9]*.csv" ;
      s:format [ s:delimiter "|" ] ;
      ?creationDate (xsd:dateTime) ;
      ?id (xsd:string) ;
      ?imageFile (xsd:string) ;
      ?locationIP (xsd:string) ;
      ?browserUsed (xsd:string) ;
      ?language (xsd:string) ;
      ?content (xsd:string) ;
      ?length(xsd:string) .
   BIND(IRI("http://www.ldbc.eu/ldbc_socialnet/1.0/data/Post/{{?id}}") as ?postIRI)
  }
}
```

The example below is similar to the query above but it specifies the formats to use for the `xsd:date` values.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX kd:    <http://cambridgesemantics.com/ont/autogen/Rh/Kaggle_Diabetes#>

INSERT {
   GRAPH ${targetGraph}
```

```
  {
      ?URI a kd:Diagnosis ;
      kd:Diagnosis_DiagnosisGuid ?diagnosis_guid ;
      kd:Diagnosis_PatientGuid ?patient_guid ;
      kd:Diagnosis_ICD9Code ?icd9Code ;
      kd:Diagnosis_DiagnosisDescription ?diagnosisDescription ;
      kd:Diagnosis_StartDate ?cus_start_date ;
      kd:Diagnosis_EndDate ?Date_End ;
      kd:Diagnosis_Acute ?acute ;
      kd:Diagnosis_UserGuid ?UserGuid .
  }
}
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?csv a s:FileSource ;
      s:url "/opt/shared-files/source_data/kaggle_diabetes/" ;
      s:pattern "Diagnosis.csv" ;
      s:format [ s:delimiter "," ] ;
      ?diagnosis_guid ("DiagnosisGuid" xsd:string) ;
      ?patient_guid ("PatientGuid" xsd:string) ;
      ?icd9Code ("ICD9Code" xsd:string) ;
      ?diagnosisDescription ("DiagnosisDescription" xsd:string) ;
      ?acute ("Acute" xsd:int ) ;
      ?UserGuid ("UserGuid" xsd:string) ;
      ?cus_start_date ("CUSTOMER_START_DATE" xsd:date "yyyy-MM-dd") ;
      ?Date_End ("Date End" xsd:date "MM/dd/yy") .
  }
  BIND(IRI(CONCAT("urn://cambridgesemantics.com/kaggle_diabetes/patient/",ENCODE_FOR_
URI(?diagnosis_guid))) as ?URI)
}
```

Querying JSON and NDJSON Files

This topic provides details about the structure to use when writing GDI queries to read or ingest data from JSON or NDJSON files. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Query Syntax
- Hierarchical Bindings and Arrays

- Capturing Property Keys
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for JSON sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:FileSource ;
        s:url "string" ;
        [ s:options [ file_storage_connection_options ] ; ]
        [ s:pattern "string" ; ]
        [ s:maxDepth int ; ]
```

```
        [ s:format [ source_format_options ; ] ; ]
        [ s:mimetype "string" ; ]
        [ s:username "string" ; ]
        [ s:password "string" ; ]
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ]
        # Mapping variables and hierarchical bindings
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific

Option	Type	Description
		declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of

Option	Type	Description
		the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
url	string	This property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the pattern and/or maxDepth properties.
options	RDF list	If additional connection information needs to be provided to access the file storage system, include the `options` property to list any storage-specific connection parameters. See File Storage Connection Options for information about the supported properties for each storage type.
pattern	string	This property is used to specify a wildcard pattern for matching file names. For example, `s:pattern "common_prefix*.json"`. You can include one `s:pattern` property per FileSource. The GDI supports Unix file globbing syntax outside of parentheses.

Option	Type	Description
		Within parentheses, full Java regular expression language is supported. For example, including `s:pattern "data/**/customer_*.json"` tells the GDI to load all files that match the pattern "customer_*.json" from any number of subdirectories under the `data` directory. Similarly `s:pattern "(\d+)/transaction_*.json"` tells the GDI to load all files that match the pattern "transaction_*.json" in all subdirectories.
maxDepth	int	This property can be used to limit the directory traversal depth. By default, when `s:url` specifies a directory (and a `s:pattern` that limits that traversal depth is not specified), all subdirectories are processed. To process only the files in the top level directory, set maxDepth to 0 (`s:maxDepth 0`). To process the files in the top level directory plus the first-level subdirectories, set maxDepth to 1 (`s:maxDepth 1`), and so on.
format	RDF list	You can include the `format` property to add parameters that describe the source files. See File Source Format Options for details about the supported parameters.
mimetype	string	This property can be included to specify the MIME type of the data.
username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean	This property can be used to disable batching, or it can be used to

Option	Type	Description
	or int	change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of

Option	Type	Description

		requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used for JSON path extraction to traverse nested structures and target specific data. For example, `s:selector "projects"` targets the `projects` class of data. To express a hierarchy, use dot notation. For example, `s:selector "region.state.city"` navigates a hierarchy to

Option	Type	Description
		target `city` data. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.

Option	Type	Description
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_ variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Tip** See Hierarchical Bindings and Arrays below for more information about configuring mapping variables and unpacking JSON files with nested objects and arrays.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a

Option	Type	Description
		source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column. For example for CSV, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline ("AIRLINE").` **Note** For FileSource, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern could be written as `?averagePerDay ("average\\/day").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`

Option	Type	Description
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Hierarchical Bindings and Arrays

When configuring the mapping variables in a query, the GDI provides syntax for unpacking JSON files with nested objects and arrays. One way to express hierarchies in queries is to use brackets ([ ]) to group objects into binding trees. For example, the WHERE clause snippet below organizes mapping variable objects into an `hourly/data` hierarchy by nesting the `?data` patterns inside the `?hourly [ ]` tree:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
```

```
      ?data a s:FileSource;
        s:url "/mnt/data/json/weather.json" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        ?hourly
[
          ?data
[
            ?time (xsd:long) ;
            ?rainProbability ("precipProbability" xsd:double) ;
            ?temperature (xsd:double) ;
            ?feelsLike ("apparentTemperature" xsd:double) ;
            ?windSpeed (xsd:double) ;
        ] ;
    ] .
  }
}
```

When constructing object binding trees, if you choose to introduce the hierarchy with a variable name that is not an exact match to the source label, include a **selector** property to list the value from the source. For example, in the WHERE clause snippet below, `s:selector` is included to select `eventHeader` in the source as `?event` in the query and `statLocation` as `?location`.

```
WHERE
{
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?data a s:FileSource ;
      s:url "/mnt/data/json/part_1.json" ;
      ?event
[
         s:selector "eventHeader" ;
           ?eventId (xsd:string) ;
           ?eventName (xsd:string) ;
           ?eventVersion (xsd:string) ;
           ?eventTime (xsd:dateTime) ;
      ] ;
      ?location
[
         s:selector "statLocation" ;
           ?locationId (xsd:string) ;
```

```
            ?lineNo (xsd:int) ;
            ?statNo (xsd:int) ;
            ?statId (xsd:int) ;
        ] .
    }
}
```

As an alternative to grouping objects in binding trees, the **selector** property also supports using dot notation to specify paths. For example, the WHERE clause snippet below rewrites the first example query to express the same `hourly/data` hierarchy as a path in the `s:selector` value:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:FileSource;
        s:url "/mnt/data/json/weather.json" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        s:selector: "hourly.data" ;
        ?time (xsd:long) ;
        ?rainProbability ("precipProbability" xsd:double) ;
        ?temperature (xsd:double) ;
        ?feelsLike ("apparentTemperature" xsd:double) ;
        ?windSpeed (xsd:double) .
  }
}
```

In addition to object binding trees and selectors, the GDI offers additional syntax for reading or ingesting JSON sources with nested objects and arrays. For example, following the JSON sample file below is a query that captures each value in the arrays:

```
{
   "payload" :
   {
      "IBP_IndEvent_MSR" :
      {
         "unit" : "ms",
         "value" : [ 0, 1 ]
      },
      "IBP_IndEvent_RMF" :
```

```
      {
         "unit" : "-",
         "value" : [ 0.012, 1.398, 3.1415 ]
      }
   }
}
```

To read the JSON file above, the following query uses an object binding (`?values [ ]`) to drill down to the `value` arrays in the source. An `@` selector is specified in the `?value` variable binding (`?value ("@" xsd:double)`) to retrieve each of the array values. For an array of primitive values, the @ selector captures each value in the array. If the source `value` was an array of objects, the @ selector would retrieve a JSON representation for each object in the array. In addition to creating a new binding context for the primitive array values, the `?values` object binding also includes `?index ("!array::index")` to capture the index array with the primitive value.

```
PREFIX s:   <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:FileSource ;
      s:url "/mnt/data/json/array-index.json" ;
      s:selector "payload.*" ;
      ?unit (xsd:string) ;
      ?values [
         s:selector "value" ;
         ?value ("@" xsd:double) ;
         ?index ("!array::index") ;
      ] .
  }
}
```

The results of the query are shown below:

```
unit | value  | index
-----+--------+-------
ms   |      0 |     0
ms   |      1 |     1
-    |  0.012 |     0
-    |  1.398 |     1
-    | 3.1415 |     2
```

If you do not want to retrieve all of the values in an array, you can include the specific index number to retrieve instead of using the @ symbol. In the variable binding, the index number is appended in brackets ([ ]) to the binding column name. For example, the following variable binding retrieves the second index value (the third value in the array) from a "projects" array: `?project ("projects[2]")`. The next example uses the following JSON file:

```
{
   "field1" : "value1" ,
   "arrayfield" : [
        "arrayvalue1",
        "arrayvalue2"
   ]
}
```

To retrieve only the second value in the array, the following query appends the index value `1` to the array column name, `arrayfield:`

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
SELECT *
WHERE {
   SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
       ?json a s:FileSource ;
       s:url "/mnt/data/json/array-index-2.json" ;
       ?field1 (xsd:string) ;
       ?arrayval ("arrayfield[1]" xsd:string) .
  }
}
```

The results of the query are shown below:

```
field1   | arrayval
---------+----------
value1   |arrayvalue2
```

Capturing Property Keys

In GDI Generator queries, the names of property keys can be captured from files by including a variable as the `s:selector` and using the same variable as the `s:key`. For example, the GDI query below ingests the following simple JSON file.

```
# company.json
{
  "AAPL": {
    "name": "Apple Corp"
  },
  "MSFT": {
    "name": "Microsoft"
  },
  "IBM": {
   "name": "IBM"
  }
}
```

In the query, the keys "AAPL," "MSFT," and "IBM" are selected as the ?TickerSymbol variable and the key is set to the same value.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT ${targetGraph} {
  ?s ?p ?o .
  }
}
WHERE {
    SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:FileSource ;
        s:url "/opt/shared/data/company.json" ;
        s:selector "?TickerSymbol" ;
        s:key (?TickerSymbol) ;
        s:model "Company" ;
        ?TickerSymbol (xsd:string) ;
        ?name (xsd:string) .

      ?rdf a s:RdfGenerator, s:OntologyGenerator;
        s:as (?s ?p ?o) ;
        s:ontology <http://cambridgesemantics.com/ontologies/company> ;
        s:base ${targetGraph} .
  }
}
```

Selecting the predicates and objects from the graph shows the tickerSymbol predicate and value.

```
SELECT ?p ?o
${usingSources}
WHERE { ?s ?p ?o . }
ORDER BY desc(?o)
```

```
p                                                              | o
---------------------------------------------------------------+-------------
http://anzograph.com/ontologies/company#Company.name           | Microsoft
http://anzograph.com/ontologies/company#Company.tickerSymbol   | MSFT
http://anzograph.com/ontologies/company#Company.name           | IBM
http://anzograph.com/ontologies/company#Company.tickerSymbol   | IBM
http://anzograph.com/ontologies/company#Company.name           | Apple Corp
http://anzograph.com/ontologies/company#Company.tickerSymbol   | AAPL
...
```

Query Examples

The example query below reads a JSON file that contains data about weather. Since the file is hierarchical, the `s:selector` property is included to specify the path to `data` in the hourly/data hierarchy:

```
PREFIX s:   <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:FileSource;
        s:url "/mnt/data/json/weather.json" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        s:selector: "hourly.data" ;
        ?time (xsd:long) ;
        ?rainProbability ("precipProbability" xsd:double) ;
        ?temperature (xsd:double) ;
        ?feelsLike ("apparentTemperature" xsd:double) ;
        ?windSpeed (xsd:double) .
  }
}
```

The following example query ingests data from a JSON file that contains data about the New York Times best selling books.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX books: <http://cambridgesemantics.com/ontologies/NYT_Bestsellers_Ontology#>
INSERT {
    GRAPH ${targetGraph}{
      ?book a books:Book ;
      books:p_Title ?title ;
      books:p_Description ?description ;
      books:p_Author ?author ;
      books:p_Publisher ?publisher ;
      books:p_Date ?rawdate .
  }
}
WHERE {
    SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?data a s:FileSource ;
      s:url "/mnt/data/json/nyt_best_sellers.json" ;
      ?title () ;
      ?author () ;
      ?description () ;
      ?publisher () ;
      ?price() ;
      ?rawdate ("bestsellers_date.$date.$numberLong").
  }
  BIND(IRI(CONCAT("http://cambridgesemantics.com/ontologies/NYT_Bestsellers_Ontology/",
ENCODE_FOR_URI(?title))) AS ?book) .
}
```

A snippet of the file's contents is shown below:

```
{
  "_id": {
    "$oid": "5b4aa4ead3089013507db18b"
  },
  "bestsellers_date": {
    "$date": {
      "$numberLong": "1211587200000"
    }
  },
  "published_date": {
```

```
    "$date": {
      "$numberLong": "1212883200000"
    }
  },
  "amazon_product_url": "http://www.amazon.com/Odd-Hours-Dean-
Koontz/dp/0553807056?tag=NYTBS-20",
  "author": "Dean R Koontz",
  "description": "Odd Thomas, who can communicate with the dead, confronts evil forces
in a California coastal town.",
  "price": {
    "$numberInt": "27"
    },
  "publisher": "Bantam",
  "title": "ODD HOURS",
  "rank": {
    "$numberInt": "1"
    },
  "rank_last_week": {
    "$numberInt": "0"
    },
  "weeks_on_list": {
    "$numberInt": "1"
    }
}
```


Querying XML Files

This topic provides details about the structure to use when writing GDI queries to read or ingest data from XML files. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

- Query Syntax
- Hierarchical Bindings and Arrays
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for XML sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:FileSource ;
        s:url "string" ;
        [ s:options [ file_storage_connection_options ] ; ]
        [ s:pattern "string" ; ]
        [ s:maxDepth int ; ]
        [ s:format [ source_format_options ; ] ; ]
        [ s:mimetype "string" ; ]
        [ s:username "string" ; ]
        [ s:password "string" ; ]
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
```

```
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ]
        # Mapping variables and hierarchical bindings
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Note

For readability, the parameters below exclude the base URI `<http://cambridgesemantics.com/ontologies/DataToolkit#>` as well as the `s:` prefix. As shown in the examples, however, the `s:` prefix or full property URI does need to be included in queries.

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH	N/A	Include the `GRAPH` keyword and target graph parameter

Option	Type	Description
${targetGraph}		`${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataTool`

Option	Type	Description
		`kitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
url	string	This property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the pattern and/or maxDepth properties.
options	RDF list	If additional connection information needs to be provided to access the file storage system, include the `options` property to list any storage-specific connection parameters. See File Storage Connection Options for information about the supported properties for each storage type.
pattern	string	This property is used to specify a wildcard pattern for matching file names. For example, `s:pattern "common_prefix*.xml"`. You can include one `s:pattern` property per FileSource. The GDI supports Unix file globbing syntax outside of parentheses. Within parentheses, full Java regular expression language is supported. For example, including `s:pattern "data/**/customer_*.xml"` tells the GDI to load all files that match the pattern "customer_*.xml" from any number of subdirectories under the `data` directory. Similarly `s:pattern "(\d+)/transaction_*.xml"` tells the GDI to load all files that match the pattern "transaction_*.xml" in all subdirectories.

Option	Type	Description
maxDepth	int	This property can be used to limit the directory traversal depth. By default, when `s:url` specifies a directory (and a `s:pattern` that limits that traversal depth is not specified), all subdirectories are processed. To process only the files in the top level directory, set maxDepth to 0 (`s:maxDepth 0`). To process the files in the top level directory plus the first-level subdirectories, set maxDepth to 1 (`s:maxDepth 1`), and so on.
format	RDF list	You can include the `format` property to add parameters that describe the source files. See File Source Format Options for details about the supported parameters.
mimetype	string	This property can be included to specify the MIME type of the data.
username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.

Option	Type	Description
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`

Option	Type	Description
		`s:rate "10000 every 6 hours" ;` To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate. Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used for XML path extraction to traverse nested structures and target specific data. For example, `s:selector "projects"` targets the `projects` class of data. To express a hierarchy, use dot notation. For example, `s:selector "region.state.city"` navigates a hierarchy to target `city` data. For more information about binding components and the selector property, see Hierarchical Bindings and Arrays below.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and

Option	Type	Description
		you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example,

Option	Type	Description
		`s:count ?count.`
offset	int	This property can be used to offset the data that is returned by a number of rows.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_ variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options. **Tip** See Hierarchical Bindings and Arrays below for more information about configuring mapping variables and unpacking files with nested objects and arrays.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column. For example, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline`

Option	Type	Description
		`("AIRLINE").` **Note** For FileSource, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern could be written as `?averagePerDay ("average\\/day").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note**

Option	Type	Description
		The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is specified, `02-04-99` is parsed to `02-04-1999`.

Hierarchical Bindings and Arrays

When configuring the mapping variables in a query, the GDI provides syntax for unpacking XML files with nested objects and arrays. One way to express hierarchies in queries is to use brackets ([ ]) to group objects into binding trees. For example, the WHERE clause snippet below organizes mapping variable objects into an `hourly/data` hierarchy by nesting the `?data` patterns inside the `?hourly [ ]` tree:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:FileSource;
        s:url "/mnt/data/xml/weather.xml" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        ?hourly
[
          ?data
[
```

```
            ?time (xsd:long) ;
            ?rainProbability ("precipProbability" xsd:double) ;
            ?temperature (xsd:double) ;
            ?feelsLike ("apparentTemperature" xsd:double) ;
            ?windSpeed (xsd:double) ;
        ] ;
    ] .
  }
}
```

When constructing object binding trees, if you choose to introduce the hierarchy with a variable name that is not an exact match to the source label, include a **selector** property to list the value from the source. For example, in the WHERE clause snippet below, `s:selector` is included to select `eventHeader` in the source as `?event` in the query and `statLocation` as `?location`.

```
WHERE
{
   SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
      ?data a s:FileSource ;
      s:url "/mnt/data/xml/part_1.xml" ;
      ?event
[
         s:selector "eventHeader" ;
           ?eventId (xsd:string) ;
           ?eventName (xsd:string) ;
           ?eventVersion (xsd:string) ;
           ?eventTime (xsd:dateTime) ;
      ] ;
      ?location
[
         s:selector "statLocation" ;
           ?locationId (xsd:string) ;
           ?lineNo (xsd:int) ;
           ?statNo (xsd:int) ;
           ?statId (xsd:int) ;
      ] .
  }
}
```

As an alternative to grouping objects in binding trees, the **selector** property also supports using dot notation to specify paths. For example, the WHERE clause snippet below rewrites the first example query to express the same `hourly/data` hierarchy as a path in the `s:selector` value:

```
WHERE
{
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
    {
      ?data a s:FileSource;
        s:url "/mnt/data/xml/weather.xml" ;
        ?latitude (xsd:double) ;
        ?longitude (xsd:double) ;
        ?timezone (xsd:string) ;
        s:selector: "hourly.data" ;
        ?time (xsd:long) ;
        ?rainProbability ("precipProbability" xsd:double) ;
        ?temperature (xsd:double) ;
        ?feelsLike ("apparentTemperature" xsd:double) ;
        ?windSpeed (xsd:double) .
  }
}
```

Query Examples

The following example query ingests data from an XML file that contains hierarchies.

```
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fmcsa: <http://census.gov/ontologies/FMCSA#>

INSERT {
  GRAPH ${targetGraph} {
  }
}
${usingSources}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
      ?data a s:FileSource ;
        s:url "file:///opt/shared/data/xml/define.xml" ;
```

```
        ?ItemGroupDef [
           ?OID (xsd:string) ;
           ?Name (xsd:string) ;
           ?Repeating (xsd:string) ;
           ?IsReferenceData (xsd:string) ;
           ?Purpose (xsd:string) ;
           ?Label (xsd:string) ;
           ?Structure (xsd:string) ;
           ?DomainKeys (xsd:string) ;
           ?Class (xsd:string) ;
           ?ArchiveLocationID (xsd:string) ;
           ?Comment (xsd:string) ;
           ?ItemRef [
              ?ItemOID (xsd:string) ;
              ?OrderNumber (xsd:int) ;
              ?Mandatory (xsd:string) ;
           ] ;
        ] .
   }
}
```

Querying Parquet and SAS Files

This topic provides details about the structure to use when writing GDI queries to read or ingest data from Parquet and SAS files. It also includes example queries that may be useful as a starting point for writing your own GDI queries.

> **Note**
> Loading parquet files from Amazon S3 is not supported. Amazon does not provide a VFS driver that supports random access reads, which are necessary for reading parquet files.

- Query Syntax
- Query Examples

Query Syntax

The following query syntax shows the structure of a GDI query for Parquet and SAS sources. The clauses, patterns, and placeholders that are links are described below.

```
# PREFIX Clause
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>

# Result Clause
{
   [ GRAPH ${targetGraph} { ]
   triple_patterns
 [ } ]
}
[ ${usingSources} ]

WHERE
{
   # SERVICE Clause: Include the following service call when reading or inserting data.
    SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>

   # View SERVICE Clause: Or use the service call below when constructing a view.
    SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})

    {
      ?data a s:FileSource ;
        s:url "string" ;
        [ s:options [ file_storage_connection_options ] ; ]
        [ s:pattern "string" ; ]
        [ s:maxDepth int ; ]
        [ s:format [ source_format_options ; ] ; ]
        [ s:mimetype "string" ; ]
        [ s:username "string" ; ]
        [ s:password "string" ; ]
```

```
        [ s:timeout int ; ]
        [ s:batching boolean | int ; ]
        [ s:paging [ pagination_options ; ]
        [ s:concurrency int | [ list_of_properties ] ; ]
        [ s:rate int | "string" ; ]
        [ s:locale "string" ; ]
        [ s:sampling int ; ]
        [ s:selector "string" | [ list ] ; ]
        [ s:model "string" ; ]
        [ s:key ("string") ; ]
        [ s:reference [ s:model "string" ; s:using ("string") ]
        [ s:formats [ datatype_formatting_options ] ; ]
        [ s:normalize boolean | [ normalization_rules ] ; ]
        [ s:count ?variable ; ]
        [ s:offset int ; ]
        [ s:limit int ; ]
        # Mapping variables and hierarchical bindings
        ?mapping_variable ( [ "binding" ] [ datatype ] [ "datetime_format" ] ) ;
        ... ;
        .
    # Additional clauses such as BIND, VALUES, FILTER
  }
}
```

Option	Type	Description
PREFIX Clause	N/A	The PREFIX clause declares the standard and custom prefixes for GDI service queries. Generally, queries include the prefixes from the query template (or a subset of them) plus any data-specific declarations.
Result Clause	N/A	The result clause defines the type of SPARQL query to run and the set of results to return, i.e., whether you want to read (SELECT or CONSTRUCT) from the source or ingest the data into Anzo (INSERT).
GRAPH ${targetGraph}	N/A	Include the `GRAPH` keyword and target graph parameter `${targetGraph}` when writing an INSERT query to ingest data into a graphmart. Anzo automatically populates the query with the

Option	Type	Description
		appropriate target URIs when the query runs.
${usingSources}	N/A	Include the source graph parameter `${usingSources}` when writing a "topdown" query that passes values from the data that is in the graphmart to the data source. Anzo automatically populates the query with the appropriate FROM clauses when the query runs. When passing literal values to the remote source, you do not need to include the source graph parameter. The SERVICE Clause description below includes more information about passing input to data sources.
SERVICE Clause	N/A	Include the SERVICE call `SERVICE [ TOPDOWN ] <http://cambridgesemantics.com/services/DataToolkit>` to invoke the GDI service when you are running a SELECT, INSERT, or CONSTRUCT query that is not creating a view. When writing a CONSTRUCT query in a View Step, use the `DataToolkitView` service call, as described below in View SERVICE Clause. Include the optional TOPDOWN keyword when you want to pass input values from the graphmart to the data source. When you include TOPDOWN in the service call, it indicates that the rest of the query produces values to send to the source. In this case, the GDI makes repeated calls to pass in each of the specified values and retrieve the data that is based on those values.
View SERVICE Clause	N/A	When writing a CONSTRUCT query that creates a view of the data (usually in a View Step), include the following SERVICE call: `SERVICE <http://cambridgesemantics.com/services/DataToolkitView>(${targetGraph})`. Using the `DataToolkitView` call optimizes query execution because it tells the GDI to inspect

Option	Type	Description
		the query and determine which filters to push to the data source. It also limits the result set and retrieves only the data that is needed, i.e., the source data is fully mapped but all of the mapped data is not necessarily returned.
url	string	This property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"` or `s:url "gs://shared-files/my-files/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the pattern and/or maxDepth properties.
options	RDF list	If additional connection information needs to be provided to access the file storage system, include the `options` property to list any storage-specific connection parameters. See File Storage Connection Options for information about the supported properties for each storage type.
pattern	string	This property is used to specify a wildcard pattern for matching file names. For example, `s:pattern "common_prefix*.parquet"`. You can include one `s:pattern` property per FileSource. The GDI supports Unix file globbing syntax outside of parentheses. Within parentheses, full Java regular expression language is supported. For example, including `s:pattern "data/**/customer_*.parquet"` tells the GDI to load all files that match the pattern "customer_*.parquet" from any number of subdirectories under the `data` directory. Similarly `s:pattern "(\d+)/transaction_*.xpt"` tells the GDI to load all files that match the pattern "transaction_*.xpt" in all subdirectories.
maxDepth	int	This property can be used to limit the directory traversal depth. By

Option	Type	Description
		default, when `s:url` specifies a directory (and a `s:pattern` that limits that traversal depth is not specified), all subdirectories are processed. To process only the files in the top level directory, set maxDepth to 0 (`s:maxDepth 0`). To process the files in the top level directory plus the first-level subdirectories, set maxDepth to 1 (`s:maxDepth 1`), and so on.
format	RDF list	You can include the `format` property to add parameters that describe the source files. See File Source Format Options for details about the supported parameters.
mimetype	string	This property can be included to specify the MIME type of the data.
username	string	If authentication is required to access the source, include this property to specify the user name.
password	string	This property lists the password for the given username.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
paging	RDF list	This property can be used to configure paging so that the GDI can

Option	Type	Description
		access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`

Option	Type	Description
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate. Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
sampling	int	This property can be used to configure the number of records in the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.

Option	Type	Description
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
normalize	RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
offset	int	This property can be used to offset the data that is returned by a number of rows.

Option	Type	Description
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
mapping_variable	variable	The mapping variables, in `?mapping_variable (["binding"] [datatype] ["datetime_format"])` format, define the triple patterns to output. When the specified `?variable` matches the source column name, the GDI uses the variable as the source data selector. If you specify an alternate variable name, a binding needs to be specified to map the new variable to the source. You also have the option to transform the data using the datatype and datetime_format options.
binding	string	The `binding` is a literal value that binds a ?mapping_variable to a source column. If you specify a ?variable that matches the source column name, then that variable name is the data selector and it is not necessary to specify a binding. If you specify an alternate variable name or there is a hierarchical path to the source column, then the binding is needed to map the new variable to that source column. For example, the following pattern simply binds the source column AIRLINE to the lowercase variable ?airline: `?airline ("AIRLINE")`. **Note** For FileSource, periods (.), forward slashes (/), and brackets ([ ]) are parsed as path notation. Therefore, if a source column name includes any of those characters they must be escaped in the binding. Use two backslashes (\\) as an escape character. For example, if a column name is **average/day**, the variable and binding pattern

Option	Type	Description
		could be written as `?averagePerDay ("average\\/day").`
datatype	URI	The `datatype` is the data type to convert the column to. If you do not specify a data type, the GDI infers the type. The GDI supports the following types: `xsd:int, xsd:long, xsd:float, xsd:double, xsd:boolean, xsd:time, xsd:dateTime, xsd:date, xsd:duration, xsd:dayTimeDuration, xsd:yearMonthDuration, xsd:gMonthDay, xsd:gMonth, xsd:gYearMonth, xsd:anyURI`
datetime_ format	string	This option is used to specify the format to use for date and time data types. The GDI supports Java date and time formats. Specify days as "d," months as "M," and years as "y." For the time, specify "H" for hours, "m" for minutes, and "s" for seconds. For example, `"yyyyMMdd HH:mm:ss"` or `"ddMMMyy"` to display date values such as "01JAN19." **Note** The GDI's default base year is **2000**. If the source data has years with only two digits, such as `02-04-99`, the GDI prepends **20** to the digits. The value `02-04-99` is parsed to `02-04-2099`. To specify an alternate base year to use for two-digit values, you can include the notation `^nnnn (e.g., ^1900)` in the format value. For example, to set the base year to 1900 instead of 2000, use a format value such as `xsd:date "dd-MMM-yy^1900"` or `xsd:date "dd-MMM-yy^1990"`. When one of those values is

Option	Type	Description
		specified, `02-04-99` is parsed to `02-04-1999`.

Query Examples

The following query inserts data from a Parquet file on the shared file system and uses a binding tree to express hierarchical data.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>

INSERT {
   GRAPH ${targetGraph} {
      ?s ?p ?o
  }
}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
    ?data a s:FileSource ;
      s:url "file:///opt/shared/data/parquet/part-c000.snappy.parquet" ;
      s:model "BID" ;
      ?bidid (xsd:string) ;
      ?cur (xsd:string) ;
      ?id (xsd:string) ;
      ?nbr (xsd:long) ;
      ?seatbid [
        ?seat (xsd:string) ;
        ?group (xsd:long) ;
        ?bid [
          ?adid (xsd:string) ;
          ?cid (xsd:string) ;
```

```
          ?crid (xsd:string) ;
          ?dealid (xsd:string) ;
          ?h (xsd:long) ;
          ?w xsd:long ;
          ?bundle (xsd:string) ;
          ?price (xsd:double) ;
        ] ;
     ] .
  }
}
```

The query below reads data from a Parquet file and filters out data by zip code.

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX anzo:  <http://openanzo.org/ontologies/2008/07/Anzo#>
PREFIX zowl:  <http://openanzo.org/ontologies/2009/05/AnzoOwl#>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>

SELECT *
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit>
  {
    ?data a s:FileSource ;
      s:url "file:///opt/shared/data.gov/FMCSA_CENSUS1_2023Nov.parquet" ;
      ?DOT_NUMBER (xsd:string) ;
      ?LEGAL_NAME (xsd:string) ;
      ?CARRIER_OPERATION (xsd:string) ;
      ?HM_FLAG (xsd:string) ;
      ?PC_FLAG (xsd:string) ;
      ?PHY_STREET (xsd:string) ;
      ?PHY_CITY (xsd:string) ;
      ?PHY_STATE (xsd:string) ;
      ?PHY_ZIP (xsd:string) ;
      ?PHY_COUNTRY (xsd:string) .
    FILTER(?PHY_ZIP = "78735" || ?PHY_ZIP = "77040" || ?PHY_ZIP = "78705")
  }
}
```

File Source Format Options

For file sources, you can include the **format** property to list additional parameters that describe the source. The supported format parameters are described below.

```
s:format [
   s:delimiter "string" ;
   s:headers boolean ;
   s:columns "string" ;
   s:start int ;
   s:skip int ;
   s:comment "string" ;
   s:quote "string" ;
   s:escape "string" ;
   s:maxColumns int ;
   s:segment boolean ;
] ;
```

Option	Type	Description
delimiter	string	This property specifies the string that is used to delimit columns in the file(s). For example, `s:delimiter "|"`.
headers	boolean	This property indicates whether or not the file(s) include headers. By default the headers value is true (`s:headers true`). For files that do not have headers, specify `s:headers false`.
columns	string	If you want the GDI to target only certain columns in the source file(s), you can include the **columns** property to list the names of columns to include. The value is a single string that is a comma-separated list. For example, `s:columns "employee_id, name, address, start date, title"`.
start	int	If the file includes headers that take up more than one row, include the `start` property to specify the row number where the

Option	Type	Description
		data starts to exclude headers. For example, `s:start 8`.
skip	int	This property can be used to specify the number of rows/records to skip before reading or ingesting the file(s). By default, skip is set to 0 (`s:skip 0`).
comment	string	This property specifies the string that is used as the comment character in the file(s). The comment value is set to `#` by default (`s:comment "#"`).
quote	string	This property is used to specify the string that is used as the quote character.
escape	string	This property is used to specify the escape string that is used in the file(s). For example, `s:escape "\"`.
maxColumns	int	This property can be used to set a limit on the maximum number of columns to read or ingest. The maxColumns property is set to `-1` (unlimited) by default (`s:maxColumns -1`).
segment	boolean	This property indicates whether or not the file(s) can be segmented (partitioned). The default value is `s:segment true`. If you have CSV files that contain embedded new lines, include `s:segment false` in the query as those files cannot be segmented.

File Storage Connection Options

If you are querying a **FileSource** and additional connection information needs to be provided to access the file storage system, include the **options** property in the query and define the necessary

storage-specific connection parameters. The parameters that the GDI supports for each type of storage system are pulled directly from the Java API for that system. The supported properties for each storage type are listed below.

- Amazon S3
- FTP & FTPS
- Google Cloud Storage
- HDFS
- SFTP
- WebDAV

Amazon S3

> **Note**
> Loading parquet files from S3 is not supported. Amazon does not provide a VFS driver that supports random access reads, which are necessary for reading parquet files.

```
s:options [
   s:accessKey "string" ;
   s:region "string" ;
   s:secretKey "string" ;
   s:serviceName "string" ;
   s:sessionToken "string" ;
   s:createBucket boolean ;
   s:disableChunkedEncoding boolean ;
   s:serverSideEncryption boolean ;
   s:useHttps boolean ;
] ;
```

Option	Type	Description
accessKey	string	The `accessKey` property can be included to specify the access key.

Option	Type	Description
region	string	The `region` property can be included to specify the region.
secretKey	string	The `secretKey` property can be included to specify the secret key.
serviceName	string	For connections to AWS service endpoints, the `serviceName` property can be included to specify the service name.
sessionToken	string	The `sessionToken` property can be included to specify the session token.
createBucket	boolean	Refer to the S3 API documentation.
disableChunkedEncoding	boolean	For increased performance, Amazon S3 requests use chunked encoding by default. To disable chunked encoding, you can include `s:disableChunkedEncoding true` in the query.
serverSideEncryption	boolean	Refer to the S3 API documentation.
useHttps	boolean	Refer to the S3 API documentation.

FTP & FTPS

```
s:options [
   s:autodetectUtf8 boolean ;
   s:connectTimeout int ;
   s:controlEncoding "string" ;
   s:dataTimeout int ;
```

```
   s:defaultDateFormat "string" ;
   s:entryParser "string" ;
   s:fileType "string" ;
   s:passiveMode boolean ;
   s:proxy "string" ;
   s:recentDateFormat "string" ;
   s:remoteVerification boolean ;
   s:serverLanguageCode "string" ;
   s:serverTimeZoneId "string" ;
   s:shortMonthNames "string" ;
   s:socketTimeout int ;
   s:userDirIsRoot boolean ;
   s:dataChannelProtectionLevel "string" ;
   s:ftpsMode "string" ;
   s:keyManager "string" ;
   s:trustManager "string" ;
] ;
```

Option	Type	Description
autodetectUtf8	boolean	For FTP connections, the `autodetectUtf8` property can be included to indicate whether the FTP server is set to UTF-8 mode or Auto-detect encoding.
connectTimeout	int	For FTP connections, you can include the `connectTimeout` property to specify the maximum number of seconds to hold a connection before timing out.
controlEncoding	string	Refer to the FTP API documentation.
dataTimeout	int	For FTP connections, you can include the `dataTimeout` property to specify the maximum number of seconds to transfer data before timing out.

Option	Type	Description
defaultDateFormat	string	Refer to the FTP API documentation.
entryParser	string	Refer to the FTP API documentation.
fileType	string	Refer to the FTP API documentation.
passiveMode	boolean	For FTP connections, the `passiveMode` property can be included to indicate whether the data transfer mode is passive or active. If you use passive mode, set passiveMode to true (`s:passiveMode true`).
proxy	string	If you are using an FTP proxy, include the `proxy` property to specify the proxy connection details.
recentDateFormat	string	Refer to the FTP API documentation.
remoteVerification	boolean	For FTP connections, the `remoteVerification` property can be included to indicate whether remote authentication is enabled. If you use remote authentication, set remoteVerification to true (`s:remoteVerification true`).
serverLanguageCode	string	If the FTP server language is not set to English, include the `serverLanguageCode` property to specify the language code for the server. For example, `s:serverLanguageCode "ES"`.
serverTimeZoneId	string	For FTP connections, the `serverTimeZoneId` property can be included

Option	Type	Description
		to specify the timezone ID for the server.
shortMonthNames	string	Refer to the FTP API documentation.
socketTimeout	int	For FTP connections, you can include the `socketTimeout` property to specify the maximum number of seconds to transfer data before timing out.
userDirIsRoot	boolean	Refer to the FTP API documentation.
dataChannelProtectionLevel	string	For FTPS connections, the `dataChannelProtectionLevel` property specifies the Data Channel Protection Level for the server.
ftpsMode	string	For FTPS connections, the `ftpsMode` property specifies whether the FTPS is in implicit or explicit mode.
keyManager	string	For FTPS connections, the `keyManager` property specifies the KeyManager value for making an SSL connection to the server.
trustManager	string	For FTPS connections, the `trustManager` property specifies the TrustManager value for the SSL connection to the server.

Google Cloud Storage

```
s:options [
   s:serviceAccountKey "string" ;
] ;
```

Option	Type	Description
serviceAccountKey	string	For connections to GCS, the `serviceAccountKey` property can be included to specify the key for the service account.

HDFS

```
s:options [
   s:configName "string" ;
   s:configPath "string" ;
   s:configURL "string" ;
] ;
```

Option	Type	Description
configName	string	For connections to HDFS, the `configName` property can be included to specify the name of the configuration file to read.
configPath	string	For connections to HDFS, the `configPath` property can be included to list the path to the specified configuration file.
configURL	string	Refer to the HDFS API documentation.

SFTP

```
s:options [
   s:compression "string" ;
   s:configRepository "string" ;
   s:fileNameEncoding "string" ;
   s:identityProvider "string" ;
   s:identityRepositoryFactory "string" ;
   s:keyExchangeAlgorithm "string" ;
   s:knownHosts "string" ;
   s:loadOpenSSHConfig boolean ;
   s:preferredAuthentications "string" ;
   s:sessionTimeout int ;
   s:strictHostKeyChecking "string" ;
   s:userInfo "string" ;
] ;
```

Option	Type	Description
compression	string	Refer to the SFTP API documentation.
configRepository	string	Refer to the SFTP API documentation.
fileNameEncoding	string	Refer to the SFTP API documentation.
identityProvider	string	Refer to the SFTP API documentation.
identityRepositoryFactory	string	Refer to the SFTP API documentation.
keyExchangeAlgorithm	string	For SFTP connections, you can include the `keyExchangeAlgorithm` property to specify the key exchange algorithm to use.
knownHosts	string	Refer to the SFTP API documentation.

Option	Type	Description
loadOpenSSHConfig	boolean	For SFTP connections, the `loadOpenSSHConfig` property indicates whether to read the `~/.ssh/config` file.
preferredAuthentications	string	For SFTP connections, the `preferredAuthentications` property can be included to specify the authentication order to use.
sessionTimeout	int	For SFTP connections, you can include the `sessionTimeout` property to specify the maximum number of seconds to leave the session open before timing out.
strictHostKeyChecking	string	For SFTP connections, you can include the `strictHostKeyChecking` property to specify how host keys are checked.
userInfo	string	Refer to the SFTP API documentation.

WebDAV

```
s:options [
   s:creatorName "string" ;
   s:versioning boolean ;
] ;
```

Option	Type	Description
creatorName	string	For WebDAV connections, the `creatorName` property can be included to add a description of the creator of the resource.

Option	Type	Description
versioning	boolean	Refer to the WebDAV API documentation.

GDI Property Reference

This topic describes the Graph Data Interface (GDI) properties that are available to use in queries. The first section describes the options that are available regardless of data source type, and the second section describes the source-specific options.

- Universal Properties
- DbSource Properties
- FileSource Properties
- HttpSource Properties
- ElasticSource Properties

Universal Properties

The table below lists the properties that are valid in queries against all data source types.

Option	Type	Description
batching	boolean or int	This property can be used to disable batching, or it can be used to change the default the batch size. By default, batching is set to 5000 (`s:batching 5000`). To disable batching, you can include `s:batching false` in the query. Typically users do not change the batching size. However, it can be useful to control the batch size when performing updates. To configure the size, include `s:batching int` in the query. For example, `s:batching 3000`.
concurrency	int or RDF list	This property can be included to configure the maximum level of concurrency for the query. The value can be an integer, such as `s:concurrency 8`. If the value is an integer, it configures a maximum limit on the number of slices that can execute the query. For finer-grained control over the number of nodes and slices to use, concurrency can also be included as an object with `limit`, `nodes`, and/or `executorsPerNode` properties. For

Option	Type	Description

		example, the following object configures a concurrency model that allows a maximum of 24 executors distributed across 4 nodes with 8 executors per node:
		`s:concurrency [`
		`  s:limit 24 ;`
		`  s:nodes 4 ;`
		`  s:executorsPerNode 8 ;`
		`] ;`

count	variable	If you want to turn the query into a COUNT query, you can include this property with a `?variable` to perform a count. For example, `s:count ?count`.
errors	boolean	Controls whether the GDI ignores errors (such as query or file errors) or stops processing the query when an error is encountered. This property is set to `true` by default (`s:errors true`). Processing stops when an error is encountered. To ignore errors, you can include `s:errors false`.
formats	RDF list	To give users control over the data types that are used when coercing strings to other types, this property can be included in GDI queries to define the desired types. In addition, it can be used to describe the formats of date and time values in the source to ensure that they are recognized and parsed to the appropriate date, time, and/or dateTime values. For details about the `formats` property, see Data Type Formatting Options.
key	string	This property can be used to define the primary key column for the source file or table. This column is leveraged in a resource template for the instances that are created from the source. For

Option	Type	Description
		example, `s:key ("EMPLOYEE_ID")`. For more information about `key`, see Data Linking Options.
limit	int	You can include this property to limit the number of results that are returned. `s:limit` maps to the SPARQL LIMIT clause.
locale	string	This property can be used to specify the locale to use when parsing locale-dependent data such as numbers, dates, and times.
model	string	This property defines the class (or table) name for the type of data that is generated from the specified data source. For example, `s:model "employees"`. Model is optional when querying a single source. If your query targets multiple sources, however, and you want to define resource templates (primary keys) and object properties (foreign keys), you must specify the model value for each source.
normalize	boolean and/or RDF list	To give users control over the labels and URIs that are generated, the GDI offers several options for normalizing the model and/or the fields that are created from the specified data source(s). For details about the `normalize` property, see Model Normalization Options.
offset	int	This property can be used to offset the data that is returned by a number of rows.
paging	RDF list	This property can be used to configure paging so that the GDI can access large amounts of data across a number of smaller requests. For details about the `paging` property, see Paginating Requests.

Option	Type	Description
password	string	This property lists the password for the given username.

rate	int or string	This property can be included to control the frequency with which a request is sent to the source. The limit applies to the number of requests a single slice can make. If you specify an integer for the rate, then the value is treated as the maximum number of requests to issue per minute. If you specify a string, you have more flexibility in configuring the rate. The sample values below show the types of values that are supported:
		`s:rate "90/minute" ;`
		`s:rate "90 per minute" ;`
		`s:rate "200000 every week" ;`
		`s:rate "10000 every 6 hours" ;`
		To enforce the rate limit, the GDI introduces a sleep between requests that is equal to the rate delay. The more executing slices, the longer the rate delay needs to be to enforce the limit in aggregate.
		Given the example of `s:rate "90/minute"`, the GDI would optimize the concurrency and only use 1 slice for execution with a rate delay of 666ms between requests. If `s:rate "240/minute"`, the GDI would use 3 executors with a rate delay of 750ms between requests.

reference	RDF list	This property can be used to specify a foreign key column. The reference property is an RDF list that includes the `model` property to list the target table and a `using` property that defines the foreign key column. For more information about `reference`, see Data Linking Options.
sampling	int	This property can be used to configure the number of records in

Option	Type	Description
		the source to examine for data type inferencing.
selector	string or RDF list	This property can be used as a binding component to identify the path to the source objects. For example, `s:selector "Sales.SalesOrderHeader"` targets the SalesOrderHeader table in the Sales schema. For more information about binding components and the selector property, see Using Binding Trees and Selector Paths.
strict	boolean	This property can be used to force the GDI to limit the data to strictly what is stated in the query. For example, when ingesting data from a CSV file, you can include `s:strict true` on the `s:FileSource` to ensure that the GDI only ingests columns for which a variable binding exists in the query. In addition, this property can be included in `s:formats` to control the automatic data type conversion feature (as described in Data Type Formatting Options). The default value is `false`.
timeout	int	This property can be used to specify the timeout (in milliseconds) to use for requests against the source. For example, `s:timeout 5000` configures a 5 second timeout.
url	string	This property specifies the URL for the data source, such as the database URL, Elasticsearch URL, or HTTP endpoint URL. For file-based sources, the `url` property specifies the file system location of the source file or directory of files. When specifying a directory (such as `s:url "/opt/shared-files/loads/"`), the GDI loads all of the file formats it recognizes. To specify a directory but limit the number or type of files that are read, you can include the pattern and/or maxDepth properties. **Important**

Option	Type	Description

		For security, it is a best practice to reference connection information (such as the url, username, and password) from a Query Context so that the sensitive details are abstracted from any requests. In addition, using a Query Context makes connection details reusable across queries. See Using Query Contexts for more information. For example, the triple patterns below reference keys from a Query Context:
		`?data a s:DbSource ;`
		`s:url "{{@db.eca4bf...3ff9a.url}}" ;`
		`s:username "{{@db.eca4bf...3ff9a.user}}" ;`
		`s:password "{{@db.eca4bf...3ff9a.password}}" ;`

username	string	If authentication is required to access the source, include this property to specify the user name.

DbSource Properties

The table below lists the properties that are available for queries against database data sources. For more information about database sources, see Querying a Database Source.

Option	Type	Description
database	string	This property can be used to specify the database to target in the source if the database is not listed in the `s:url` or `s:selector` strings.
driver	string	This property can be included to specify the JDBC driver to

Option	Type	Description
		use.
orderBy	string, variable, list	You can include this property to order the result set by a field name, a bound variable, or a list of names or bound variables.
maxConnections	int	This property can be used to set a limit on the maximum number of active connections to the source. For example, `s:maxConnections 16` sets the limit to 16 connections. The default value is `10`.
partitionBy	string, variable, list	The GDI attempts to partition queries automatically across the available cores (slices) in AnzoGraph. To determine how to partition the query, the GDI uses metadata from the source database. It looks for any column in an index, preferring the primary key column if it is interpolable. However, it only considers the first column in any index on the table. After determining the partition column, the GDI does a MIN/MAX on the column as well as a basic sizing query. To specify which column or columns the GDI should partition on, you can include the `partitionBy` property in the query. The property supports a list of source field names, bound variables, or the object `s:auto`, which forces the GDI to partition the data when the source does not define partitioning metadata.

property	RDF list	This property can be included to list any JDBC driver-specific connection properties. To incorporate `property`, use the following syntax:
		`s:property [`
		`s:name "custom_driver_property_name" ;`

Option	Type	Description
		`s:value "custom_value"` `]`
query	string	If you want to access the source data by running an SQL query, you can include this property to specify the query string to run. The language does not have to be SQL if the source supports another language. However, some GDI features where the query is dynamically altered may not work with a non-SQL language. Including `{{?variable}}` substitutions is supported within `s:query` strings. **Important** If you include `s:query`, you must also specify table and partitionBy. Specify the table name in `s:table` and the column to partition the table on in `s:partitionBy`. If the table and partition column are not specified, the GDI will not partition the query and query execution may fail or perform very poorly.
schema	string	This property can be included to specify the target schema to query. If you include `s:schema "schema_name"` without specifying `s:table` (described below) or `s:query`, all tables in the schema are queried.
table	string	This property can be included to specify the target table or tables for the query.

FileSource Properties

The table below lists the properties that are available for queries against file-based data sources. For more information about file sources, see Querying a File Source.

Option	Type	Description
format	RDF list	You can include the `format` property to add parameters that describe the source files. See File Source Format Options for details about the supported parameters.
maxDepth	int	This property can be used to limit the directory traversal depth. By default, when `s:url` specifies a directory (and a `s:pattern` that limits that traversal depth is not specified), all subdirectories are processed. To process only the files in the top level directory, set maxDepth to 0 (`s:maxDepth 0`). To process the files in the top level directory plus the first-level subdirectories, set maxDepth to 1 (`s:maxDepth 1`), and so on.
mimetype	string	This property can be included to specify the MIME type of the data. If you are querying TSV files that do not have a .tsv file extension, include the `mimetype` property with a value of `text/tsv` (`s:mimetype "text/tsv"`).
options	RDF list	If additional connection information needs to be provided to access the file storage system, include the `options` property to list any storage-specific connection parameters. See File Storage Connection Options for information about the supported properties for each storage type.
pattern	string	This property can be used to specify a wildcard pattern for matching file names. For example, `s:pattern "common_prefix*.csv"`. You can include one `s:pattern` property per FileSource. The GDI supports Unix file globbing syntax outside of parentheses. Within parentheses, full Java regular expression language is supported. For

Option	Type	Description
		example, including `s:pattern "data/**/customer_*.csv"` tells the GDI to load all files that match the pattern "customer_*.csv" from any number of subdirectories under the `data` directory. Similarly `s:pattern "(\d+)/transaction_*.csv"` tells the GDI to load all files that match the pattern "transaction_*.csv" in all subdirectories.

HttpSource Properties

The table below lists the properties that are available for queries against HTTP data sources. For more information about HTTP sources, see Querying an HTTP Source.

Option	Type	Description
authorization	RDF list	This property specifies the type of authorization to use and the values for authentication. The options are BearerToken, AWSSignature, or BasicAuth. `s:authorization [ a s:BearerToken | s:AWSSignature | s:BasicAuth ]`
AWSSignature	RDF list	For authorization to AWS service endpoints, specify this property and include the appropriate authentication properties from the list below: • **accessKey**: Include this property to specify the AWS access key. • **region**: Include this property to specify the AWS region. • **secretKey**: Include this property to specify the AWS secret key. • **serviceName**: Include this property to specify the AWS service name.

Option	Type	Description
		• **sessionToken**: Include this property to specify the AWS session token. `s:authorization [ a s:AWSSignature ; s:accessKey "string" ; s:region "string" ; s:secretKey "string" ; s:serviceName "string" ; s:sessionToken "string" ; ]`
BasicAuth	RDF list	Specify this property when basic authentication is used, and include the username and password properties. `s:authorization [ a s:BasicAuth ; s:username "string" ; s:password "string" ; ]`
BearerToken	string	Specify this property when a bearer token is used for authentication, and include the token property. `s:authorization [ a s:BearerToken ; s:token "string" ]`
content	string or RDF list	This property can be included to send content to the source in the body of the request. For example, `content` can be a SPARQL query, JSON arrays, or a list of key-value pairs. Content can also be configured with an inline object (blank node) that gets translated to JSON. For more information, see Mapping the Content Property to JSON.

Option	Type	Description
contentType	string	Include this property to specify the content type of the body of the request. For example, `s:contentType "application/sparql-query"` or `s:contentType "application/json"`.
encoding	string	When targeting a file, you can include this property to specify the character encoding used by the file. The default value is `s:encoding "utf8"`.
form	RDF list	To send data to the HTTP endpoint, you can use this property to post the data. Form is a list of name-value pairs. When including `s:form`, you must also include `s:contentType "multipart/form-data"`. The GDI sends the form object as an `application/x-www-form-urlencoded` string that contains the specified parameters. See Example form Parameter Usage in Querying an HTTP Source for sample usage.
format	RDF list	If the data is file-based, you can include the `format` property to add parameters that describe the source. See File Source Format Options for details about the supported parameters.
header	RDF list	You can use this property to specify name-value pairs to include as headers in the request. For example: `s:header [ s:name "Accept" ; s:value "application/json" ]` If you are creating a view, you can include variables in the `s:header` list. When another query is run against a view with variables, that query can map the variables through the view by including predicates in the CONSTRUCT clause.

Option	Type	Description
method	string	You can include this property to specify the HTTP method. For example, `s:method "GET"` or `s:method "POST"`.
mimetype	string	You can include this property to specify the MIME type of the source. For example, `s:mimetype "text/html"`.
orderBy	string, variable, list	You can include this property to order the result set by a field name, a bound variable, or a list of names or bound variables.

parameter	RDF list	You can include this property to list any URL parameters as name-value pairs. For example, the `s:parameter` property below adds `format` to return results in CSV format and the `named-graph-uri` parameter to target a specific layer in a graphmart.
		`s:parameter [ s:name "format" ; s:value "csv" ] ,`
		`[ s:name "named-graph-uri" ;`
		`s:value "http://cambridgesemantics.com/Layer/d541..." ]`
		If you are creating a view, you can include variables in the `s:parameter` list. When another query is run against a view with variables, that query can map the variables through the view by including predicates in the CONSTRUCT clause.

partitionBy	string, variable, list	The GDI attempts to partition queries automatically across the available cores (slices) in AnzoGraph. To determine how to partition the query, the GDI uses metadata from the source. It looks for any column in an index, preferring the primary key

Option	Type	Description
		column if it is interpolable. However, it only considers the first column in any index on the table. After determining the partition column, the GDI does a MIN/MAX on the column as well as a basic sizing query. To specify which column or columns the GDI should partition on, you can include the `partitionBy` property in the query. The property supports a list of source field names, bound variables, or the object `s:auto`, which forces the GDI to partition the data when the source does not define partitioning metadata.
proxy	string or RDF list	Include this property to specify proxy information if a proxy is used. The value can be a string, such as `s:proxy "host_url:port_number"`, or an RDF list that includes `host` and `port` properties, such as `s:proxy [ s:host "host_url" ; s:port port_number ]`.
trust	string	Include this property to set the level of trust for the source's SSL certificate. The value can be either `"system"` or `"all"`.

ElasticSource Properties

The table below lists the properties that are available for queries against Elasticsearch data sources. For more information about Elasticsearch sources, see Querying an Elasticsearch Source.

Option	Type	Description
aggregations	object	You can include this property to calculate aggregations over the specified bindings. For information about aggregations, see Aggregations in the Elasticsearch documentation.
config	string	To enable you to use explicit mappings, you can include this property to specify the URL to the index configuration file to employ. For example, `es:config`

Option	Type	Description
		`"/opt/shared/elastic/mapping.json"`.
document	string	This property lists the document(s) to search.
field	string or variable	This property defines the field to operate on. The value can be a string or bound variable.
highlight	RDF list	You can include this property to define how results are highlighted. For information about the available properties, see Highlighting Elasticsearch Results.
html	boolean	This property controls whether to output HTML for highlighted results. Defaults to `true`.
index	string	This property can be included to specify the indexes to search. Specify multiple indexes in a comma-separated list. For example, `es:index "projectA_mar", "projectA_apr" ;`.
minScore	float	This property defines the minimum score for matching documents. Documents with a lower score are not included in the search results.
query	string or RDF list	This property defines the query to execute. The value can be a string or a query object that maps to the Elasticsearch Query DSL. To generate the final query, the GDI combines `es:query` with any filters it can push to the Elasticsearch DSL. For more information about the `query` property and mapping Elasticsearch filters to SPARQL FILTER clauses, see Query DSL and Filter Mapping.
routing	string	This property can be included to route a document to a specific

Option	Type	Description
		shard or to limit the search to a particular shard.
searchAfter	RDF list	You can include this property to define the key values to start searching from.
size	int	This property maps to the `size` parameter in the Elasticsearch Search API and configures the batch size or maximum number of hits to return in a single call. Defaults to `10` and typically does not need to be changed.
source	boolean or RDF list	This property can be included to specify the source data to include in results. The value can be a boolean, list of fields, or a list of variable bindings. When `true`, all source data is returned. When `false`, no source data is returned.
url	string	The Elasticsearch endpoint URL.

Onboard Unstructured Data

Anzo processes unstructured data using an Anzo Distributed Unstructured cluster and Elasticsearch. Configurable text analytics and natural language processing (NLP) pipelines find and extract data and convert it to the graph data model. Anzo can process all common file types such as Office documents, PDFs, web pages, and email messages, and can analyze text within Excel, databases, and knowledgebases, or XML columns, properties, and fields. Anzo finds, analyzes, extracts, and ingests concepts, entities, sentiment, topics, classifications, events, facts, and thousands of types of relationships.

In this section:

Unstructured Onboarding Process Overview 356

Creating an Unstructured Pipeline 358

Running an Unstructured Pipeline 379

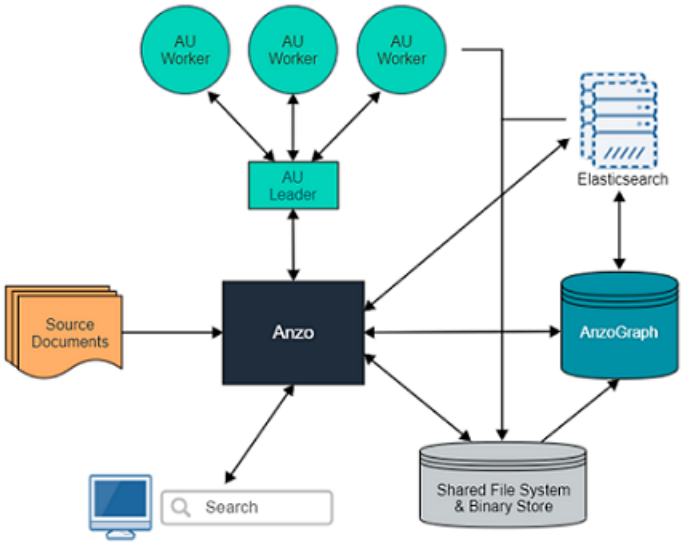
Pipeline Settings Reference 381

Annotator Settings Reference 387

Unstructured Onboarding Process Overview

Anzo onboards unstructured data through pipelines that run in a distributed environment where a cluster of worker nodes process the incoming documents and generate output artifacts. This topic provides an overview of the Anzo Distributed Unstructured (DU) pipeline process and infrastructure.

The diagram below provides a high level overview of the Anzo platform architecture with integration of DU and Elasticsearch. The description below the diagram describes the unstructured data onboarding process and resulting artifacts.



When an unstructured pipeline is run, a crawler service streams data to a pipeline service. The pipeline service reads the stream of files and constructs the appropriate request payloads—one request per document to process. Anzo sends the requests to the DU leader instance, and the leader queues the requests and distributes them to the worker instances to process in parallel. When each worker processes a document, it creates a temporary output artifact on the shared file system. The artifact includes the following items:

- An RDF file that describes the text annotations and general metadata about the processed document.
- A binary store artifact for Anzo.
- A JSON artifact that contains a reference to the extracted text of the document. Elasticsearch uses this artifact to generate the document index.

When the DU workers have processed all of the documents, Anzo completes the following post-processing steps:

- Consolidate the RDF artifacts from the workers and create a file-based linked data set (FLDS) for loading to AnzoGraph.
- Read the JSON artifacts and instruct the Elasticsearch server to build an index with the text extracted from the documents. A snaphsot of the index is saved on the file system with the FLDS. Any time a graphmart that includes that FLDS is loaded to an AnzoGraph instance, Anzo loads the corresponding snapshot into the Elasticsearch server that is associated with the AnzoGraph connection.

When the post-processing is finished, the pipeline service finalizes the FLDS metadata to store in its catalog. The new unstructured data set becomes available in the Datasets catalog, and it can be added to a graphmart and loaded to AnzoGraph for use in Hi-Res Analytics dashboards.

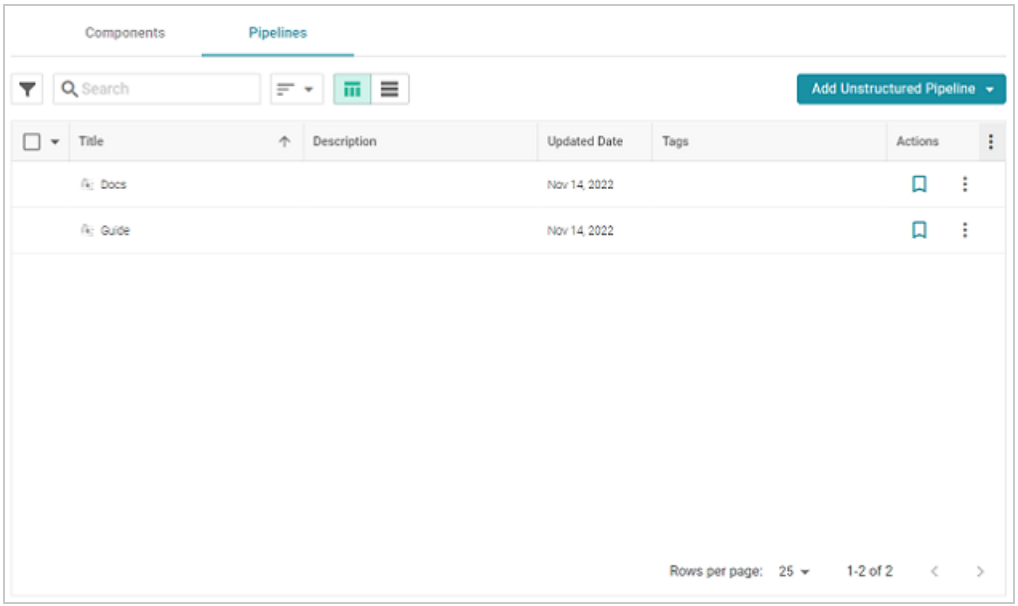
Creating an Unstructured Pipeline

Follow the instructions below to create and run a new unstructured pipeline.

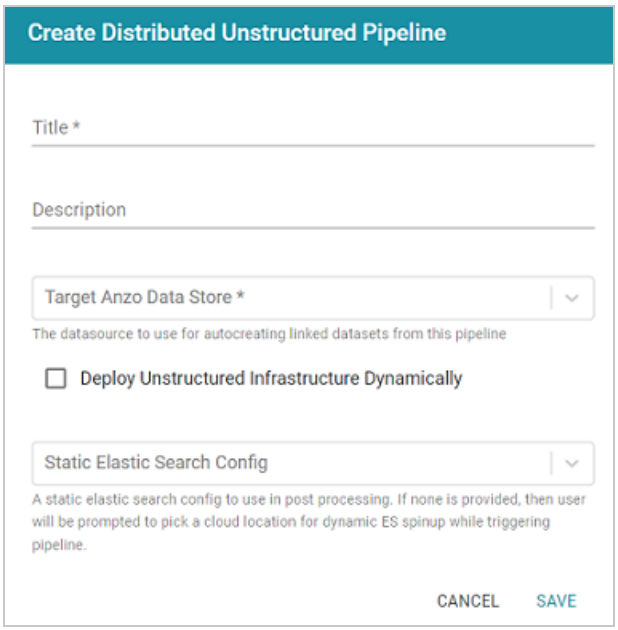
1. Create the Pipeline
2. Add Crawlers to the Pipeline
3. Add Annotators to the Pipeline
4. Run the Pipeline

Create the Pipeline

1. In the Anzo application, expand the **Onboard** menu and click **Unstructured Data**. Anzo displays the Pipelines screen, which lists any existing unstructured pipelines. For example:



2. Click the **Add Unstructured Pipeline** button and select **Distributed Unstructured Pipeline**. Anzo opens the Create Distributed Unstructured Pipeline dialog box. For example:

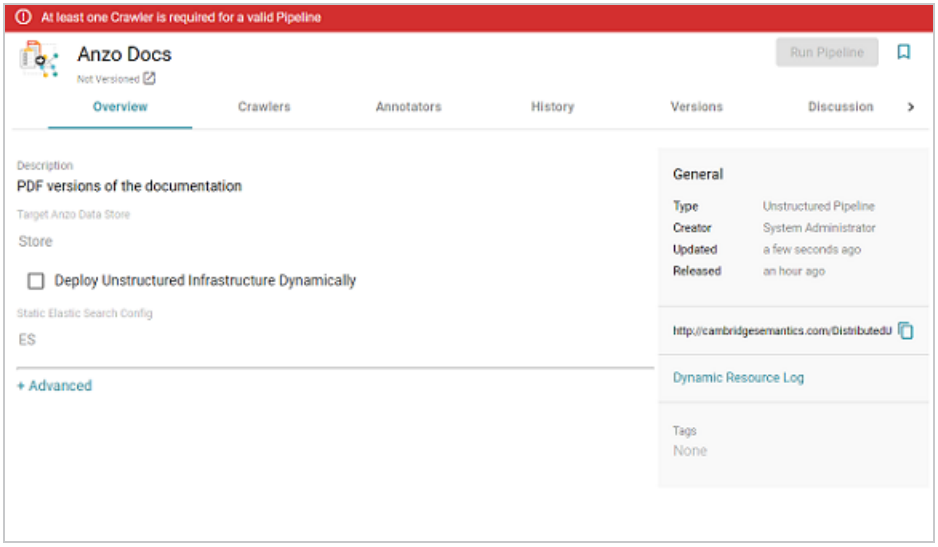


3. In the **Title** field, type a name for the pipeline.

> **Note**
> The title serves as a key to identify this pipeline and its corpus in multiple contexts. Specify a title that is unique and stable. The pipeline's corpus dataset name is derived from this value.

4. Type an optional description for the pipeline in the **Description** field.
5. If necessary, click the **Target Anzo Data Store** field and select the Anzo Data Store for this pipeline.
6. If the environment is configured for dynamic Kubernetes-based deployments of the infrastructure, select the **Deploy Unstructured Infrastructure Dynamically** checkbox and leave the **Static Elasticsearch Config** field blank.
7. If necessary, click the **Static Elasticsearch Config** field and select the Elasticsearch connection to use for this pipeline. If you use dynamic deployments to deploy Elasticsearch instances on-demand, leave this field blank. Anzo prompts the user to choose a Cloud Location when the pipeline is run.

8. Click **Save** to create the pipeline. Anzo displays the pipeline Overview screen. For example:



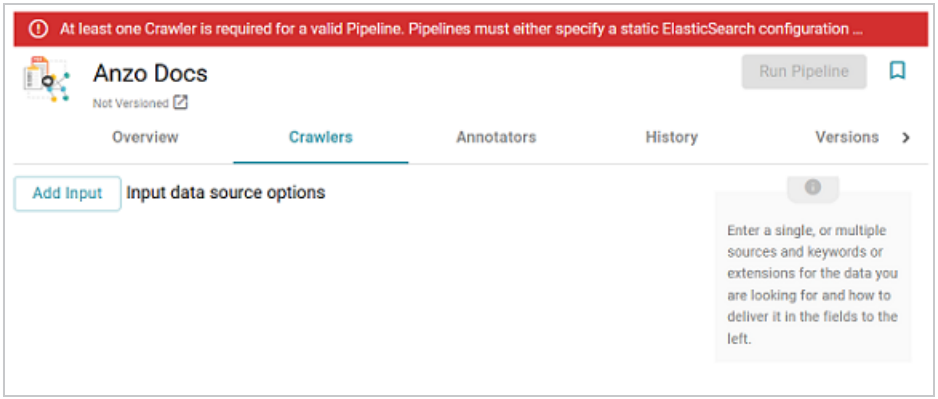
> **Note**
> A pipeline saves automatically and constantly undergoes validation to make sure that it is valid based on the current configuration. Anzo displays validation issues in red on the top of the screen. The warnings will disappear as you add components to the pipeline.

9. If necessary, click **Advanced** to configure the advanced pipeline settings. For details about the advanced settings, see Pipeline Settings Reference.
10. Next, follow the instructions in Add Crawlers to the Pipeline to add one or more crawlers to the pipeline.

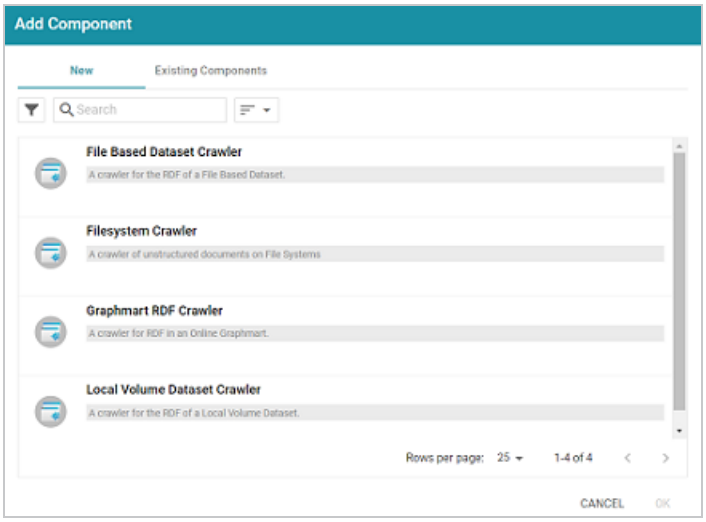
Add Crawlers to the Pipeline

After creating a pipeline, the next step is to add one or more crawlers. Crawlers determine what text to process.

1. In the pipeline, click the **Crawlers** tab.



2. Next, click the **Add Input** button. Anzo displays the Add Component dialog box. The **New** tab is selected and lists all available crawlers. The **Existing Components** tab lists crawlers that have been previously configured for other pipelines.

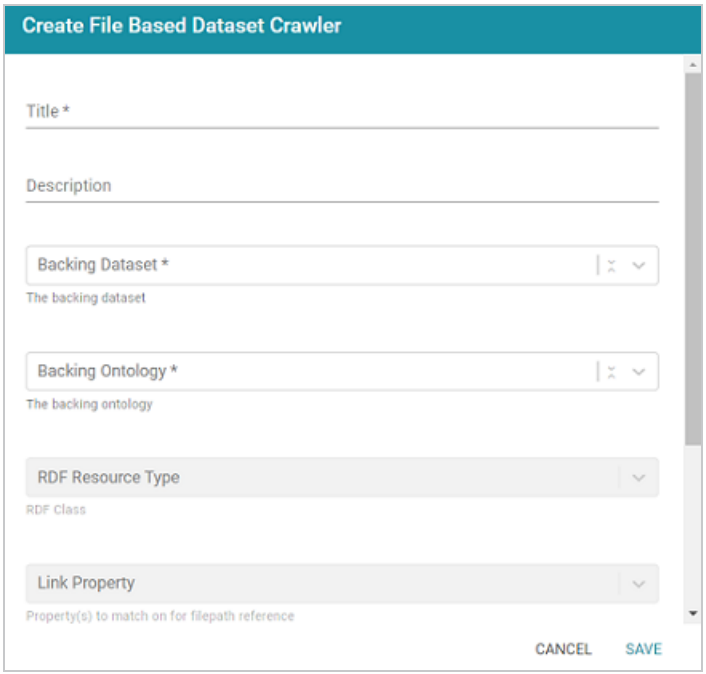


3. To add a new crawler, select the crawler. To add an existing crawler, click the **Existing Components** tab and select a crawler. The list below describes each of the crawlers:
   - **File Based Dataset Crawler**: Include this crawler to process data from a file-based linked data set (FLDS) on a file store.
   - **Filesystem Crawler**: Include this crawler to process documents, such as email messages, PDF, XML, PowerPoint, Excel, OneNote, or Word files, and images, that are available on a file store.

- **Graphmart RDF Crawler**: Include this crawler to process RDF in an online graphmart or specific data layer.
- **Local Volume Dataset Crawler**: Include this crawler to process RDF data that is stored as a linked data set (LDS) in an Anzo journal.

4. After selecting a crawler, click **OK**. Anzo opens the Create dialog box for that crawler so that you can configure it. Click a crawler name in the list below to view the details for that component:
   - File Based Dataset Crawler
   - Filesystem Crawler
   - Graphmart RDF Crawler
   - Local Volume Dataset Crawler

   File Based Dataset Crawler



   - **Title**: Required field that specifies the unique name for this crawler.
   - **Description**: Optional field that provides a description of this crawler.

- **Backing Dataset**: Required field that specifies the Anzo dataset to crawl.
- **Backing Ontology**: Required field that specifies the model for the dataset.
- **RDF Resource Type**: Required field that specifies the resource type or class of data to target with this crawler.
- **Link Property**: Optional field that specifies any link properties to crawl. A link property is a property whose value identifies the location of a linked document. When linked properties are specified, the crawler will crawl the linked documents. For example, in the triples below, **fileLocation** is a link property:

```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://fileLocation> "/path/to/file.pdf" .
```

> **Note**
> In typical use cases, this crawler is configured to define either a Link Property or a Content Property but not both.

- **Content Property**: Optional field that identifies any content properties to crawl. A content property is a property whose value is a string literal and you want the crawler to crawl and annotate those strings. For example, in the triples below, **longDescription** is a content property:

```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://longDescription> "this is some interesting, likely long,
unstructured text
with a lot of information, and I want it to be annotated" .
```

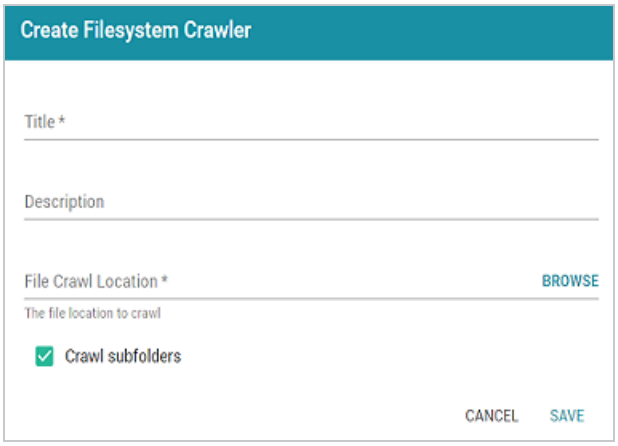
- **Base Path Connection**: Required field whose value depends on whether you specified a Link Property or a Content Property:
  - If a **Link Property** was specified, the Base Path Connection is the base path to use for resolving relative file paths in the Link Property values. For example, using the example triples:

```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata"
;
<urn://fileLocation> "/path/to/file.pdf" .
```

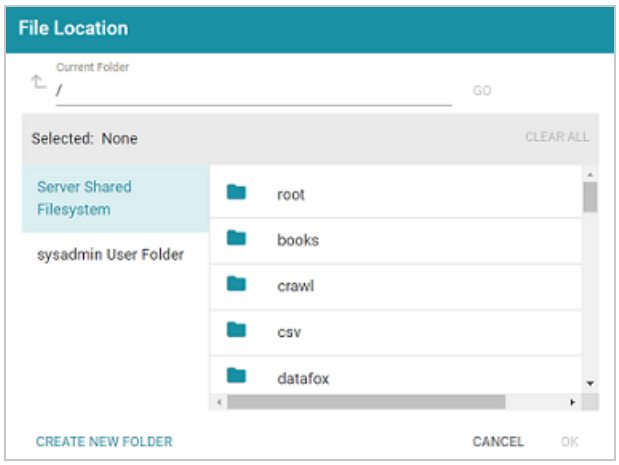
The `<urn://fileLocation>` value of `/path/to/file.pdf` could be a relative path to a location like `s3://location/bucket/path/to/file.pdf` or `/opt/anzoshare/data/path/to/file.pdf`. Therefore, the Base Path needs to be specified to resolve any relative paths and locate the linked documents.

- If a **Content Property** was specified, the Base Path Connection is a directory on the file store where the crawler can save a copy of the Content Property strings for the Anzo Unstructured worker instances. Saving the content to a shared file location avoids the overhead of sending the strings to the workers over the network.

Filesystem Crawler



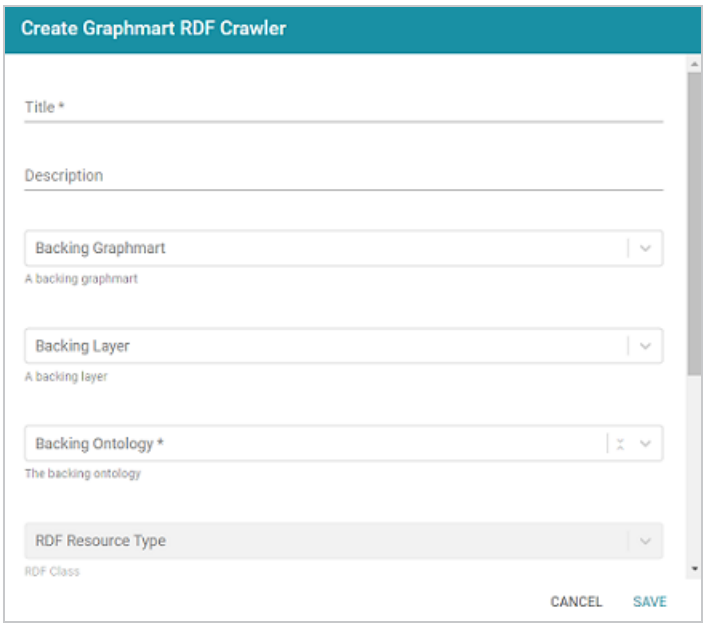
- **Title**: Required field that specifies the unique name for this crawler.
- **Description**: Optional field that provides a description of this crawler.
- **File Crawl Location**: Required field that specifies the file system crawl location. Click the field to open the File Location dialog box:



On the left side of the screen, select the storage location for the files to crawl. On the right side of the screen, navigate to the directory that contains the files. Select a directory, and then click **OK**.

- **Crawl subfolders**: Optional field that specifies whether to crawl the subdirectories under the VFS Crawl Location. To crawl the subdirectories, select the **Crawl subfolders** checkbox. To ignore subdirectories, clear the **Crawl subfolders** checkbox.

Graphmart RDF Crawler



- **Title**: Required field that specifies the unique name for this crawler.
- **Description**: Optional field that provides a description of this crawler.
- **Backing Graphmart**: Optional field that specifies the graphmart to crawl. To configure the grawler to crawl at the graphmart level, select one or more graphmarts in the **Backing Graphmart** field and leave the **Backing Layer** field blank.
- **Backing Layer**: Optional field that specifies the data layer or layers that you want the pipeline to crawl. To crawl specific layers and not an entire graphmart, make sure that you leave the **Backing Graphmart** field blank and select the layers to crawl in the **Backing Layer** field. If you specify both a Backing Graphmart and a Backing Layer, the Backing Graphmart value supersedes the Backing Layer value, resulting in the entire graphmart being crawled.
- **Backing Ontology**: Required field that specifies the model for the Backing Graphmart or Data Layer.
- **RDF Resource Type**: Required field that specifies the resource type or class of data to target with this crawler.
- **Link Property**: Optional field that specifies any link properties to crawl. A link property is a property whose value identifies the location of a linked document. When linked properties are specified, the crawler will crawl the linked documents. For example, in the triples below, **fileLocation** is a link property:

```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://fileLocation> "/path/to/file.pdf" .
```

> **Note**
> In typical use cases, this crawler is configured to define either a Link Property or a Content Property but not both.

- **Content Property**: Optional field that identifies any content properties to crawl. A content property is a property whose value is a string literal and you want the crawler to

crawl and annotate those strings. For example, in the triples below, **longDescription** is a content property:

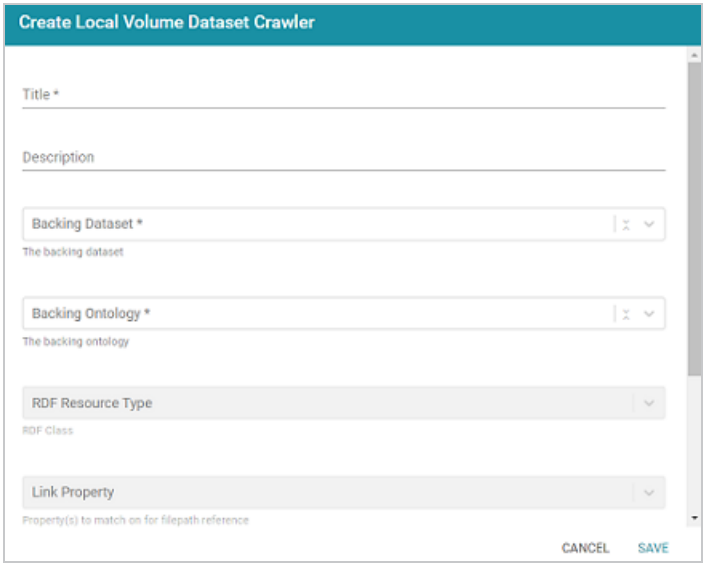
```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://longDescription> "this is some interesting, likely long,
unstructured text
with a lot of information, and I want it to be annotated" .
```

- **Base Path Connection**: Required field whose value depends on whether you specified a Link Property or a Content Property:
  - If a **Link Property** was specified, the Base Path Connection is the base path to use for resolving relative file paths in the Link Property values. For example, using the example triples:

    ```
    <urn://someUnstructuredDocument> <urn://someProperty> "file metadata"
    ;
    <urn://fileLocation> "/path/to/file.pdf" .
    ```

    The `<urn://fileLocation>` value of `/path/to/file.pdf` could be a relative path to a location like `s3://location/bucket/path/to/file.pdf` or `/opt/anzoshare/data/path/to/file.pdf`. Therefore, the Base Path needs to be specified to resolve any relative paths and locate the linked documents.
  - If a **Content Property** was specified, the Base Path Connection is a directory on the file store where the crawler can save a copy of the Content Property strings for the Anzo Unstructured worker instances. Saving the content to a shared file location avoids the overhead of sending the strings to the workers over the network.

Local Volume Dataset Crawler



- **Title**: Required field that specifies the unique name for this crawler.
- **Description**: Optional field that provides a description of this crawler.
- **Backing Dataset**: Required field that specifies the Anzo dataset to crawl.
- **Backing Ontology**: Required field that specifies the model for the dataset.
- **RDF Resource Type**: Required field that specifies the resource type or class of data to target with this crawler.
- **Link Property**: Optional field that specifies any link properties to crawl. A link property is a property whose value identifies the location of a linked document. When linked properties are specified, the crawler will crawl the linked documents. For example, in the triples below, **fileLocation** is a link property:

```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://fileLocation> "/path/to/file.pdf" .
```

> **Note**
> In typical use cases, this crawler is configured to define either a Link Property or a Content Property but not both.

- **Content Property**: Optional field that identifies any content properties to crawl. A content property is a property whose value is a string literal and you want the crawler to crawl and annotate those strings. For example, in the triples below, **longDescription** is a content property:

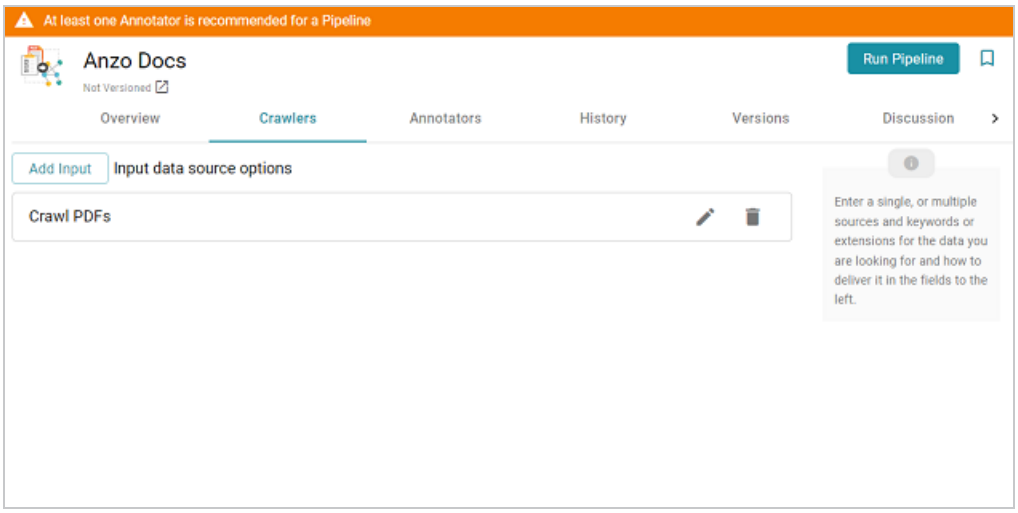
```
<urn://someUnstructuredDocument> <urn://someProperty> "file metadata" ;
<urn://longDescription> "this is some interesting, likely long,
unstructured text
with a lot of information, and I want it to be annotated" .
```

- **Base Path Connection**: Required field whose value depends on whether you specified a Link Property or a Content Property:
  - If a **Link Property** was specified, the Base Path Connection is the base path to use for resolving relative file paths in the Link Property values. For example, using the example triples:

    ```
    <urn://someUnstructuredDocument> <urn://someProperty> "file metadata"
    ;
    <urn://fileLocation> "/path/to/file.pdf" .
    ```

    The `<urn://fileLocation>` value of `/path/to/file.pdf` could be a relative path to a location like `s3://location/bucket/path/to/file.pdf` or `/opt/anzoshare/data/path/to/file.pdf`. Therefore, the Base Path needs to be specified to resolve any relative paths and locate the linked documents.
  - If a **Content Property** was specified, the Base Path Connection is a directory on the file store where the crawler can save a copy of the Content Property strings for the Anzo Unstructured worker instances. Saving the content to a shared file location avoids the overhead of sending the strings to the workers over the network.

5. When you have finished configuring the crawler, click **Save**. Anzo adds the crawler to the pipeline and returns to the Crawlers screen. For example:



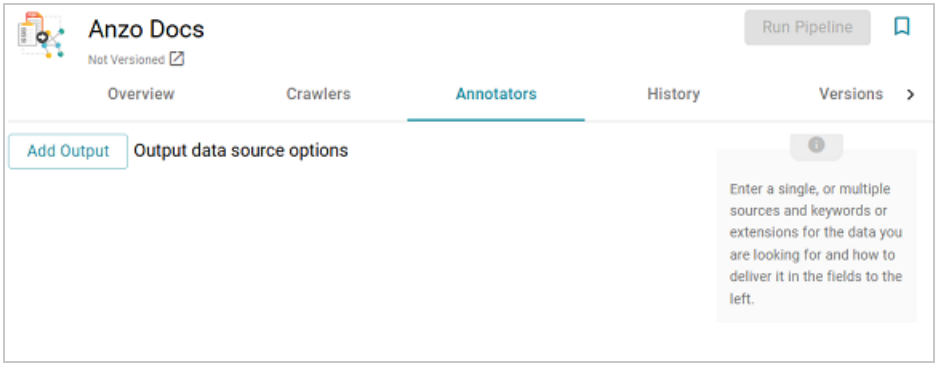
6. If you want to change the crawler configuration, click the Edit icon (✎) for the crawler and modify the settings as needed. If you want to add another crawler to the pipeline, repeat the steps above.

7. When you have finished adding crawlers, follow the instructions in Add Annotators to the Pipeline to add one or more annotators to the pipeline.

Add Annotators to the Pipeline

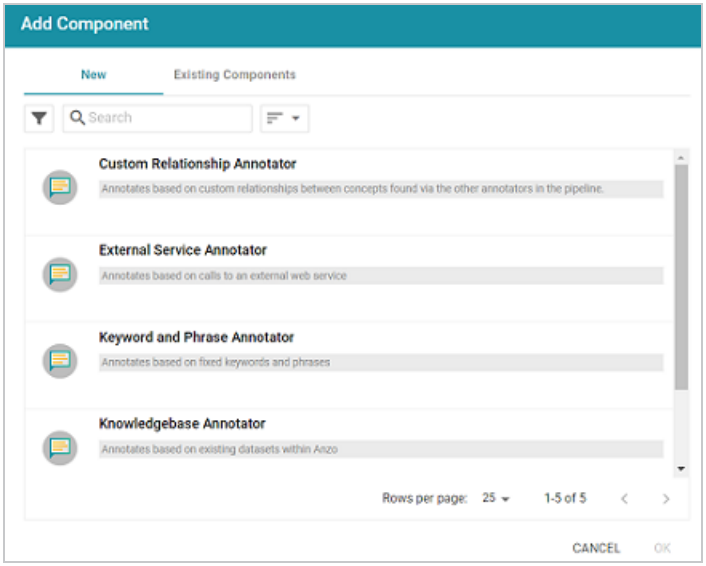
After adding crawlers, the next step is to add one or more annotators. Annotators extract facts or references in the text as annotations.

1. In the pipeline, click the **Annotators** tab.



2. Next, click the **Add Output** button. Anzo opens the Add Component dialog box. The **New** tab is selected and lists the available annotators and the **Existing Components** tab lists

annotators that have been previously configured for other pipelines.

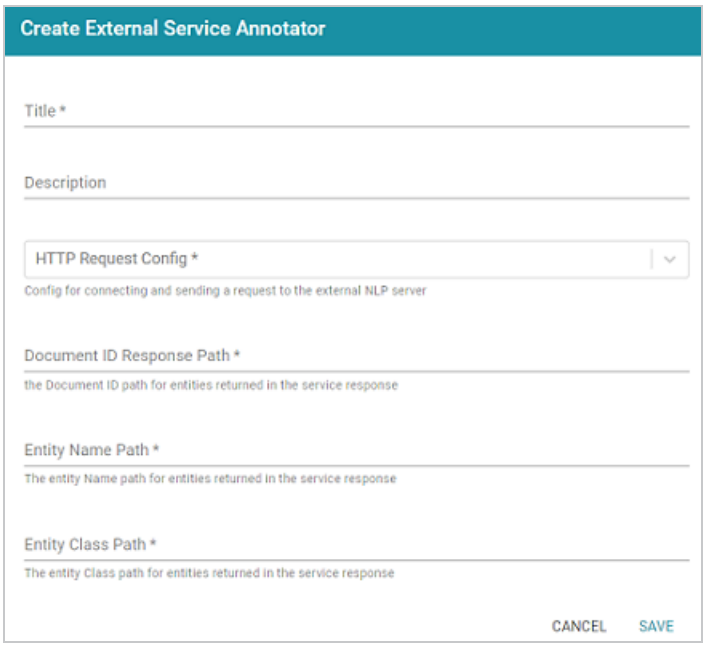


3. To add a new annotator to the pipeline, click the annotator name to select it. To add an existing annotator to the pipeline, click the **Existing Components** tab, and then select an annotator. The list below describes each of the default annotators:
   - **Custom Relationship Annotator**: Include this annotator to map relationships between annotations based on the number of characters between the annotations.
   - **External Service Annotator**: Include this annotator to hit an HTTP endpoint that provides annotations.
   - **Keyword and Phrase Annotator**: Include this annotator to create annotations based on the phrases that you specify.
   - **Knowledgebase Annotator**: Include this annotator to link structured and unstructured data by finding instances in data layers, graphmarts, or Anzo linked datasets. Based on the names and aliases of entities present or patterns that are indicative of the entities, this annotator marks up the documents with the structured entities linked.
   - **Regex Annotator**: Include this annotator to use regular expression rules to identify entities such as email addresses, URLs, phone numbers, or any other entity that can be matched using a regular expression.

4. After selecting an annotator, click **OK**. Anzo opens the Create dialog box for the component. Complete the fields to configure the annotator. The list below provides details about the settings for the annotators that are typically used in pipelines. Click an annotator name to view the details for that component:

   - External Service Annotator
   - Keyword and Phrase Annotator
   - Knowledgebase Annotator
   - Regex Annotator

   External Service Annotator

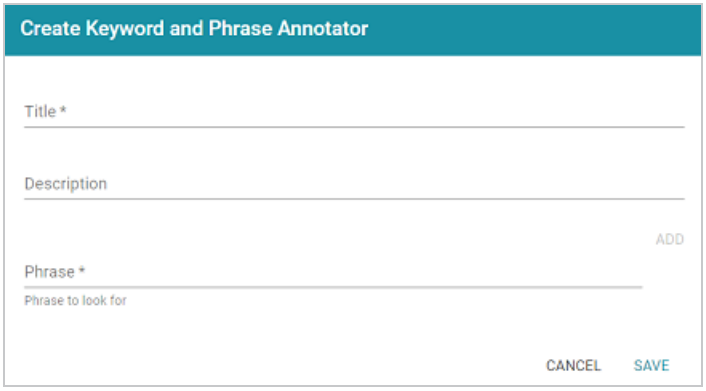


> **Tip**
> For information about the options that are presented when you edit an External Service Annotator, see Annotator Settings Reference.

   - **Title**: Required field that specifies the unique name for this annotator.
   - **Description**: Optional field that provides a description of this annotator.

- **HTTP Request Config**: Required field that specifies the HTTP source object that contains the URL and method to use when sending data for annotations.
- **Document ID Response Path**: Required field that specifies where to find the document ID in the response.
- **Entity Name Path**: Required field that specifies the annotation object name path.
- **Entity Class Path**: Required field that specifies the class URI for an annotation.

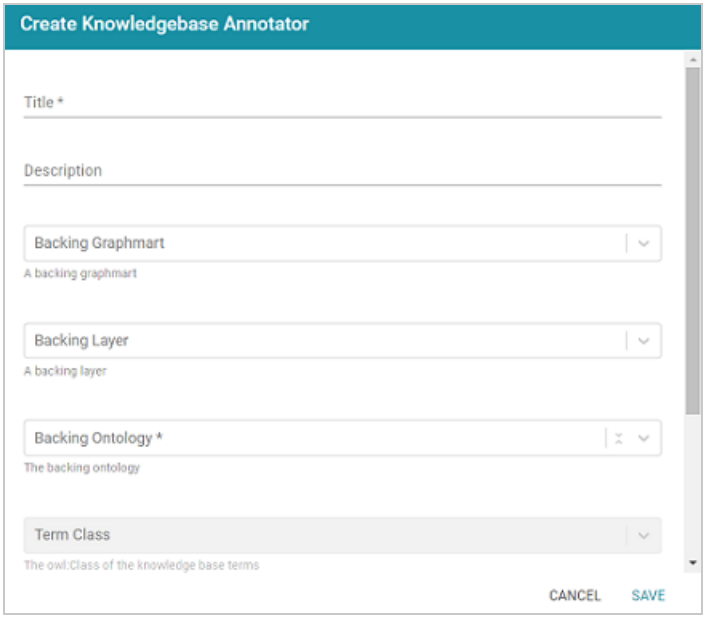
Keyword and Phrase Annotator



> **Tip**
> For information about the options that are presented when you edit a Keyword and Phrase Annotator, see Annotator Settings Reference.

- **Title**: Required field that specifies the unique name for this annotator.
- **Description**: Optional field that provides a description of this annotator.
- **Phrase**: Required field that specifies the terms or phrases to annotate. Type a word or phrase in the field and then click **Add** to add the phrase. You can add any number of phrases.

Knowledgebase Annotator



> **Tip**
> For information about the options that are presented when you edit a Knowledgebase Annotator, see Annotator Settings Reference.

- **Title**: Required field that specifies the unique name for this annotator.
- **Description**: Optional field that provides a description of this annotator.
- **Backing Graphmart**: Optional field that specifies the graphmart or graphmarts to annotate.

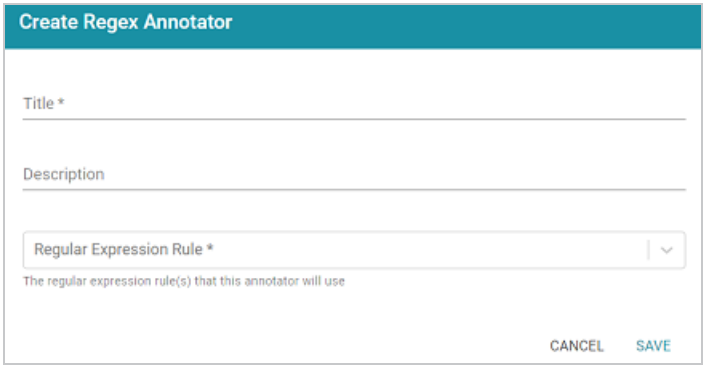
  > **Note**
  > If you want the annotator to run against a linked dataset or Anzo knowledgebase instead of a data layer or graphmart, leave the Backed Layer and Backed Graphmart fields blank. After saving the pipeline, you can edit the pipeline and specify a **Backed Dataset** at that time.

- **Backing Layer**: Optional field that specifies the data layer or layers to annotate.

  > **Note**
  > The Backing Layer and Backing Graphmart fields are treated independently. Layers that you select do not have to be part of the graphmart that you specify in **Backing Graphmart**. And specifying a layer does not mean that you must select a Backing Graphmart. However, any layers or graphmarts that you select must contain classes and properties from the **Backing Ontology** or the data will not be annotated.

- **Backing Ontology**: Required field that specifies the model for the backing data layers and/or graphmart. Click the field and select a model from the drop-down list.
- **Term Class**: Required field that specifies the class of data for the annotations.
- **Term Label Property**: Required field that lists the primary name or label property of the resources.
- **Term Identifying Properties**: Required field that specifies the properties that contain names, aliases, or other identifiers to use for identifying the resources.

Regex Annotator



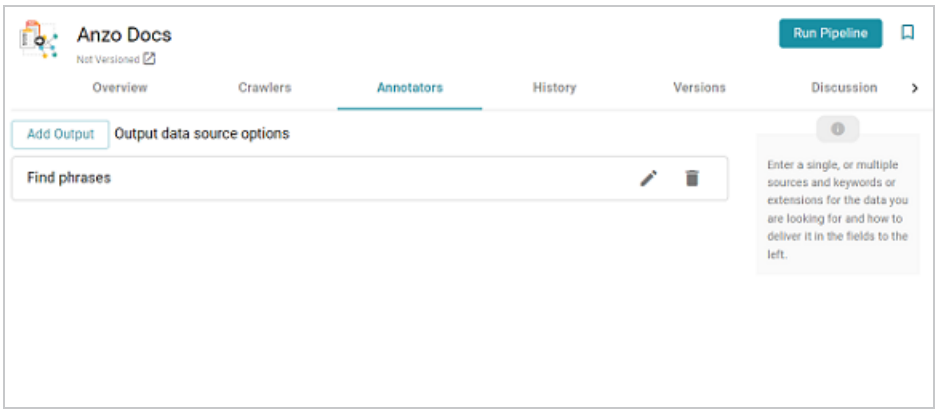
> **Tip**
> For information about the options that are presented when you edit a Regex Annotator, see Annotator Settings Reference.

- **Title**: Required field that specifies the unique name for this annotator.
- **Description**: Optional field that provides a description of this annotator.
- **Regular Expression Rule**: Required field that lists the regular expression rules for this annotator. To add a rule, click drop-down field and select **Create New**. Anzo opens the Create Regular Expression Rule dialog box where you can define the rule:



  - **Title**: Required field that specifies the name of the rule.
  - **Class Structure**: Required field that specifies the class in the model that should be created for this rule. The value should be in the format `group_number:class_name`, where `group_number` corresponds to a group in the regex capture. Each rule should start with group `0`. Include groups `1` and higher if needed to represent parts of the expression that are contained in parentheses. The `class_name` is a label that describes the type of data the rule will find. For example, for a rule that finds hyphenated words `0:Hyphens`.
  - **Description**: Optional field that describes the rule.
  - **Regular Expression**: Required field that specifies the regular expression to use for finding matching entities.

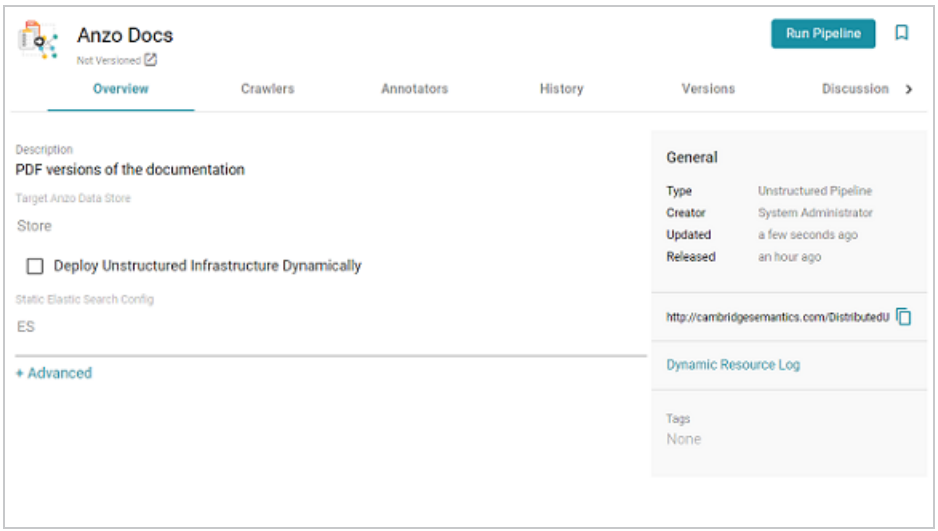
5. When you have finished configuring the annotator, click **Save**. Anzo adds the annotator to the pipeline and returns to the Annotators screen. For example:



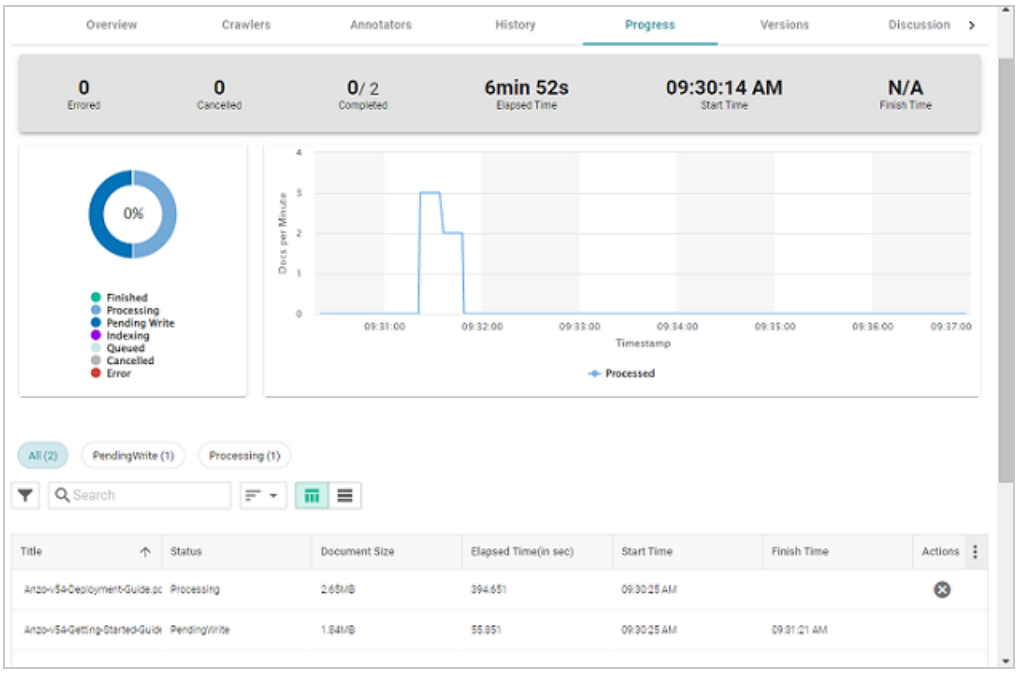
6. If you want to change the annotator configuration, click the Edit icon (✎) for the annotator and modify the settings as needed (see Annotator Settings Reference for information about settings). If you want to add another annotator to the pipeline, repeat the steps above.
7. When you have finished adding annotators to the pipeline, proceed to Run the Pipeline below.

Run the Pipeline

When you are ready to run the pipeline, click the **Run Pipeline** button on the top right of the screen. For example:



The process can take several minutes to complete. You can click the **Progress** tab to view details such as the pipeline status, runtime, number of documents processed, and errors. For example:

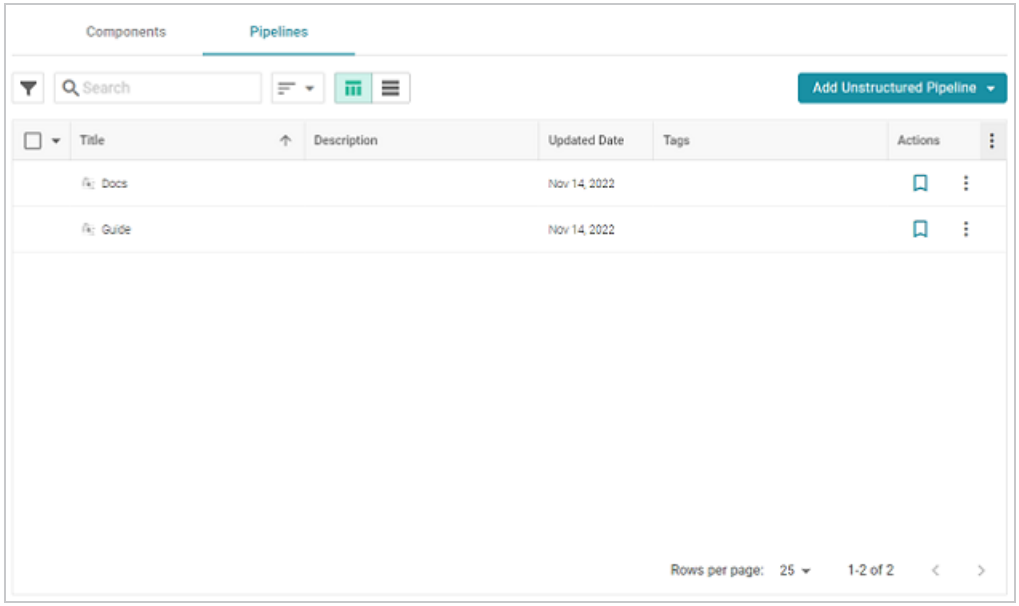


When the pipeline finishes, a new dataset becomes available in the Datasets catalog. From the catalog, you can create a graphmart from the dataset so that you can explore and analyze the data. For instructions, see Creating a Graphmart from a Dataset. You can also add the dataset to an existing graphmart by following the steps in Adding a Dataset to a Graphmart.

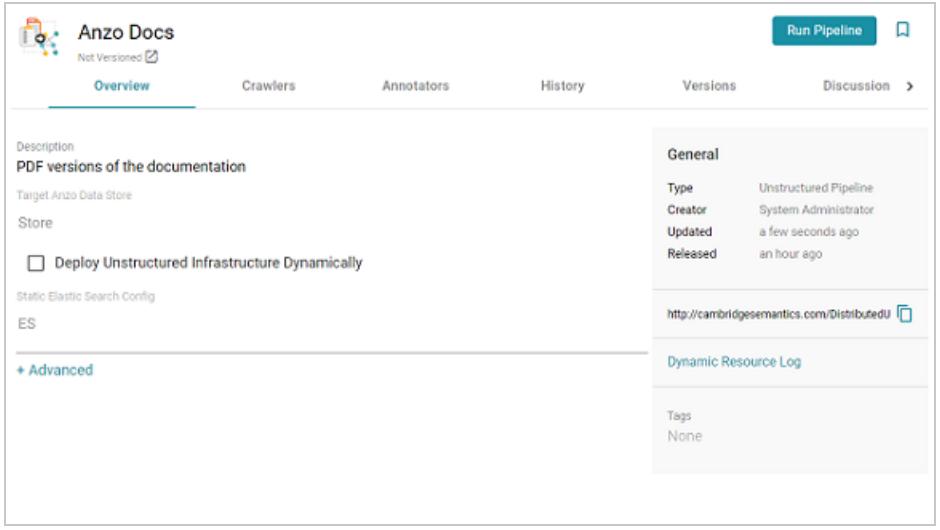
Running an Unstructured Pipeline

This page provides instructions for running an unstructured pipeline.

1. In the Anzo application, expand the **Onboard** menu and click **Unstructured Data**. Anzo displays the Pipelines screen, which lists any existing unstructured pipelines. For example:

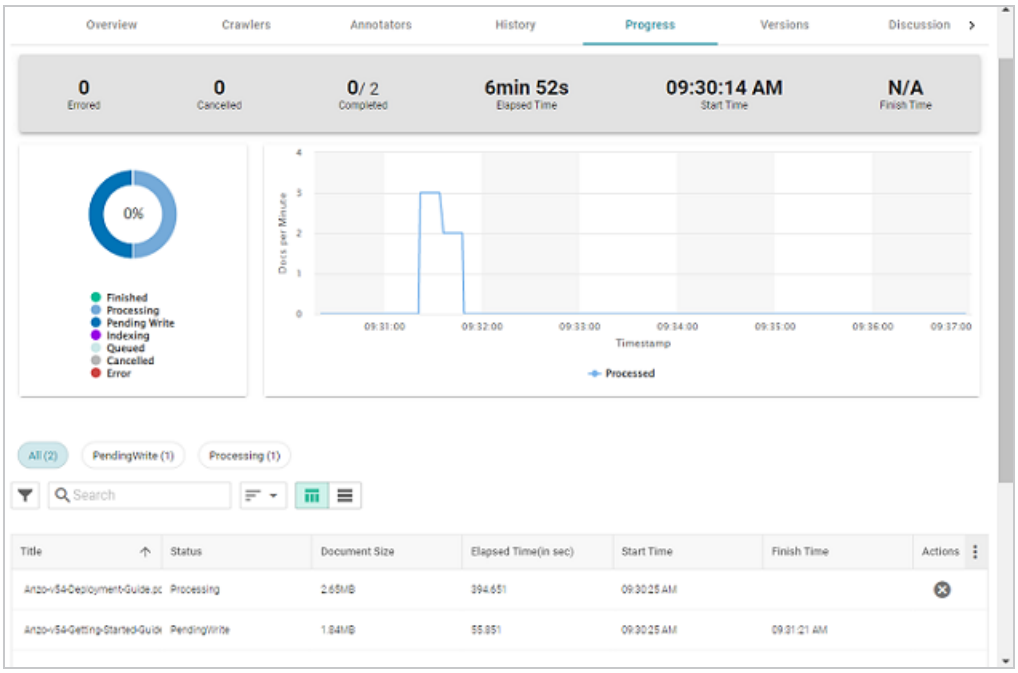


2. Click the name of the pipeline that you want to run. Anzo displays the pipeline Overview screen. For example:



3. Click **Run Pipeline** to run the pipeline.

The process can take several minutes to complete. You can click the **Progress** tab to view details such as the pipeline status, runtime, number of documents processed, and errors. For example:



If this is the first time the pipeline was run, a new dataset becomes available in the Datasets catalog. From the catalog, you can create a graphmart from the dataset so that you can explore and analyze the data. For instructions, see Creating a Graphmart from a Dataset. If the pipeline was run previously, the existing dataset was updated and you can refresh or reload the graphmart that contains the dataset to make the new data available for analytics.

Pipeline Settings Reference

The table below defines the Advanced settings that are available on the Overview tab when viewing an unstructured pipeline.

Setting	Description
Append Timestamp	Controls whether to add a timestamp to unstructured document URIs. This setting is enabled by default.
Diagnostic Logging	Controls whether verbose diagnostic logging is enabled for the pipeline. This setting is disabled by default. When enabled, debug-level logging is performed for the duration of the pipeline.
Current Pipeline Run	This setting is a pointer to the pipeline run object that tracks the ongoing execution of the pipeline.
Pipeline Network Connection	This setting specifies the network connection configuration to be used by the pipeline's worker nodes to connect to the Anzo server. If not specified, this setting defaults to the Unstructured Cluster connection configuration.
Persist Extracted Text	Controls whether to persist the extracted text from documents. This setting is enabled by default.
Persist HTML	Controls whether to persist the extracted highlighted/annotated HTML from documents. This setting is enabled by default.
Persist Original Binary	Controls whether to persist the binary from the original documents. This setting is enabled by default.
Persist Hit Spans	Controls whether to persist the hit spans for the annotations of unstructured documents. This setting is disabled by default.
Persist Nothing	Controls whether RDF data about the documents or annotations are saved or

Setting	Description
	persisted. This setting is disabled by default.
Skip Elastic Search Indexing	Controls whether to skip Elasticsearch indexing. This setting is disabled by default.
Skip Elastic Search JSON creation	Controls whether to skip creating Elasticsearch JSON. This setting is disabled by default.
Is Corpus Cumulative	Controls whether to add the components of each pipeline run to the working edition of the dataset. This setting is disabled by default.
Skip Text Extraction	Controls whether to skip text extraction. This setting is disabled by default.
Delete Elastic Search JSON files	Controls whether to delete the Elasticsearch JSON files after they are indexed. This setting is enabled by default.
Allow Empty Documents	Controls whether to allow documents that have no text to proceed through the pipeline. This setting is disabled by default.
Archive and Host Content	Controls whether to download, cleanse, encapsulate, archive, and host complete document content with inline artifacts. This setting is enabled by default.
HTTP Fetch in Archive	Controls whether the archiving process should resolve and download HTTP URLs that are specified in documents. This setting is disabled by default.
Corpus Linked Dataset	Specifies the FLDS used for documents and annotations from this pipeline. This setting defaults to the name of the pipeline.

Setting	Description
Corpus Name	Specifies the name of the corpus (collection of documents) for the pipeline.
Phase Status Persistence	Specifies how phase status metadata is persisted for each document in the pipeline.
Write Status Updates to Jnl	Controls whether status updates for pipeline runs are written to the journal. This setting is enabled by default.
Write Status Updates to FLDS	Controls whether status updates for pipeline runs are written to an FLDS. This setting is disabled by default.
Write Original Binary On Timeout	Controls whether the original binary is written if the pipeline times out or errors. This setting is disabled by default.
RamDisk Directory Location	Specifies an optional RamDisk base directory to create temporary files under. Using a RamDisk may speed up the pipeline.
Use File Name as Document Title	Controls whether to use the file's name on disk as the document title. This setting is disabled by default.
RDF Statement Buffer Size	Specifies the maximum number of statements to buffer before writing. The default value is 10,000.
RDF File Statement Count	Specifies the maximum number of statements to include in each RDF output file.
Batch Size	Specifies the number of documents to include in one batch.
Maximum	Specifies the maximum number of issues that can be encountered in a run of

Setting	Description
Allowed Session Issues	this pipeline before failing the pipeline.
UI Update Interval (in milliseconds)	The interval of time to wait between running queries to update the data on the pipeline Progress screen. The default value is 30,000 milliseconds (30 seconds).
Document Processing Timeout	Specifies the timeout in milliseconds for each document batch to be processed. Leave this value unset (or set it to 0) to use the microservice cluster's default timeout value.
Error On No Documents Found	Controls whether to fail the pipeline if no documents are found. This setting is enabled by default.
Maximum Pipeline Run Status Journals	Specifies the maximum number of pipeline run status journals to keep before aging them off to an FLDS. By default, only the status of the most recent run of a pipeline remains stored in a status journal. All previous reports are automatically converted to an FLDS and the original status journal is deleted.
Elastic Search Bulk Actions	Specifies the maximum number of indexing actions to queue during Elasticsearch indexing. The default value is 2,000.
Elastic Search Bulk Size	Specifies the maximum size of the document queue during Elasticsearch indexing. The default value is 5.
Elastic Search Bulk Concurrent Requests	This setting specifies the maximum number of concurrent bulk requests to allow during Elasticsearch indexing. The default value is 1.
Elastic Search Bulk Max	This setting specifies the maximum number of threads to use for Elasticsearch indexing. The default value is 1.

Setting	Description
Threads	
Elastic Search Mapping	This setting specifies (in JSON format) the mapping to use when indexing unstructured documents in Elasticsearch.
Elastic Search Pipeline Configuration	This setting specifies (in JSON format) the Elasticsearch pipeline configuration to use when indexing unstructured documents.
Elastic Search Directory Write-all	Controls whether to give write-all permission to the `esi` directory in the output corpus FLDS.
Elasticsearch Index Settings	This setting specifies (in JSON format) the index settings to use when indexing unstructured documents in Elasticsearch.
Skip Teardown Of Dynamic Resources	Controls whether dynamic K8s-based resources associated with the pipeline are left running after the pipeline is complete. This setting is disabled by default. Enabling it can result in increased cloud resource usage.
Default Finish Pending Writes On Pipeline Cancellation	Controls whether to finish any pending writes for documents during pipeline cancellation. There is a flag in the cancellation request that can be used to override this setting. This setting is enabled by default.
Post-persist Postprocessor	Specifies any post-persist semantic postprocessors in the pipeline.
Rich Text Extractor	Lists the HTML extractors to use in the pipeline.
Post Worker	Specifies a service to invoke on documents after they are successfully

Setting	Description
Service	processed by the pipeline worker processes.
Pre-persist Postprocessor	Specifies any pre-persist semantic postprocessors in the pipeline.
Status Journal Base Path	Specifies the base path for storage of the status journal. By default, status journals are written to a `status_journals` subdirectory in the Anzo Data Store that is specified for the pipeline.
Content Transformer	Specifies any content transformation and metadata extraction components to use in the pipeline.
Document Crawler Thread Count	Specifies the number of threads to use for document crawling. The default value is 4.
Worker Service ID	Specifies the worker service ID to send requests to. If not specified, the default is `pipelineWorkerService`.

Annotator Settings Reference

When you edit an existing annotator, additional options become available for refining the annotation criteria or customizing the generated model or instance data. This topic describes the advanced settings that are available when editing each type of annotator.

- External Service Annotator
- Keyword and Phrase Annotator
- Knowledgebase Annotator
- Regex Annotator

External Service Annotator

The table below defines the settings that are displayed when an External Service Annotator is edited.

Setting	Description
Title	Required field that specifies the unique name for the annotator.
Description	Optional field that provides a description of the annotator.
HTTP Request Config	Required field that specifies the HTTP source object that contains the URL and method to use when sending data for annotations.
Document ID Response Path	Required field that specifies where to find the document ID in the response.
Entity Name Path	Required field that specifies the annotation object name path.
Entity Class Path	Required field that specifies the base class URI for an annotation.
Result Path Root	The path to the object that contains the annotation results.

Setting	Description
Store NLP Service Response	Controls whether the service's response is stored in the binary store.
Result Field Path	The external NLP-specific result configuration for returned entities.
Socket Timeout	Specifies the socket timeout (in milliseconds) to use for requests against the source.
Entity Snippet Path	The snippet path for entities returned in the service response.
Entity End Offset Path	The end text offset location in the document for entities returned in the service response.
Entity Begin Offset Path	The start text offset location in the document for entities returned in the service response.
Entity Span Path	The text offset location in the document for entities returned in the service response.
Entity Text Path	The text path for entities returned in the service response.
Entity ID Path	The ID path for entities returned in the service response.
Document ID Request Field	The Document ID parameter for the external service.
Class Name Property	Specifies an annotation property whose value you want to map to the name of the class. For example, if a `Category` property has the value `Disease` and you want the name of the class to be "Disease," add `Category` to this field. When Class Name Property is not defined, the class name is auto-generated.

Setting	Description
Unintended Property Names	A list of any property names to filter out. Type a name in the field and then click **Add** to add the value.
Unintended Classes	A list of any classes to filter out. Type a class in the field and then click **Add** to add the value.
Unintended Instances	A list of any entities or instances of the class to filter out. Type an instance in the field and then click **Add** to add the value.
Is Combine Annotation Instances	Controls whether to combine multiple instances of an extraction into one annotation.
Create Additional General Annotation Type	Controls whether the annotator creates a general shared annotation type in addition to the specific annotation types that are created.
Explicit Property Datatypes/Objecttypes	A list of keys that map property names to a particular object property type.
Output the Detections	Controls whether to include specific detections as an annotation property.
Unintended Property Values	A list of any property values to filter out. Type a value in the field and then click **Add** to add the value.
Use General Annotator Name in Ontology URI	Controls whether to use `ExternalServiceAnnotator` or the specific annotator name in the ontology URI.
Entity URI Property	Specifies an annotation property whose value you want to map to the URIs for instances of the class. For example, if a `Disease_ID` property has the value `http://example.com/Asthma` and you want to use `http://example.com/Asthma` as the base URI for instances of the class, add `Disease_ID` to this field. When Entity

Setting	Description
	URI Property is not defined, the URI is auto-generated based on the name.
Entity Name Property	Specifies an annotation property whose value you want to map to the names for instances of the class. For example, if a `Preferred_Label` property includes disease names and you want to use those label values as the names for instances of the Disease class, add `Preferred_Label` to this field. When Entity Name Property is not defined, the name is auto-generated.
Domain Object Base Class URI	If creating a general annotation type (Create Additional General Annotation Type is enabled), this setting specifies the class to use as the base type for the annotator's domain objects.
Class URI Property	Specifies an annotation property whose value you want to map to the class URI in the model. For example, if a `Category_ID` property has the value `http://example.com/Disease` and you want to use `http://example.com/Disease` as the base class URI, add `Category_ID` to this field. When Class URI Property is not defined, the URI is auto-generated based on the name.
Is Error Fatal	Controls whether to fail the pipeline if this annotator fails to create annotations.

Keyword and Phrase Annotator

The table below defines the settings that are displayed when a Keyword and Phrase Annotator is edited.

Setting	Description
Only Consider Text	Controls whether to find phrases via a simplified format of the

Setting	Description
	document. Enabling this setting can be beneficial for a document such as a rich HTML file. Enabling this option is less ideal for documents with multibyte characters.
Require Nonstandard Word Boundaries	Indicates whether the specified phrase can be present with or without surrounding character breaks (e.g. for Chinese) or with regex-nonstandard word boundaries (e.g. for Tagalog).
Title	Required field that specifies the unique name for the annotator.
Description	Optional field that provides a description of the annotator.
Phrase	Required field that specifies the terms or phrases to annotate. Type a word or phrase in the field and then click **Add** to add the phrase. You can add any number of phrases.
Unintended Property Names	A list of any property names to filter out. Type a name in the field and then click **Add** to add the value.
Create Additional General Annotation Type	Controls whether the annotator creates a general shared annotation type in addition to the specific annotation types that are created.
Entity URI Property	Specifies an annotation property whose value you want to map to the URIs for instances of the class. For example, if a `Disease_ID` property has the value `http://example.com/Asthma` and you want to use `http://example.com/Asthma` as the base URI for instances of the class, add `Disease_ID` to this field. When Entity URI Property is not defined, the URI is auto-generated based on the name.
Explicit Property Datatypes/Objecttypes	A list of keys that map property names to a particular object property type.

Setting	Description
Entity Name Property	Specifies an annotation property whose value you want to map to the names for instances of the class. For example, if a `Preferred_Label` property includes disease names and you want to use those label values as the names for instances of the Disease class, add `Preferred_Label` to this field. When Entity Name Property is not defined, the name is auto-generated.
Domain Object Base Class URI	If creating a general annotation type (Create Additional General Annotation Type is enabled), this setting specifies the class to use as the base type for the annotator's domain objects.
Unintended Classes	A list of any classes to filter out. Type a class in the field and then click **Add** to add the value.
Class Name Property	Specifies an annotation property whose value you want to map to the name of the class. For example, if a `Category` property has the value `Disease` and you want the name of the class to be "Disease," add `Category` to this field. When Class Name Property is not defined, the class name is auto-generated.
Unintended Instances	A list of any entities or instances of the class to filter out. Type an instance in the field and then click **Add** to add the value.
Is Combine Annotation Instances	Controls whether to combine multiple instances of an extraction into one annotation.
Class URI Property	Specifies an annotation property whose value you want to map to the class URI in the model. For example, if a `Category_ID` property has the value `http://example.com/Disease` and you want to use `http://example.com/Disease` as the base class URI, add `Category_ID` to this field. When Class URI Property is not defined, the URI is auto-generated based on the name.

Setting	Description
Output the Detections	Controls whether to include specific detections as an annotation property.
Unintended Property Values	A list of any property values to filter out. Type a value in the field and then click **Add** to add the value.
Is Error Fatal	Controls whether to fail the pipeline if this annotator fails to create annotations.

Knowledgebase Annotator

The table below defines the settings that are displayed when a Knowledgebase Annotator is edited.

Setting	Description
Title	Required field that specifies the unique name for the annotator.
Description	Optional field that provides a description of the annotator.
Backing Graphmart	Optional field that specifies the graphmart or graphmarts to annotate.
Backing Layer	Optional field that specifies the data layer or layers to annotate. **Note** The Backing Layer and Backing Graphmart fields are treated independently. Layers that you select do not have to be part of the graphmart that you specify in Backing Graphmart. And specifying a layer does not mean that you must select a Backing Graphmart. However, any layers or graphmarts that you select must contain classes and

Setting	Description
	properties from the Backing Ontology or the data will not be annotated.
Backing Ontology	Required field that specifies the model for the backing data layers and/or graphmart.
Term Class	Required field that specifies the class of data for the annotations.
Term Label Property	Required field that lists the primary name or label property of the resources.
Term Identifying Properties	Required field that specifies the properties that contain names, aliases, or other identifiers to use for identifying the resources.
Backing Dataset	Optional field that specifies the dataset or datasets to annotate.
Case Sensitive	Controls whether matches must be case-sensitive.
Invalidating Properties	A list of any properties for which you do not want to find matching resources.
Discard Matches Of Common Words	Controls whether to discard matches of the most common words.
Discard Matches of Substrings	A list of the substrings for which you want matches to be discarded. Type a string in the field and then click **Add** to add the value.
Text Search Query Pattern Precedence	When text search query properties are specified, this setting controls whether resource names or aliases are included as matches. When enabled, resource names and aliases will not be matched.

Setting	Description
Lucene Pattern Properties	A list of properties that contain Lucene query syntax for document categorization.
Approximate Label Properties	A list of properties that contain phrases that may be matched only approximately, i.e., fault-tolerantly, via slightly alternate spellings or misspellings.
Simplified Regex Pattern Properties	A list of properties that contain simplified regular expressions.
Regex Pattern Properties	A list of properties that contain regular expressions.
Strip Characters for Match	Characters to strip out before determining if there is a match.
Clear Caches	Controls whether to clear any existing caches when the pipeline is run.
Rows Per Query	The maximum number of rows to query at a time when paging through the knowledgebase.
Minimum Hit Length	The minimum span length that can count as a match.
Domain Object Base Class URI	If creating a general annotation type (Create Additional General Annotation Type is enabled), this setting specifies the class to use as the base type for the annotator's domain objects.
Class URI Property	Specifies an annotation property whose value you want to map to the class URI in the model. For example, if a `Category_ID` property has the value `http://example.com/Disease` and you want to use `http://example.com/Disease` as the base class URI, add `Category_ID` to this field. When Class URI Property is

Setting	Description
	not defined, the URI is auto-generated based on the name.
Entity URI Property	Specifies an annotation property whose value you want to map to the URIs for instances of the class. For example, if a `Disease_ID` property has the value `http://example.com/Asthma` and you want to use `http://example.com/Asthma` as the base URI for instances of the class, add `Disease_ID` to this field. When Entity URI Property is not defined, the URI is auto-generated based on the name.
Is Combine Annotation Instances	Controls whether to combine multiple instances of an extraction into one annotation.
Unintended Instances	A list of any entities or instances of the class to filter out. Type an instance in the field and then click **Add** to add the value.
Explicit Property Datatypes/Objecttypes	A list of keys that map property names to a particular object property type.
Unintended Classes	A list of any classes to filter out. Type a class in the field and then click **Add** to add the value.
Unintended Property Names	A list of any property names to filter out. Type a name in the field and then click **Add** to add the value.
Create Additional General Annotation Type	Controls whether the annotator creates a general shared annotation type in addition to the specific annotation types that are created.
Output the Detections	Controls whether to include specific detections as an annotation property.
Unintended Property	A list of any property values to filter out. Type a value in the field and

Setting	Description
Values	then click **Add** to add the value.
Entity Name Property	Specifies an annotation property whose value you want to map to the names for instances of the class. For example, if a `Preferred_Label` property includes disease names and you want to use those label values as the names for instances of the Disease class, add `Preferred_Label` to this field. When Entity Name Property is not defined, the name is auto-generated.
Class Name Property	Specifies an annotation property whose value you want to map to the name of the class. For example, if a `Category` property has the value `Disease` and you want the name of the class to be "Disease," add `Category` to this field. When Class Name Property is not defined, the class name is auto-generated.
Is Error Fatal	Controls whether to fail the pipeline if this annotator fails to create annotations.

Regex Annotator

The table below defines the settings that are displayed when a Regex Annotator is edited.

Setting	Description
Title	Required field that specifies the unique name for the annotator.
Description	Optional field that provides a description of the annotator.
Regular Expression Rule	Required field that lists the regular expression rules for this annotator.
Case-Insensitive	Enables or disables case-insensitive matching. By default, case-

Setting	Description
	insensitive matching assumes that only characters in the US-ASCII character set are being matched. Unicode-aware case-insensitive matching can be enabled by enabling Unicode Case Folding in conjunction with this option.
Multiline Mode	Enables or disable multiline mode. When multiline mode is enabled, the expressions `^` and `$` match immediately after or before a line terminator or the end of the input sequence. When multiline mode is disabled, these expressions only match at the beginning and end of the entire input sequence.
Allow Comments	Controls whether whitespace and comments are allowed in a pattern. When enabled, whitespace and embedded comments starting with `#` are ignored until the end of a line.
Canonical Equivalence	Controls whether canonical equivalence is taken into account when finding matches. When enabled, characters are considered a match if and only if their full canonical decompositions match. For example, the expression `a\u030A` will match the string `\u00E5`.
Enable Dotall	Controls whether dotall mode is used. When enabled, the expression `.` matches any character, including a line terminator. When disabled, `.` does not match line terminators.
Literal Parsing	Controls whether literal parsing is employed. When enabled, the input string that specifies the pattern is treated as a sequence of literal characters and metacharacters and escape sequences have no special meaning.
Unicode Case Folding	Controls whether case-insensitive matching is done in a manner that is consistent with the Unicode Standard. By default, Case-Insensitive matching assumes that only characters in the US-ASCII

Setting	Description
	set are being matched.
Unix Lines	Enables or disables Unix line mode. When enabled, only the `\n` line terminator is recognized in the behavior of `.`, `^`, and `$`.
Is Combine Annotation Instances	Controls whether to combine multiple instances of an extraction into one annotation.
Class URI Property	Specifies an annotation property whose value you want to map to the class URI in the model. For example, if a `Category_ID` property has the value `http://example.com/Disease` and you want to use `http://example.com/Disease` as the base class URI, add `Category_ID` to this field. When Class URI Property is not defined, the URI is auto-generated based on the name.
Entity URI Property	Specifies an annotation property whose value you want to map to the URIs for instances of the class. For example, if a `Disease_ID` property has the value `http://example.com/Asthma` and you want to use `http://example.com/Asthma` as the base URI for instances of the class, add `Disease_ID` to this field. When Entity URI Property is not defined, the URI is auto-generated based on the name.
Class Name Property	Specifies an annotation property whose value you want to map to the name of the class. For example, if a `Category` property has the value `Disease` and you want the name of the class to be "Disease," add `Category` to this field. When Class Name Property is not defined, the class name is auto-generated.
Unintended Classes	A list of any classes to filter out. Type a class in the field and then click **Add** to add the value.

Setting	Description
Entity Name Property	Specifies an annotation property whose value you want to map to the names for instances of the class. For example, if a `Preferred_Label` property includes disease names and you want to use those label values as the names for instances of the Disease class, add `Preferred_Label` to this field. When Entity Name Property is not defined, the name is auto-generated.
Unintended Instances	A list of any entities or instances of the class to filter out. Type an instance in the field and then click **Add** to add the value.
Unintended Property Values	A list of any property values to filter out. Type a value in the field and then click **Add** to add the value.
Create Additional General Annotation Type	Controls whether the annotator creates a general shared annotation type in addition to the specific annotation types that are created.
Output the Detections	Controls whether to include specific detections as an annotation property.
Explicit Property Datatypes/Objecttypes	A list of keys that map property names to a particular object property type.
Unintended Property Names	A list of any property names to filter out. Type a name in the field and then click **Add** to add the value.
Domain Object Base Class URI	If creating a general annotation type (Create Additional General Annotation Type is enabled), this setting specifies the class to use as the base type for the annotator's domain objects.
Is Error Fatal	Controls whether to fail the pipeline if this annotator fails to create annotations.

Model

The topics in this section provide introductory information about data models, describe model requirements, and include instructions for working with models.

In this section:

Model Concepts and Vocabulary 402

Managed Model Concepts 405

Model Requirements 407

Uploading a Model 412

Creating a Model 415

Editing a Custom Model 418

Editing a Managed Model 428

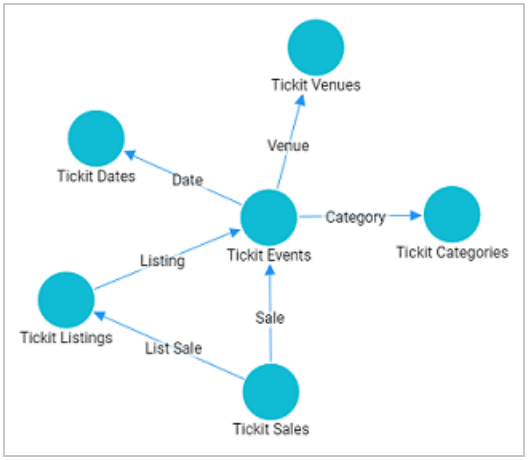
Downloading a Model 434

Defining Resource Templates 438

Model Concepts and Vocabulary

Models define the business meaning of the source data. They describe the concepts, attributes, and relationships in or across data sets. Instead of reflecting the format or schema of the source data, models reflect the desired structure of the data after it is onboarded to Anzo. Anzo links data to models to provide flexibility for capturing data coming from various sources and structures and to enable users to search for and visualize data in Hi-Res Analytics dashboards or other applications. Users can create models in the Models editor or import existing or external models (OWL ontologies) to Anzo. Models can be shared and reused.

The following image shows a portion of the model for a dataset that captures sales activity for a fictional website where people buy and sell tickets for sporting events, shows, and concerts.



Model Vocabulary

The table below defines key terms to know when working with models.

Term	Description
Class	Models are made up of classes. Classes describe a concept or a group of related objects. For example, the model above contains Tickit Events, Tickit Dates, Tickit Categories, Tickit Sales, and Tickit Listings classes.
Base Class	A base class is a more general version of a class. For example, a film model might

Term	Description
	have a Person base class with subclasses that categorize types of people.
Subclass	A subclass is a more specific version of another class. For example, in a film model a Person class might have subclasses of Actors, Directors, and Producers. Subclasses share the properties from the base class and can include additional properties.
Property	Properties are attributes that describe the data in a class. For example the Tickit Events class has properties such as eventName, dateID, and startTime. There are two types of properties: • **Data property**: Relates a class to a simple or literal value. For example, in the events class, the eventName and startTime properties relate to simple values. • **Object property**: Relates a class to another class. For example, the listID property relates to the Tickit Sales and Tickit Events classes.
Property Type	A property type is the data type of the values for the property, also known as the Property Range.
Instance	Instances are concrete occurrences of a class. For example, an event's name is an instance of the Tickit Events class.
Simple Value	A simple value is also known as a literal value. The list below describes literal values: • Numbers (for example, 15, -9, 10.35) • Text strings (for example, "Jane Doe" or "a long description") • Dates and times (for example, "13-Dec-2021", or "April, 2022") • Boolean (true or false)

Term	Description
Type	Either a class or a simple value.

For conceptual information about Anzo-generated models, called managed models, see Managed Model Concepts.

Managed Model Concepts

This topic provides an overview of managed models and includes important concepts to know when working with them.

What is a Managed Model?

When you onboard a data source with the automated direct load workflow (as described in Onboarding Data with the Automated Workflow) or manually onboard a source with a Direct Load Step (as described in Onboarding Data with a Direct Load Step), a *Managed Model* is produced. A managed model is generated, owned, and managed by the data layer that contains the auto-generated or user-created Direct Load Step. If a Direct Load Step query is changed, additional Direct Load Steps are added to the same layer, or the underlying source schema changes, the managed model is automatically updated when the graphmart or layer is reloaded or refreshed.

Managed Models Cannot be Edited Outside of Direct Load Steps

Though a model that is generated in a Direct Load Step is registered in Anzo and is available for viewing in the Model editor, the model is owned and managed by the layer that contains the Direct Load Step. That means any manual changes made to the model outside of the step, such as from the Model editor, will be overwritten any time the graphmart or layer is refreshed or reloaded. **Do not modify managed models except by editing (or adding) Direct Load Step queries**. For guidance on editing managed models, see Editing a Managed Model.

There is One Managed Model Per Data Layer

If you include multiple Direct Load Steps in the same layer, they will all update the same model. This functionality can be useful if you want to align the data and generated model across multiple steps. If you have multiple sources that are not intended to align or update the same model, create separate data layers.

Deleting a Layer Deletes the Model

If you delete a layer that includes a managed model, the model is also deleted. Use caution when referencing managed models outside of graphmarts. For example, if you create a dataset and reference a managed model when you select the ontology, the reference will break if the data layer that manages the model is deleted.

Model Requirements

To ensure that data structures are properly defined, Anzo requires that data models include certain information and avoid unsupported information. This topic provides details about the requirements and guidelines to follow when uploading or creating models.

Requirements

This section lists the requirements or rules to follow when uploading or creating a data model. Models that are generated by Anzo during the auto-ingest process conform to these rules.

- Define each model as an owl:Ontology
- Define the model name with rdfs:label
- The named graph URI must match the ontology URI
- Define classes and concepts with owl:Class
- Define taxonomy with rdfs:subClassOf
- Define properties as owl:DatatypeProperty or owl:ObjectProperty
- Include rdfs:domain and rdfs:range for all properties
- Reference only Anzo-stored models

Define each model as an owl:Ontology

Define each data model as an **owl:Ontology**. To do so, include the following triple in the model:

```
<myOntology> a owl:Ontology
```

Where `myOntology` is the URI that names the model. The URI must be unique. To avoid unexpected results when saving a model, do not include a hash (#) character at the end of the model URI.

Define the model name with rdfs:label

Use an **rdfs:label** property to define name of the model as a string. Include the following triple:

```
<myOntology> rdfs:label "My Ontology"^^xsd:string .
```

For example, you can use the following statement as a template for inserting owl:Ontology and rdfs:label into the model:

```
<myOntology> a owl:Ontology ;
  rdfs:label "My ontology"^^xsd:string .
```

The named graph URI must match the ontology URI

Make sure that the named graph URI for the model matches the ontology URI. For example:

```
<myOntology> { <myOntology> a owl:Ontology . }
```

Like a linked data set, an ontology is a core component that is used throughout the system. The registries that store and track the graphs for core components, such as the ontology registry, expect that each graph contains a resource that matches the graph URI and specifies the type of graph. Having a mismatched graph and ontology URI can break core Anzo functionality.

Define classes and concepts with owl:Class

Use **owl:Class** for class or concept definitions. Do NOT include skos:Concept or rdfs:Class. For example, the following statement requires modification to make it valid in an Anzo model:

```
<myConcept> a skos:Concept
```

Changing the statement as follows correctly uses owl:Class instead of skos:Concept:

```
<myConcept> a owl:Class ;
  rdfs:label <businessFacingClassLabel> .
```

Define taxonomy with rdfs:subClassOf

Use **rdfs:subClassOf** for taxonomy. Do NOT use skos:broader. For example, the following statement requires modification to make it valid in an Anzo model:

```
<childSkosConcept> skos:broader <parentSkosConcept> .
```

Changing the statement as follows correctly uses rdfs:subClassOf instead of skos:broader:

```
<childOwlClass> rdfs:subClassOf <parentOwlClass> .
```

Define properties as owl:DatatypeProperty or owl:ObjectProperty

Define properties using **owl:DatatypeProperty** or **owl:ObjectProperty**. For example:

```
<myObjectProperty> a owl:ObjectProperty .
```

Or

```
<myDataTypeProperty> a owl:DatatypeProperty .
```

Include rdfs:domain and rdfs:range for all properties

Define **rdfs:domain** and **rdfs:range** for all properties. For example, the following property definition is incomplete:

```
<myObjectProperty> a owl:ObjectProperty .
```

The statement below completes the definition by adding rdfs:label, rdfs:domain, and rdfs:range:

```
<myObjectProperty> a owl:ObjectProperty ;
  rdfs:label <businessFacingPropertyLabel> ;
  rdfs:domain <myClass> ;
  rdfs:range <myOtherClass> .
```

The example below shows a valid data type definition:

```
<myDataTypeProperty> a owl:DatatypeProperty ;
  rdfs:label <businessFacingPropertyLabel> ;
  rdfs:domain <myClass> ;
  <myDataTypeProperty> rdfs:range <literal> .
```

> **Note**
> When defining the property range for integer values, use **xsd:int** instead of xsd:integer.

Reference only Anzo-stored models

Models must be self-contained or include references only to models that are stored in Anzo.

Guidelines

This section lists additional guidelines and important information to know when working with data models in Anzo.

- Property Range Guidelines
- TriG is the preferred format for models to upload
- Load RDFS and OWL vocabularies as graphs
- Axiomatically defined classes and property hierarchies are not processed

Property Range Guidelines

When creating or editing properties in the model editor, Anzo offers several RDF property ranges or data types to choose from. Certain types are preferred over others, however, because they are treated consistently and predictably across systems. Cambridge Semantics recommends that you specify one of the following preferred property range values:

- **Boolean**: For true or false values.
- **Byte**: For 1-byte integers from -128 to 127.
- **Date**: For date values that follow a format such as YYYY-MM-DD.
- **Date time**: For 8-byte date and time values that follow a format such as YYYY-MM-DDThh:mm:ss. Note that dateTime values are normalized to GMT.
- **Double**: For up to 8-byte double floating point values.
- **Duration**: For a duration of time expressed as a number of years, months, days, hours, minutes, and seconds in a format such as PnYnMnDTnHnMnS.
- **Float**: For up to 4-byte floating point values with potential decimal places.
- **Int**: For up to 4-byte integers from -2,147,483,648 to 2,147,483,647.
- **Long**: For up to 8-byte integers from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

- **Short**: For up to 2-byte integers from -32,768 to 32,767.
- **String**: For character values of varying length.
- **Time**: For time values that follow a format such as hh:mm:ss.

TriG is the preferred format for models to upload

Anzo accepts model files in OWL (.owl), RDF (.rdf), TriG (.trig), TTL (.ttl), and XML (.xml) format. The preferred format for models that will be uploaded to Anzo is **TriG** (.trig) format.

Load RDFS and OWL vocabularies as graphs

Anzo loads but does not process additional vocabulary data (such as rdf:subPropertyOf, owl:sameAs, and owl:intersectionOf, etc.) if they are encoded in models. Models that contain vocabularies rather than structural information should be loaded as RDF graphs instead. Anzo can load any valid RDF data. Since RDFS, SKOS, and OWL are valid RDF formats, the vocabulary information can be loaded as a graph, and the data can be interpreted with SPARQL in data layers and Hi-Res Analytics.

Axiomatically defined classes and property hierarchies are not processed

When models include axiomatically defined classes or property hierarchies, Anzo loads the information but does not process the data. For example, Anzo does not infer information from axiomatically defined classes.

Uploading a Model

This topic provides instructions for uploading an existing model to Anzo. Follow these instructions if you have a model that was created outside of Anzo or was downloaded from Anzo as described in Downloading a Model. Anzo accepts model files in OWL (.owl), RDF (.rdf), TriG (.trig), TTL (.ttl), and XML (.xml) format.

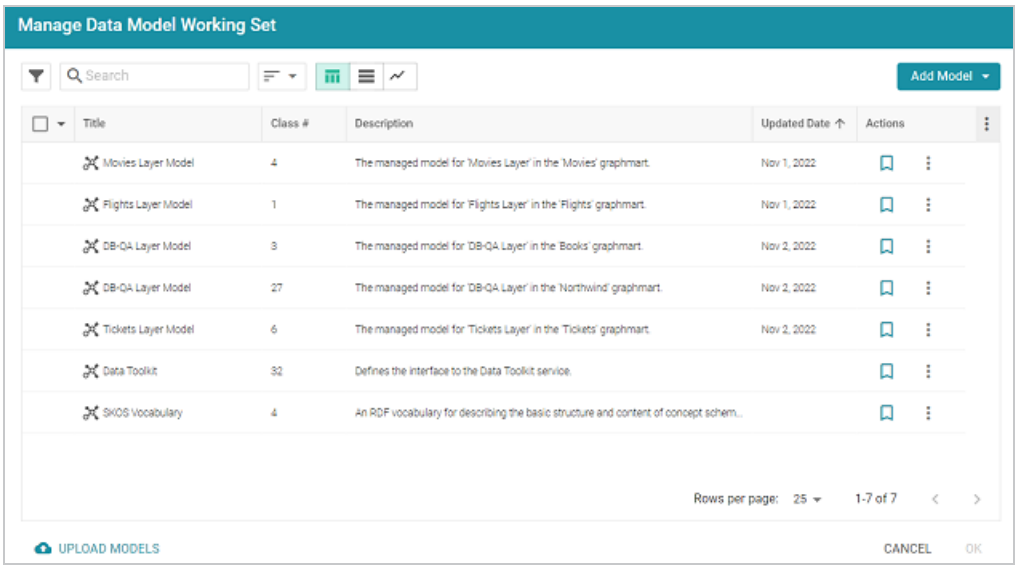
Important
When uploading a model to Anzo, follow the requirements and guidelines defined in Model Requirements.

If you want to import a version of a model that was exported from Anzo (as described in Exporting an Artifact), follow the instructions in Importing Exported Versions of Artifacts to import the model.

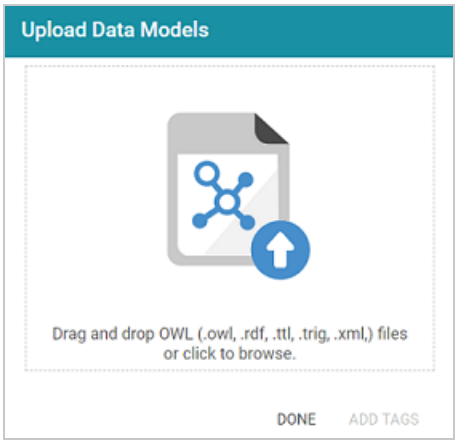
Note
One of the following outcomes will occur if two users upload the same data model:

- If the second user does not have permission to modify the model that the first user uploaded, the second user receives an access denied error and cannot upload the model.
- If the second user does have permission to modify the model that the first user uploaded, Anzo overwrites the existing model with the version from user two.

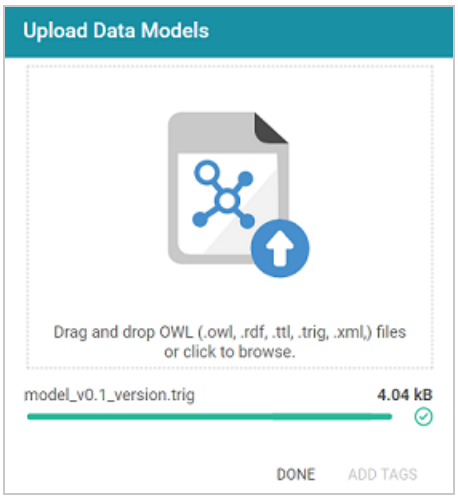
1. In the Anzo application, click **Model**. Anzo displays the Manage Data Model Working Set screen. For example:



2. On the bottom left corner of the screen, click **Upload Models**. The Upload Data Models dialog box opens.



3. To upload a model, drag and drop the file onto the dialog box or click the text to browse and select the file on your computer. Anzo uploads the model that you selected and displays the file name and size. For example:



If you want to upload additional models, you can repeat the process and drag and drop or select files on the Upload Data Models dialog box.

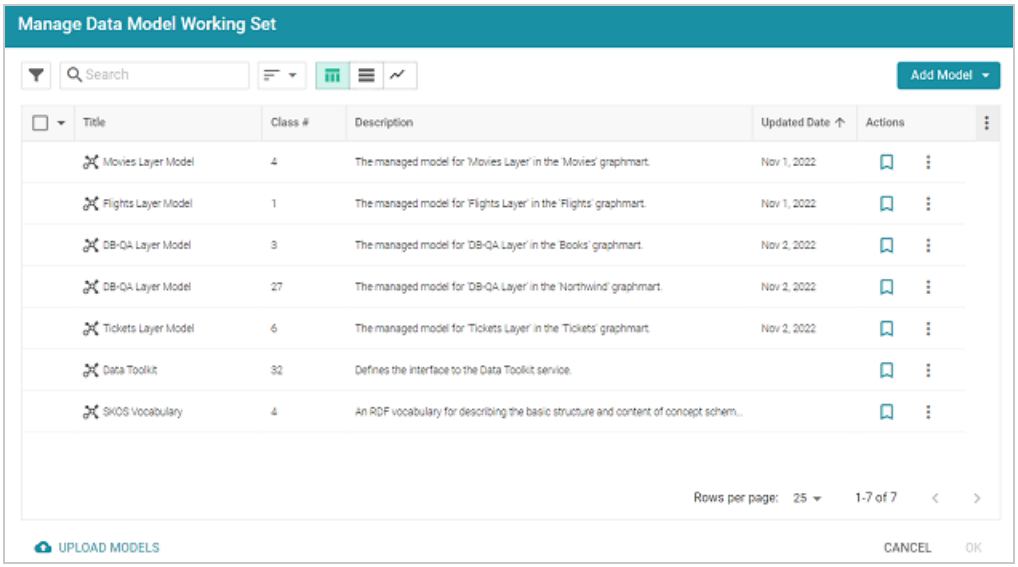
4. If you want to add a tag or edit the tag that was specified in the uploaded model, you can click **Add Tags** and specify the tag in the dialog box. Then click **OK**.
5. Click **Done** when you finish uploading models. The new models become available on the Manage Data Model Working Set screen.

For information about editing models using the model editor, see Editing a Custom Model.

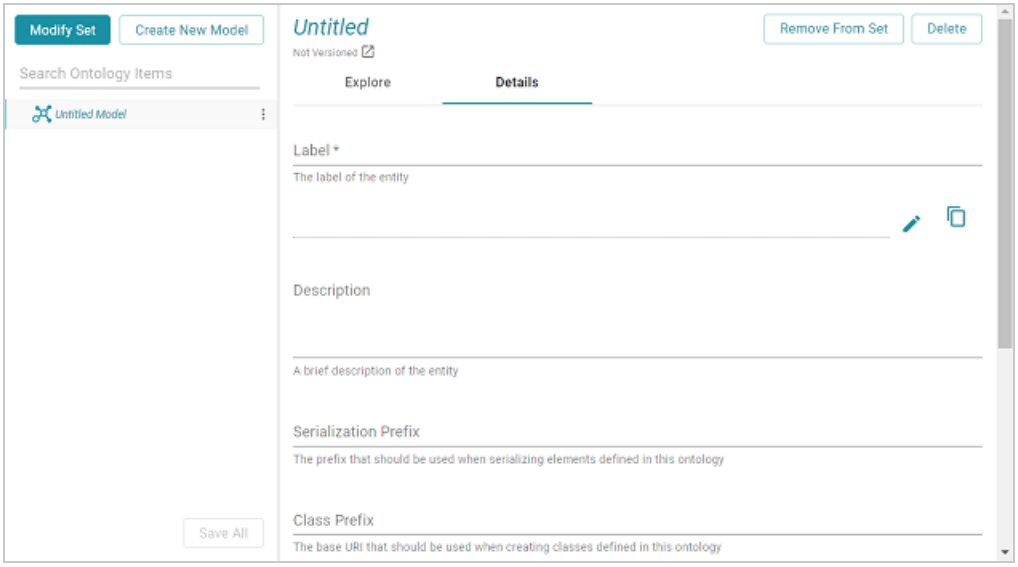
Creating a Model

This topic provides instructions for creating a new data model in the Anzo application. For instructions on uploading an existing model to Anzo, see Uploading a Model.

1. In the Anzo application, click **Model**. Anzo displays the Manage Data Model Working Set screen. For example:



2. Click the **Add Model** button on the top right of the screen and select **New Model**. Anzo displays the Model editor.



3. In the **Label** field, type a unique name for the model.
4. Provide the following optional information as needed:
   - **Description**: A brief description of the model.
   - **Serialization Prefix**: The prefix to use for this model when Anzo serializes it. For example, the prefix for the Friend of a Friend (FOAF) model is "foaf," and the prefix for Dublin Core is "dc."

     > **Tip**
     > The Prefix value is also used to provide hints when typing queries in the Query Builder. When writing a query against a data source that has this model in scope, typing in the PREFIX clause presents this Prefix value as a suggestion.

   - **Class Prefix**: The custom URI template to follow for classes in this model. The value must be a valid URI. When the Class Prefix is set, the URIs for the classes in this model will follow the specified scheme. For example, if Class Prefix is set to **http://cambridgesemantics.com/class/** and a class called **Employees** is created in the model, the URI that is generated for the Employees class will be **http://cambridgesemantics.com/class/Employees**.
   - **Property Prefix**: The custom URI template to follow for properties in this model. The value must be a valid URI. When the Property Prefix is set, the URIs for the properties in this model will follow the specified scheme. For example, if Property Prefix is set to **http://cambridgesemantics.com/property/** and a property called **LastName** is created in the model, the URI that is generated for the LastName property will be **http://cambridgesemantics.com/property/LastName**.
   - **Imports**: Lists any definitions that you want to import from another model into this model. To select models to import, click in the **Imports** field and select a model from the drop-down list. Select the field again to select additional models.
   - **System Model**: Indicates that the data model is a system model only and not related to

business data.

- **Hidden Model**: Hides the data model so that it is not associated with business data.

5. Click **Save** to save the model.

For information about adding classes and properties to the new model, see Editing a Custom Model.

Editing a Custom Model

This topic provides information about using the Anzo model editor to open a data model and modify it to add, edit, or remove classes, properties, data ranges, and annotations.

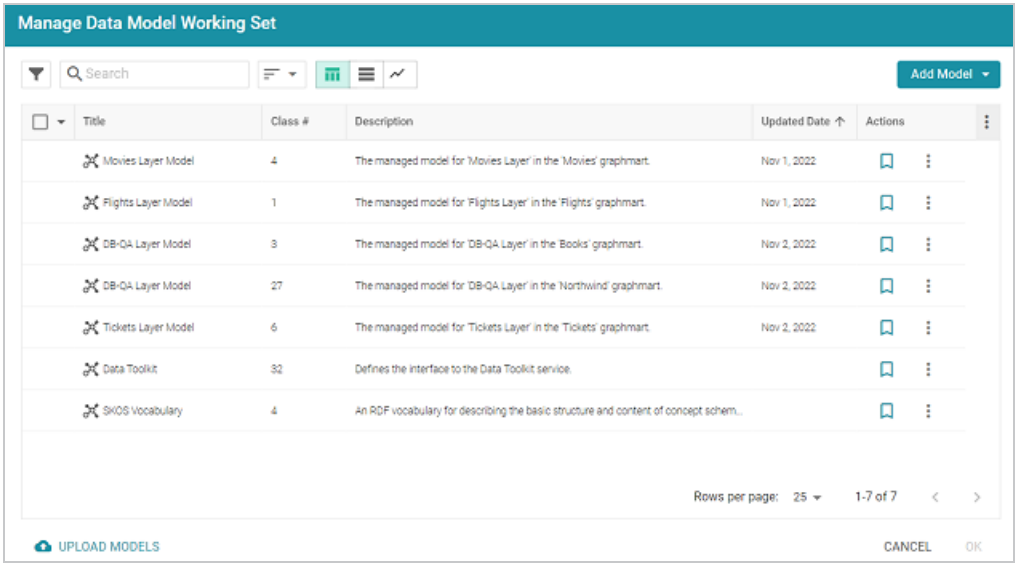
Important
Do not modify auto-generated, layer-managed models. Changes will be overwritten whenever the host graphmart or layer is reloaded or refreshed. For more information, see Managed Model Concepts.

- Opening Models in the Editor
- Changing Model Components
- Class Editor Reference
- Property Editor Reference

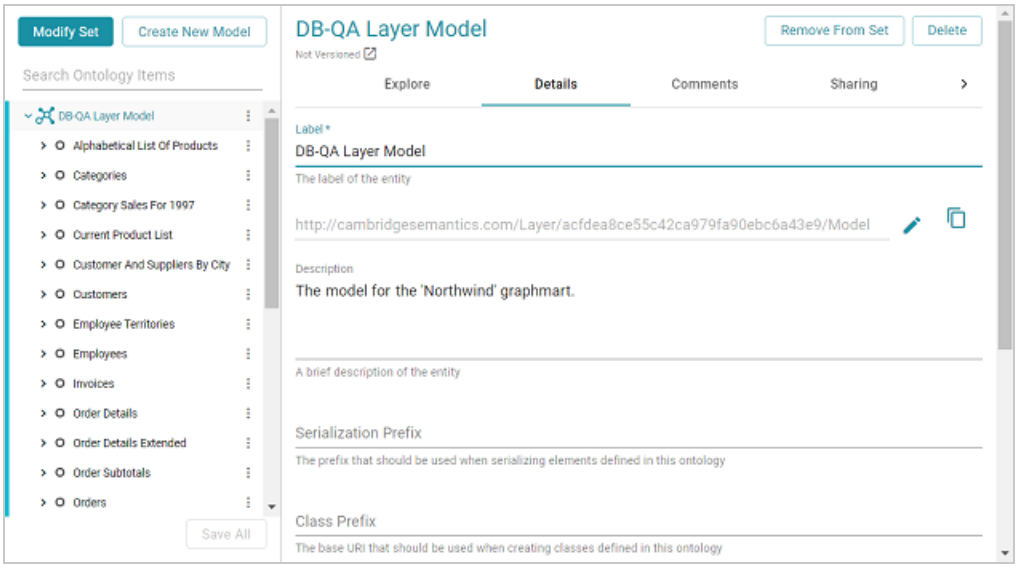
Tip
Before editing a model, you have the option to create a backup of the current version. For more information, see Creating and Restoring Versions of Artifacts.

Opening Models in the Editor

1. In the Anzo application, click **Model**. Anzo displays the Manage Data Model Working Set screen. For example:



2. On the Manage Working Set screen, select the checkbox next to the model (or models) that you want to add to the working set and edit. Then click **OK**. Anzo opens the selected model in the editor. For example:



3. You can edit the following model-level settings or view the Changing Model Components section below for information about working with classes, properties, annotations, and data

ranges.

- **Description**: A brief description of the model.
- **Serialization Prefix**: The prefix to use for this model when Anzo serializes it. For example, the prefix for the Friend of a Friend (FOAF) model is "foaf," and the prefix for Dublin Core is "dc."

  > **Tip**
  > The Prefix value is also used to provide hints when typing queries in the Query Builder. When writing a query against a data source that has this model in scope, typing in the PREFIX clause presents this Prefix value as a suggestion.

- **Class Prefix**: The custom URI template to follow for classes in this model. The value must be a valid URI. When the Class Prefix is set, the URIs for the classes in this model will follow the specified scheme. For example, if Class Prefix is set to **http://cambridgesemantics.com/class/** and a class called **Employees** is created in the model, the URI that is generated for the Employees class will be **http://cambridgesemantics.com/class/Employees**.
- **Property Prefix**: The custom URI template to follow for properties in this model. The value must be a valid URI. When the Property Prefix is set, the URIs for the properties in this model will follow the specified scheme. For example, if Property Prefix is set to **http://cambridgesemantics.com/property/** and a property called **LastName** is created in the model, the URI that is generated for the LastName property will be **http://cambridgesemantics.com/property/LastName**.
- **Imports**: Lists any definitions that you want to import from another model into this model. To select models to import, click in the **Imports** field and select a model from the drop-down list. Select the field again to select additional models.
- **System Model**: Indicates that the data model is a system model only and not related to business data.
- **Hidden Model**: Hides the data model so that it is not associated with business data.

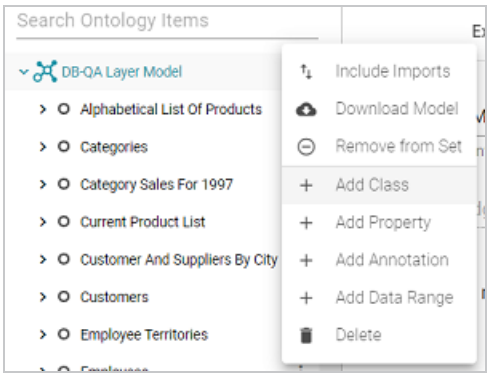
Changing Model Components

The sections below provides instructions for working with model components. When modifying models, make sure that you click **Save** periodically to save your changes.

- Creating a New Class
- Creating a New Property
- Adding an Existing Property to a Class
- Editing a Class
- Deleting a Property
- Deleting a Class
- Adding a Data Range
- Adding an Annotation

Creating a New Class

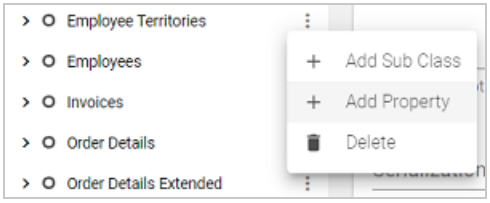
Open the model menu by clicking the menu icon (⋮) to the right of the model name. Then select **Add Class**.



Anzo opens the class editor so that you can configure the new class. See Class Editor Reference below for information about class settings.

Creating a New Property

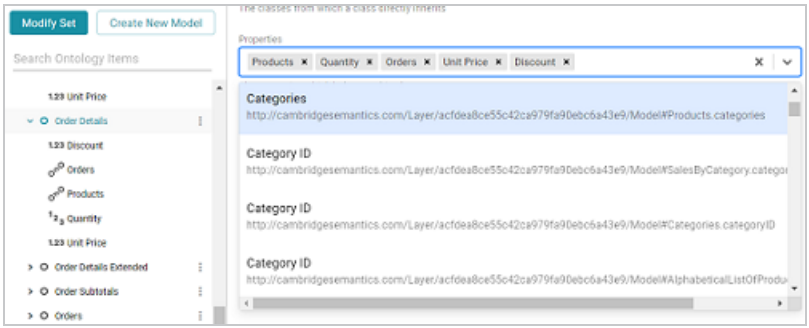
Open the class menu by clicking the menu icon (⋮) to the right of the class name. Then select **Add Property**.



Anzo opens the property editor so you can configure the new property. See Property Editor Reference below for information about property settings.

Adding an Existing Property to a Class

To add an existing property to a class, click the class in the left pane to display the class details in the editor. In the editor, click in the **Properties** field and select the property that you want to add from the drop-down list. For example:



Editing a Class

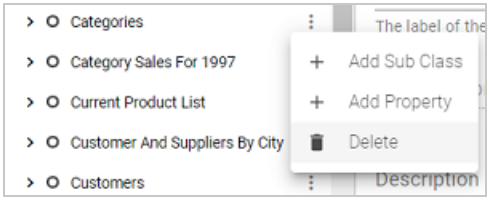
To change an existing class, select the class in the left pane. Anzo expands the class to show its properties and displays the details for that class in the editor. You can make changes in the editor. See Class Editor Reference below for information about class settings.

Deleting a Property

In the left pane of the working set, select the property that you want to delete. Anzo opens that property in the editor. To remove the property, click the **Delete** button on the top right of the screen. Then click **Delete** in the dialog box to confirm that you want to delete the property.

Deleting a Class

Click the menu icon (⋮) to the right of the class that you want to remove from the model.



Click **Delete**. Anzo displays a dialog box that asks if you want to delete only the class or all of the subclasses and properties in the class. Select the appropriate option and then click **Delete** to confirm that you want to delete the class. **This action cannot be undone.**Anzo removes the class and saves the model.

Adding a Data Range

Click the menu icon (⋮) to the right of the model name. Then select **Add Data Range**. Anzo opens the data range editor so that you can configure the new range.

Adding an Annotation

Click the menu icon (⋮) to the right of the model name. Then select **Add Annotation**. Anzo opens the editor so that you can configure the annotation.

Class Editor Reference

This section describes each of the fields that are available for configuring classes.

Field	Description
Label	The name of the class.
Description	A brief description of the class.
Parent Classes	Lists any parent classes under which this class becomes a child or subclass. Click in the field to select parent classes from the drop-down list. Or click the X to the left of a class name to remove that parent class from the list.
Properties	Lists the properties under this class. Click in the field to a property from the drop-down list. Or click the X to the right of a property name to remove that property from the list.
Inherited Properties	Properties that the class has inherited from a super class or the model.
Preview Property	Defines a property from the class to use as the "name" or entity on default displays. For example, if there is a reference to entity X, and entity X has Name, Title, and Label properties, you could specify that you want Title to display by default instead of "X."
Resource Template	Defines the Uniform Resource Identifier (URI) template to use for instances of the class. You can construct URI templates by typing a value and pressing **Enter** or by choosing an available property from the drop-down list. For more information, see Defining Resource Templates.
Graph Template	Defines the graph URI template to use for instances of the class. You can construct graph URI templates by typing a value and pressing **Enter** or by choosing an available property from the drop-down list. You can concatenate the

Field	Description
	specified graph template value with values of properties in the class. For example, `http://cambridgesemantics.com/graph/` and `Title`

Property Editor Reference

This section describes each of the fields that are available for configuring properties.

Field	Description
Label	The name of the property.
Description	A brief description of the property.
Required	Indicates whether a value is required for this property.
Multi Value	Indicates whether more than one value can exist for this property. **Note** Some business intelligence (BI) applications have limitations on the retrieval of multi-value properties. If you use the Anzo Data on Demand service to query data from BI tools, consider whether your application supports multi-value properties before creating them.
Has Data Range	Indicates whether the property has a single data type or a data range. Selecting this checkbox displays the Data Range field so that you can choose the data range.
Property Range	The data type for the property. See Property Range Guidelines for recommendations on choosing property ranges.
Domain	Lists the class or classes that the property belongs to.
Min Cardinality	The minimum number of distinct values a property can have. When Min Cardinality is blank, the number of values is unrestricted.
Max	The maximum number of distinct values a property can have. When Max

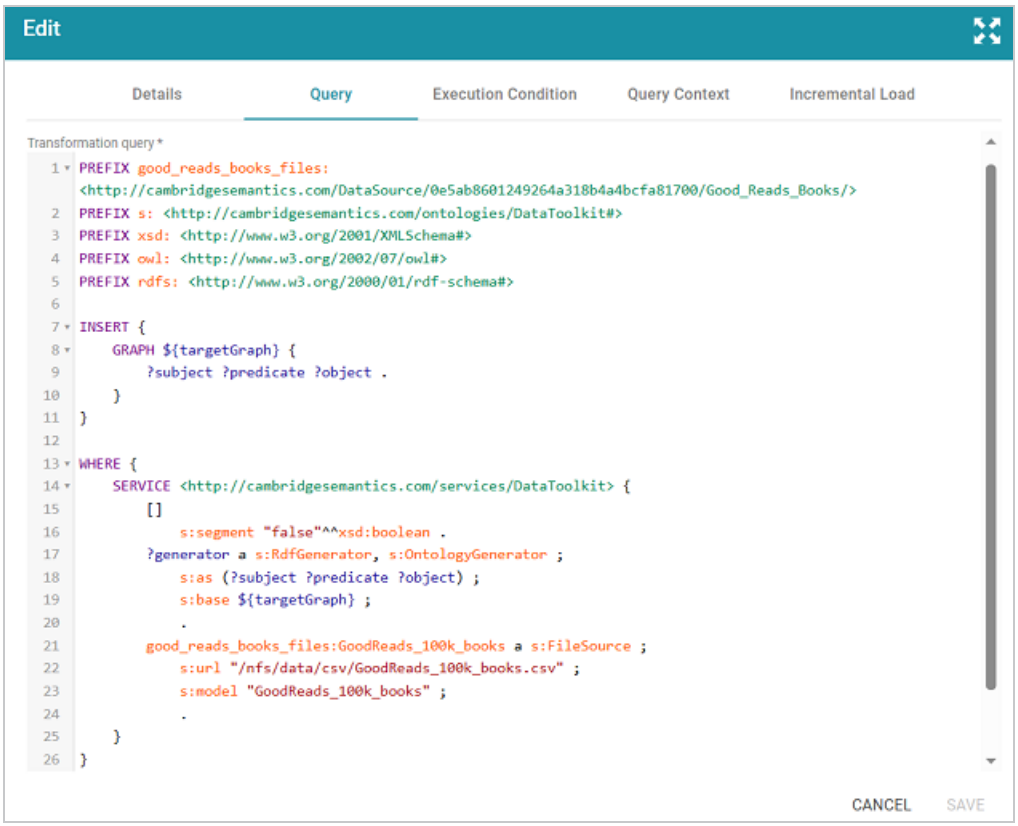
Field	Description
Cardinality	Cardinality is blank, the number of values is unrestricted.
Value Restriction	Indicates whether to restrict the property’s values to certain data types or specific values in a list.

Editing a Managed Model

Managed models that are generated by a Direct Load Step are owned and managed by the data layer that contains the Direct Load Step. Any changes made to the model outside of the step, such as from the Model editor, are overwritten any time the graphmart or layer is refreshed or reloaded. This topic provides guidance on updating the properties in a managed model by editing the Direct Load Step query.

> **Tip**
> Before starting the procedure, you might want to start another session of Anzo. In the new session, open in the Model editor the model that you will be updating. That way you can copy model, class, and property URIs from the model while you edit the Direct Load Step query.

1. In the graphmart for which you want to update a model, click the **Data Layers** tab. Expand the layer that was generated when you created the graphmart and find the Direct Load Step that inserts the data.
2. Open the Direct Load Step for editing and click the **Query** tab. For example, the image below shows the query that was generated to onboard data about books from a CSV file:



3. Make sure the following prefixes are declared in the PREFIX clause at the top of the query. If any are missing, add them to the query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

4. Next, add the following GRAPH clause statements under the SERVICE clause:

```
GRAPH <model_URI> {
  <property_URI> a owl:DatatypeProperty;
    rdfs:comment "comment" ;
    rdfs:label "property_label" ;
    rdfs:domain <class_URI> ;
    rdfs:range <datatype> .
[ <property2_URI> a owl:DatatypeProperty;
    rdfs:comment "comment" ;
    rdfs:label "property2_label" ;
    rdfs:domain <class_URI> ;
    rdfs:range <datatype> . ]
```

```
[ ... ]
}
```

- **model_URI**: The URI of the model that was generated by the Direct Load Step. You can copy the URI from the Model editor from the Details tab for the model. For example:

  ```
  <http://cambridgesemantics.com/Layer/698d17.../Model>
  ```

- **property_URI**: The URI to use for the new property or the URI of the existing property that you want to update. When adding a property, you can look at the URIs for other properties in the model and follow the same scheme. For example:

  ```
  <http://cambridgesemantics.com/Layer/6.../Model#GoodReads100kBooks.FirstPr
  int>
  ```

- **comment**: A string that describes the property. For example, `"Date the book first went to print"`.
- **property_label**: The label to give the property. For example, `"First Print Date"`.
- **class_URI**: The URI of the class that the existing property belongs to or the new property should be added to. You can copy the URI from the Model editor from the Details tab for the class. For example:

  ```
  <http://cambridgesemantics.com/Layer/698d1794.../Model#GoodReads100kBooks>
  ```

- **datatype**: The URI for the datatype of the property. For example, `xsd:date`.

For example, the following query adds one new property to the model that is generated by the Direct Load Step query shown above:

```
PREFIX good_reads_books_files:
<http://cambridgesemantics.com/DataSource/0e5ab8601249264a318b4a4bcfa81700/Goo
d_Reads_Books/>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT {
  GRAPH ${targetGraph} {
    ?subject ?predicate ?object .
```

```
  }
}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    GRAPH
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model> {

<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model#Goo
dReads100kBooks.FirstPrint> a owl:DatatypeProperty;
      rdfs:comment "Date the book first went to print" ;
      rdfs:label "First Print Date" ;
      rdfs:domain
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model#Goo
dReads100kBooks> ;
      rdfs:range xsd:date .
  }
    []
      s:segment "false"^^xsd:boolean .
    ?generator a s:RdfGenerator, s:OntologyGenerator ;
      s:as (?subject ?predicate ?object) ;
      s:base ${targetGraph} .

    good_reads_books_files:GoodReads_100k_books a s:FileSource ;
      s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
      s:model "GoodReads_100k_books" .
  }
}
```

5. Next, if you added properties and the class for any of the new properties does not have a primary key defined, you must create a key for that class. If all of the classes you referenced in the query have primary keys, you can continue to the next step. If one or more of the classes do not have primary keys, follow the instructions below:

   a. Locate in the query the statement block for each class that needs a key defintion. For example, in the query above, there is only one class, `s:model "GoodReads_100k_Books"`. If you have multiple classes, the query has several blocks, such as this example:

```
...
emrdbsmall:emr_complaint a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
```

```
  s:table "emrdbsmall.emr_complaint" ;
  s:model "emr_complaint" .

emrdbsmall:emr_patient a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
  s:table "emrdbsmall.emr_patient" ;
  s:model "emr_patient" .

emrdbsmall:emr_complaintdescription a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
  s:table "emrdbsmall.emr_complaintdescription" ;
  s:model "emr_complaintdescription" .
...
```

b. At the end of the block for the class you want to add a key to, change the period (.) after `s:model` to a semicolon (;).

c. Next, add the following line below `s:model`:

```
s:key ("key_property" [, "key_property2" ] [, ... ]) .
```

Where `key_property` is the label of the property to use as a key for the class. The property that you choose must have unique values. If there is not a property in the class with unique values, you can specify a combination of properties that would create a unique value. Make sure that the value of `key_property` matches the label for that property in the model. For example, for the query in step 4, the `Isbn` property can be used as a unique key for the `GoodReads_100k_Books` class:

```
good_reads_books_files:GoodReads_100k_books a s:FileSource ;
  s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
  s:model "GoodReads_100k_books" ;
  s:key ("Isbn") .
```

The final, completed query is shown below:

```
PREFIX good_reads_books_files:
<http://cambridgesemantics.com/DataSource/0e5ab8601249264a318b4a4bcfa81700
/Good_Reads_Books/>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT {
  GRAPH ${targetGraph} {
    ?subject ?predicate ?object .
  }
}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    GRAPH
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l> {

<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l#GoodReads100kBooks.FirstPrint> a owl:DatatypeProperty;
      rdfs:comment "Date the book first went to print" ;
      rdfs:label "First Print Date" ;
      rdfs:domain
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l#GoodReads100kBooks> ;
      rdfs:range xsd:date .
  }
    []
      s:segment "false"^^xsd:boolean .
    ?generator a s:RdfGenerator, s:OntologyGenerator ;
      s:as (?subject ?predicate ?object) ;
      s:base ${targetGraph} .

    good_reads_books_files:GoodReads_100k_books a s:FileSource ;
      s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
      s:model "GoodReads_100k_books" ;
      s:key ("Isbn") .
  }
}
```

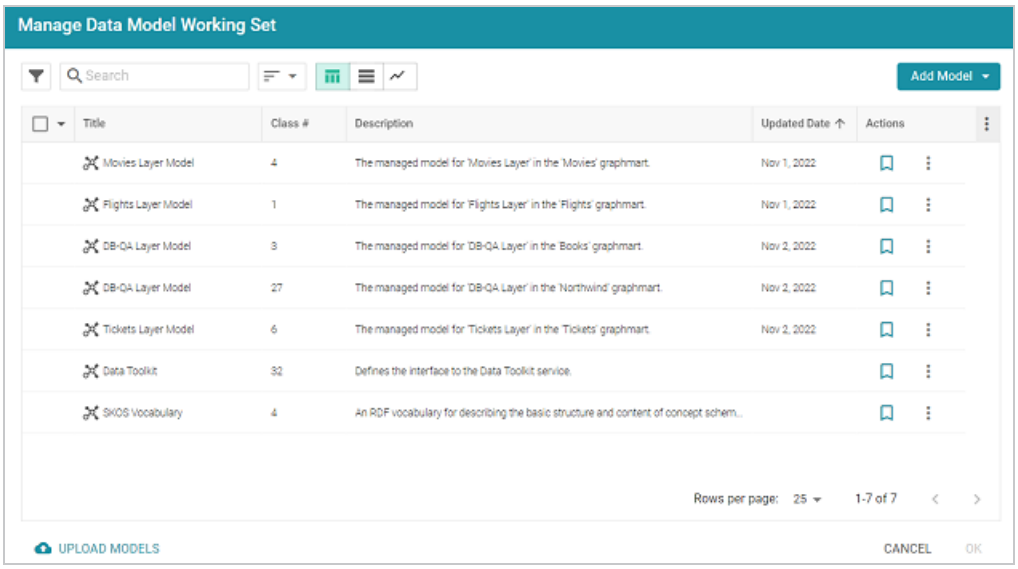
6. Save the step and then refresh or reload the graphmart or layer to update the model with the new properties.

For more information about Direct Load Steps, see Directly Load a Data Source (Direct Load Step).

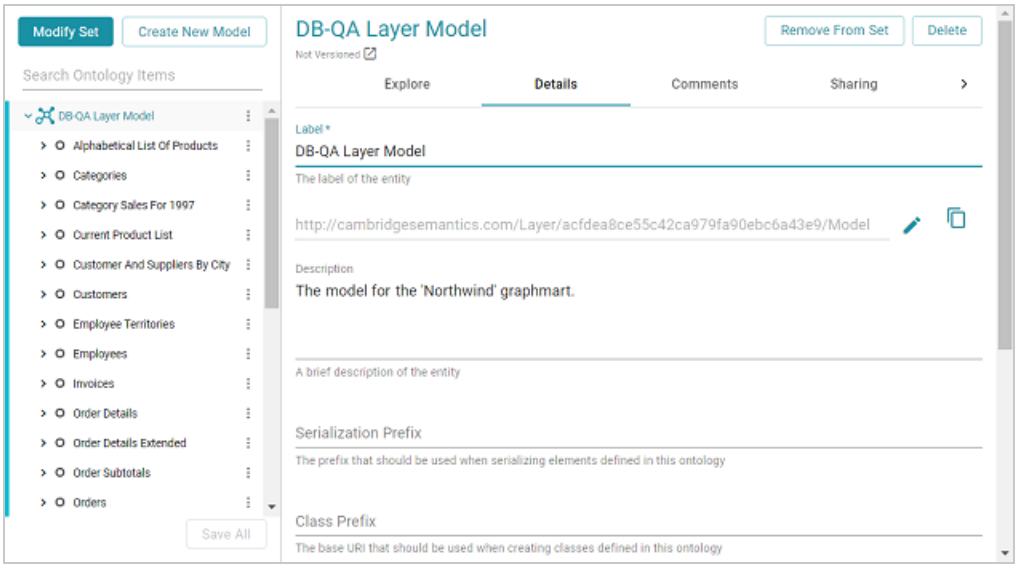
Downloading a Model

This topic provides instructions for downloading a data model from Anzo to your computer.

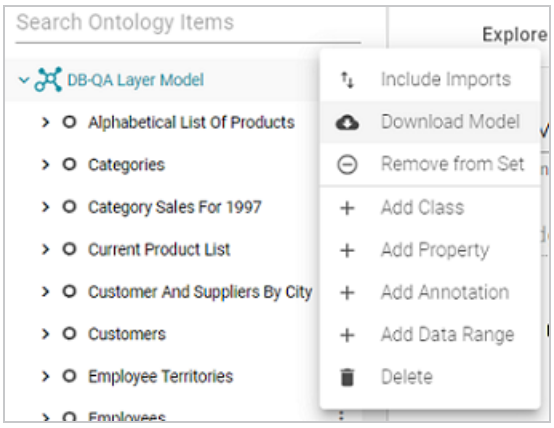
1. In the Anzo application, click **Model**. Anzo displays the Manage Data Model Working Set screen. For example:



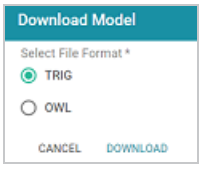
2. On the Manage Working Set screen, select the checkbox next to the model that you want to download, and then click **OK**. Anzo opens the selected model in the editor. For example:



3. Open the model menu by clicking the menu icon (⋮) to the right of the model name. Then select **Download Model**.



Anzo displays the Download Model dialog box:



4. In the Download Model dialog box, select the format to save the model in. By default Anzo saves models in **TRIG** format. If you want to save the file in OWL format, select the **OWL** radio button. Then click **Download**.

Anzo downloads the model to your computer in the selected format.

> **Note**
> When a model is downloaded from Anzo, the resulting TriG or OWL file size can be significantly larger than the file size of the original data model file that was uploaded. The original model likely includes prefix specifications and abbreviated URIs. When a model is exported, however, Anzo replaces the prefixes with full URIs. In addition, the downloaded model includes the Anzo-generated metadata for the model.

For example, the following simple example TTL content shows part of a data model that uses prefixes:

```
@prefix csi: <http://cambridgesemantics.com/2017/02/ont#> .
csi:testModel a owl:Ontology ;
  rdfs:label "Test Model"^^xsd:string .
csi:DOB a owl:Class;
  rdfs:domain csi:Demographics ;
  rdfs:label "DOB" ;
  rdfs:range xsd:string .
csi:HEIGHT a owl:Class;
  rdfs:domain csi:Demographics ;
  rdfs:label "HEIGHT" ;
  rdfs:range xsd:decimal .
```

After uploading the TTL file and then downloading the model in TriG format, the resulting file includes full URIs as well as the model's metadata:

```
<http://cambridgesemantics.com/2017/02/ont#testModel> {
  <http://cambridgesemantics.com/2017/02/ont#DOB> a
<http://www.w3.org/2002/07/owl#Class> ;
    <http://www.w3.org/2000/01/rdf-schema#domain>
<http://cambridgesemantics.com/2017/02/ont#Demographics> ;
    <http://www.w3.org/2000/01/rdf-schema#label> "DOB" ;
    <http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2001/XMLSchema#string> .

  <http://cambridgesemantics.com/2017/02/ont#HEIGHT> a
<http://www.w3.org/2002/07/owl#Class> ;
    <http://www.w3.org/2000/01/rdf-schema#domain>
<http://cambridgesemantics.com/2017/02/ont#Demographics> ;
    <http://www.w3.org/2000/01/rdf-schema#label> "HEIGHT" ;
    <http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2001/XMLSchema#decimal> .
  <http://cambridgesemantics.com/2017/02/ont#testModel> a
<http://www.w3.org/2002/07/owl#Ontology> ;
    <http://www.w3.org/2000/01/rdf-schema#label> "Test Model" .
}
<http://cambridgesemantics.com/registries/Ontologies> {
  <http://cambridgesemantics.com/registries/Ontologies>
    <http://openanzo.org/ontologies/2008/07/Anzo#defaultNamedGraph>
    <http://cambridgesemantics.com/2017/02/ont#testModel> ;
    a <http://openanzo.org/ontologies/2008/07/Anzo#Dataset> .
}
<http://openanzo.org/metadataGraphs
(http%3A%2F%2Fcambridgesemantics.com%2F2017%2F02%2Font%23testModel)> {
```

```
  <http://cambridgesemantics.com/2017/02/ont#testModel>
    <http://openanzo.org/ontologies/2008/07/Anzo#canBeAddedToBy>
    <http://openanzo.org/system/internal/sysadmin> ;
  <http://openanzo.org/ontologies/2008/07/Anzo#canBeReadBy>
    <http://openanzo.org/Role/everyone> ,
<http://openanzo.org/system/internal/sysadmin> ;
    <http://openanzo.org/ontologies/2008/07/Anzo#canBeRemovedFromBy>
    <http://openanzo.org/system/internal/sysadmin> .
  ...
}
```

Defining Resource Templates

When you open a data model in the Model editor, there is a **Resource Template** setting for each of the classes in the model. A Resource Template defines the Uniform Resource Identifier (URI) pattern that Anzo should follow when ingesting data and generating the URIs for the instances of each class. Defining a Resource Template for the classes in your models helps link and relate data by using URI patterns that express the meaning of the data and combine similar concepts. Additionally, simpler and more meaningful URIs are easier to read and therefore easier to write in queries.

Important

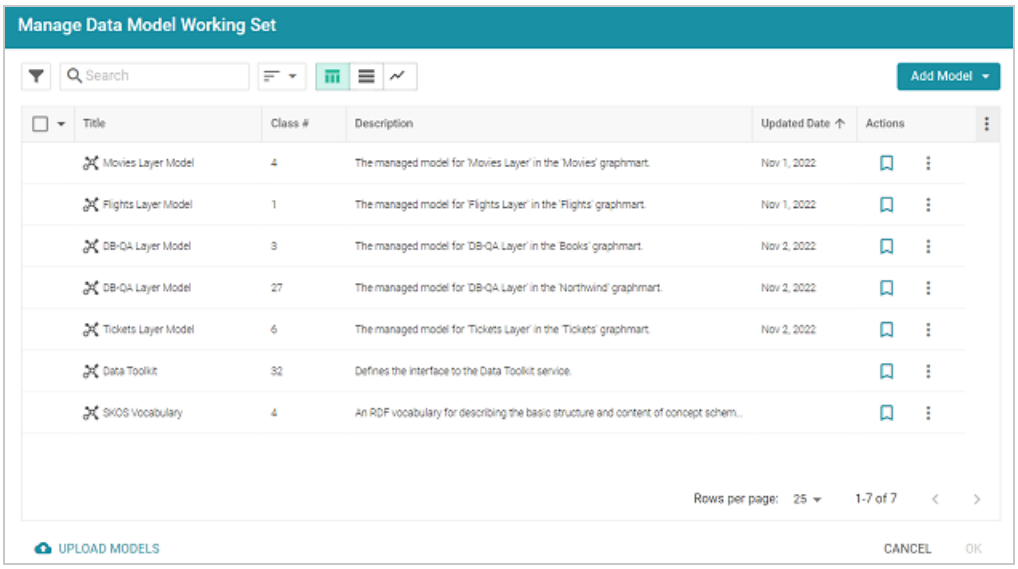
Keep the following points in mind when defining class instance URI patterns:

- Do not modify auto-generated, layer-managed models. Changes will be overwritten whenever the host graphmart or layer is reloaded or refreshed. For more information, see Managed Model Concepts.
- Avoid joining data that should not be joined. For example, using a property such as YearProduced in a movies Resource Template would group all movies from a given year as a single instance.
- Resource Templates with multiple components must have all components present. If a component is missing, Anzo generates random strings for missing Resource Template components.
- Resource templates do not work across different classes. You must define resource templates on individual classes.

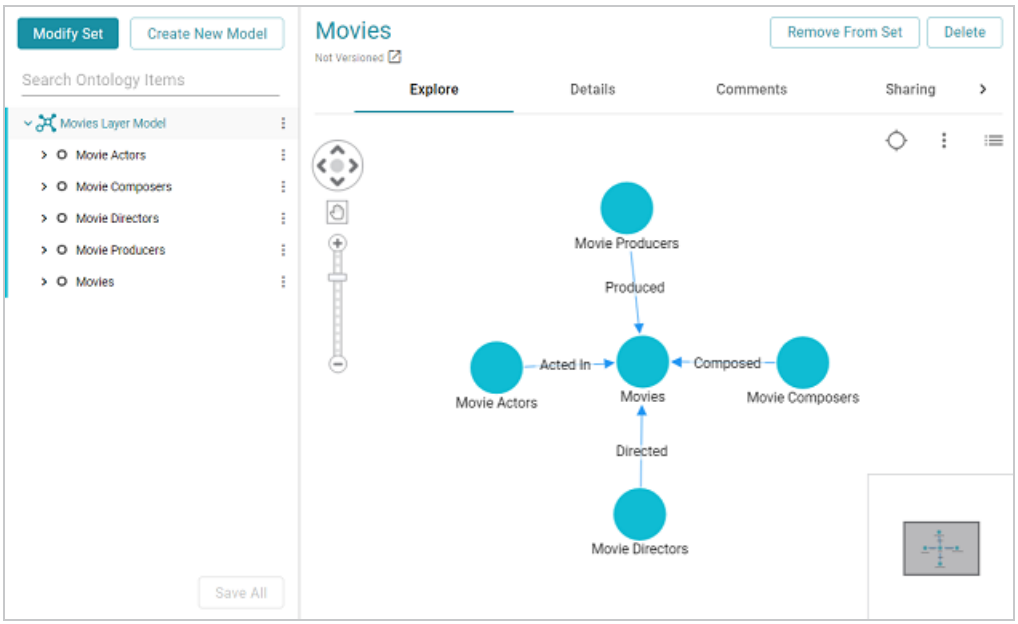
Tip

For property URIs, the default URI prefix is **http://cambridgesemantics.com/**. The value is controlled by the URI Prefix option in server settings. See Configure URI Prefix and SPARQL Options in the Administration Guide for more information.

1. In the Anzo application, click **Model**. Anzo displays the Manage Data Model Working Set screen. For example:

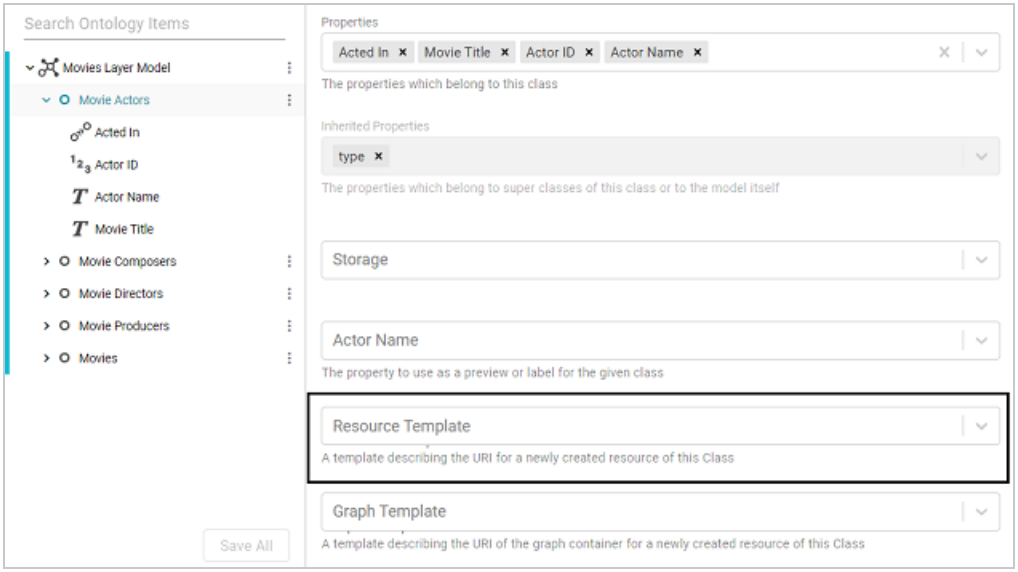


2. On the Manage Working Set screen, select the checkbox next to the model (or models) that you want to add to the working set for editing. Then click **OK**. Anzo opens the selected model in the editor. For example:

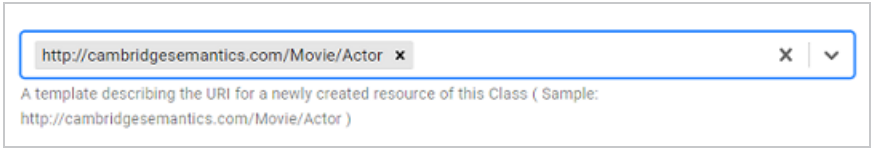


3. Click the **Details** tab.

4. Select a class in the model to display the settings for that class. Then scroll down to the **Resource Template** field. For example, the image below shows the Resource Template field for the selected Movie Actors class.



5. Click the **Resource Template** field and specify the URI pattern to use for instances of this class. First, type a base value in the field and press **Enter** to add the value to the field. For example, for Movie Actors in the step above: `http://cambridgesemantics.com/Movie/Actor`.



Then click the field again and select a property in the class that defines the class, i.e., contains unique values. For example, in the Movie Actors class, Actor ID provides unique values.



6. Click **Save** to save the change, and then select another class for which to set a Resource Template. Repeat the step above for each class in the model.

Blend

This section provides information about working with datasets and blending data in graphmarts.

In this section:

Working with Datasets 442

Working with Graphmarts 481

Profiling Datasets and Graphmarts 604

Working with Datasets

The topics in this section provide guidance on working with datasets in the Datasets catalog.

In this section:

Adding an Empty Dataset for an Export Step 443

Importing an Existing Dataset (FLDS) 446

Creating a Dataset from RDF Files 451

Managing Dataset Editions 456

Creating a Graphmart from a Dataset 466

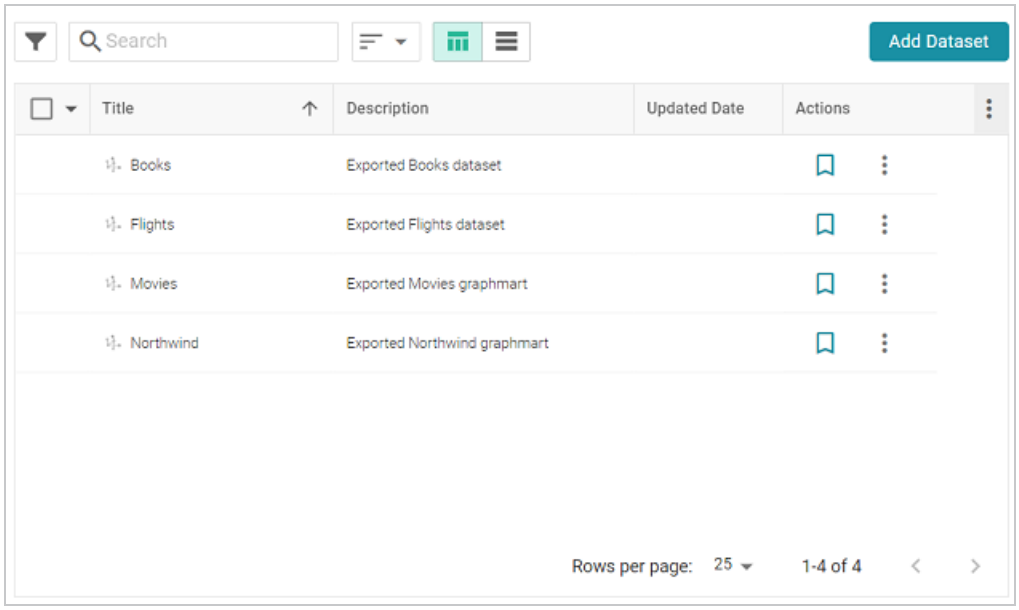
Adding a Dataset to a Graphmart 471

Dataset FAQ 475

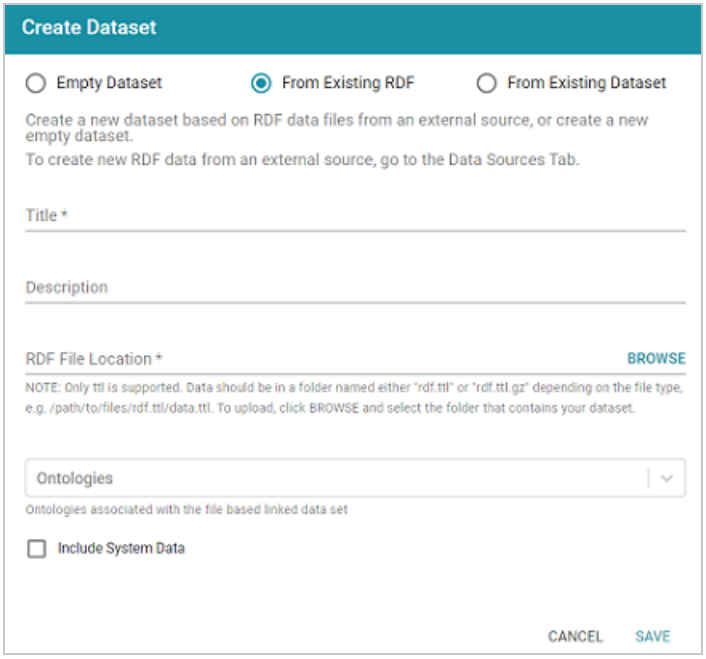
Adding an Empty Dataset for an Export Step

Follow the steps below to create an empty dataset that can be used to create an FLDS from the data that is output from an Export Step.

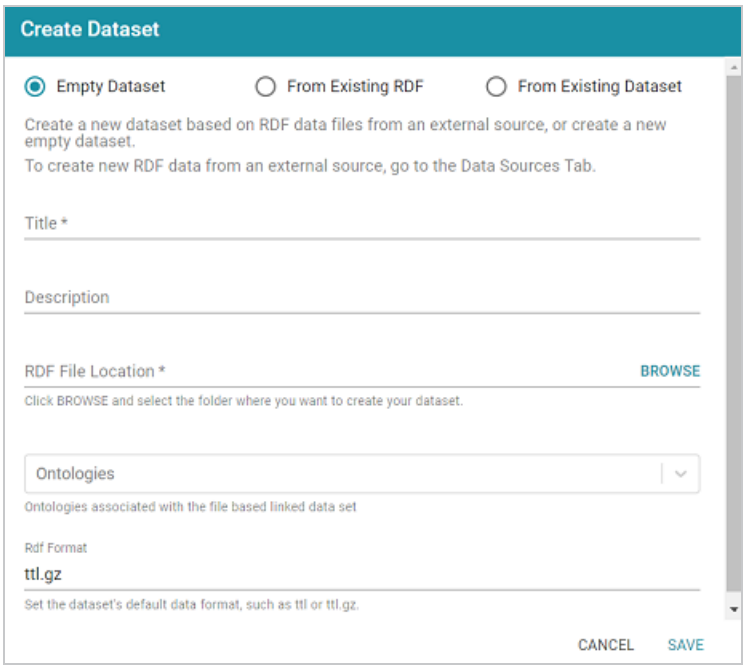
1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. On the Datasets screen, click **Add Dataset**. Anzo opens the Create Dataset dialog box.

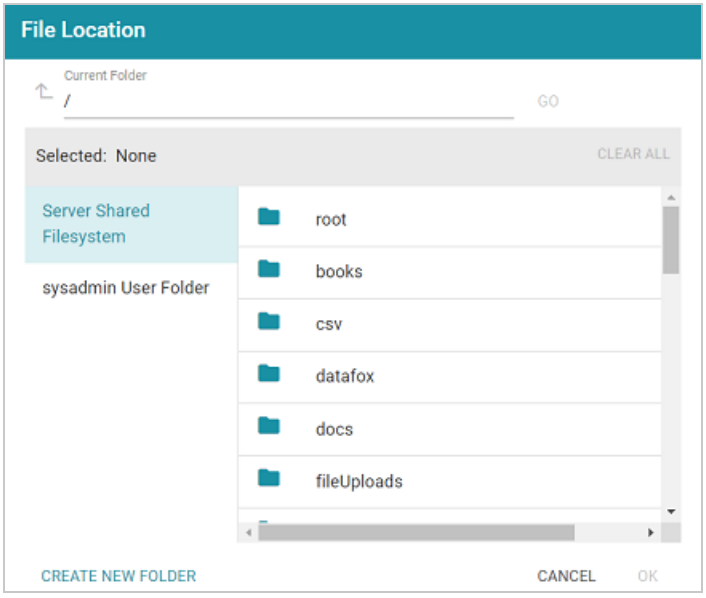


3. Select the **Empty Dataset** radio button.



4. Type a name for the new dataset in the **Title** field and an optional description in the **Description** field.

5. Click the **RDF File Location** field to open the File Location dialog box.



6. Find and select the directory where the FLDS for this dataset should be created. If needed, you can click **Create New Folder** to create a new directory. Then click **OK** to close the dialog box and return to the Create Dataset screen.
7. Next, if you know which model or models are associated with the data that you plan to export, select the models from the **Ontologies** drop-down list. Leave **Ontologies** blank if you do not know which models apply or do not want to select one at this time. When the Export steps runs, Anzo automatically exports any models that are related to the exported data.
8. Lastly, the RDF Format setting defaults to **ttl.gz**, meaning files that are output to this dataset will be in compressed Turtle format. This is the ideal format for preserving space on the file store. If you do not want the files to be compressed, you can change the value to **ttl**.
9. Click **Save** to save the empty dataset and return to the Datasets screen. The new dataset becomes available as a selection when choosing the Target FLDS for an Export Step.

For more information about Export Steps, see Export Data to an FLDS (Export Step).

Importing an Existing Dataset (FLDS)

Follow the instructions below to add an existing FLDS to the Datasets catalog. Make sure that the FLDS meets the following File Requirements.

File Requirements

To add an FLDS to the Datasets catalog, the location of the files, the file format, and the directory structure must meet the following requirements:

- **Supported File Locations**: Files can be staged on a configured file store or they can be uploaded from your computer as a .zip file.
- **Supported Directory Structure**: FLDS directories should contain an **flds.trig** file, an **onts** directory that includes the model .trig file, and an **rdf.ttl** or **rdf.ttl.gz** directory that contains the data files. For example:

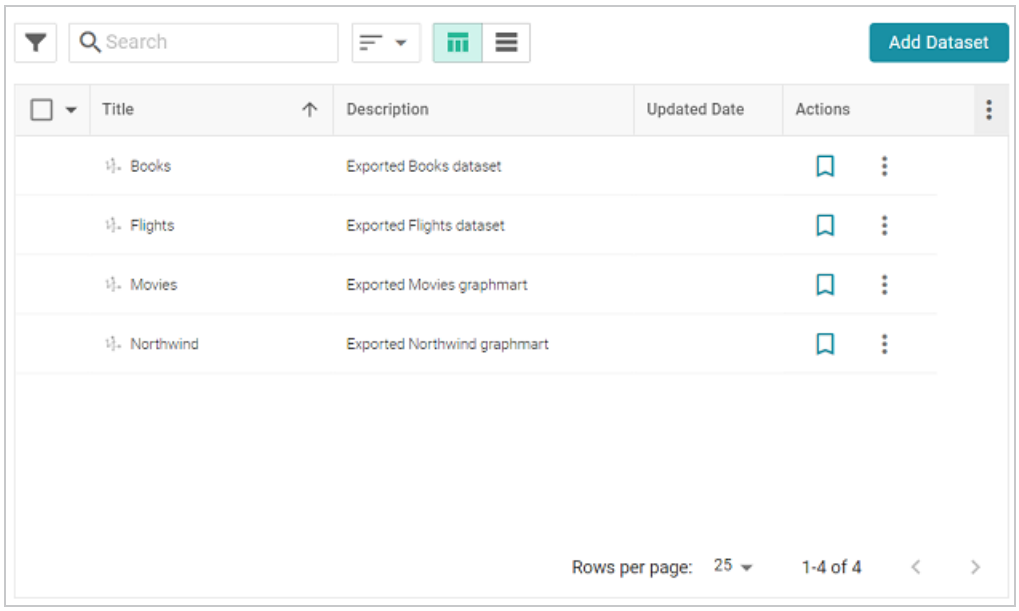
  > **Note**
  > Models must be in TriG format, regardless of the file type of the data files.

```
LoadEmployees_f7b1f
├── flds.trig
├── onts
│   └── Employees.trig
└── rdf.ttl.gz
    └── Loadnew_employees_8be23.ttl.gz
        └── 20191021034225.ttl.gz
            └── part-00000.ttl.gz
            └── part-00001.ttl.gz
             └── part-00003.ttl.gz
```

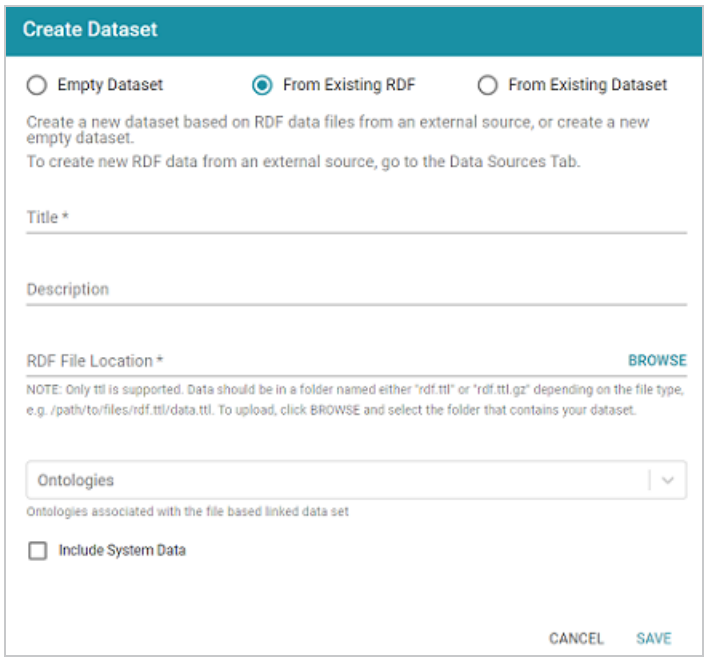
Importing an FLDS

Follow the steps below to import an FLDS from the file store or from a .zip file on your computer.

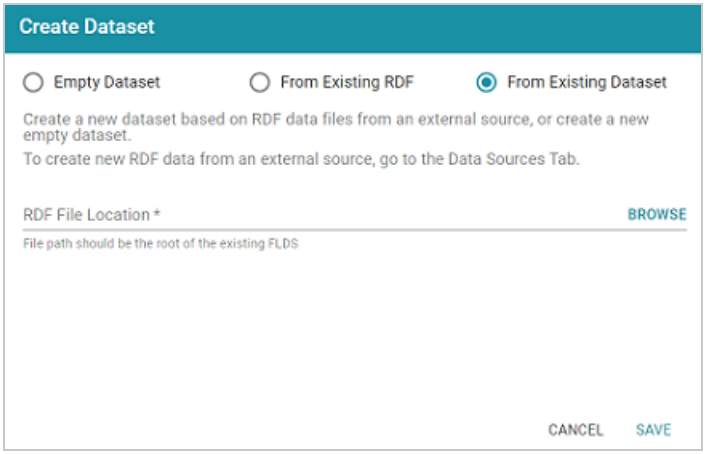
1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



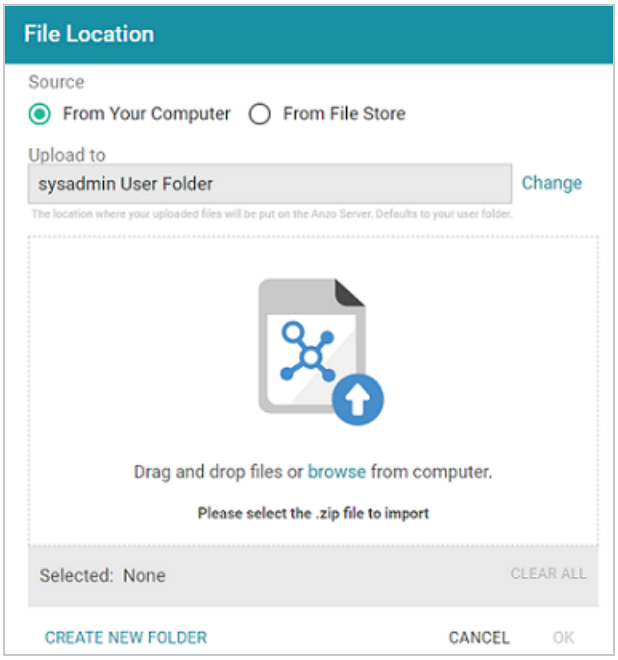
2. On the Datasets screen, click **Add Dataset**. Anzo opens the Create Dataset dialog box.



3. Select the **From Existing Dataset** radio button.



4. Click the **RDF File Location** field to open the File Location dialog box.



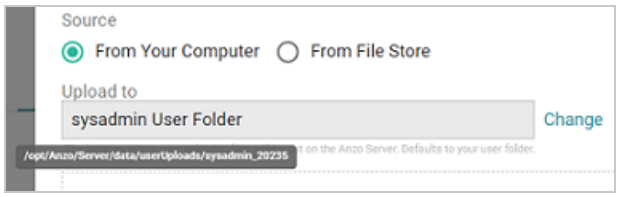
5. Follow the appropriate steps below depending on whether the FLDS is on your computer or the shared File Store:

   If the file is on your computer:

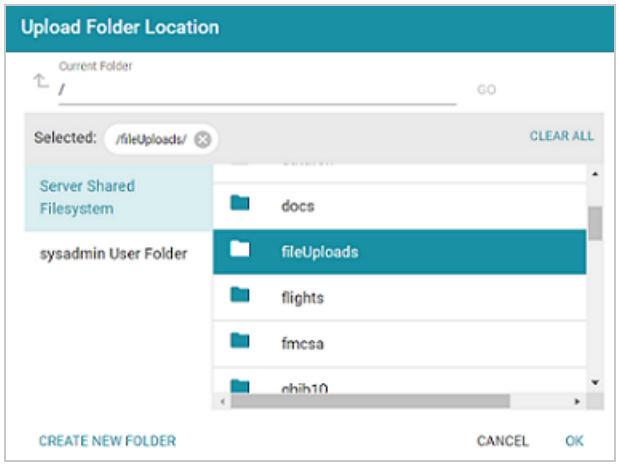
   a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is a directory on the shared file store and not in the server installation path. If the file is not

uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they typically do not have access to the file location.

For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`



If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.

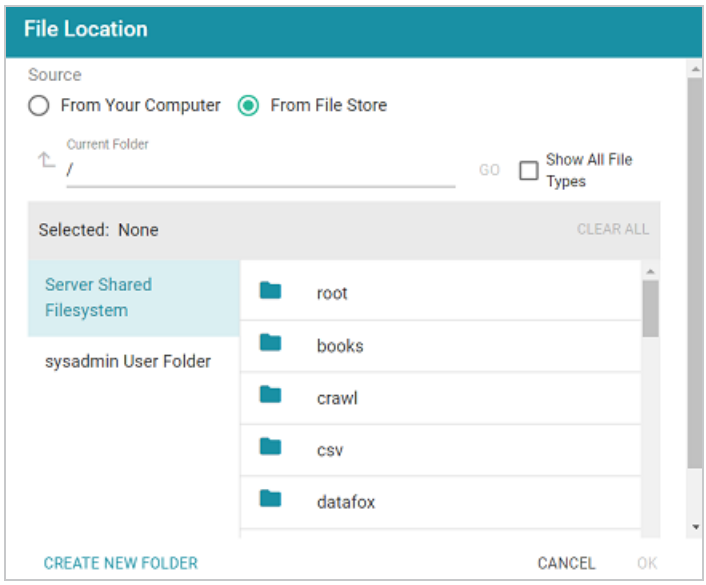


b. Drag and drop the file onto the screen or click **browse** to navigate to the file and select it. Anzo attaches the file and the **OK** button becomes active.

c. Click **OK**. The file is added to the RDF File Location field on the Create Dataset screen.

If the files are on the File Store:

a. Click the **From File Store** radio button. Anzo displays the file selection dialog box. For example:



b. On the left side of the screen, select the file store that hosts the FLDS. On the right side of the screen, navigate to the root directory for the dataset. This is the directory that contains the **flds.trig** file, the **onts** directory, and the **rdf.ttl** or **rdf.ttl.gz** directory.

c. Select the root directory for the FLDS and then **click OK**. The location is added to the RDF File Location field on the Create Dataset screen.

6. Click **Save** to import the FLDS and return to the Datasets screen.

You can now select the dataset in the catalog and create a new graphmart or add the dataset to an existing graphmart. See Creating a Graphmart from a Dataset or Adding a Dataset to a Graphmart for instructions.

Creating a Dataset from RDF Files

Source data that is not in RDF format is onboarded through the automated direct data load workflow or unstructured pipelines, where the data is converted to RDF format. If you have data that is already in RDF format in Turtle or N-Triple files, those files can be added to the Datasets catalog directly, making the data available to add to a graphmart for loading and analyzing in AnzoGraph.

> **Note**
> To import data from CSV, JSON, XML, Parquet, or SAS files, follow the processes described in Adding Data Sources.

Follow the instructions below to create a dataset from a directory of Turtle or N-Triple files. Make sure that the files and directory meet the requirements in File Requirements.

File Requirements

To add data to the Dataset catalog, the location of the files, the file format, and the directory structure must meet the following requirements:

- **Supported File Locations**: Files can be staged on a configured file store, or they can be uploaded from your computer as a .zip file.
- **Supported File Formats**: Files must be in one of the following formats:
  - Turtle (.ttl file type)
  - N-Triple (.n3 and .nt file types)

  Either of the file types listed above can be compressed in GZIP format and named as **<filename>.<filetype>.gz** files.
- **Supported Directory Structure**: When importing RDF files that are not part of an FLDS, the files must be placed in a directory named **rdf.<filetype>** or **rdf.<filetype>.gz**. Stage uncompressed TTL files in a directory called **rdf.ttl**, and stage compressed TTL files in a directory called **rdf.ttl.gz**. Stage uncompressed N-Triple files in a directory called **rdf.nt** or **rdf.n3**, depending on the file type extension. Place compressed files in an **rdf.nt.gz** or **rdf.n3.gz** directory. For example:

```
External-RDF-Top-Level-Directory
└── rdf.ttl.gz
    └── external-rdf-file1.ttl.gz
    └── external-rdf-file2.ttl.gz
    └── external-rdf-file3.ttl.gz
```

Important

All files inside an rdf.<filetype> or rdf.<filetype>.gz directory must be the same format and end in the same extension. Data in mixed formats will not load successfully. If you plan to import multiple file types, organize files into separate directories by file extension type, and then import each directory separately.

Note

To upload files from your computer, use the same directory structure as shown above. Zip the top-level directory so that the upload file is `External-RDF-Top-Level-Directory.zip` and contains the `rdf.ttl.gz` directory.

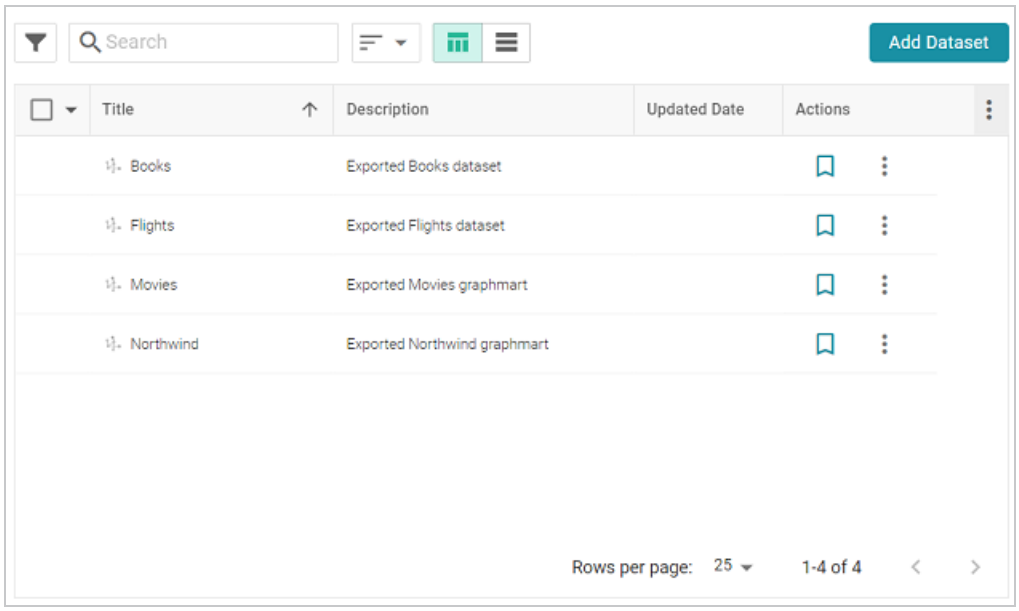
Importing RDF Files

Follow the steps below to create a dataset from RDF files.

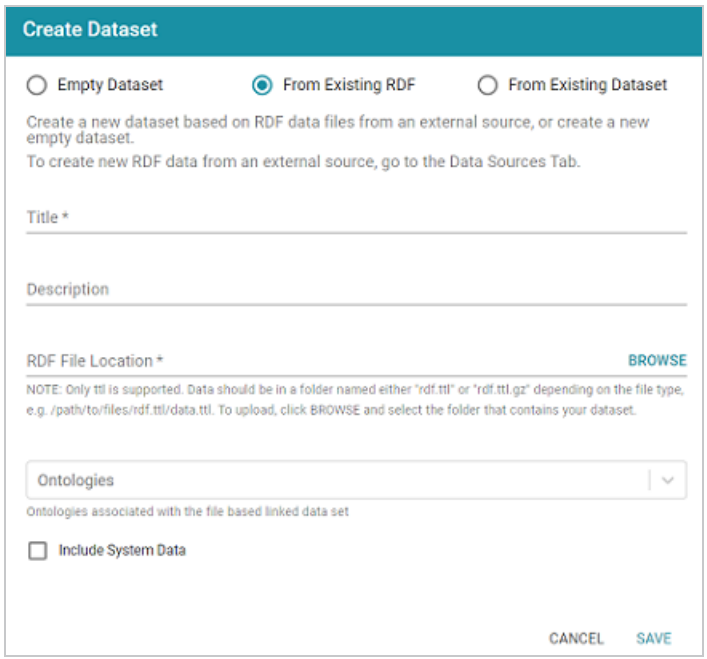
Tip

Anzo provides the option to link the files to an existing data model during the import. If the model is not yet available in Anzo, consider uploading it before importing the RDF files. See Uploading a Model for instructions. You are not required to include a model at import time; a model can be associated with a data set at any time. How do I associate a model with a dataset?

1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. On the Datasets screen, click **Add Dataset**. Anzo opens the Create Dataset dialog box.

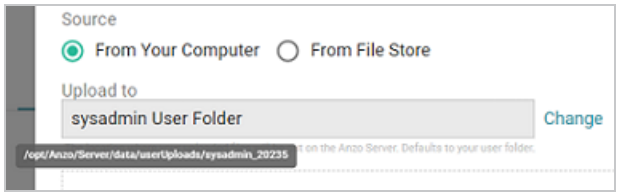


3. The **From Existing RDF** radio button is selected by default. Type a name for the new dataset in the **Title** field and an optional description in the **Description** field.
4. Click the **RDF File Location** field to open the File Location dialog box and follow the appropriate steps below depending on the location of the files.

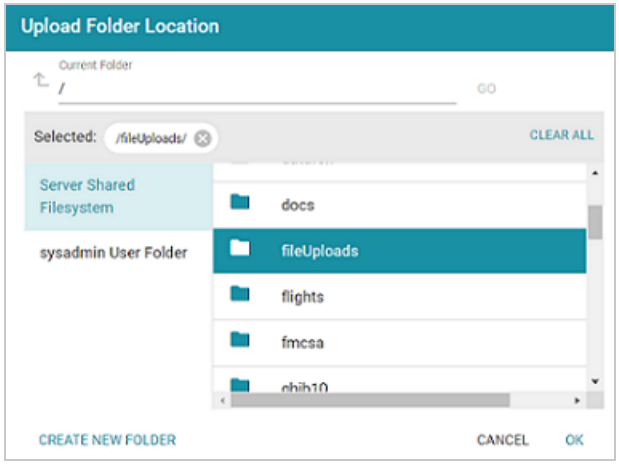
If you are uploading a .zip file from your computer:

a. As a best practice, check the upload location that is listed in the **Upload To** field by hovering your pointer over the value to view the tooltip. Make sure the upload location is a directory on the shared file store and not in the server installation path. If the file is not uploaded to the shared file store it is not accessible by applications like AnzoGraph. In addition, other users cannot create graphmarts from the data source because they typically do not have access to the file location.

   For example, viewing the Upload To location for the screen above shows that the file will be uploaded to the server installation path, `/opt/Anzo/Server/data...`



   If your Upload To location is configured to upload the file to the server installation path, click **Change** and select an upload location that is on the shared file store. For example, the image below shows the Upload Folder Location dialog box that is presented after clicking **Change**. A folder called **fileUploads** is selected on the shared store.



b. Drag and drop the .zip file with the RDF files onto the screen or click **Browse** to navigate to the file on your computer and select it.

c. Click **OK** to close the dialog box and return to the Create Dataset screen.

If the files are on the File Store

a. Select the **From File Store** radio button.

b. Find and select the **rdf.<filetype>** directory that you want to import, and then click **OK** to close the dialog box and return to the Create Dataset screen.

5. If you want to associate a model with this dataset, click the **Ontologies** drop-down list and select the model. To include a system model, select the **Include System Data** checkbox. If you do not want to associate a model with the data at this time, leave the Ontologies field blank.

> **Note**
> Datasets without a model cannot be viewed in Hi-Res Analytics dashboards, but the imported data can still be queried. A model can be associated with the data set at a later time. How do I associate a model with a dataset?

6. Click **Save**. Anzo creates the FLDS and adds the new dataset to the Datasets catalog, and return to the Datasets screen.

> **Note**
> Anzo generates an flds.trig file at the same level as the rdf.<filetype> directory. The file contains metadata about the load files.

You can now select the dataset in the catalog and create a new graphmart or add the dataset to an existing graphmart. See Creating a Graphmart from a Dataset or Adding a Dataset to a Graphmart for instructions.

Managing Dataset Editions

The topics in this section introduce the concepts to know when working with editions and provide instructions for creating, deleting, and modifying editions.

- Introduction to Editions
- Creating an Edition
- Modifying an Edition
- Deleting a Saved Edition
- Limiting the Number of Editions in a Dataset

Introduction to Editions

Editions are collections of the data components that are published by an unstructured pipeline or generated by an Export Step in a graphmart. Editions can be assembled by users and can include any subset of components. This topic introduces you to the concepts that are helpful to know when working with editions.

- What is a Data Component?
- What is the Managed Edition?
- What is a Saved Edition?

What is a Data Component?

A **Data Component** is the data that is generated by a successful run of an unstructured pipeline or by the processing of an Export Step in a graphmart. Each time a pipeline or step runs to completion, a new data component is created that contains the version of the data that was generated by that run.

All data components are automatically included in the **Managed Edition** (see What is the Managed Edition?) and any of the components can be added to a **Saved Edition** (see What is a Saved Edition?).

What is the Managed Edition?

When an unstructured pipeline runs or an Export Step is processed, the resulting data is added to the **Managed Edition**. This Edition is managed by Anzo and always contains all of the published data components.

The Managed Edition cannot be changed, but it can be cloned (via the **Actions** menu) and saved as a **Saved Edition**. Saved Editions can be modified.

What is a Saved Edition?

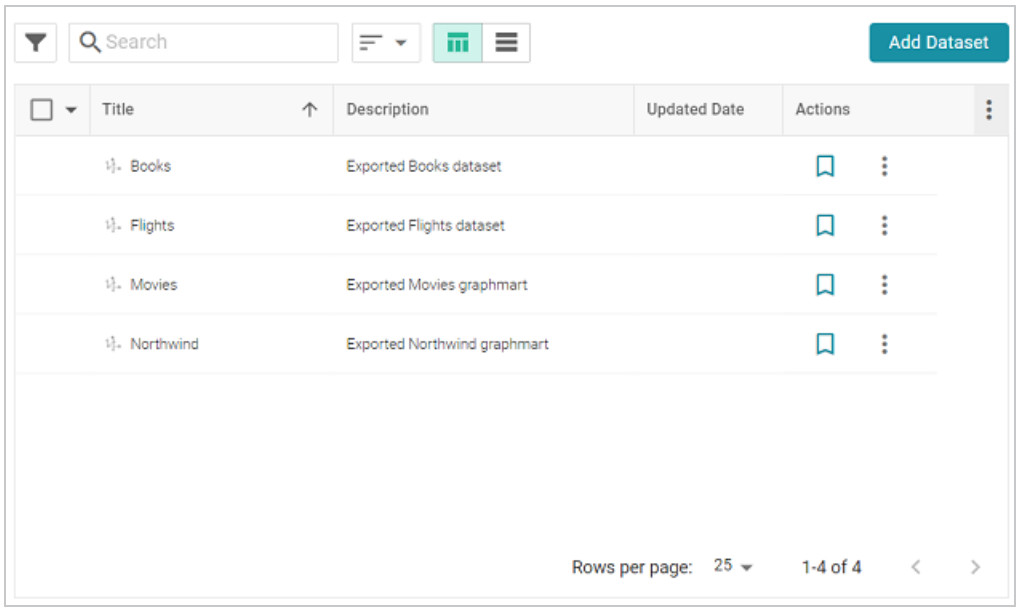
A **Saved Edition** is a user-assembled collection of data components. A Saved Edition can contain any combination of and any version of data components. Saved Editions can be created from scratch or can be cloned from the Managed Edition or another Saved Edition.

The Managed Edition or any Saved Edition can be added to a graphmart for analysis.

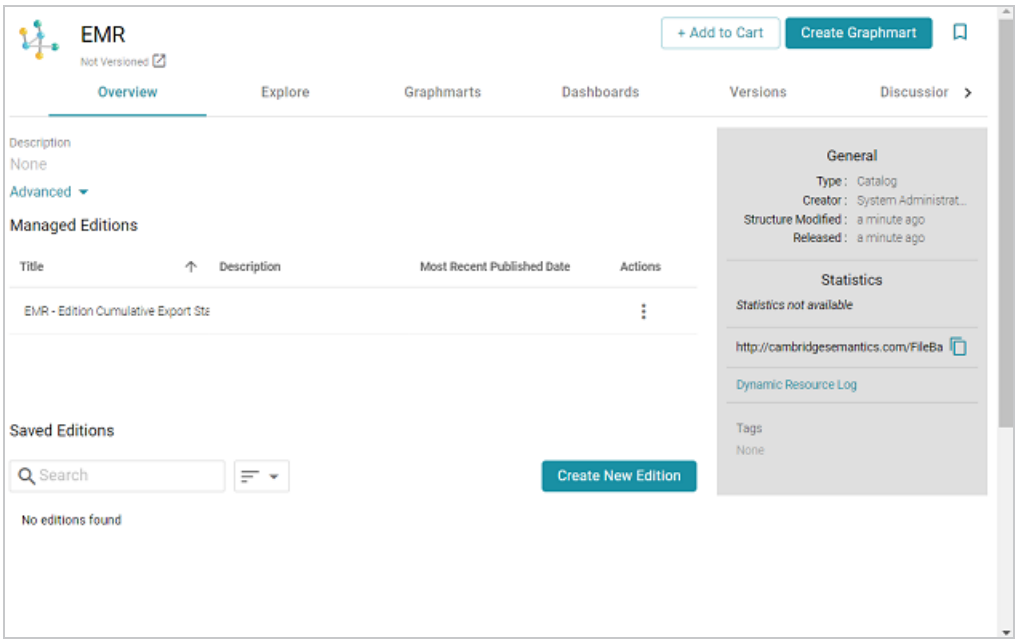
Creating an Edition

Follow the instructions below to create a new edition.

1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



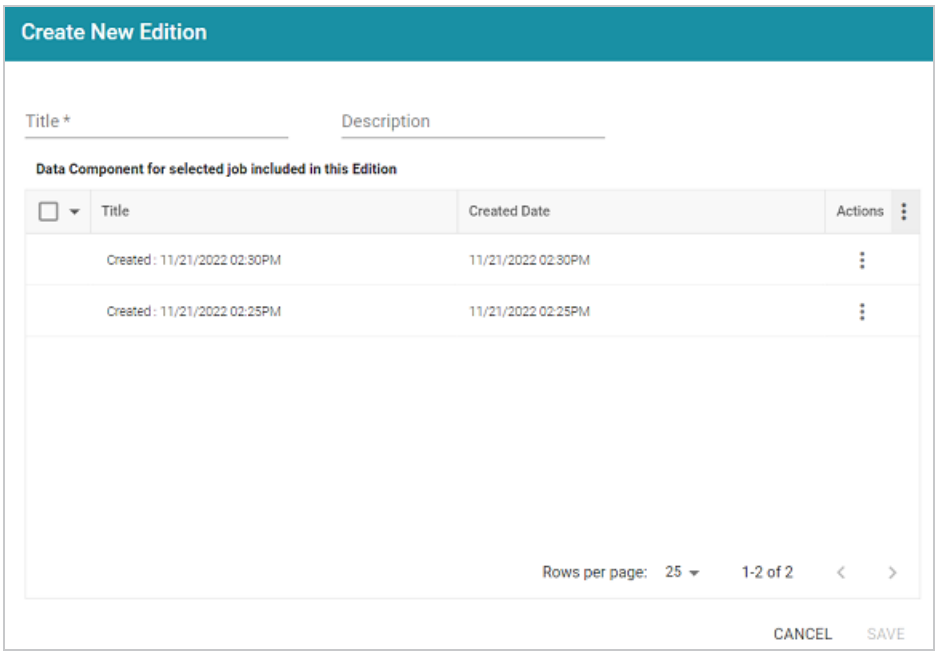
2. Click the dataset for which you want to create an edition. Anzo displays the Explore tab for the dataset. Click the **Overview** tab, which lists the existing editions. For example:



3. To create a new edition from scratch, click the **Create New Edition** button at the bottom of the screen.

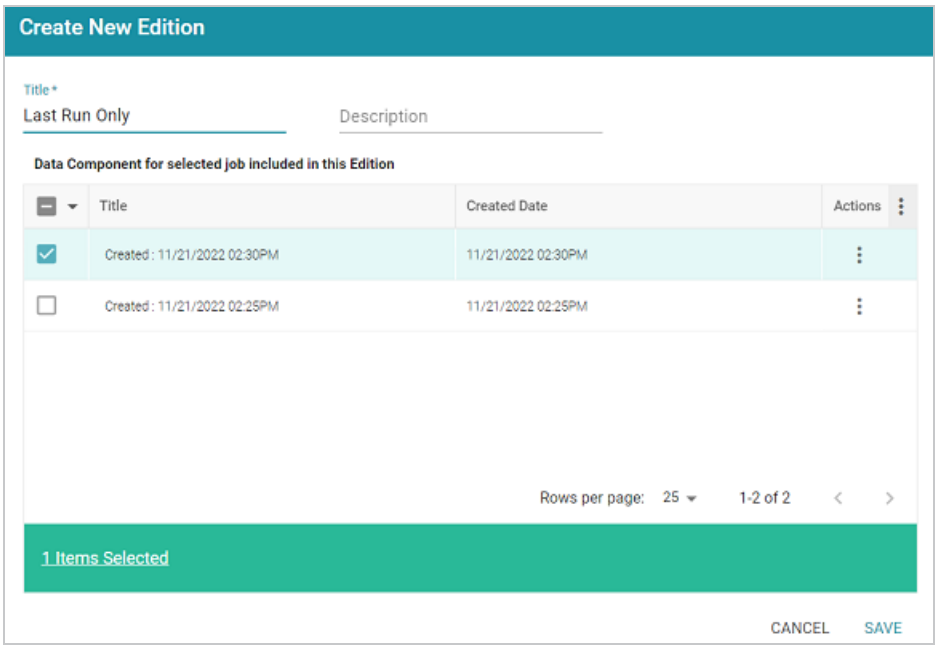
> **Tip**
> If you want to create an edition by cloning the Managed Edition, click the menu icon (⋮) in the Actions column for the Managed Edition and select **Clone Edition**. To clone a Saved Edition, click the menu icon for the Saved Edition and select **Clone Edition**.

The Create New Edition (or Clone Edition) screen is displayed. For example:

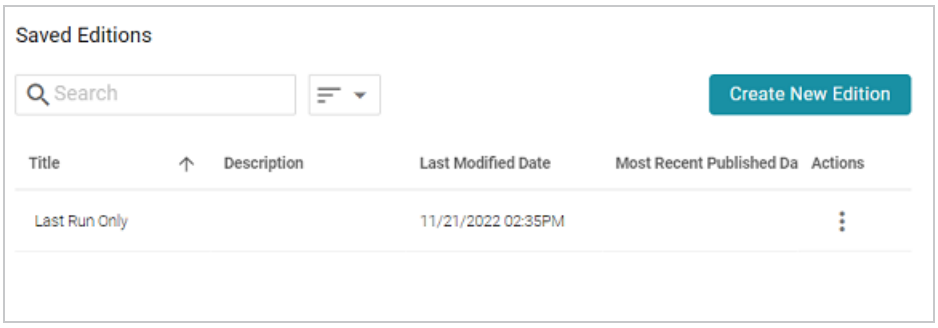


4. Specify a name for the edition in the **Title** field and include an optional description in the **Description** field.

5. In the Data Components list, select the checkbox next to each component that you want to add to this edition. For example:



6. When you are finished selecting data components, click **Save** to save the edition. The new edition is added to the list of **Saved Editions** on the Overview screen. For example:

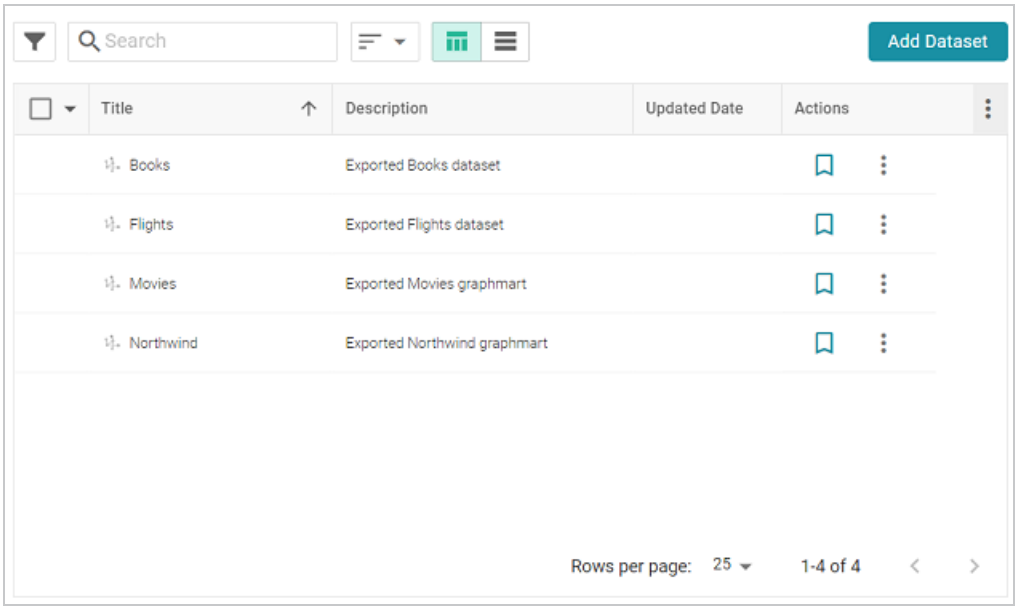


From the **Actions** menu for an edition, you can create a graphmart, or you can browse, clone, or delete the edition.

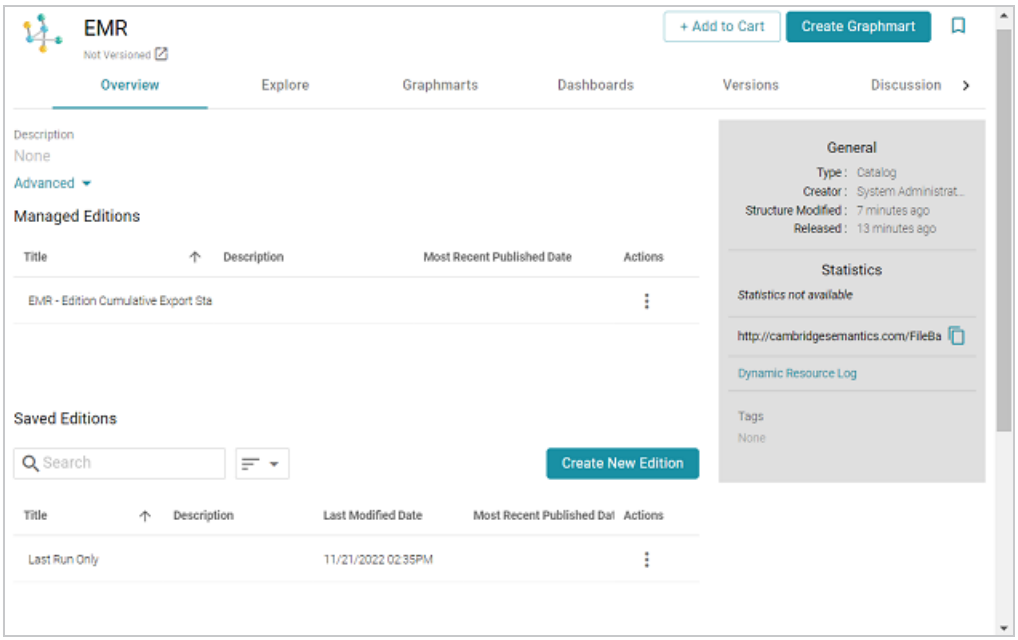
Modifying an Edition

You cannot change an existing edition, but you can clone and edit a copy of an edition. Follow the steps below to create a new edition based on an existing version.

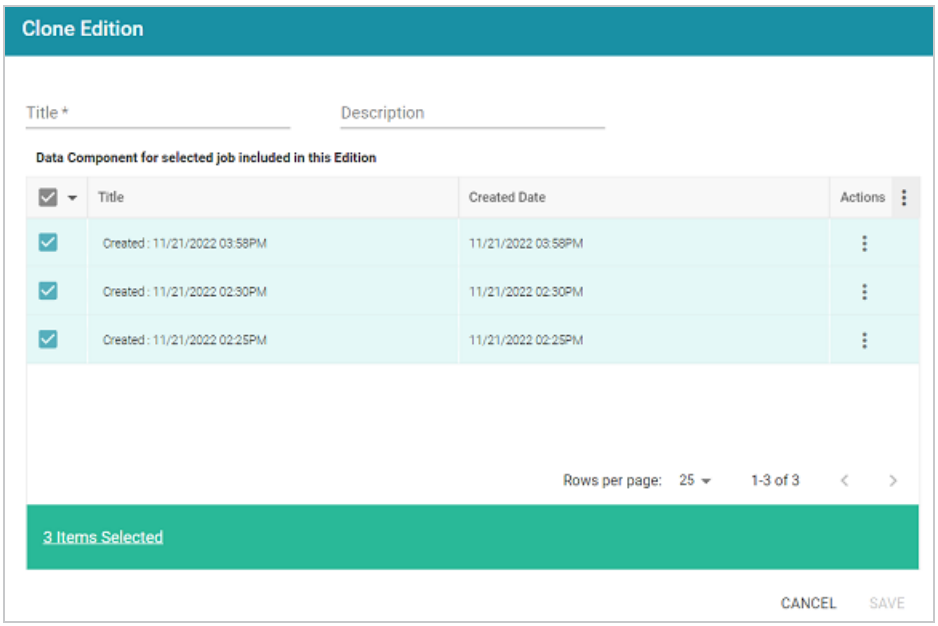
1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. Click the dataset for which you want to modify an ddition. Anzo displays the Explore tab for the dataset. Click the **Overview** tab, which lists the existing editions. For example:



3. Click the menu icon in the Actions column for the edition that you want to copy and select **Clone Edition**. Or select **Browse Edition** if you want to review the edition before making a copy. When you are ready to make a copy, click the **Edit a Copy** button. Anzo opens the Edition for editing. For example:



4. Specify a name for the edition in the **Title** field and include an optional description in the **Description** field.

5. To make changes to the edition, select or clear the Data Component checkboxes on the left side of the screen to include or exclude components.

> **Note**
> When you make changes to an edition while creating or changing a graphmart, Anzo creates a copy of the edition with the changes and uses the copy as a dataset in the graphmart. The original published edition remains unchanged.

6. When you have finished modifying the edition, click **Save**. Anzo creates the edition and adds it to the list of Saved Editions on the Overview screen.

The new edition is now available to add to a new or existing graphmart. To quickly create a new graphmart, you can click the menu icon in the Actions column for the new edition and select **Create Graphmart with this Edition**.

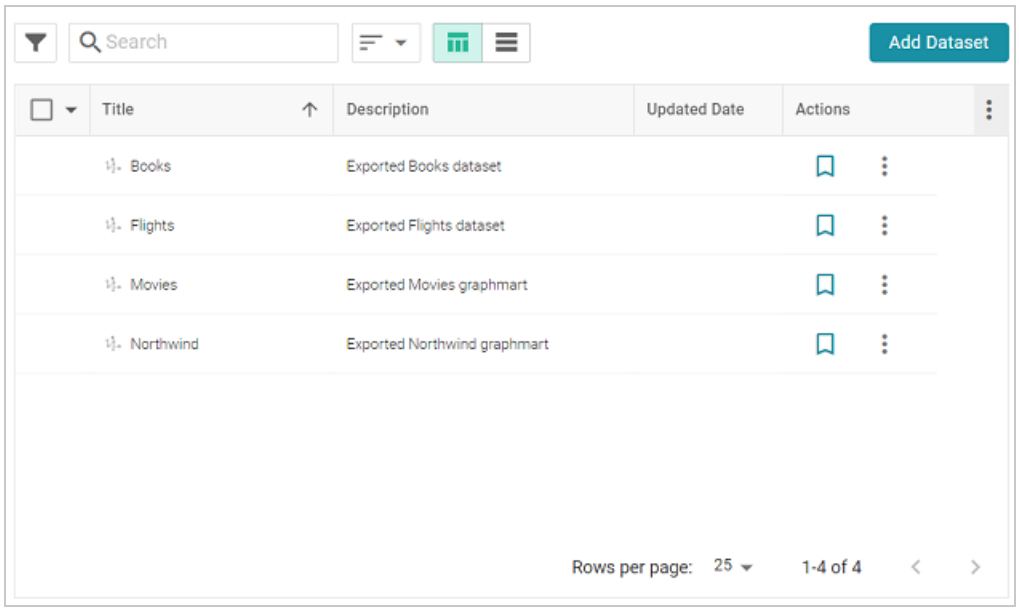
Deleting a Saved Edition

Follow the steps below to delete a Saved Edition.

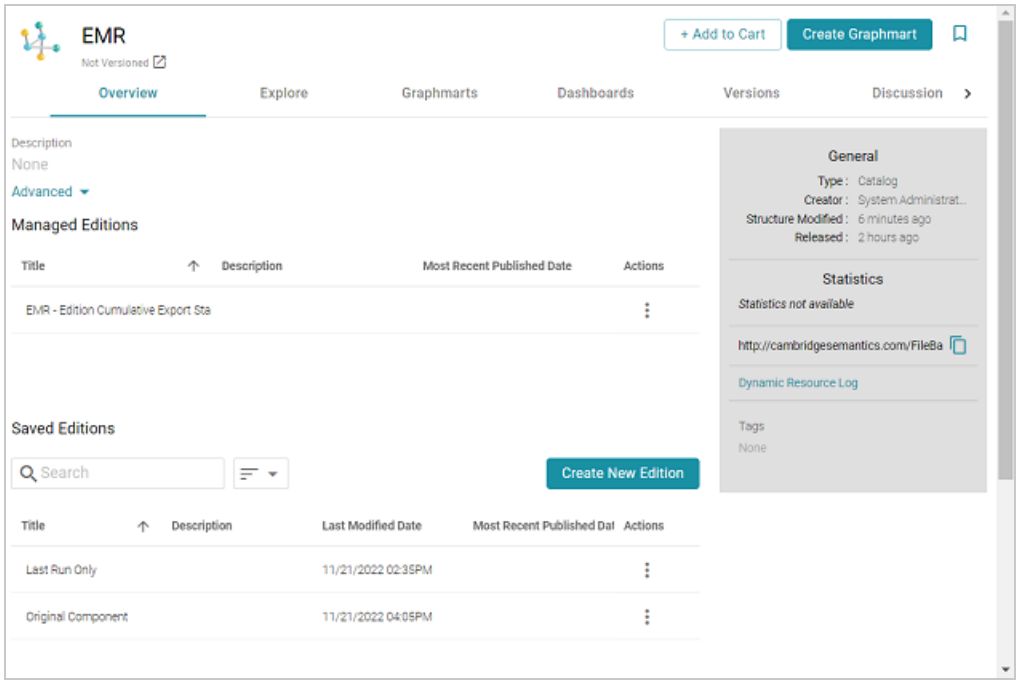
> **Note**
> Before deleting an edition, ensure that there are no graphmarts that require that edition.

> **Tip**
> You cannot delete the Managed Edition, but users with administrative privileges can clear out the existing components so that the edition is recreated from scratch the next time the pipeline is published. For instructions, see How do I clear the components from the Managed Edition?

1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. Click the dataset for which you want to delete an edition. Anzo displays the Explore tab for the dataset. Click the **Overview** tab, which lists the existing editions. For example:

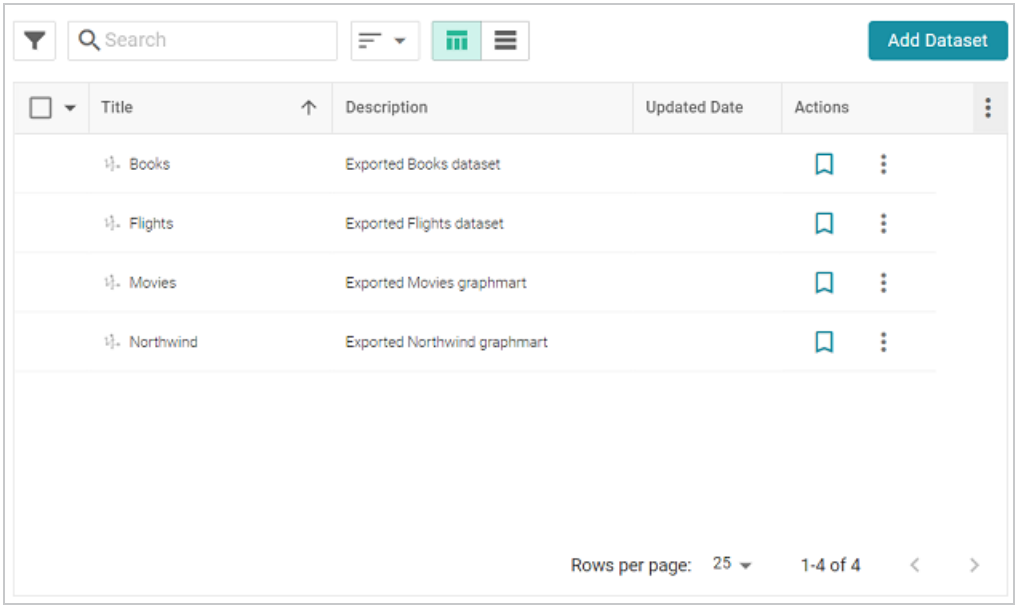


3. In the list of Saved Editions, click the menu icon in the Actions column for the edition that you want to delete and select **Delete**. Anzo displays a confirmation message. Click **OK** to confirm the delete operation and remove the edition.

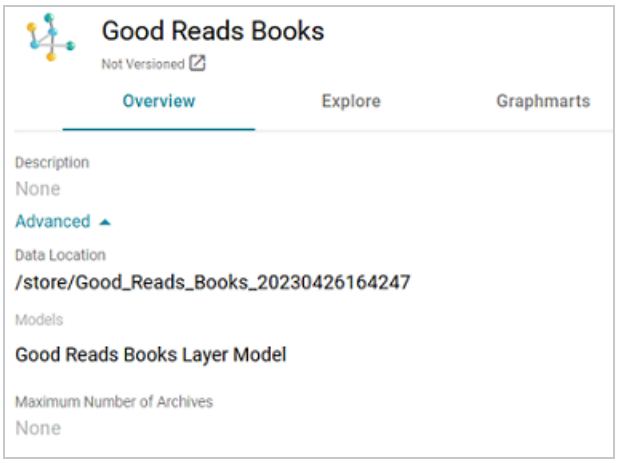
Limiting the Number of Editions in a Dataset

By default, there is no limit on the number of editions that can be created and preserved in a dataset. Each time an unstructured pipeline or Export Step is run, a new edition is generated and the TTL for that edition is written to disk. You can modify a dataset, however, to set a limit on the number of editions to archive. When a limit is set, Anzo to ages off and removes older editions from disk. Follow the steps below to limit the number of editions for a dataset.

1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. Click the dataset for which you want to configure the maximum number of editions. The Explore screen is displayed.
3. Click the **Overview** tab. Then click **Advanced** to expand the screen and show the advanced settings. The image below shows the settings.



4. Click in the **Maximum Number of Archives** field to make it editable. Then type the number of archived editions that you want to keep for the dataset. For example:



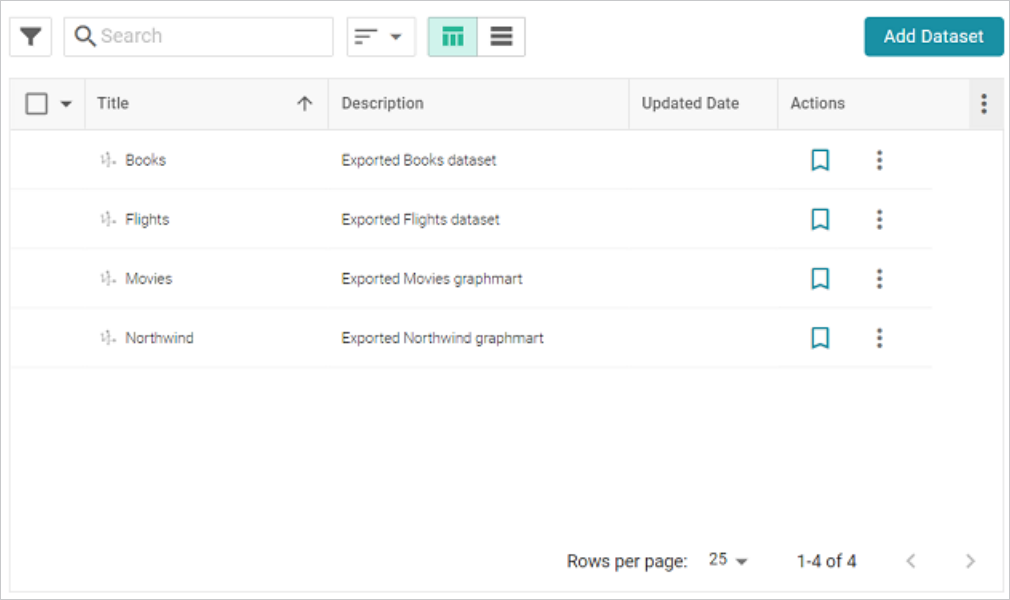
5. Click the checkmark icon (✓) to save the change.

The dataset is now configured to limit the number of editions that are retained. When the maximum number is reached, Anzo ages off and removes the oldest edition. For unstructured datasets, Anzo also removes the corresponding Elasticsearch Index JSON backups and pipeline runs.

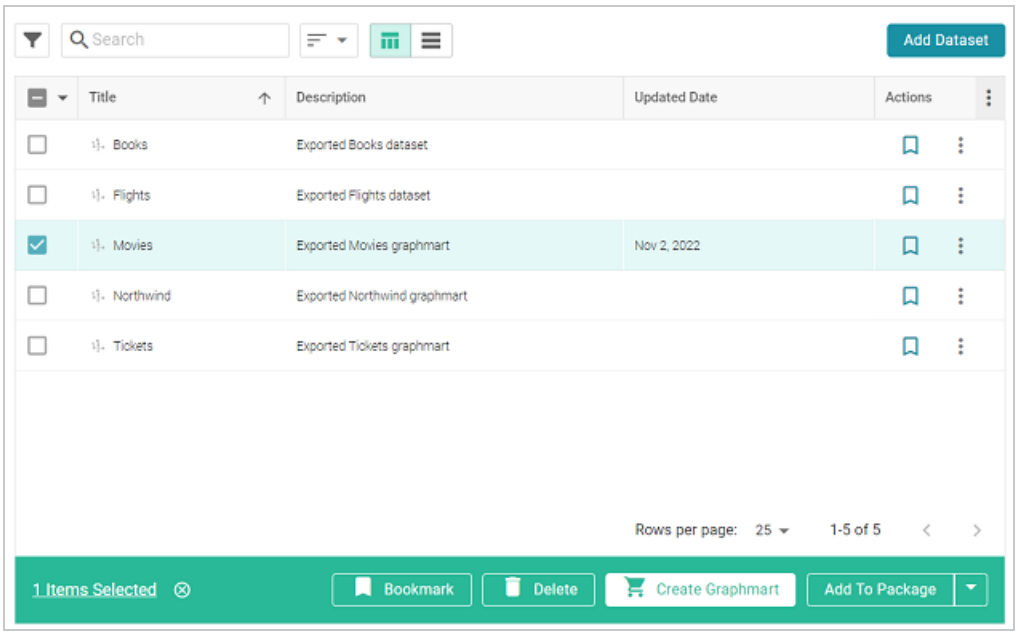
Creating a Graphmart from a Dataset

Follow the steps below to create a new graphmart from a dataset in the Datasets catalog.

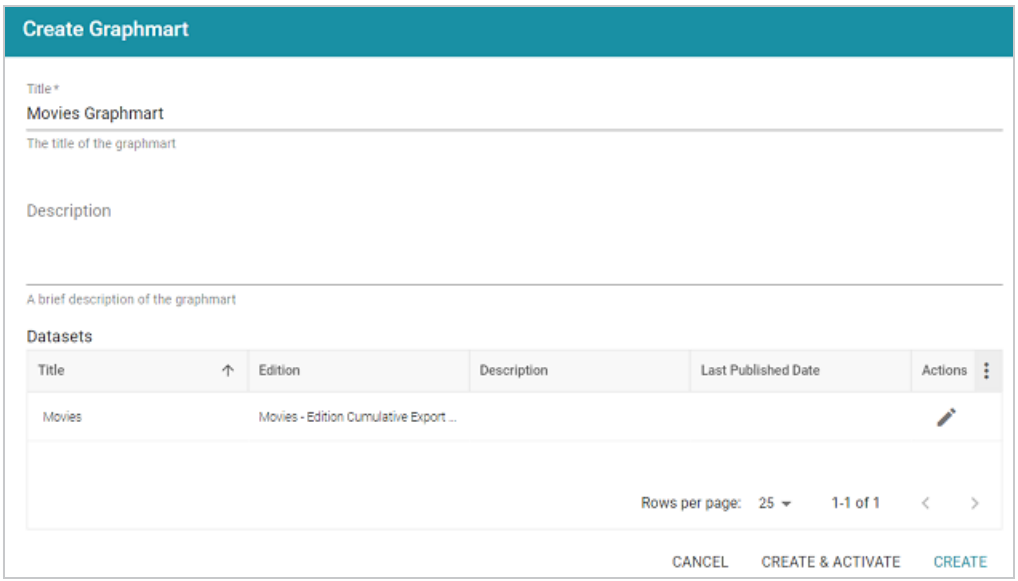
1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



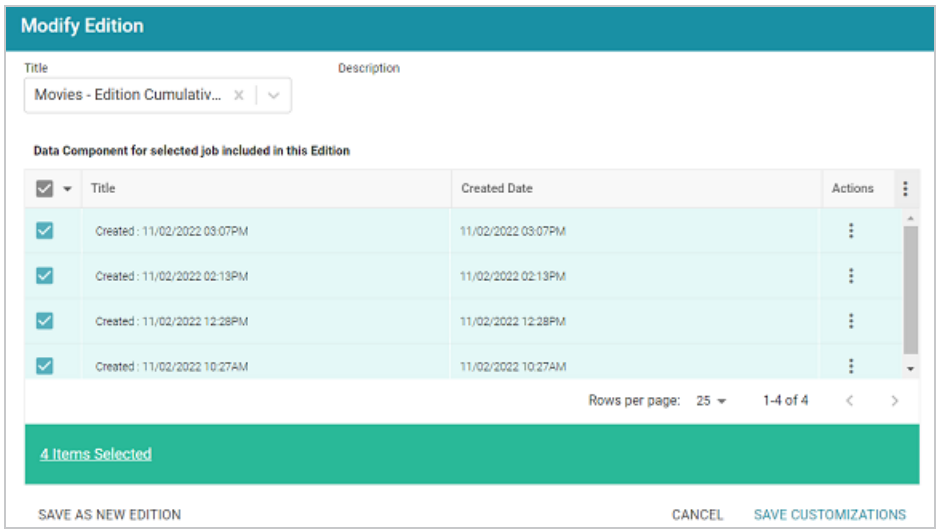
2. In the catalog, click the checkbox next to each dataset that you want to add to the new graphmart. Hover the pointer over an item to display the checkbox in the left column. Anzo adds the datasets to the shopping cart and additional icons become available at the bottom of the screen. For example:



3. Click the **Create Graphmart** button. Anzo displays the Create Graphmart screen and populates the Title field by appending "Graphmart" to the name of the dataset.



4. On the Create Graphmart screen, you can edit the **Title** and add an optional **Description**.
5. By default the current working edition (Managed Edition) of the dataset is selected. If you want to select a different edition, follow these steps:
   a. click the Modify Edition (✎) icon in the Actions column. The Modify Edition dialog box is displayed. For example:



b. To choose a different edition, click the drop-down list at the top of the screen and select the edition to use.

c. If you want to make changes to the selected edition, select or clear the Data Component checkboxes on the left side of the screen to include or exclude components.

> **Tip**
> You can also rename a component by clicking the menu icon in the Actions column and selecting **Rename Component**. In addition, you can remove a component from a dataset by clicking the Actions menu and selecting **Remove Component from Dataset**. However, note that choosing this option deletes the component from the edition and **deletes that component's data from the file system**.

d. When you are finished making changes, choose one of the following options for saving the changes:

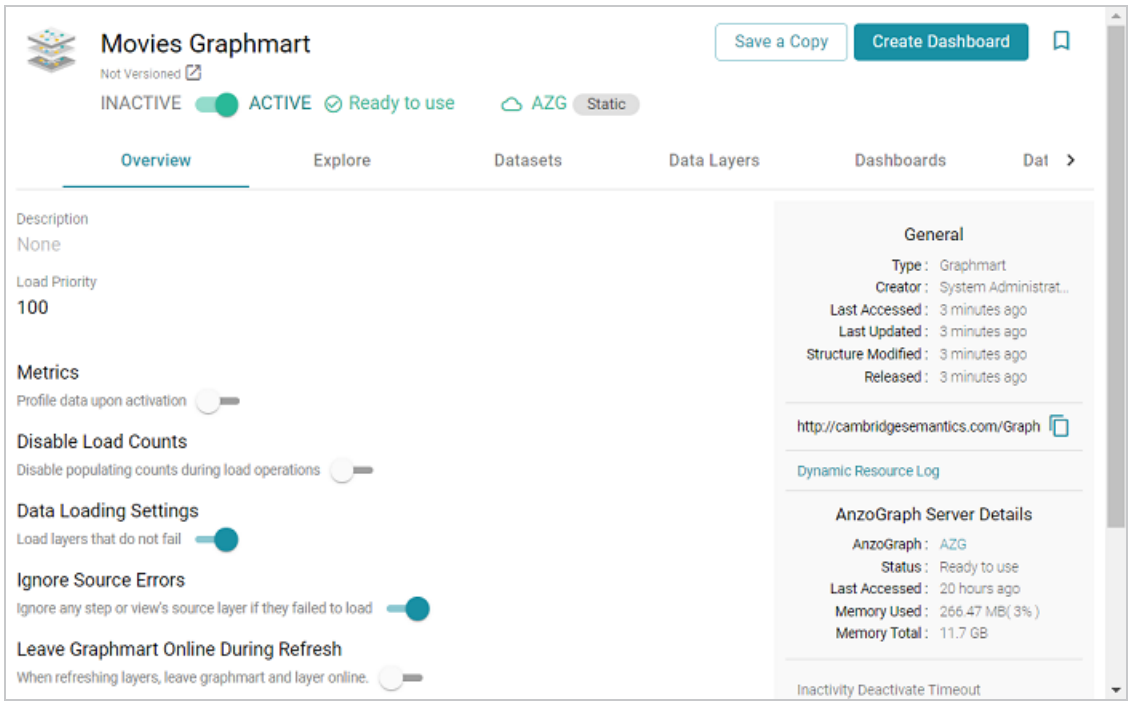
- To save the changes as a new edition, click **Save As New Edition** on the left side of the screen. The Create New Edition dialog box is displayed. Specify a Title and optional Description and click **Save**.



- To save the changes as a copy of the existing edition, click **Save Customizations**. Anzo clones the edition and adds the copy to the list on the Create Graphmart screen.

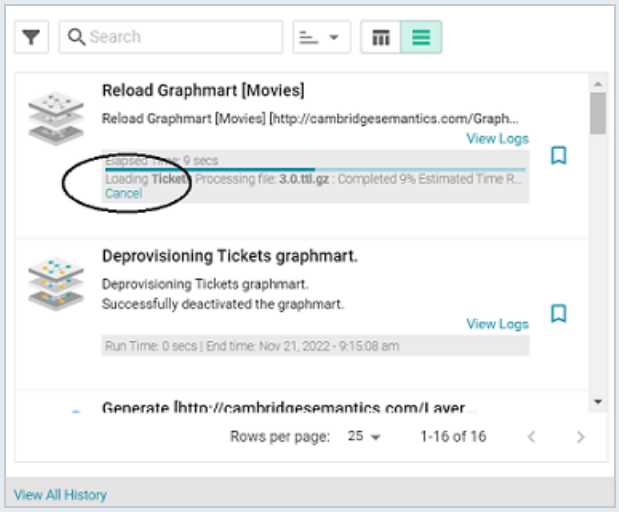
6. To create the graphmart without activating it, click **Create**. If you want to create the graphmart and activate it, click **Create & Activate**. Anzo creates the graphmart and displays the Overview screen. For example:

> **Note**
> If you activate the graphmart and have more than one static AnzoGraph engine configured or you have a Cloud Location configured for dynamic AnzoGraph deployments, Anzo displays a **Select an AnzoGraph Query Engine** dialog box. Click the drop-down list to select the engine to load the graphmart to, or select **Spin up new AnzoGraph** (if available) to deploy a new instance. Then click **OK**.



Tip

If you want to cancel graphmart activation while data is loading, open the Activity Log by clicking the Activity Log icon ( ) in the main menu bar. Then click **Cancel** for the **Provisioning...graphmart** activity. For example:

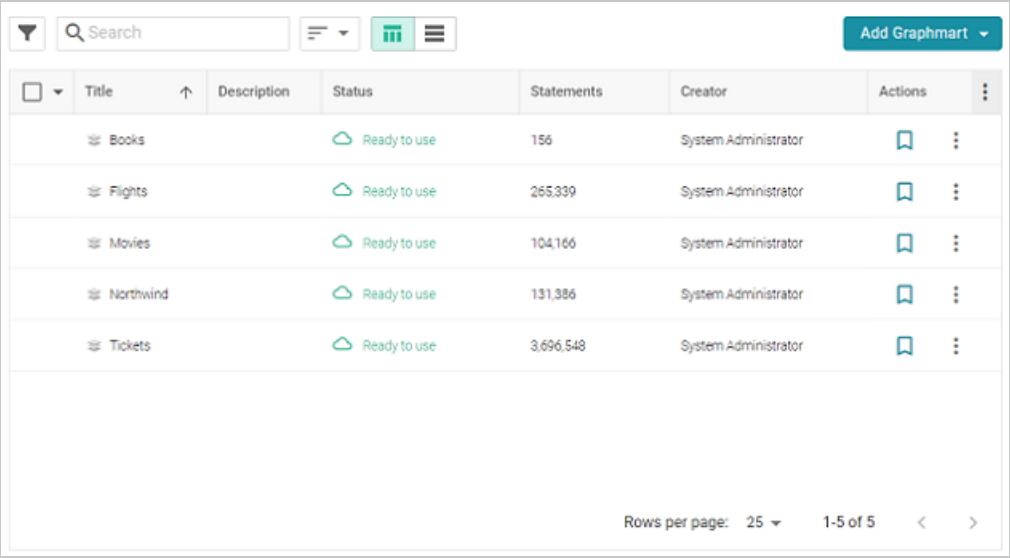


When graphmart creation is complete and the graphmart is activated, the data is available to access and analyze. For more information, see Access & Analyze. For more information about graphmarts, see Working with Graphmarts.

Adding a Dataset to a Graphmart

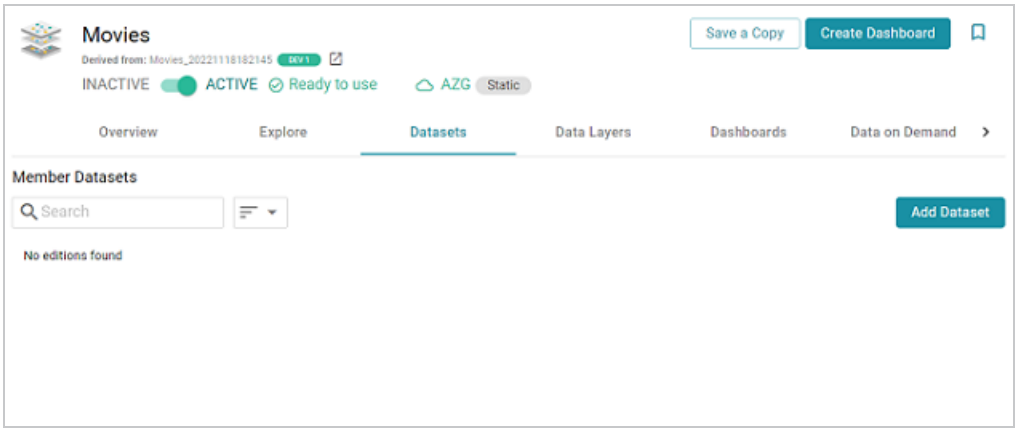
This topic provides instructions for adding a dataset from the Datasets catalog to an existing graphmart.

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:

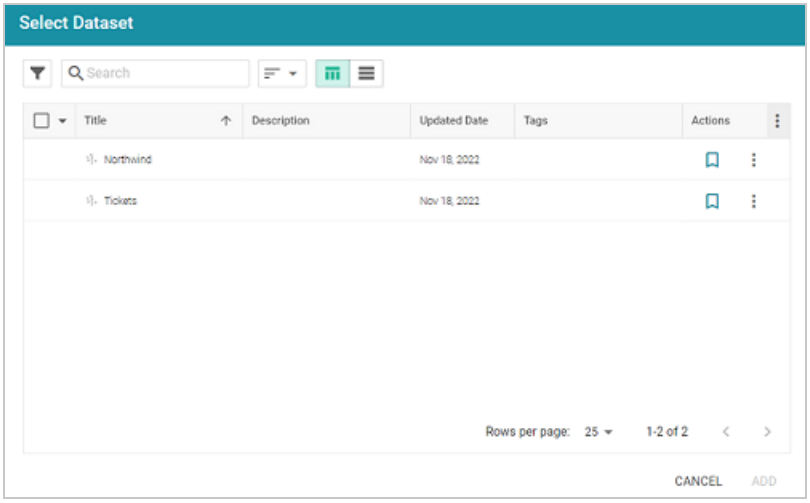


2. On the Graphmarts screen, click the name of the graphmart that you want to add data to. Anzo displays the Overview.

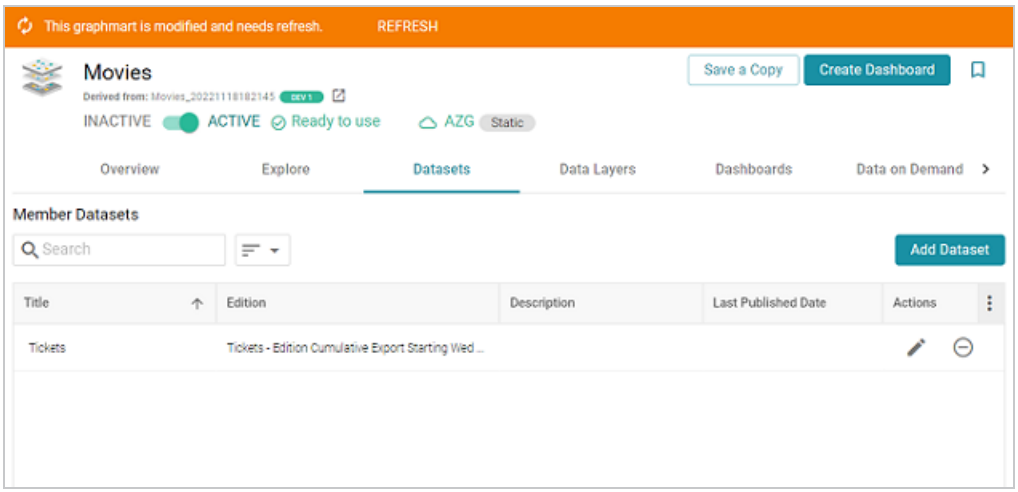
3. Click the **Datasets** tab. The screen lists the datasets in the graphmart. For example, the image below shows a graphmart without any datasets because this is a graphmart that was auto-generated from a data source:



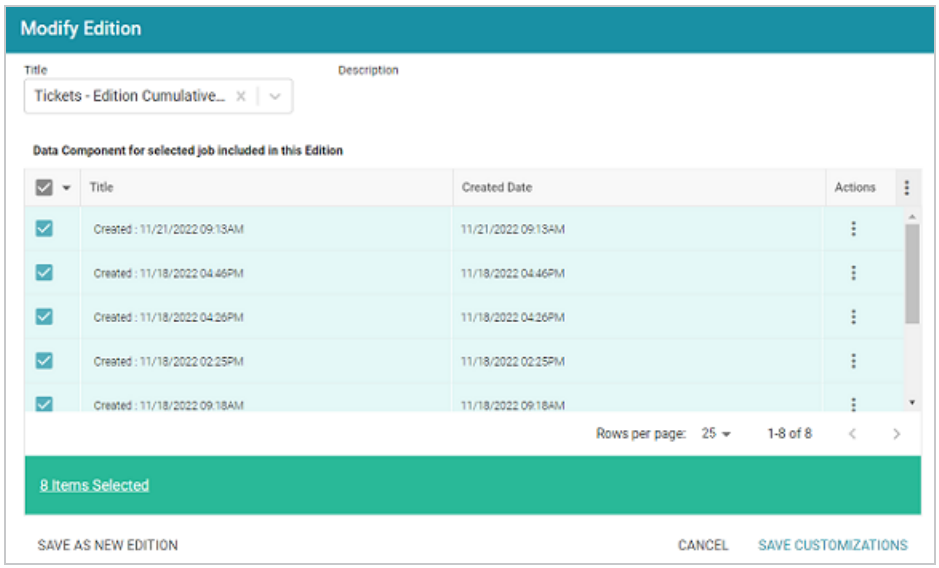
4. Click the **Add Dataset** button. The Select Dataset dialog box is displayed.



5. In the dialog box, select the checkbox next to the dataset that you want to add, then click **Add**. Anzo adds the dataset to the graphmart. (A new data layer is generated with a Load Dataset Step that will load the dataset.)



6. By default the current working edition (Managed Edition) of the dataset is selected. If you want to select a different edition, follow these steps:
   a. Click the Edit icon (✎) in the Actions column. The Modify Edition dialog box is displayed. For example:



b. To choose a different edition, click the drop-down list at the top of the screen and select the edition to use.

c. If you want to make changes to the selected edition, select (to include) or clear (to exclude) the data component checkboxes on the left side of the screen.

> **Note**
> When you make changes to an edition while creating or changing a graphmart, Anzo creates a copy of the edition (with the changes) and uses the copy as a dataset in the graphmart. The original published edition remains unchanged.

d. When you are finished making changes, choose one of the following options for saving the changes:

- If you want to save the changes as a new Saved Edition, click **Save As New Edition**. Anzo displays the Create New Edition dialog box. Specify a Title and optional Description for the edition, and click **Save**. Then click **Save Customizations** on the Modify Edition screen.

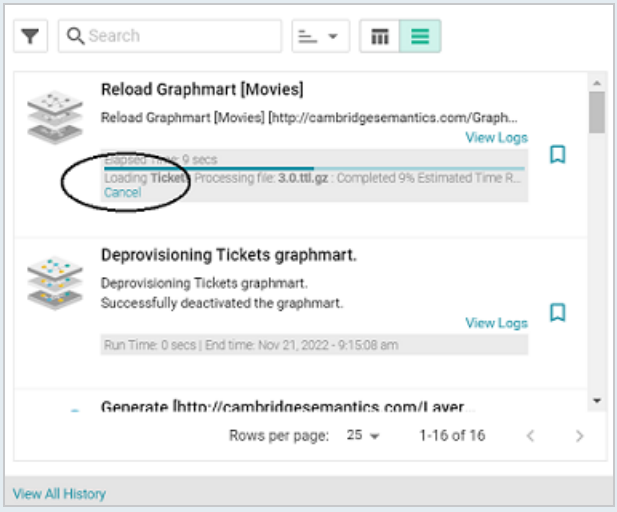


- If you want to save the changes as a copy of the existing edition, click **Save Customizations**. Anzo clones the edition and adds the copy to the list on the screen.

7. To reload the graphmart and add the new dataset to AnzoGraph, click the **Data Layers** tab, and then click the **Reload** button (↻).

Tip

If you want to cancel graphmart activation while data is loading, open the Activity Log by clicking the Activity Log icon (🕘) in the main menu bar. Then click **Cancel** for the **Provisioning...graphmart** activity. For example:



Once the graphmart is loaded into AnzoGraph, the data is available to access and analyze. For more information, see Access & Analyze.

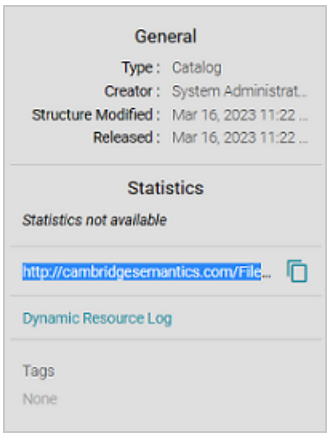
Dataset FAQ

This topic provides answers to frequently asked questions about datasets.

- How do I find the URI for a dataset?
- How do I find the catalog entry URI for a dataset?
- How do I associate a model with a dataset?
- How do I clear the components from the Managed Edition?

How do I find the URI for a dataset?

Anzo displays dataset details on the Overview screen for the dataset. To view and copy a dataset URI, go to the Overview tab for the dataset. The URI is under General information on the right side of the screen. You can click the clipboard icon ( ) to copy the URI.

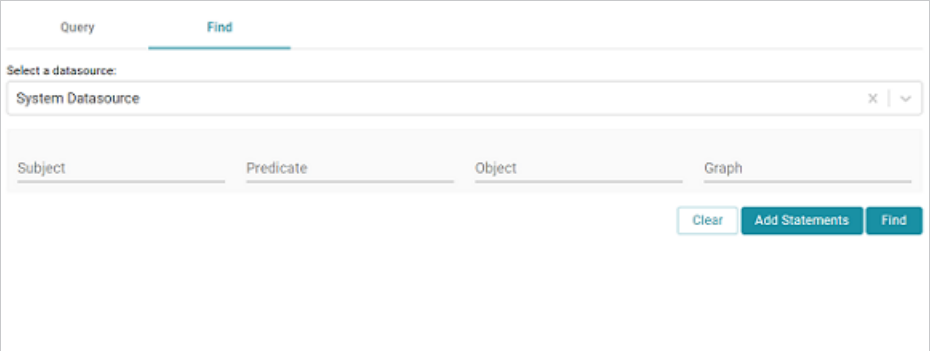


How do I find the catalog entry URI for a dataset?

To query from a remote client (such as over the SPARQL endpoint) a linked data set (LDS) that is stored in a local volume, you need to specify the catalog entry URI for that LDS as the target data set. The catalog entry URI uniquely identifies an LDS because it encodes both the LDS and its data source (local volume) in the URI. Follow the steps below to find the catalog entry for an LDS.

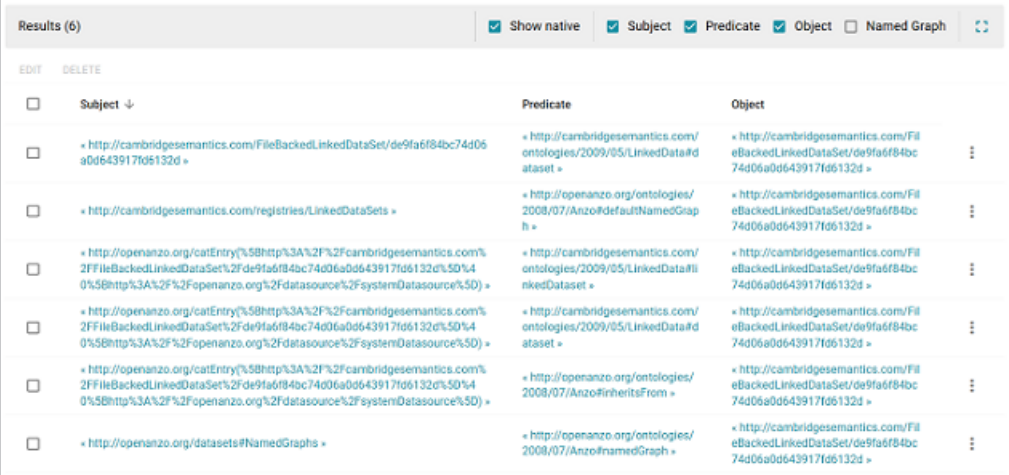
1. First, retrieve the URI for the dataset whose catalog entry URI you want to find. For instructions, see How do I find the URI for a dataset? above.

2. Next, open the Find tab in the Query Builder. In the Anzo application, expand the **Access** menu and click **Query Builder**. Then click the **Find** tab. The Find screen opens and the **System Datasource** is selected as the target data source.



3. If the LDS is in a different volume, click the **Source** drop-down list and select the appropriate volume. Typically, linked data sets are stored in the system volume.

4. Paste in the **Object** field the LDS URI that you copied in the first step. Then click **Find**. Anzo returns the set of quads for which the LDS URI is the object. For example:



5. In the **Subject** field in the results, look for a URI that begins with **http://openanzo.org/catEntry**. The value is the catalog entry URI for the LDS. For example:

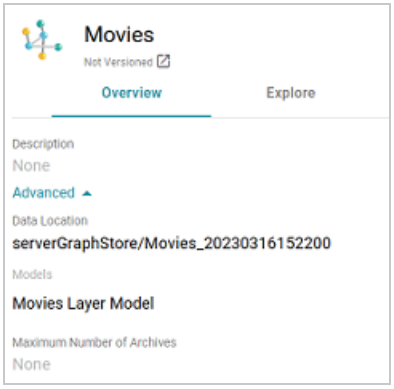
```
« http://openanzo.org/catEntry(%5Bhttp%3A%2F%2Fcambridgesemantics.com%
2FFileBackedLinkedDataSet%2Fde9fa6f84bc74d06a0d643917fd6132d%5D%4
0%5Bhttp%3A%2F%2Fopenanzo.org%2Fdatasource%2FsystemDatasource%5D) »
```

6. Copy the entire URI. This is the URI to use as the target data source for SPARQL endpoint queries against the LDS. For more information about the SPARQL endpoint, see Access the SPARQL Endpoint.

How do I associate a model with a dataset?

Follow the instructions below to associate a model that is in Anzo with an onboarded dataset.

1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets catalog, which lists the existing datasets.
2. Click the dataset that you want to add a model to. Anzo displays the Explore screen for the dataset.
3. Click the **Overview** tab. Under the Description field, click **Advanced** to display the advanced options. For example:



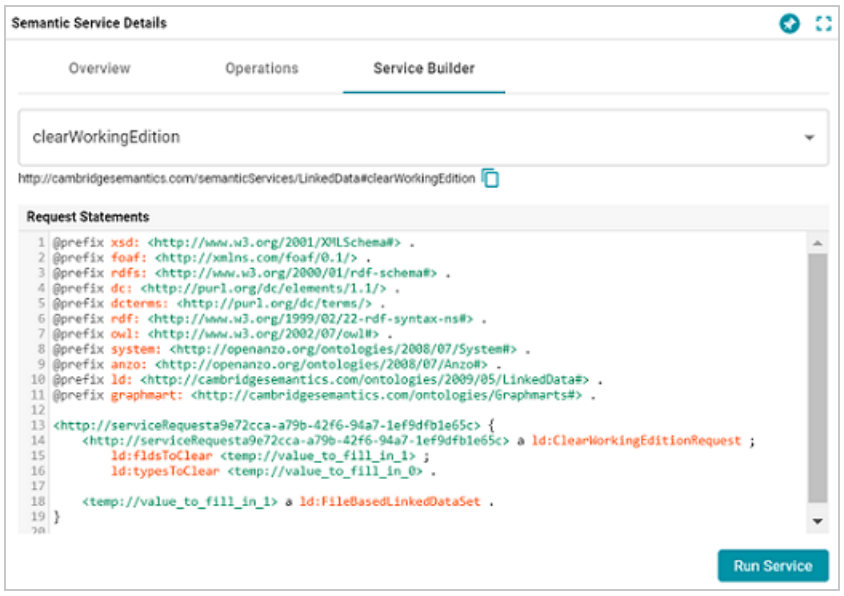
4. Click in the **Models** field to make it editable, then click the Models drop-down and select the model to add to this dataset. To include a system model, select the **Include System Data** checkbox. To select multiple models, click the drop-down list again and select another model.
5. When you have finished selecting models, click the checkmark icon (✓) to save the change and associate the model or models with the dataset.

How do I clear the components from the Managed Edition?

Follow the instructions below if you want to clear out all of the existing components from the Managed Edition so that the edition is recreated from scratch the next time the pipeline is published or the dataset is exported from a graphmart.

> **Note**
> Permission to **Manage Semantic Services** is required to complete this task.

1. First, copy the URI of the dataset for which you want to clear the Managed Edition. How do I find the URI for a dataset?
2. Next, In the Administration application, expand the **Monitoring & Diagnostics** menu and select **Semantic Services**.
3. Search for the **LinkedDataService** and view its details. Then click the **Service Builder** tab in Semantic Service Details.
4. Click the **Please Select an Operation** field and select **clearWorkingEdition** from the drop-down list. The Request Statements for the service call are populated:



5. Toward the bottom of the request, replace the **<temp://value_to_fill_in_1>** placeholder URI with the URI for the dataset.

```
<http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> {
  <http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> a
ld:ClearWorkingEditionRequest ;
    ld:fldsToClear <temp://value_to_fill_in_1> ;
    ld:typesToClear <temp://value_to_fill_in_0> .

<temp://value_to_fill_in_1> a ld:FileBasedLinkedDataSet .
}
```

For example:

```
<http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> {
  <http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> a
ld:ClearWorkingEditionRequest ;
    ld:fldsToClear
<http://csi.com/FileBasedLinkedDataSet/ee8d3d5792fd218a03b70fdf850b6a4c> ;
    ld:typesToClear <temp://value_to_fill_in_0> .

<http://csi.com/FileBasedLinkedDataSet/ee8d3d5792fd218a03b70fdf850b6a4c> a
ld:FileBasedLinkedDataSet .
}
```

6. Comment out the `ld:typesToClear <temp://value_to_fill_in_0>` line. For example:

```
<http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> {
  <http://serviceRequesta9e72cca-a79b-42f6-94a7-1ef9dfb1e65c> a
ld:ClearWorkingEditionRequest ;
    ld:fldsToClear
<http://csi.com/FileBasedLinkedDataSet/ee8d3d5792fd218a03b70fdf850b6a4c> ;
  # ld:typesToClear <temp://value_to_fill_in_0> .

<http://csi.com/FileBasedLinkedDataSet/ee8d3d5792fd218a03b70fdf850b6a4c> a
ld:FileBasedLinkedDataSet .
}
```

7. Click the **Run Service** button to clear the edition. Anzo returns a response such as the following example when the request is processed:

```
@prefix n-1060687345: <http://openanzo.org/ClearWorkingEditionResponse/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ld: <http://cambridgesemantics.com/ontologies/2009/05/LinkedData#> .
@prefix ss: <http://openanzo.org/ontologies/2008/07/SemanticService#> .
```

```
n-1060687345:effbda5b-ce9c-4190-abb1-6f6efd07f5d8 {
  n-1060687345:effbda5b-ce9c-4190-abb1-6f6efd07f5d8 ld:wasWorkingEditionCleared
    "true"^^<http://www.w3.org/2001/XMLSchema#boolean> .
  n-1060687345:effbda5b-ce9c-4190-abb1-6f6efd07f5d8 rdf:type
ld:ClearWorkingEditionResponse .
  n-1060687345:effbda5b-ce9c-4190-abb1-6f6efd07f5d8 rdf:type ss:ServiceResponse
}
```

Now, if you browse the Managed Edition for the dataset, you will see that the edition does not contain any data components.

Working with Graphmarts

The topics in this section provide guidance on working with graphmarts.

In this section:

Creating a Graphmart 482

Copying a Graphmart 483

Graphmart Settings Reference 485

Creating an Elasticsearch Index from a Graphmart 489

Adding Data Layers to Graphmarts 497

Adding Steps to Layers 512

Creating Data on Demand Endpoints 574

Sharing Access to Graphmarts 592

Graphmart FAQ 601

Creating a Graphmart

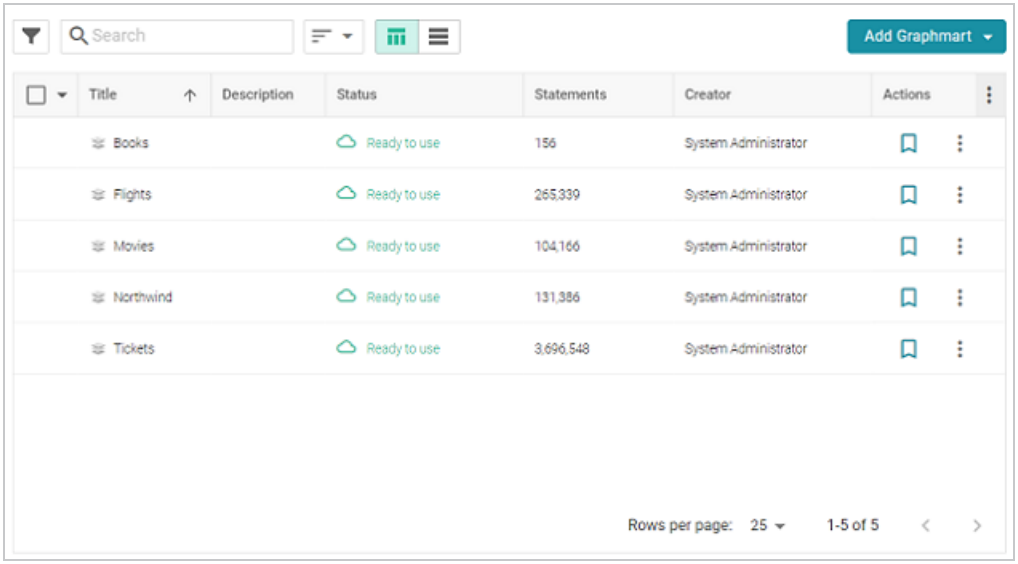
Besides importing an exported graphmart, there are four ways to create a new graphmart:

1. You can create a graphmart directly from a connected data source. For instructions, see Creating a Graphmart from a Data Source.
2. You can create a graphmart from a dataset in the Datasets catalog. For instructions, see Creating a Graphmart from a Dataset.
3. You can create a graphmart by copying an existing one. For instructions, see Copying a Graphmart.
4. Or you can create an empty graphmart and build it from scratch by adding data layers and steps that load and transform data. To create an empty graphmart, go to the Graphmarts screen in the Anzo application, click **Add Graphmart**, and select **Create New Graphmart**. See Graphmart Settings Reference for information about graphmart settings, and see Adding Data Layers to Graphmarts for information about adding layers and steps.

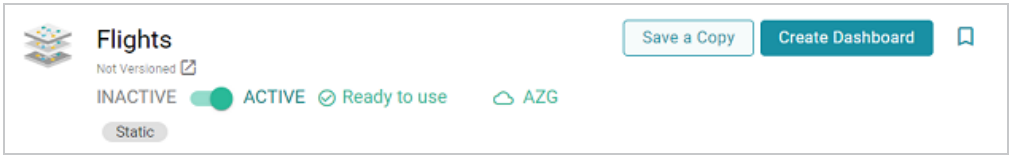
Copying a Graphmart

Follow the instructions below if you want to create a graphmart by making a copy of an existing graphmart and its data layers and steps. For instructions on creating a new graphmart from scratch, see Creating a Graphmart.

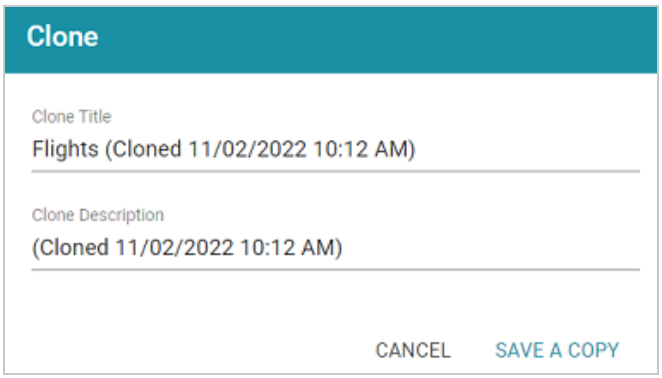
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



2. On the Graphmarts screen, click the name of the graphmart that you want to copy. Anzo displays the graphmart Overview. At the top of the screen, click the **Save a Copy** button.



The Clone dialog box is displayed. For example:

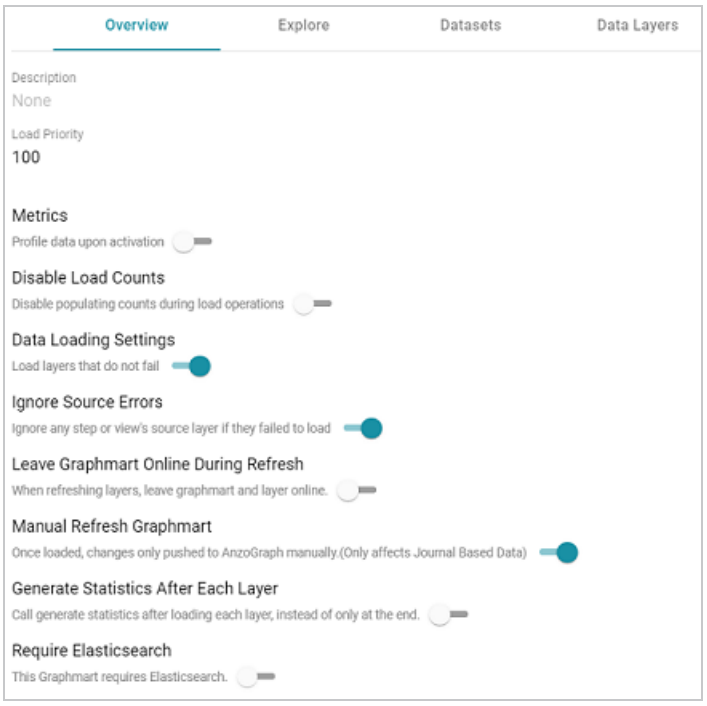


3. In the Clone dialog box you have the option to edit the new graphmart name in the **Clone Title** field and modify the description in the **Clone Description** field.
4. When you are ready to save the copy, click **Save a Copy**.

Anzo creates and displays the new graphmart. You can proceed with modifying the graphmart settings, changing and adding datasets, data layers, or Data on Demand endpoints, or activating the new graphmart.

Graphmart Settings Reference

This topic describes the graphmart configuration settings that are available on the graphmart Overview tab.



Setting	Description
Load Priority	If you want Anzo to prioritize the order in which graphmarts are activated when reconnecting to AnzoGraph or resetting and reloading AnzoGraph, you can designate a Load Priority for each graphmart. When reloading AnzoGraph, Anzo activates the graphmarts in sequence, starting with the lowest Load Priority number. The default value is **100**.
Metrics	**Profile Data Upon Activation** This setting is disabled by default and controls whether a data profile is automatically generated each time the graphmart is activated. For information about data profiles, see Generating a Graphmart Data Profile.
Disable Load	**Disable Populating Counts During Load Operations**

Setting	Description
Counts	This setting is disabled by default controls whether Anzo periodically sends `select (count(*) as ?count)...` queries to AnzoGraph to count the total number of statements that are being loaded in each data layer. Disabling the load counts may increase load performance as it decreases the number of queries that run during graphmart activation.
Data Loading Settings	**Load Layers that Do Not Fail** This setting is enabled by default and controls what to do if a data layer fails during graphmart activation. When enabled (the default setting), the graphmart is configured to load all layers that succeed and skip any layers that fail. When disabled, the entire graphmart activation is aborted if any layer fails.

Ignore Source Errors	**Ignore any Step or View's Source Layer if they Failed to Load**
	This setting is enabled by default and controls what to do if a source that is referenced by a step or view fails to load. For example, if the source for a Query Step is set to "All Previous Layers Within Graphmart" and one of the previous layers fails to load, this setting controls whether to run the Query Step but ignore the failed layer or fail the step since one of the sources failed.
	If **Ignore Source Errors** is enabled (the default setting), Anzo ignores the failed source and runs the step against the sources that did not fail. For example, if `${usingSources}` in a Query Step translates to
	`USING <layer1>`
	`USING <layer2>`
	`USING <layer3>`
	And layer1 failed to load, Anzo runs the Query Step but ignores layer1 and automatically changes the query to

Setting	Description

	`USING <layer2>`
	`USING <layer3>`
	
	If **Ignore Source Errors** is disabled and a source layer fails, any steps or views that use that source will also fail since the source is not available.

Leave Graphmart Online During Refresh	**When Refreshing Layers, Leave Graphmart and Layer Online** This setting is disabled by default and controls whether a graphmart remains online while it is being refreshed in AnzoGraph. When this option is enabled, if a user clicks the **Refresh** button to refresh a graphmart (or the Refresh icon on a data layer), Anzo copies the existing layers into temporary graphs so that the data remains online while the original graphs are refreshed. When the refresh is complete, the temporary graphs are deleted. **Note** This setting applies only to **Refresh** operations. If **Leave Graphmart Online During Refresh** is enabled and a user clicks **Reload**, the data layers will not remain online. During reloads all of the data is dropped and then loaded again.
Manual Refresh Graphmart	**Once Loaded, Changes only Pushed to AnzoGraph Manually (Only Affects Journal Based Data)** This setting is enabled by default and controls whether changes to a dataset in this graphmart are automatically deployed to AnzoGraph without requiring a manual refresh or reload of the graphmart. This setting only applies to graphmarts with Load Dataset Steps that load a journal-based data set, such as a system metadata graph. When this option is enabled, changes to the journal-based data set are only deployed to AnzoGraph when the graphmart is manually reloaded or refreshed. When this option is

Setting	Description
	disabled, changes to the dataset are automatically loaded to AnzoGraph without requiring a manual refresh.
Generate Statistics After Each Layer	**Call Generate Statistics after Loading Each Layer, Instead of Only at the End** Typically the AnzoGraph connection is configured to automatically initiate AnzoGraph's internal statistics gathering queries after loading a graphmart. However, if a user refreshes individual layers rather than the entire graphmart, those queries are not triggered. Enabling this setting initiates the statistics gathering queries each time a layer is loaded. This helps the AnzoGraph query planner generate ideal query execution plans for queries that are run against the refreshed data layers.
Require Elasticsearch	**This Graphmart Requires Elasticsearch** This setting is disabled by default. If you plan to include unstructured datasets in this graphmart or configure a data layer that creates an Elasticsearch index, you can enable this option to ensure that Anzo validates the connection to Elasticsearch whenever this graphmart is activated, reloaded, or refreshed.

Creating an Elasticsearch Index from a Graphmart

By associating an Elasticsearch index with a data layer, you can load data from a graphmart to an Elasticsearch index, enabling you to perform free-text and pattern searches on your knowledge graphs. This topic provides instructions for configuring a workflow that generates an Elasticsearch index and snapshot from a graphmart.

- Prerequisites
- Add an Export Step to the Graphmart
- Add a Layer to Manage the Index
- Add a Step to Create the Index
- Add a Step to Take a Snapshot of the Index

Prerequisites

Before configuring a graphmart to create an Elasticsearch index, make sure that the following requirements are met:

1. A supported version of Elasticsearch is installed and configured. For more information, see Elasticsearch Requirements in the Deployment Guide.
2. The Elasticsearch instance is connected to Anzo. See Connecting to Elasticsearch in the Administration Guide for more information.
3. The AnzoGraph instance that you will load the graphmart to is also connected to Elasticsearch. Configure the connection by selecting the Elasticsearch instance in the **Elasticsearch Configuration** field in the AnzoGraph configuration. For more information, see Connecting to AnzoGraph in the Administration Guide.

Add an Export Step to the Graphmart

First, if a file-based linked data set (FLDS) has not been generated for this graphmart, add an **Export Step** to the **last layer** in the graphmart so that all of the graph data is exported to an FLDS on the file store. For instructions on adding an Export Step, see Export Data to an FLDS (Export

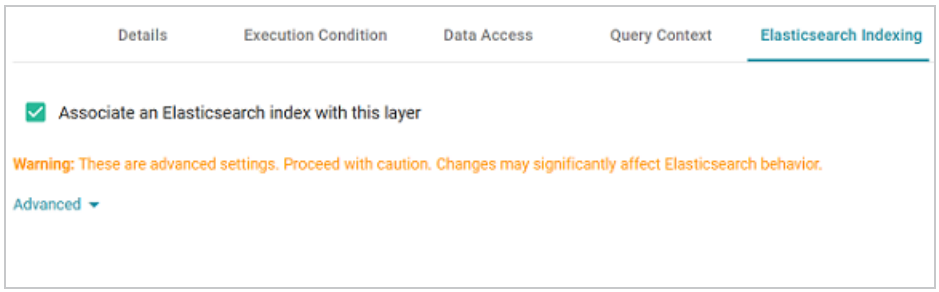
Step).

If the graphmart has an Export Step, note the Target FLDS for the step and proceed to Add a Layer to Manage the Index below.

Add a Layer to Manage the Index

Follow the steps below to add a layer to a graphmart and configure it to manage an Elasticsearch index.

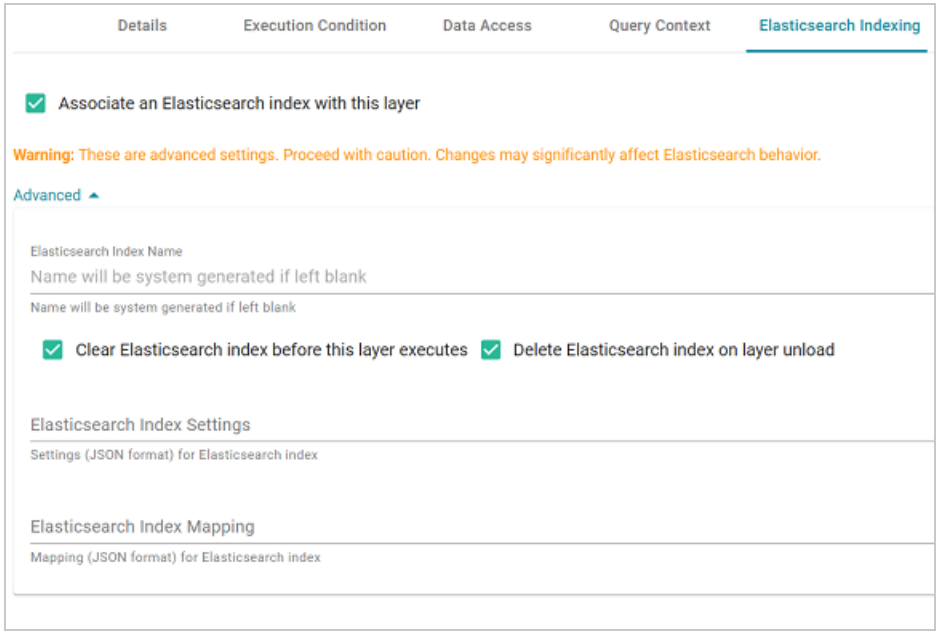
1. Add a new data layer to the graphmart. This new layer will be associated with the Elasticsearch index and contain steps to generate the index and snapshot. For instructions on adding a layer, see Creating a New Layer.
2. In the new layer, click the **Elasticsearch Indexing** tab. Then select the **Associate an Elasticsearch index with this layer** checkbox. When this setting is enabled, Anzo creates an index on the Elasticsearch server and links this layer to that index.



> **Note**
> Though the index will be populated with data from other layers in the graphmart, you do not need to modify those layers to associate them with the index. **Associate an Elasticsearch index with this layer** should be disabled on all layers that do not contain Elasticsearch Indexing Steps.

3. The Advanced settings on this screen relate to managing the index's life cycle. It is not necessary to modify the settings to successfully generate an index for the graphmart. If you have an advanced use case, such as a case that requires linking this layer to an existing index or needing to add custom Elasticsearch-specific index or mapping configurations, expand

Advanced to access the options. The list below describes the available settings.



- **Elasticsearch Index Name**: By default, the new index is given a system-generated name. If you are linking this layer to an existing index, add the existing index name to this field. Or, if you plan to reference this index elsewhere and want to give it a human-readable name, you can specify a custom name.
- **Clear Elasticsearch index before this layer executes**: This option is enabled by default and configures the layer so that the index is cleared and recreated each time this layer is run.
- **Delete Elasticsearch index on layer unload**: This option is enabled by default and configures the layer so that the index is deleted any time the layer is deactivated or reloaded. The index is recreated during the reload or when the graphmart is activated again. If you do not want the index to be deleted when the layer is offline, clear the checkbox.
- **Elasticsearch Index Settings**: This field can be used to add any Elasticsearch-specific index settings that you want to apply. Add the settings in the following JSON format:

```
{
  "index": {
```

```
    "<settings_and_values>"
  }
}
```

For example:

```
{
 "index": {
   "number_of_shards": "1",
   "number_of_replicas": "0",
   "routing": {
    "allocation": {
     "include": {
      "_tier_preference": null
     }
    }
   }
  }
}
```

- **Elasticsearch Index Mapping**: This field can be used to add any Elasticsearch-specific mapping properties that you want to apply. Add the properties in the following JSON format:

```
{
  "properties": {
    "<properties_and_values>"
  }
}
```

For example:

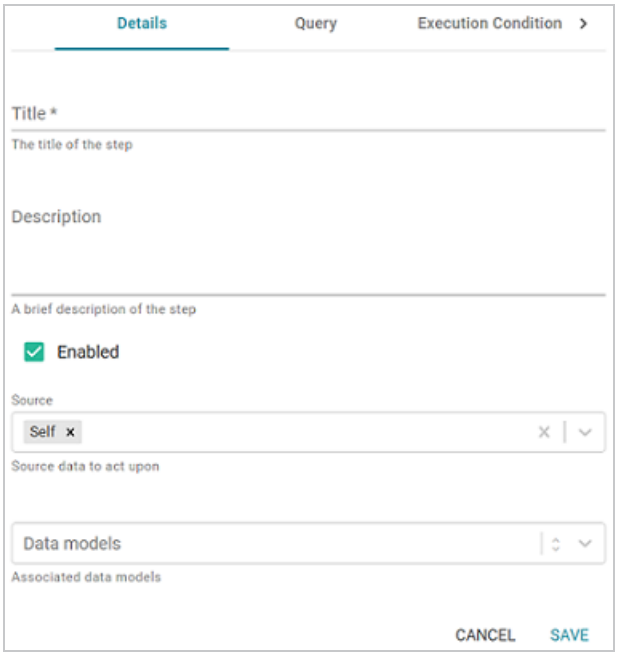
```
{
 "properties": {
  "movie_Abstract": { "type": "keyword" },
  "movie_ID": { "type": "long" },
  "movie_Title": { "type": "text" }
 }
}
```

4. When you have finished configuring the layer, click **Save** to add it to the graphmart and return to the Data Layers screen. Then proceed to Add a Step to Create the Index below.

Add a Step to Create the Index

Follow the steps below to add an Elasticsearch Indexing Step to the new layer and configure it to generate an index.

1. On the Data Layers screen, click the menu icon (⋮) for the new layer and select **Add Step/View**.
2. On the Add Step/View screen, select **Elasticsearch Indexing Step** and click **OK**. The Create dialog box is displayed.
3. On the Details tab, add a name for the step in the **Title** field, and configure any optional settings. For details about the settings, see Create an Elasticsearch Index (Elasticsearch Indexing Step).



4. When you have finished configuring the Details tab, click the **Query** tab. The tab includes a template for writing a SPARQL SELECT query that incorporates the Graph Data Interface (GDI) service to generate an index on the Elasticsearch server.



Unlike queries for other steps, this step does not run an INSERT query because the data is not being inserted into AnzoGraph. Edit the template as needed. You can click the **Preview in Query Builder** button to open the query in the Query Builder, where you can perform practice runs to see results without having to refresh the layer. For more information about writing GDI queries against an Elasticsearch source, see Querying an Elasticsearch Source.

> **Important**
> Do not include Elasticsearch connection or index parameters in the query. Anzo automatically populates that information from the AnzoGraph configuration when the step is run.

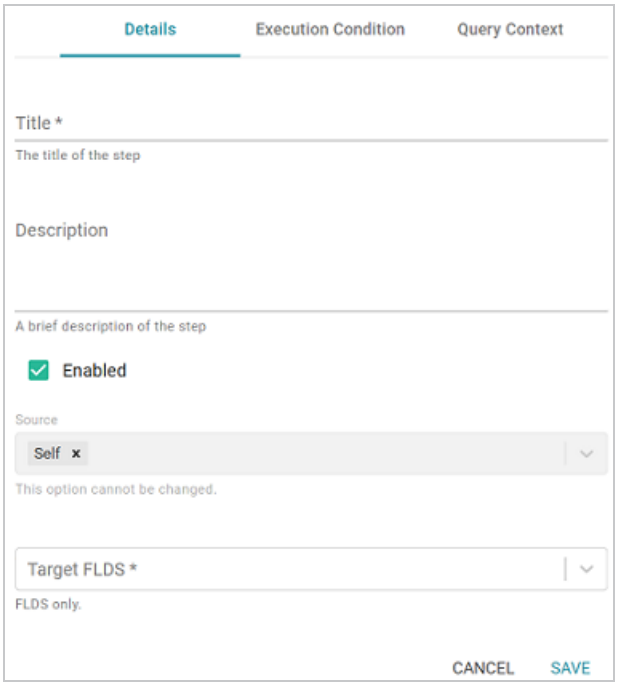
5. When you have completed the indexing query, click **Save** to save the configuration and add the step to the layer.
6. Next, users typically add an Elasticsearch Snapshot Step to the same layer. The step takes a snapshot of the index and saves it to the dataset (FLDS) on disk. Storing the snapshot with

the FLDS ensures that the index is included if the dataset is added to another graphmart. If you want to create a snapshot, continue to Add a Step to Take a Snapshot of the Index below.

Add a Step to Take a Snapshot of the Index

Follow the steps below to add an Elasticsearch Snapshot Step to the new layer and configure it to save a snapshot of the index to the FLDS.

1. On the Data Layers screen, click the menu icon (⋮) for the new layer and select **Add Step/View**.
2. On the Add Step/View screen, select **Elasticsearch Snapshot Step** and click **OK**. The Create dialog box is displayed.



3. On the Details tab, add a name for the step in the **Title** field.
4. Click the **Target FLDS** field and select the FLDS that is created by the Export Step in the graphmart.
5. Configure any other optional settings on the Details tab. For information about the settings, see Take a Snapshot of an Index (Elasticsearch Snapshot Step).
6. Click **Save** to save the configuration and add the step to the layer.

Once you have finished configuring the new layer and steps, reload the graphmart to generate (or update) the FLDS, create the Elasticsearch index, and save a snapshot of the index to the FLDS.

Adding Data Layers to Graphmarts

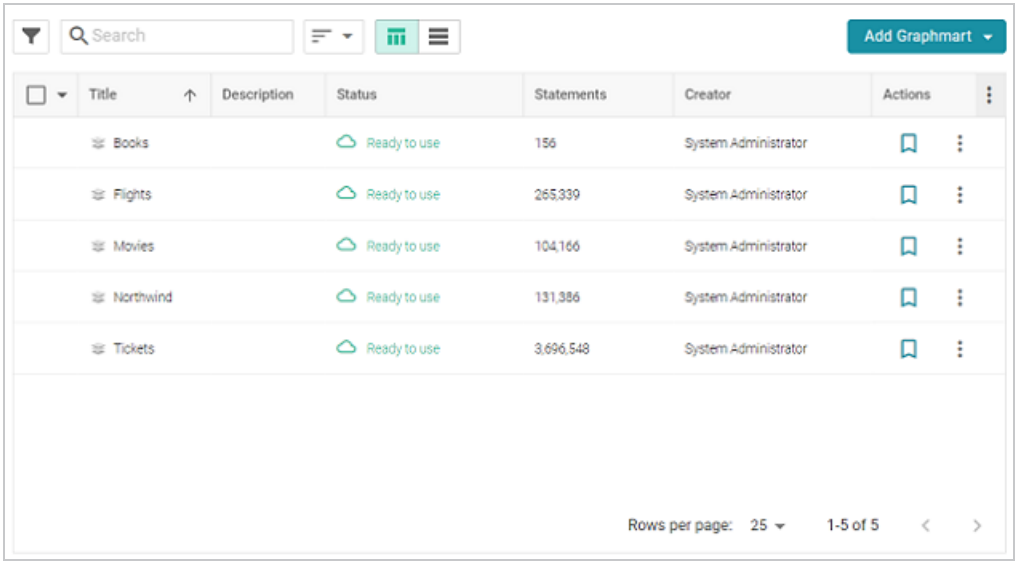
The topics in this section provide instructions for creating and configuring data layers. For conceptual information about graphmarts, layers, and steps, see Graphmart Concepts in the Getting Started Guide.

- Creating a New Layer
- Cloning a Layer
- Using Query Contexts
- Defining Execution Conditions
- Advanced Data Access Settings

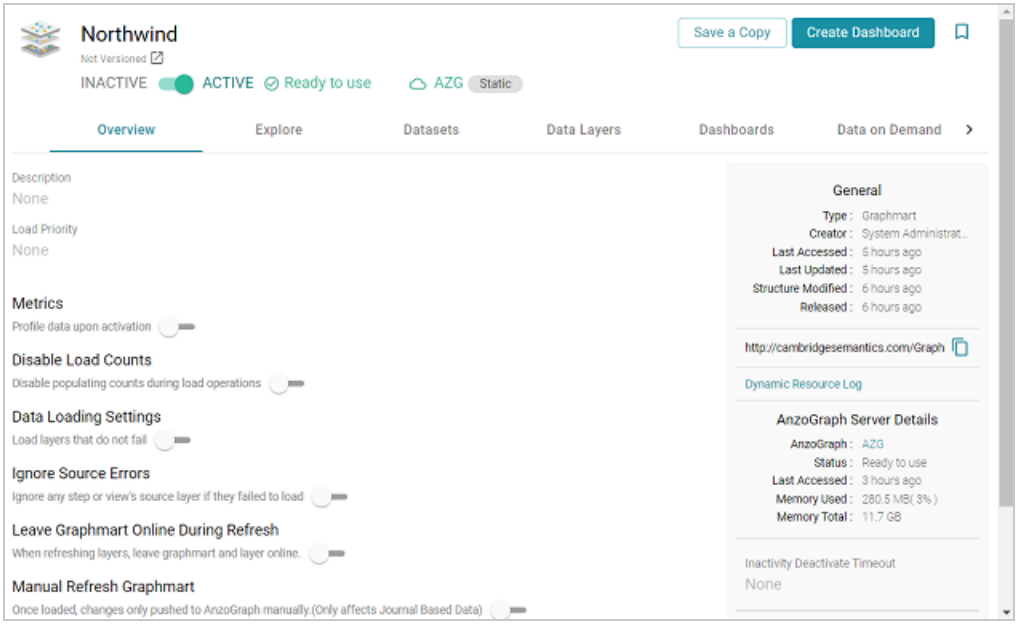
Creating a New Layer

Follow the steps below to create a new data layer in a graphmart. For instructions on creating a new layer by copying an existing one, see Cloning a Layer.

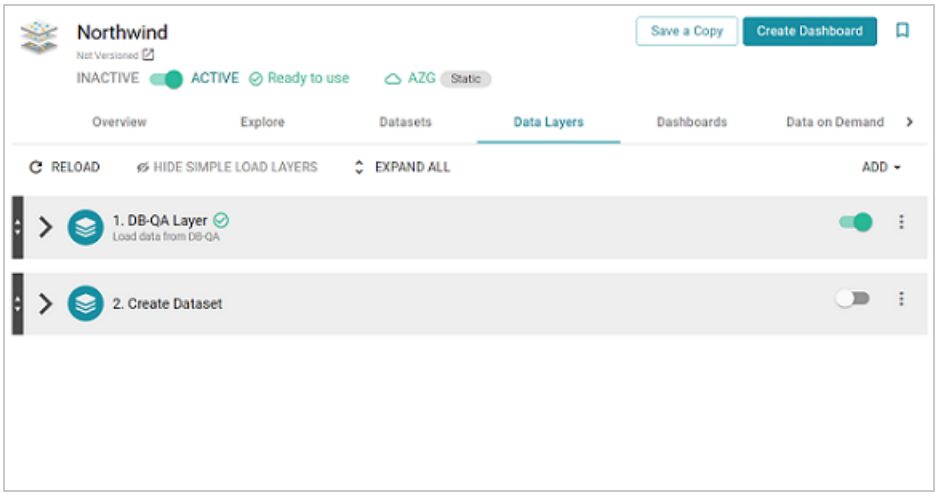
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



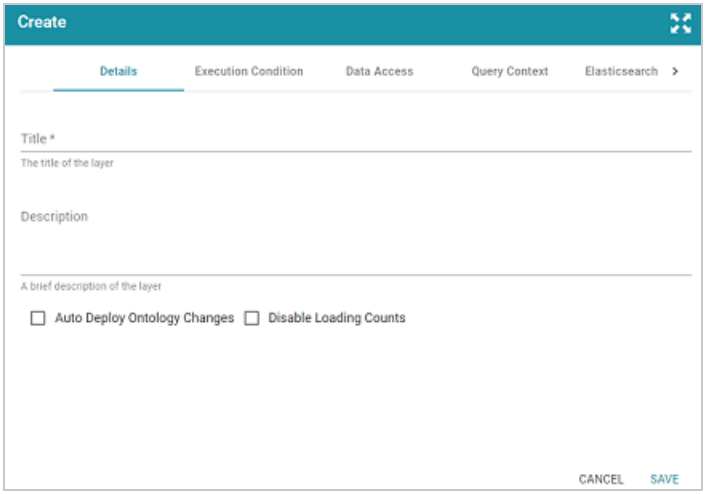
2. Click the name of the graphmart that you want to add a layer to. The Overview tab is displayed. For example:



3. Click the **Data Layers** tab. Anzo displays the existing data layers. For example:



4. Click **Add** and select **New Layer**. Anzo displays the Create dialog box.



5. Specify a name for the layer in the **Title** field and an optional description in the **Description** field.
6. Determine how you want to control changes to this layer's dependent data models:
   - If you want Anzo to automatically deploy to AnzoGraph any changes to the related models without having to manually refresh the layer or graphmart, select the **Auto Deploy Ontology Changes** checkbox.

     > **Note**
     > The **Manual Refresh Graphmart** setting on the graphmart must be **disabled** for automatic deployment of models to work. See Graphmart Settings Reference for information about graphmart settings.

   - If you want model changes to be deployed to AnzoGraph only when this layer (or entire graphmart) is refreshed or reloaded, leave the **Auto Deploy Ontology Changes** checkbox empty (disabled).
7. Determine whether to **Disable Loading Counts** for this layer. This setting controls whether Anzo periodically sends COUNT queries to AnzoGraph while this layer is reloading or refreshing. Disabling the load counts may increase load performance as it decreases the number of queries that are executed during loads.

8. Click **Save** to add the new layer to the graphmart and return to the Data Layers screen.

The new layer becomes the last layer in the graphmart. If you want to change the order of the layers, you can click the black bar on the left side of a layer and drag the layer up or down. Data layers are processed from top to bottom.

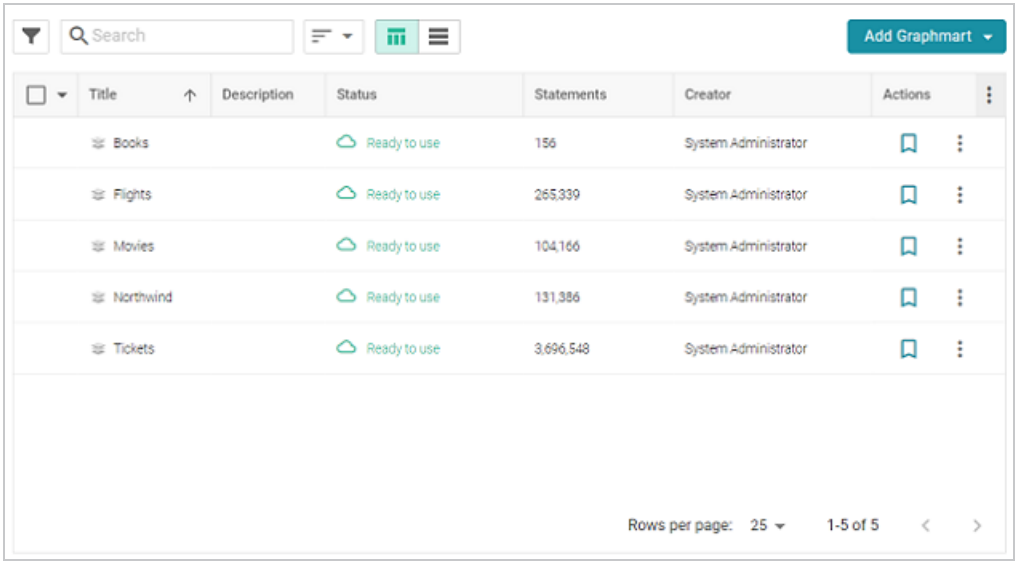
> **Tip**
> The Refresh icon (↻) on the new layer indicates that the layer is out of sync with the data that is in AnzoGraph. Once you configure the new layer and add data processing steps, you can click the **Reload** button (↻) at the top of the screen to reload the entire graphmart, or you can click the **Refresh** icon (↻) on the layer to reload only that layer.

See Adding Steps to Layers for instructions on creating steps.

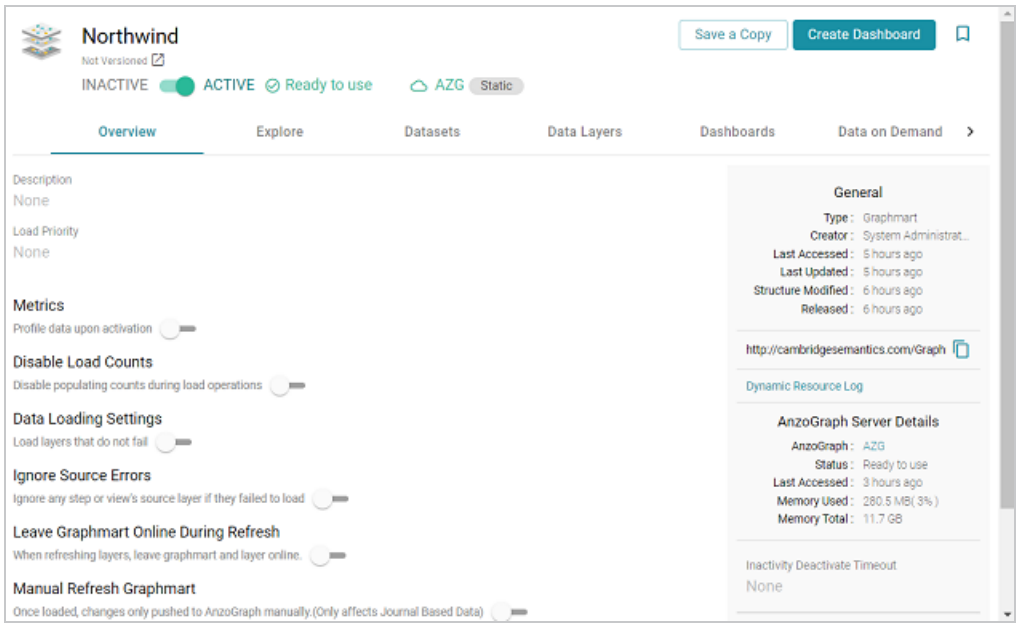
Cloning a Layer

Follow the steps below to create a new data layer by copying an existing one from any graphmart. For instructions on creating a layer from scratch, see Creating a New Layer.

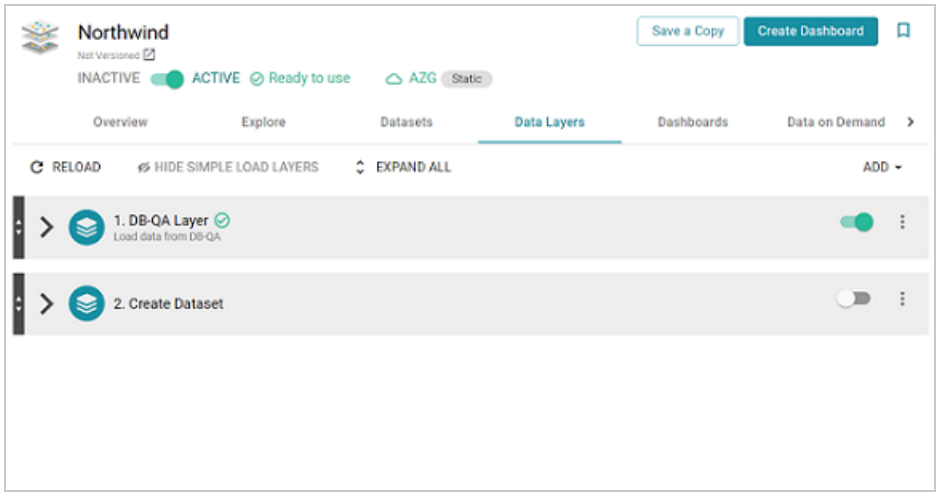
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



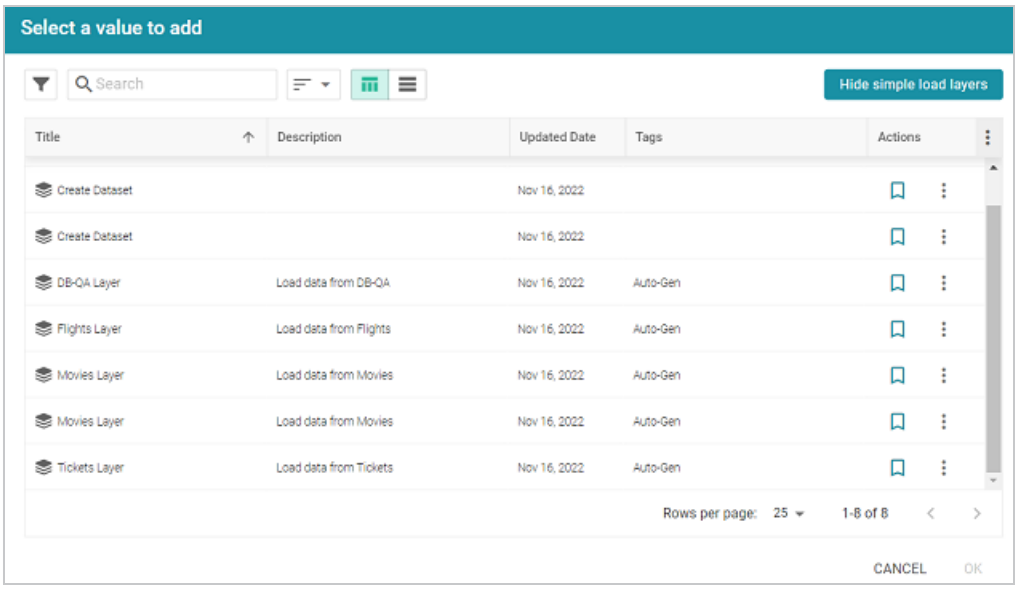
2. Click the name of the graphmart that you want to add a layer to. The Overview tab is displayed. For example:



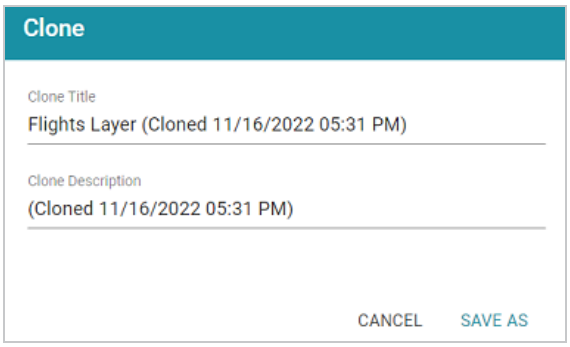
3. Click the **Data Layers** tab. Anzo displays the existing data layers. For example:



4. click **Add** and select **Add Existing**. Anzo opens the Select a value to add dialog box, which lists the existing layers for all graphmarts. For example:



5. Select the layer that you want to copy and click **OK**. The Clone dialog box is displayed. For example:



6. In the Clone dialog box, you have the option to edit the **Clone Title** and/or **Clone Description**.
7. Click **Save As** to add the cloned layer and any steps that the layer contains to the graphmart.

The new layer becomes the last layer in the graphmart. If you want to change the order of the layers, you can click the black bar on the left side of a layer and drag the layer up or down. Data layers are processed from top to bottom.

Tip

The Refresh icon ( ) on the new layer indicates that the layer is out of sync with the data that is in AnzoGraph. Once you configure the new layer and add data processing steps, you can click the **Reload** button ( ) at the top of the screen to reload the entire graphmart, or you can click the **Refresh** icon ( ) on the layer to reload only that layer.

See Adding Steps to Layers for instructions on creating steps.

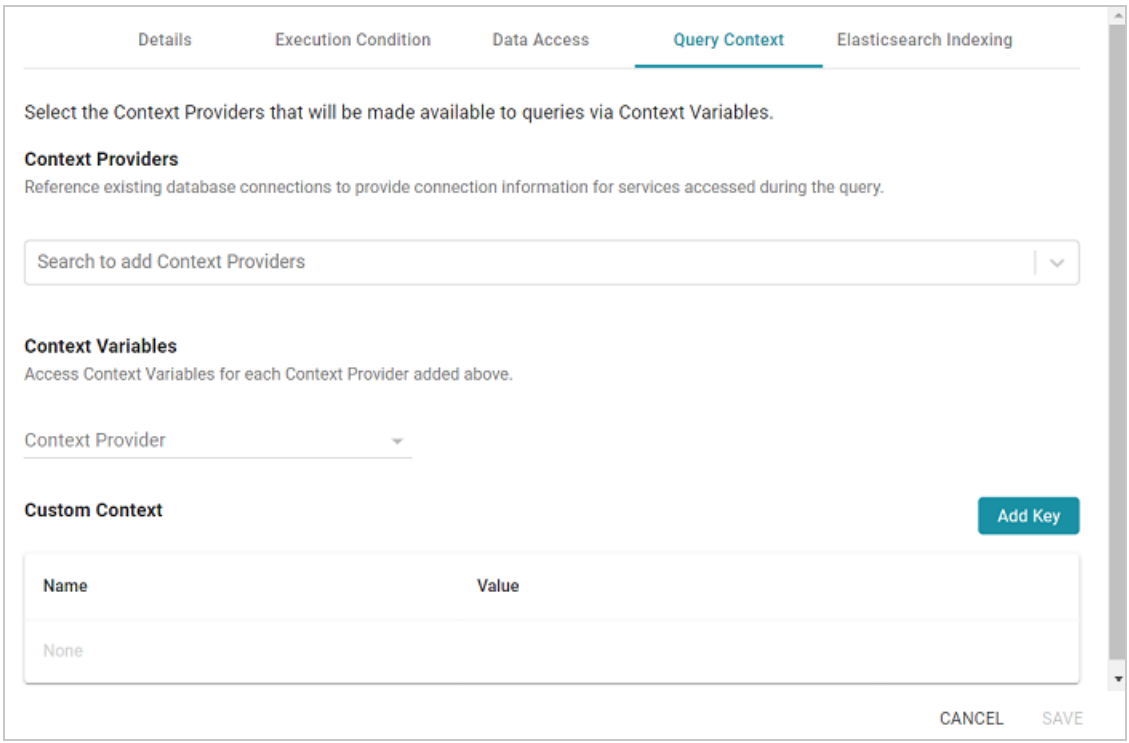
Using Query Contexts

When you use the Graph Data Interface (GDI) for querying data sources, you may connect to sources that require input of sensitive connection and authorization information such as keys, tokens, and user credentials. When configuring a step that runs a GDI query, Cambridge Semantics recommends that you refer to a Query Context whenever possible. Contexts store sensitive information as key-value pairs. Queries reference only the keys from the context and the sensitive values are abstracted from the requests that are sent to the data source and AnzoGraph. This topic provides information on configuring Query Contexts and referring to Context Variables in a query.

- Overview of Query Contexts
- Referencing Context Variables in a Query

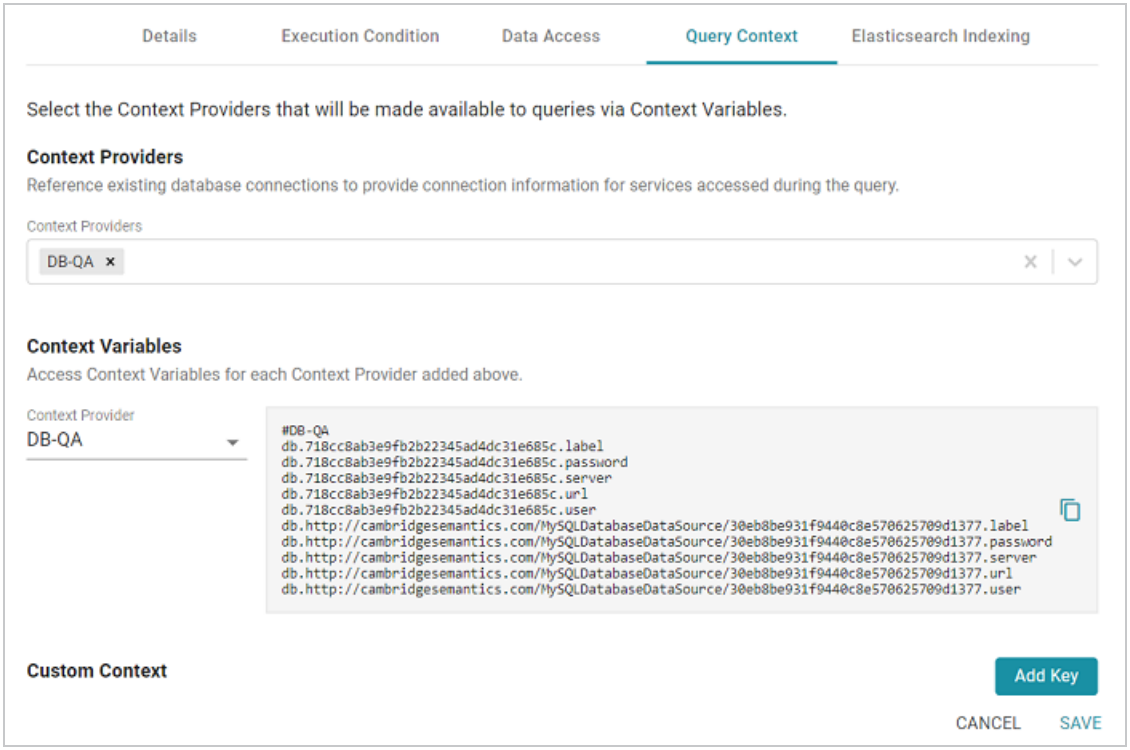
Overview of Query Contexts

Query contexts are accessible from the **Query Context** tab that is available when creating or editing a data layer or step. The image below shows the Query Context tab for a layer.



Context Providers

Connections in Anzo implement the **Context Provider** interface. For example, file store connections and data source connections provide contexts (in the form of JSON objects) that contain key-value pairs. The contexts contain the source connection details such as URLs, database names, user names, passwords, and tokens. A context is passed to the data source when a request is made against that source. To use one of the Anzo-generated Context Providers that was created for a pre-existing connection, select that provider from the drop-down list. When you select a provider, the variables from that context are displayed under Context Variables, as shown in the image below:

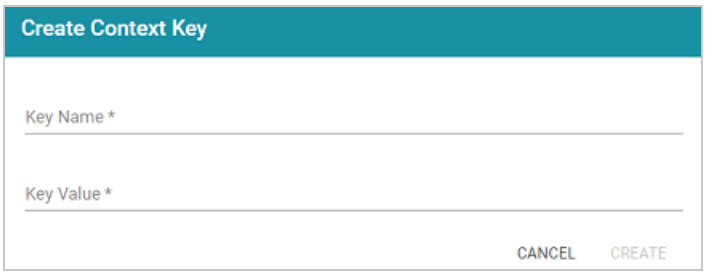


Context Variables

When you select a Context Provider, the variables from the selected context are displayed under Context Variables. You can copy the text and then use any key in the list as a variable in any query that connects to this data source. Typically there are two versions of each variable. Either version can be used in a query. The short versions are generated from the long versions for ease of use and readability in queries. For information about referencing variables in a query, see Referencing Context Variables in a Query below.

Custom Context

Custom Contexts are user-defined key-value pairs that are not associated with a particular Context Provider. To add a key and define its value, click the **Add Key** button. Then specify the **Key Name** and **Key Value** in the Create Context Key dialog box. Click **Create** to add the key-value pair to the context.



Tip

When defining custom keys at the layer level, make sure that the key names are unique. A layer may have different steps that connect to different sources but reference the same custom context. The image below, for example, creates URL, username, and password Context Keys for a MySQL database.



Referencing Context Variables in a Query

The format that you use for referencing a Context Variable in a query depends on the type of AnzoGraph plugin or extension that is being called by the query. Generally, contexts are only used in steps that contain GDI queries. When referencing context keys in GDI queries, use the following format:

```
{{@context_key_name}}
```

For example, the following GDI query references three variables from the Context Provider shown above: url, user, and password.

```
PREFIX s:     <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT *
WHERE
  {
    SERVICE <http://cambridgesemantics.com/services/DataToolkit>
```

```
      {
         ?data a s:DbSource ;
            s:url "{{@db.718cc8ab3e9fb2b22345ad4dc31e685c.url}}" ;
            s:username "{{@db.718cc8ab3e9fb2b22345ad4dc31e685c.user}}" ;
            s:password "{{@db.718cc8ab3e9fb2b22345ad4dc31e685c.password}}" ;
            s:selector "[dbo].[FILM]" ;
            ?year ("[YEAR]" xsd:int);
            ?length (xsd:int) ;
            ?title (xsd:string) ;
            ?subject ("[dbo].[FILM].[SUBJECT]" xsd:string) ;
            ?actor ("[ACTOR]" xsd:string) ;
            ?actress (xsd:string) ;
            ?director (xsd:string) ;
            ?popularity (xsd:int) ;
            ?awards (xsd:string) ;
            ?image (xsd:string) .
         FILTER(?year >= 1990 && ?year < 2000)
         FILTER(?subject = "Drama" || ?subject = "Action")
         FILTER(?length <= 90)
  }
}
```

Defining Execution Conditions

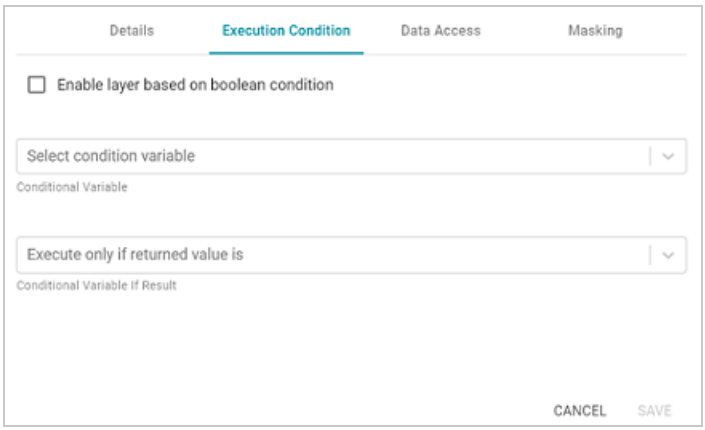
Execution conditions can be defined at the data layer or step level and are used to conditionalize the execution of the layer or step based on the result of a specified Validation Condition.

> **Note**
> Validation Conditions are defined in Validation Steps. In order to set up an Execution Condition, the Graphmart needs to have at least one **Validation Step** that defines a **Condition Variable**. Validation Conditions can be used across all Data Layers in the same Graphmart. For guidance on configuring a Validation Step, see Validate the Data (Validation Step).

This topic focuses on configuring an execution condition at the data layer level.

Configuring an Execution Condition

Execution Conditions are configured from the **Execution Condition** tab that is available when creating or editing a layer or step. The image below shows the Execution Condition tab for a layer.



Enable Layer Based on Boolean Condition

This setting indicates whether to enable this data layer only if the returned value from the Validation Condition is either **true** or **false**. You specify true or false in the **Conditional Variable If Result** field. If the Validation Condition fails, the layer is disabled.

Conditional Variable

This field specifies the variable that you want to base this execution condition on. If the list is empty, that means either there are no Validation Steps in the graphmart or a Validation Step exists but it does not include a Condition Variable (defined on the Options tab). Refer to Validate the Data (Validation Step) for more information.

Conditional Variable If Result

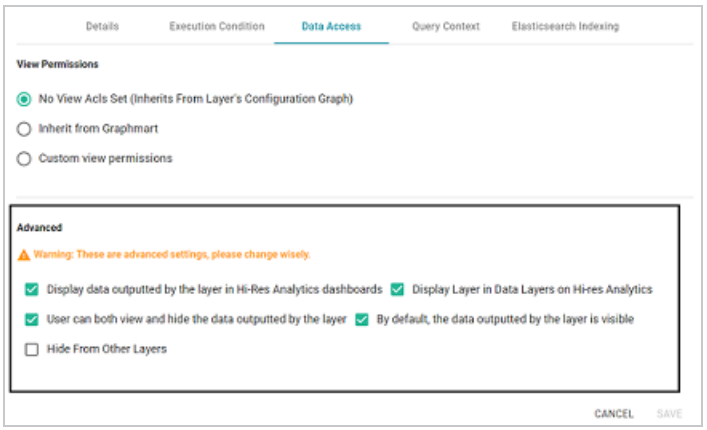
If you enabled the **Enable Layer Based on Boolean Condition** setting, select **true** or **false** from the drop-down list. The data layer will be enabled only if the result of the Validation Step Query matches the value that you specified. If `Enable Layer Based on Boolean Condition` is disabled, leave this field blank.

Advanced Data Access Settings

This topic provides reference information about the Advanced Data Layer Hi-Res Analytics settings that control how a layer is exposed to and affects Hi-Res Analytic dashboards.

> **Important**
> Changing these settings can have unexpected consequences.

The Advanced Hi-Res Analytics settings are available on the **Data Access** tab when you create or edit a data layer:



The sections below describe each of the available settings:

- Display data outputted by the layer in Hi-Res Analytics dashboards
- Display Layer in Data Layers in Hi-Res Analytics
- User can both view and hide the data outputted by the layer
- By default, the data outputted by the layer is visible
- Hide from Other Layers

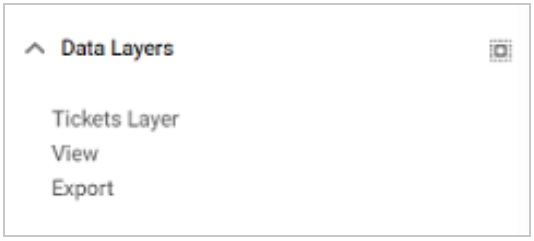
Display data outputted by the layer in Hi-Res Analytics dashboards

This setting controls whether the data generated by the steps is available to query and display in Hi-Res Analytics dashboards:

- When the setting is **enabled** (the default value), the layer's data is available to dashboards.
- When the setting is **disabled**, other data layers in the graphmart can use the layer's data, but the data is not available to dashboards.

Display Layer in Data Layers in Hi-Res Analytics

This setting controls whether Anzo displays the layer in the Data Layers panel on Hi-Res Analytics dashboards. The image below shows an example Data Layers panel:



- When the setting is **enabled** (the default value), the layer is listed in the Data Layers panel in dashboards.
- When the setting is **disabled**, the layer's data is always used in dashboards but users do not see the layer listed in the Data Layers panel.

User can both view and hide the data outputted by the layer

This setting controls whether users have the option to show and hide the layer in the Data Layers panel on dashboards:

- When the setting is **enabled** (the default value), the layer is listed in the Data Layers panel in dashboards and users have the option to show and hide the layer.
- When the setting is **disabled**, whether the layer shows up in the Data Layers panel depends on the **By default, the data outputted by the layer is visible** setting. If the layer is visible in the Data Layers panel ("By default, the data outputted by the layer is visible" is enabled), users cannot toggle it on and off.

By default, the data outputted by the layer is visible

This setting controls whether the data that is generated by the steps in the layer is visible in queries and Hi-Res Analytics dashboards:

- When the setting is **enabled** (the default value), the layer is listed in the Data Layers panel in dashboards and is selected by default. The layer's data is also included by default when queries are run against the graphmart.
- When the setting is **disabled**, the layer shows up in the Data Layers panel but is not selected. To include the layer's data in Hi-Res Analytic queries, users must select the layer. In addition, the layer's data is automatically excluded from queries that are run against the graphmart. To include data from the layer in results, the queries must explicitly list the layer's URI.

Hide from Other Layers

This setting controls whether the other layers in the graphmart can act upon the source data in this layer.

- When the setting is **disabled** (the default value), this layer is available as a choice in the **Source data to act upon** drop-down list when a step's source is configured.
- When the setting is **enabled**, this layer is not listed as a choice in the **Source data to act upon** list when the a step's source configured.

Adding Steps to Layers

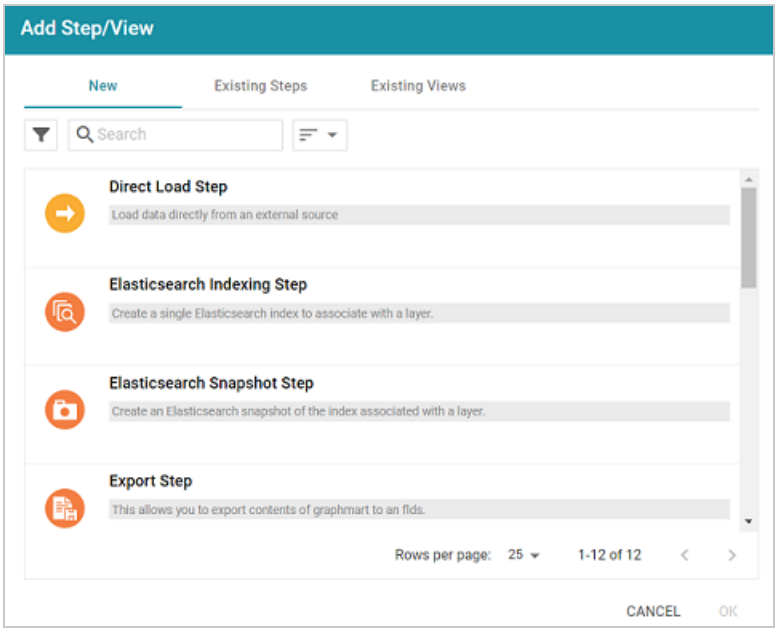
The steps in a data layer perform the data operations, such as loading, creating, deleting, changing, or exporting data. You can add any number of steps to a layer. The topics in this section provide information about configuring each type of step.

- Directly Load a Data Source (Direct Load Step)
- Create an Elasticsearch Index (Elasticsearch Indexing Step)
- Take a Snapshot of an Index (Elasticsearch Snapshot Step)
- Export Data to an FLDS (Export Step)
- Load a Dataset from the Catalog (Load Dataset Step)
- Pre-Compile a Query (Pre-Compile Query Step)
- Create a Reusable Query Template
- Run a Transformation Query (Query Step)
- Infer New Data (RDFS+ Inference Step)
- Validate the Data (Validation Step)
- Construct a View of the Data (View Step)

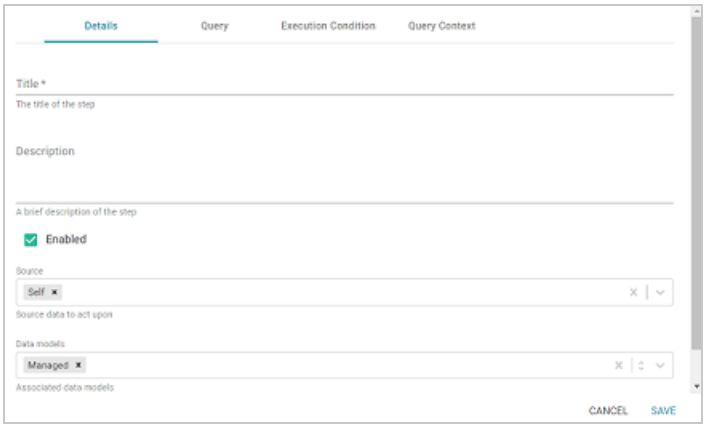
Directly Load a Data Source (Direct Load Step)

With no mapping required, a Direct Load Step can be used to automatically generate a graph and model for a data source. The Direct Load Step is the only type of step with the ability to manage generated models. A model that is generated by a Direct Load Step is automatically registered in Anzo is linked to and managed by the layer that contains the step. If a query is changed, additional Direct Load Steps are added to the same layer, or the underlying source schema changes, the managed model is automatically updated when the graphmart is reloaded or refreshed. Follow the steps below to create a Direct Load Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new Direct Load step, select **Direct Load Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:

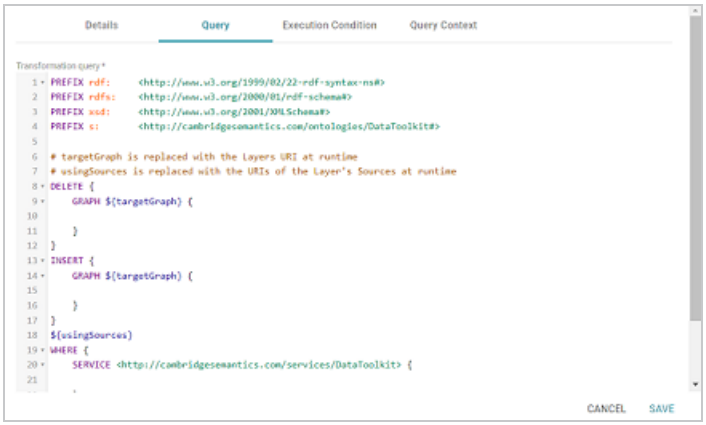


4. On the Details tab, configure the following options as needed:
    - **Title**: The required name of the step.
    - **Description**: An optional short description of the step.
    - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
    - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
        - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
        - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
        - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
        - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.
    - **Data models**: This optional field specifies the model or models to associate with this step. By default, **Managed** is selected. If you are onboarding a source that does not have a model, make sure Managed remains specified so that the step generates a model. See Important Notes on Managed Models below for more information about managed models.

The Data Models list displays all of the available models. By default, the field is set to **Exclude System Data** (⇕). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (✕). When system data is included, the drop-down list displays the system models in addition to the user-generated models.

- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. When you have finished configuring the Details tab, click the **Query** tab. This tab defines the query that this step should run.



6. Typically Direct Load Step queries are GDI RDF and Ontology Generator queries. Using a relatively simple SPARQL query, the GDI Generators recognize the structure of a data source and automatically generate the necessary statements. Invoking the Generators is preferable when the structure of the data is very complex, such as a JSON data source with many inner repeating structures or a database with many tables and keys. When the source contains complex structures, only the required statements are generated, avoiding cross-products and optimizing query execution and memory usage. For details about writing GDI Generator queries, see GDI Generator Query Syntax.

> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.

7. When you have finished writing the query, click **Save** to save the step configuration.

Once the Details tab is configured and the query is written, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Important Notes on Managed Models

Though an ontology that is generated in a Direct Load Step is registered in Anzo and is available for viewing in the Model editor, **the model is owned and managed by the data layer that contains the Direct Load Step**. That means any manual changes made to the model outside of the step, such as from the Model editor, will be overwritten any time the graphmart or layer is refreshed or reloaded. **Do not modify generated managed models except by editing (or adding) Direct Load Step queries**. For information on updating managed models, see Editing a Managed Model.

There is only one managed model per layer. If you include multiple Direct Load Steps in the same layer, they will all update the same ontology. This functionality can be useful if you want to align the data and generated model across multiple steps. If you have multiple sources that are not intended to align or update the same model, create separate layers.

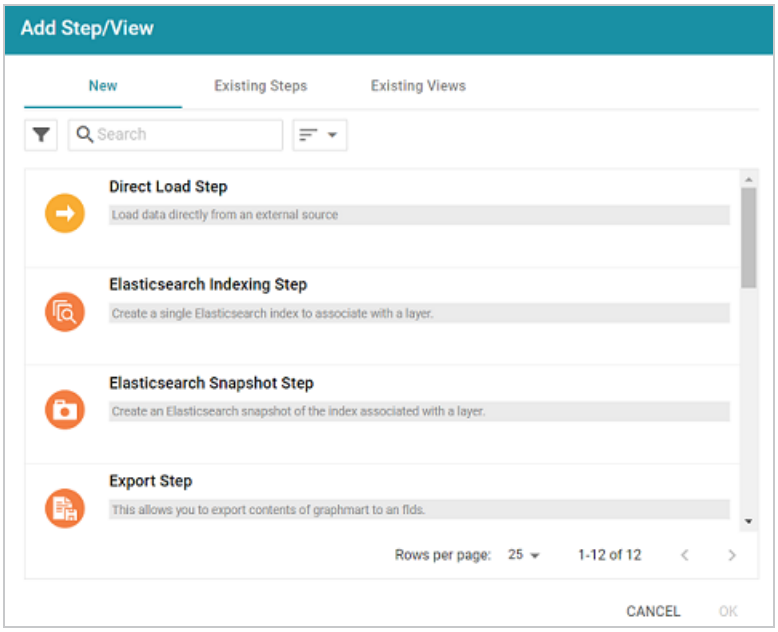
If you delete a layer that includes a managed model, the model is also deleted. Use caution when referencing a managed model outside of a graphmart. For example, if you create a dataset and reference a managed model when you select the ontology, the reference will break if the data layer that manages the model is deleted.

Create an Elasticsearch Index (Elasticsearch Indexing Step)

This topic provides guidance on configuring an Elasticsearch Indexing Step that generates an Elasticsearch index from data in a graphmart. Follow the steps below to create an Elasticsearch Indexing Step.

> **Tip**
> For an overview on creating an Elasticsearch index and snapshot from a graphmart, see Creating an Elasticsearch Index from a Graphmart.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.

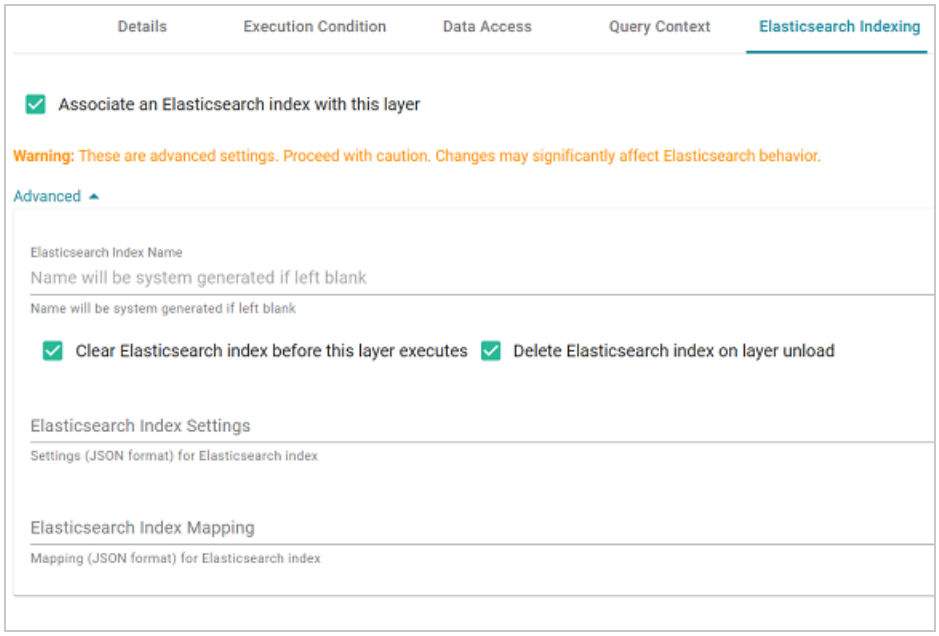


3. To create a new Elasticsearch Indexing step, select **Elasticsearch Indexing Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step.

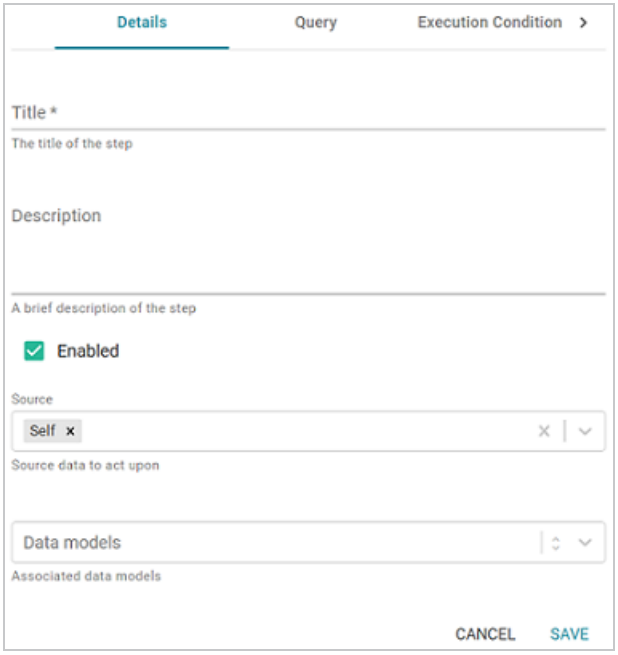
   If this is the first Elasticsearch-related step in the layer, the following warning is displayed:



The warning is to inform you that the Elasticsearch index-related configuration settings at the layer level are automatically adjusted to the default settings for layers that contain Elasticsearch Indexing or Snapshot Steps. Those settings are on the layer's Elasticsearch Indexing tab. The image below shows the tab with the default settings:



4. If necessary, click **OK** to close the warning. The Details tab is displayed:



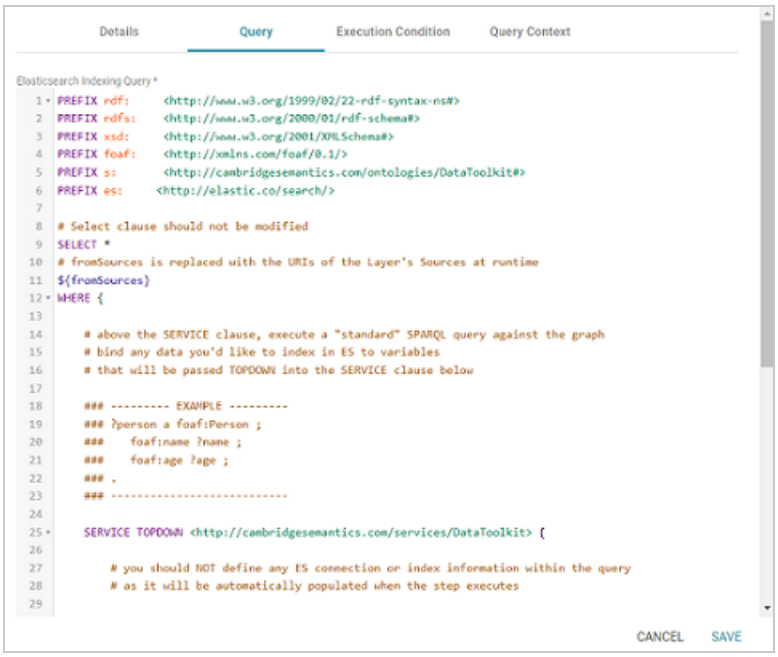
5. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
     - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
     - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.

- **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
- ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.

- **Data models**: This optional field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (⇕). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (✕). When system data is included, the drop-down list displays the system models in addition to the user-generated models.

6. When you have finished configuring the Details tab, click the **Query** tab. This tab contains the Elasticsearch indexing query to run. The template includes the syntax for writing a SPARQL SELECT query that uses the Graph Data Interface (GDI) to generate the index on the Elasticsearch server. Unlike queries for other steps, this step does not run an INSERT query because the data is not being inserted into AnzoGraph. For general information about writing GDI queries against Elasticsearch sources, see Querying an Elasticsearch Source.

> **Note**
> Do not include Elasticsearch connection or index parameters in the query. Anzo automatically populates that information from the AnzoGraph configuration when the step is run.



7. Click **Save** to save the step configuration.

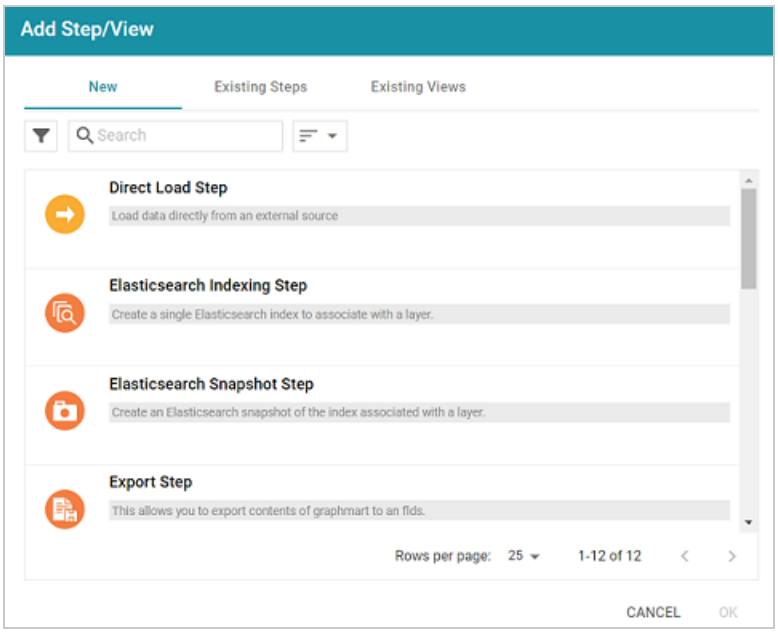
Once the Details tab is configured and the query is written, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Take a Snapshot of an Index (Elasticsearch Snapshot Step)

This topic provides guidance on configuring an Elasticsearch Snapshot Step that generates a snapshot of an Elasticsearch index that was created from a graphmart. Follow the steps below to create an Elasticsearch Snapshot Step.

> **Tip**
> For an overview on creating an Elasticsearch index and snapshot from a graphmart, see Creating an Elasticsearch Index from a Graphmart.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.

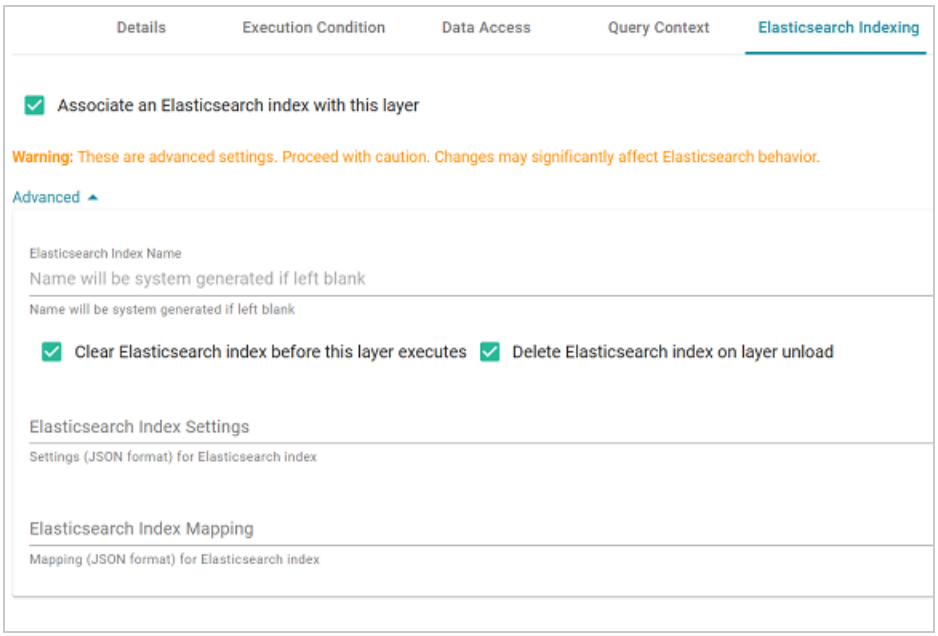


3. To create a new Elasticsearch Snapshot step, select **Elasticsearch Snapshot Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step.

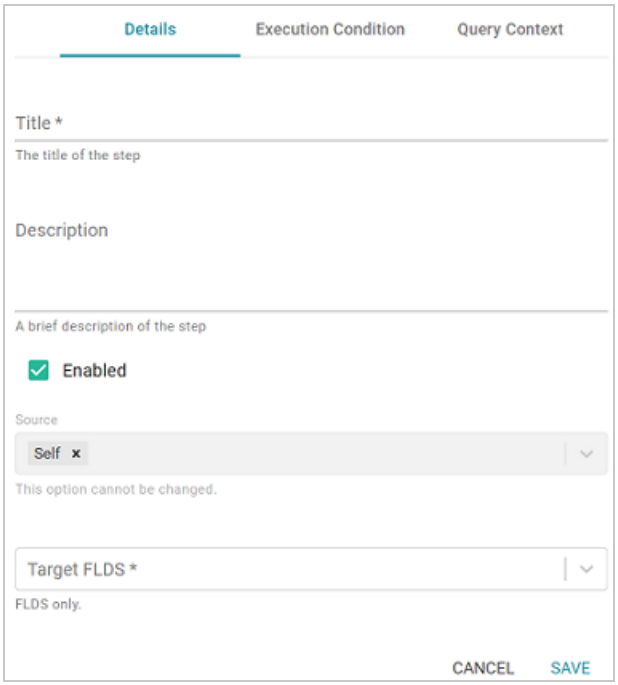
   If this is the first Elasticsearch-related step in the layer, the following warning is displayed:



   The warning is to inform you that the Elasticsearch index-related configuration settings at the layer level are automatically adjusted to the default settings for layers that contain Elasticsearch Indexing or Snapshot Steps. Those settings are on the layer's Elasticsearch Indexing tab. The image below shows the tab with the default settings:



4. If necessary, click **OK** to close the warning. The Details tab is displayed:



5. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.

- **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
- **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
  - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
  - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
  - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
  - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.
- **Target FLDS**: This is the target FLDS to save the snapshot to. Typically this is the FLDS that was created by an Export Step in a previous layer. If an FLDS does not exist, you can select **-Create New-** to create an empty dataset. See Adding an Empty Dataset for an Export Step for instructions.

6. Click **Save** to save the step configuration.

Once the Details tab is configured, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

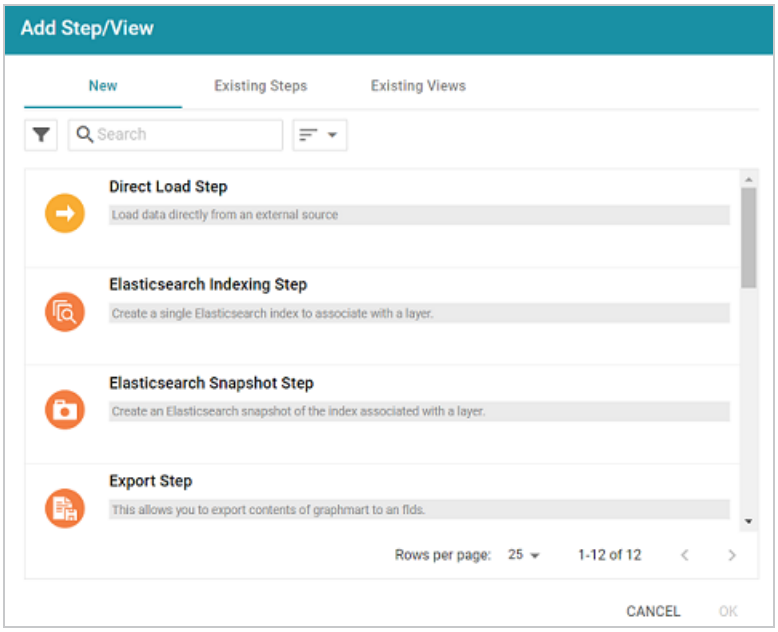
Export Data to an FLDS (Export Step)

This topic provides guidance on configuring an Export Step to use for exporting the knowledge graphs in memory to a file-based linked data set (FLDS) on the shared file store. Follow the steps below to create an Export Step.

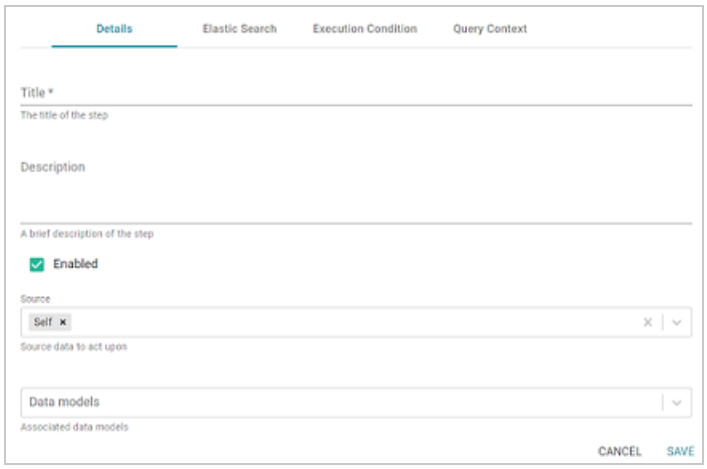
Important

If you add an Export Step to a graphmart that has been activated, you must reload the entire graphmart after adding the step. Simply refreshing the layer or graphmart after adding the step does not create the ontology graph that the Export Step requires.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new Export step, select **Export Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:



4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
     - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
     - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
     - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
     - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.

- **Data models**: This optional field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (). When system data is included, the drop-down list displays the system models in addition to the user-generated models.
- **Target FLDS**: This is the target FLDS for this export. If an FLDS does not exist, you can select **-Create New-** to create an empty dataset. See Adding an Empty Dataset for an Export Step for instructions. If you select an existing target FLDS, you also have the option to specify whether or not to overwrite the existing dataset.
- **Overwrite FLDS**: This setting controls whether the existing FLDS is replaced with the exported files each time the Export Step runs or whether the exported files are added to the existing FLDS:
  - If you want Anzo to replace the current edition of the dataset, select the **Overwrite FLDS** checkbox. When Overwrite FLDS is enabled, Anzo archives the existing files in a new timestamped **export** subdirectory directory under the Target FLDS directory. Each time the Export step runs, Anzo archives the current edition, and creates a new export directory. If you add this dataset to a graphmart, only the latest version of the exported data will be loaded to AnzoGraph.
  - If you want Anzo to add the exported files to the existing FLDS, leave the **Overwrite FLDS** checkbox unchecked. When Overwrite FLDS is disabled, Anzo adds all of the exported components to a **cumulative export** directory under the Target FLDS directory. The dataset will contain the original files as well as all cumulative working editions. If you subsequently add this dataset to a graphmart, all of the data from all of the subdirectories will be loaded into AnzoGraph.
- **Export Binary Store Contents**: This option applies to exports of unstructured graphmarts and controls whether the binary store is exported along with the data.
- **Always Move Binary Store**: This option also applies to exports of unstructured graphmarts and controls whether the binary store is moved or copied during the export.

Since the binary store can be large and have a nested structure, copying the data can take a very long time. Since moving the binary store is almost instantaneous, however, enabling **Always Move Binary Store** can reduce the time it takes to complete the export.

- **Export Elasticsearch Contents**: This option is enabled by default and controls whether Elasticsearch contents, such as associated indexes, are copied to disk in the Target FLDS. If you do not want Elasticsearch contents to be exported to the FLDS, clear the checkbox.

  > **Note**
  > When exporting data from an AnzoGraph instance that does not have an associated Elasticsearch connection, clear the **Export Elasticsearch Contents** checkbox.

- **Keep Elasticsearch Index Online**: This option controls whether the Elasticsearch index that is associated with the dataset remains stored in Elasticsearch or is removed from Elasticsearch once it is exported.

  > **Tip**
  > For information about advanced Elasticsearch options that are available for Export Steps, see Elastic Search Tab below.

- **Generate Metrics**: This option controls whether a data profile is generated before the data is exported. Enabling this option increases the time it takes to run the step, but enabling it ensures that the information on the Dataset Explore is complete if the dataset is viewed. If you load the exported files in the future, the data profile is also loaded.
- **Do Not Create New Edition in Dataset on Export**: This option controls whether or not a new edition is created for the dataset each time the Export Step is run. By default this option is disabled, which means each export results in a new edition. If you do not want a

new edition to be created on each export, select the **Do Not Create New Edition in Dataset on Export** checkbox.

- **Maximum Number of Components in Edition**: This option controls the maximum number of components to retain in an edition. The default value is blank, which means unlimited. If you specify a number in this field and the limit is reached, Anzo ages off the oldest components as new ones are created.

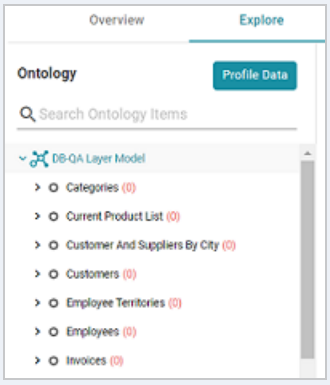
5. Click **Save** to save the step configuration.

Once the Details tab is configured, the step can be run. If you added the step to an active graphmart, make sure that you deactivate and then reactivate the graphmart. Simply refreshing the layer or graphmart after adding the step does not create the ontology graph that the Export Step requires. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Once the graphmart is reactivated, the new dataset becomes available in the Datasets catalog.

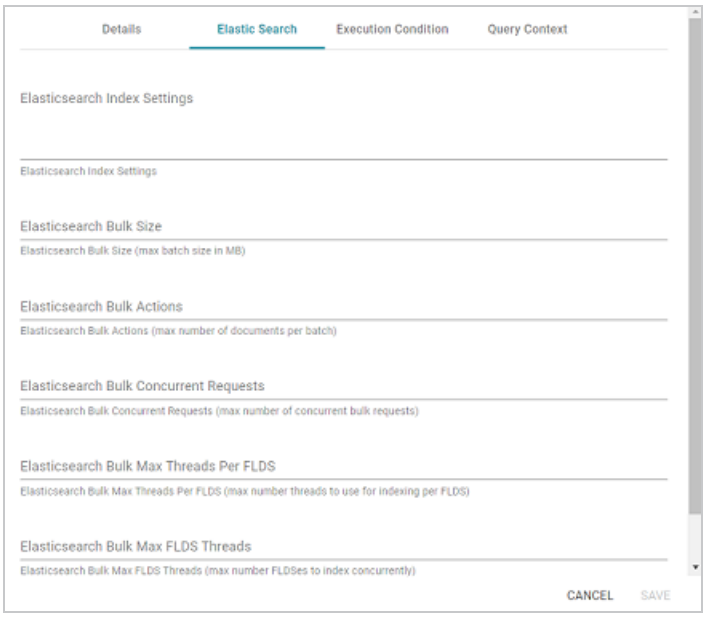
Note

Unless you enable **Generate Metrics**, when you view in the Datasets catalog a dataset that was created by an Export Step, the counts in the Ontology panel on the Explore tab remain **0** until a Data Profile is generated for the dataset. For information about generating a profile, see Generating a Dataset Data Profile. For example, the image below shows the Explore tab for a dataset that was created by an Export Step and has not been profiled.



Elastic Search Tab

The **Elastic Search** tab contains optional settings that you can use to set any desired limits on Elasticsearch indexing processes.

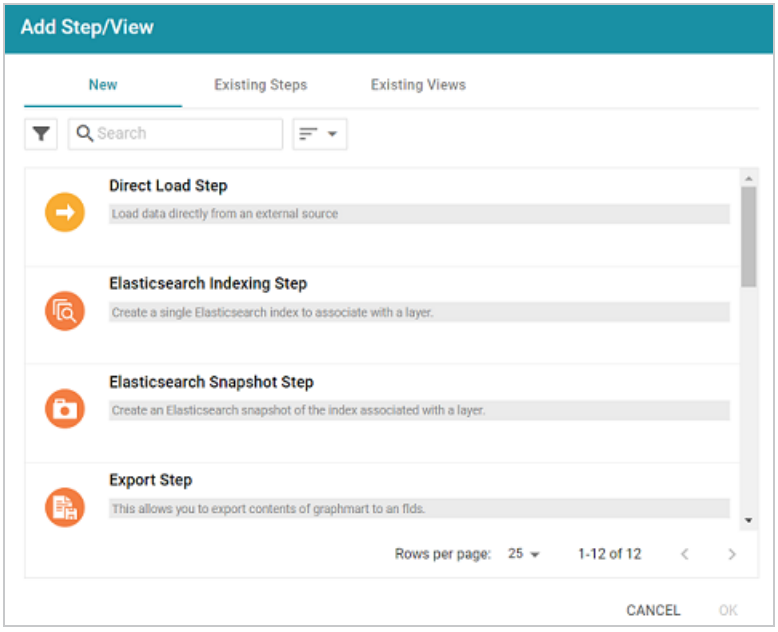


Setting	Description
Elasticsearch Index Settings	A custom JSON list of any Elasticsearch index settings that you want to apply to the export.
Elasticsearch Bulk Size	The maximum batch size in MB.
Elasticsearch Bulk Actions	The maximum number of documents to include in each batch.
Elasticsearch Bulk Concurrent Requests	The maximum number of bulk requests that can run concurrently.
Elasticsearch Bulk Max Threads Per FLDS	The maximum number of threads to use for indexing per file-backed linked data set (FLDS).
Elasticsearch Bulk Max FLDS Threads	The maximum number of FLDSes to index concurrently.

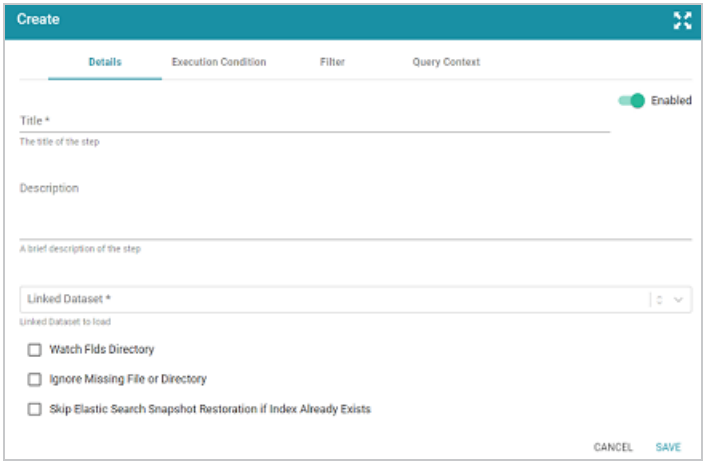
Load a Dataset from the Catalog (Load Dataset Step)

This topic provides guidance on configuring a Load Dataset Step to use for adding a dataset from the Datasets catalog to a graphmart. Follow the steps below to create a Load Dataset Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. Select **Load Dataset Step** and then click **OK**. Anzo creates the step and displays the Details tab:



4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, slide the **Enabled** slider to the left.
   - **Linked Dataset**: This field specifies the dataset to load. The list displays all of the datasets in the Dataset catalog. By default, the field is set to **Exclude System Data** (⇕). If you want to choose a system dataset, click the toggle button on the right side of the field to change it to **Include System Data** (✕). When you select a dataset, the current working edition (Managed Edition) of the dataset is selected as the data to load. If you want to change the edition, you can click **Modify Edition** and follow the steps in Modifying an Edition.
   - **Watch FLDS Directory**: This option controls whether the FLDS directory is monitored for changes. If Watch FLDS Directory is enabled and changes to the files in the FLDS directory are detected , Anzo will mark this step (and layer) as needing a refresh.
   - **Ignore Missing File or Directory**: This option controls whether to ignore missing files or subdirectories in the FLDS directory and proceed with the load or fail the step if files or directories are missing.

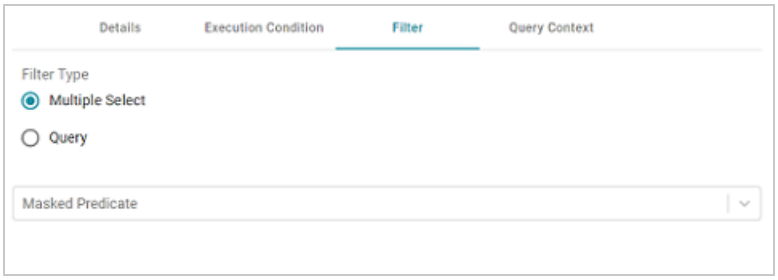
- **Skip Elastic Search Snapshot Restoration if Index Already Exists**: This option applies to graphmarts with Elasticsearch indexes and controls whether Anzo first checks to see if an index with the alias for the dataset already exists in Elasticsearch. If this setting is enabled and the index does exist, Anzo will not reload the index snaphsot into Elasticsearch.

5. Typically when users add a dataset to a graphmart, they want to load the entire dataset. However, if you are familiar with the data and want to exclude certain predicates from the dataset or write an INSERT query that filters the data, you can configure filtering options on the Filter tab. For information, see Filter Tab below.
6. Click **Save** to save the step configuration.

Once the Details tab is configured, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Filter Tab

The **Filter** tab includes options for filtering out some of the data in the dataset. If you want to load all of the statements in the dataset, do not configure Filter options. If you want to exclude some statements, configure the Filter options.



Multiple Select

This option enables you to exclude certain triples from the load by selecting the predicates to filter out. These are known as **Masked Predicates**. To exclude predicates, select the **Multiple Select** radio button, then click the **Masked Predicate** drop-down list and select a predicate to add it to the Masked Predicate field. Click the field again to select additional predicates. You can remove a property from the masked list by clicking the X next to the predicate name.

Query

If you want to hand-pick the data to load, you can use this option to write a SPARQL query that inserts specific values or filters out certain values. To write a query, select the **Query** radio button, and then type an INSERT query in the text box. For example, you can use the following format to filter out properties from the files:

```
INSERT {
  GRAPH ${targetGraph}{
    ?s ?p ?o.
  }
}
${usingSources}
WHERE {
  ?s ?p ?o .
  FILTER EXISTS { ?s a ?type . }
  FILTER(?type = <URI>)
}
```

Note

Including the **${targetGraph}** and **${usingSources}** parameters are required. Anzo automatically populates the query with the appropriate graph URIs when the step is run.

Important

In load filter queries, URIs are not supported in the object position. To specify a URI as an object, include the standard ?s ?p ?o triple pattern in the WHERE clause and then apply FILTER statements with URIs as needed. URIs are supported in the subject or predicate position.

For example, the following query filters the data in a sample dataset that includes information about people and the events they buy tickets for. The WHERE clause filters the data to load only the triples that are related to person1 (personid=1):

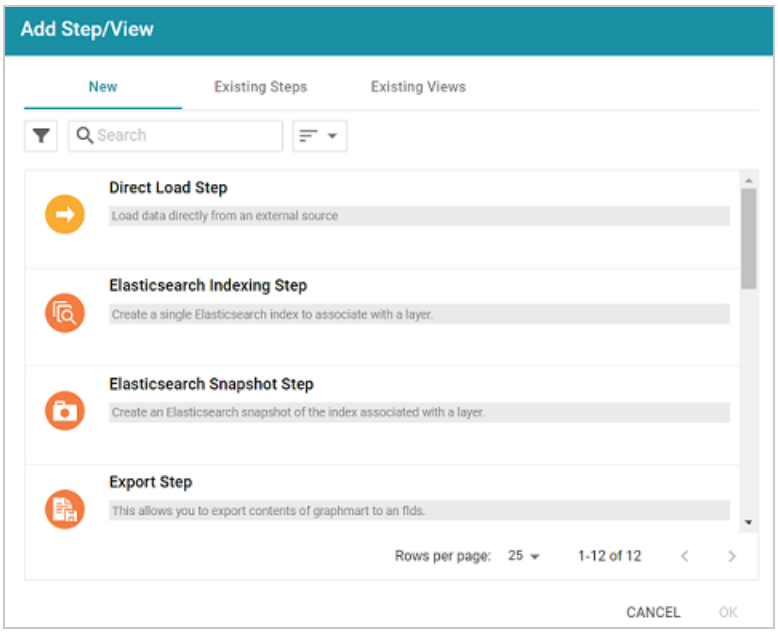
```
INSERT { GRAPH ${targetGraph} {
  ?s ?p ?o
 }
```

```
}
${usingSources}
WHERE {
  ?s ?p ?o ;
  <http://cambridgesemantics.com/ont/autogen/c89d/Tickets#tickit_users_personid> ?id
.
  FILTER (?id=1)
}
```

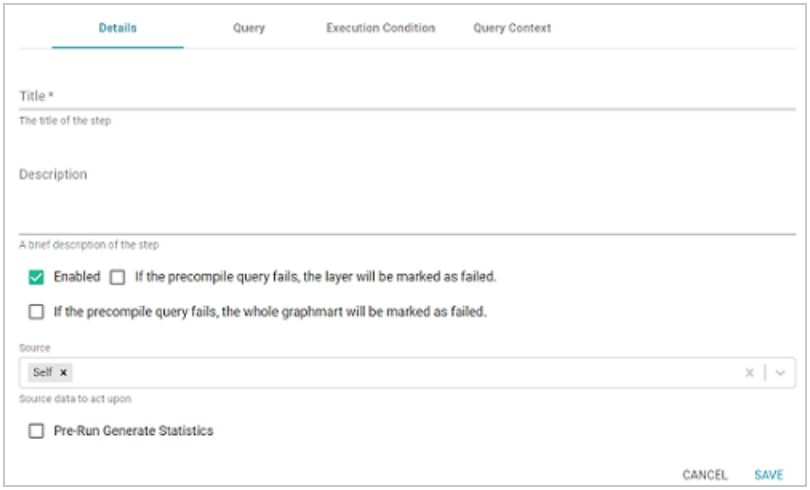
Pre-Compile a Query (Pre-Compile Query Step)

The first time a user runs an analytic query against AnzoGraph, AnzoGraph performs a code compilation process to generate the code for running that query. It then executes the query using that compiled code, and the same code is reused for subsequent runs of the query. If you determine that a particular query has a long code compilation time, you can add that query to a Pre-Compile Query Step. That way the query is run during the graphmart load and the compiled code is available before an end-user runs that query. Follow the steps below to create a Pre-Compile Query Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new Pre-Compile Query Step, select **Pre-Compile Query Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:



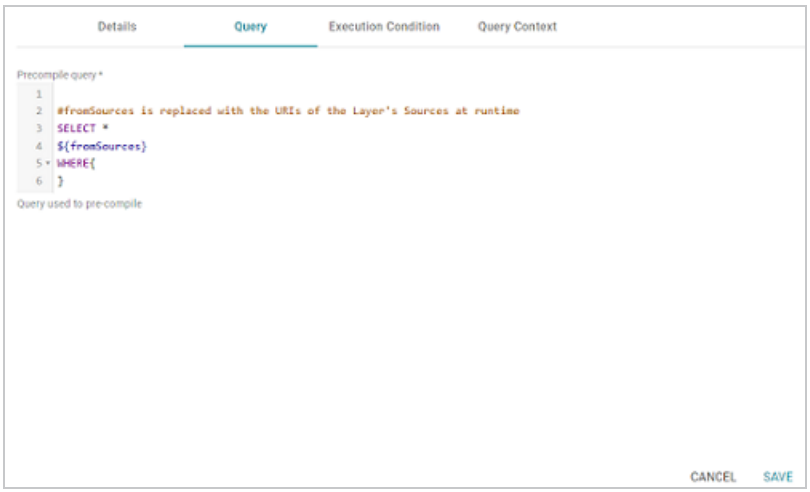
4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **If the precompile query fails, the layer will be marked as failed**: This option controls whether loading the data layer that contains this step is completed or aborted if this step fails. Select this option if you want Anzo to fail the layer if this step fails.
   - **If the precompile query fails, the whole graphmart will be marked as failed**: This option controls whether loading the graphmart is completed or aborted if this step fails. Select this option if you want Anzo to fail the entire graphmart if this step fails.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:

- **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
- **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
- **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
- ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.

- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. When you have finished configuring the Details tab, click the **Query** tab. This tab contains the query to pre-compile. The tab provides a template for writing a SPARQL SELECT query. You can edit the template to write the query, or you can paste in query contents that you copied from a log file, dashboard, the Query Builder, etc.

   Note
   Make sure that you include the `${fromSources}` parameter in the query. Anzo automatically populates the query with the appropriate source graph URIs according to the Source configured from the Details tab.



> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.

6. Click **Save** to save the step configuration.

Once the Details tab is configured and the query is written, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Create a Reusable Query Template

There are two types of steps that enable you to create a query template that is reusable across data layers and graphmarts:

Templated Step

Templated Steps use user-defined key-value pairs. The keys are represented by parameters in the query. Creating the key-value pairs requires familiarity with the data and properties defined in the model. When this step is reused, users do not need to rewrite the query; they modify the values for

the keys.

Query-Driven Templated Step

Query-Driven Templated Steps are similar to Templated Steps in that they provide a way to create query templates that use parameters to represent key-value pairs. The queries are reusable across datasets because the existing parameters can be substituted for alternate key-value pairs. The difference between the two types of steps is that the key-value pairs for Templated Steps are user-defined. In Query-Driven Templated Steps, a parameter query is run that automatically generates the key-value pairs. Then the defined template query is run for each key-value solution from the parameter query.

This section includes instructions for creating both types of templated steps.

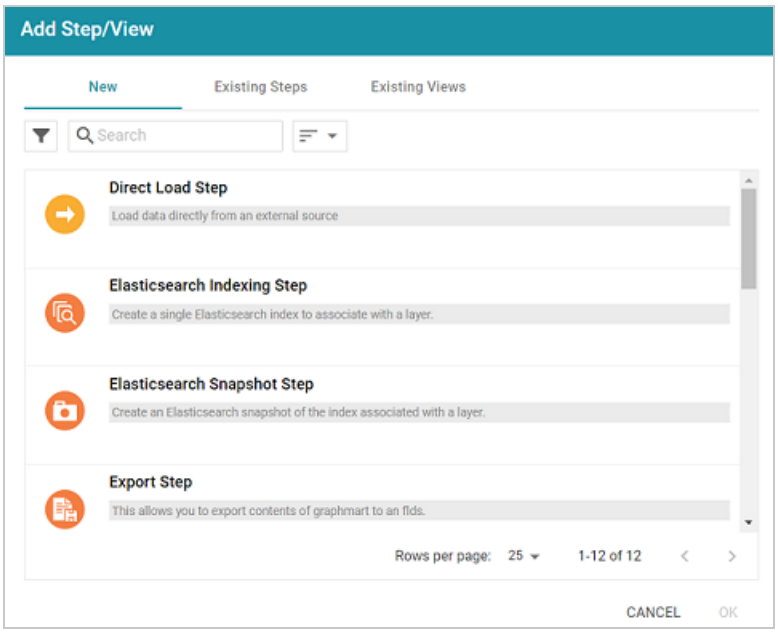
- Create a Templated Step
- Create a Query-Driven Template

Create a Templated Step

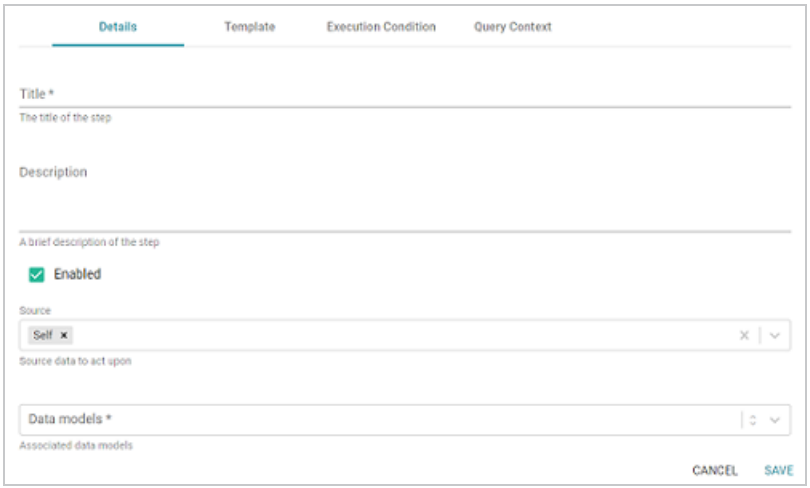
Templated Steps use user-defined key-value pairs. The keys are represented by parameters in the query. When this step is reused, users do not need to rewrite the query; they modify the values for the keys. Follow the steps below to create a Templated Step.

> **Tip**
> This type of template step uses key-value pairs that are user-defined. Creating the key-value pairs requires familiarity with the data and properties defined in the model. To create a query template that enables you to run a query and automatically generate the key-value pairs, see Create a Query-Driven Template.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



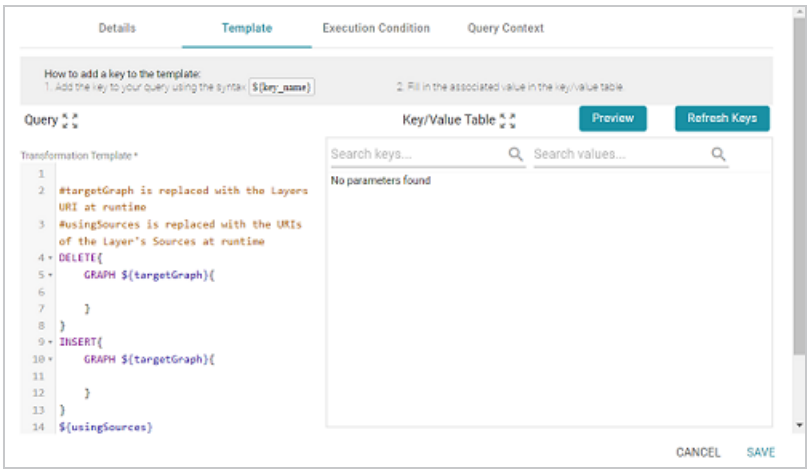
3. To create a new Templated step, select **Templated Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:



4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.

- **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
- **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
  - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
  - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
  - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
  - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.
- **Data models**: This required field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (). When system data is included, the drop-down list displays the system models in addition to the user-generated models.
- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. When you have finished configuring the Details tab, click the **Template** tab. This tab defines the template query and the key-value pairs.



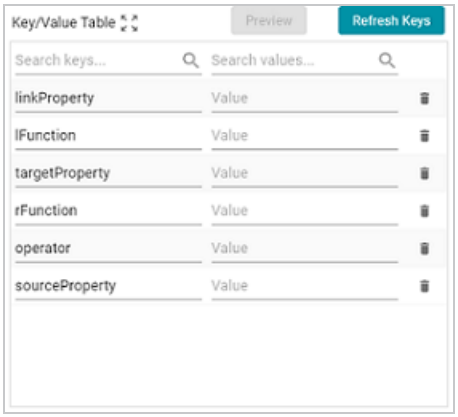
6. On the left side of the screen create the query template. The default template includes the syntax for writing SPARQL INSERT and DELETE queries and includes source and target graph parameters that Anzo replaces at runtime. In the query, include the parameters in the format `${key_name}`. Each parameter will become a key in the Key/Value Table will you click the **Refresh Keys** button. For example, the following INSERT query includes several parameters that represent properties and functions:

```
INSERT {
  GRAPH ${targetGraph}{
    ?lsubj ${linkProperty} ?rsubj
  }
}
${usingSources}
WHERE {
  ?lsubj ${sourceProperty} ?lobj .
  ?rsubj ${targetProperty} ?robj .
  FILTER (${lFunction}(?lobj) ${operator} ${rFunction}(?robj))
}
```

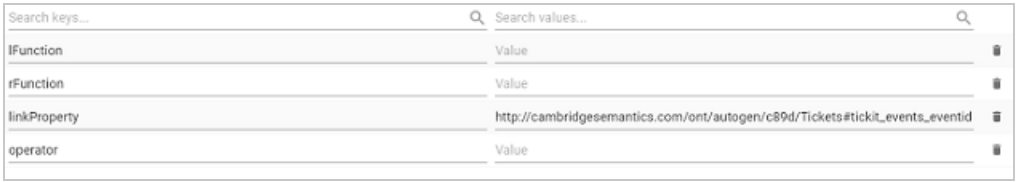
> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each

data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.

7. Once the template query has been defined, populate the Key/Value Table with the keys from the query by clicking the **Refresh Keys** button. For example, using the example query above, the Key/Value table is populated with the following keys:



8. In each row, specify the desired **Value** for the key. For example, in the image below, the property URI `http://cambridgesemantics.com/ont/autogen/c89d/Tickets#tickit_events_eventid` is specified as the Value for the `linkProperty` key.



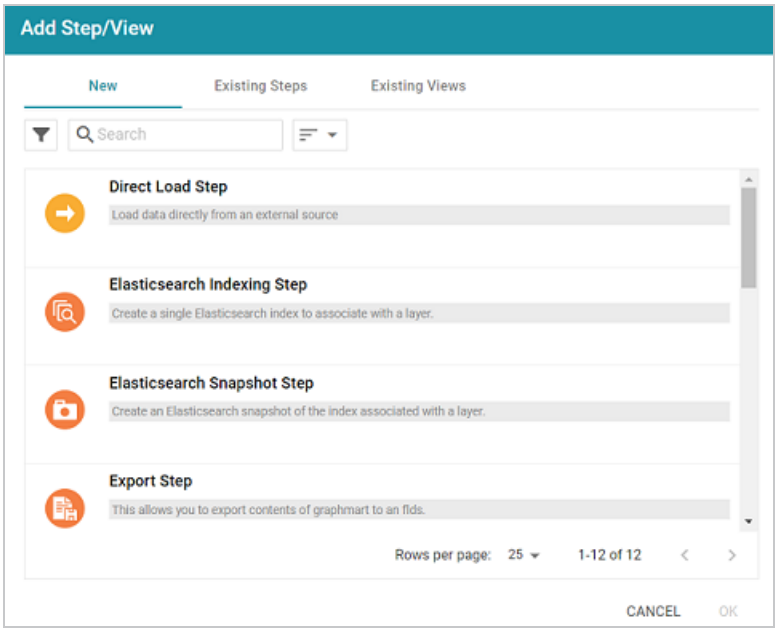
9. Click **Save** to save the step configuration.

Once the Details tab is configured and the template and key-value pairs are defined, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

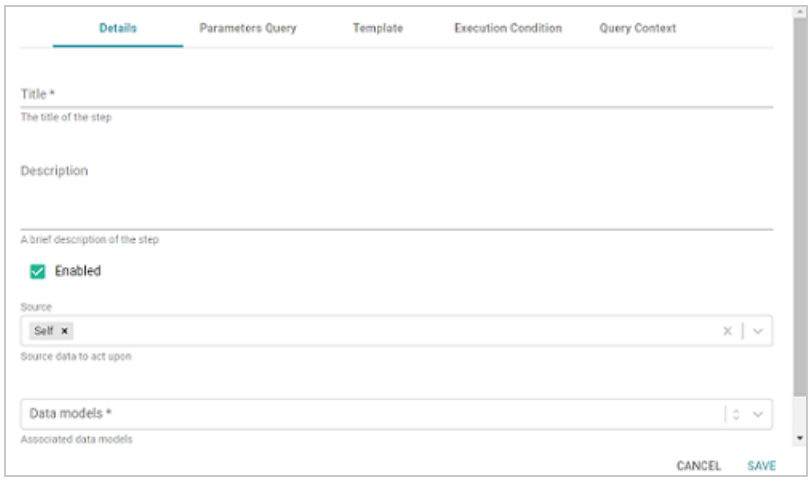
Create a Query-Driven Template

Query-Driven Templated Steps are similar to Templated Steps in that they provide a way to create query templates that use parameters to represent key-value pairs. The queries are reusable across datasets because the existing parameters can be substituted for alternate key-value pairs. The difference between the two types of steps is that the key-value pairs for Templated Steps are user-defined. In Query-Driven Templated Steps, a parameter query is run that automatically generates the key-value pairs. Then the defined template query is run for each key-value solution from the parameter query. Follow the steps below to create a Query-Driven Templated Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new Query-Driven Templated step, select **Query Driven Templated Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:

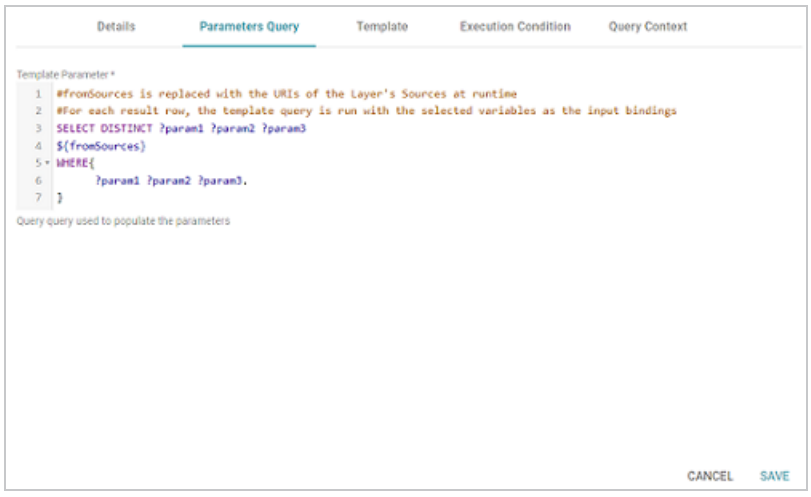


4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
     - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
     - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
     - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
     - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.

- **Data models**: This required field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (). When system data is included, the drop-down list displays the system models in addition to the user-generated models.
- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. When you have finished configuring the Details tab, click the **Parameters Query** tab. This tab defines the query to use for determining the key-value pairs for the Source that was selected on the Details tab. The tab provides a template for writing a SPARQL SELECT DISTINCT query. Edit the template as needed.

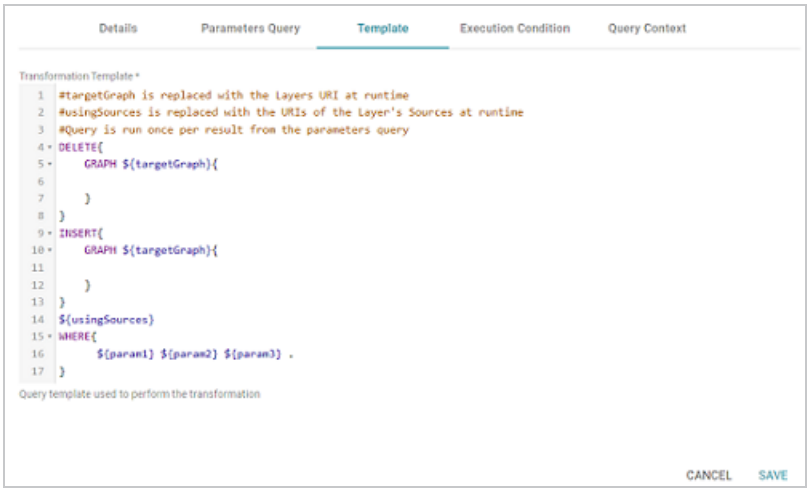
> **Note**
> Make sure that you include the `${fromSources}` parameter in the query. Anzo automatically populates the query with the appropriate source graph URIs according to the Source configured on the Details tab.



> **Note**
>
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.

6. When you have finished configuring the Parameters Query tab, click the **Template** tab. This tab contains the query to run for each of the key-value pairs identified by the Parameters Query. The template includes the syntax for writing SPARQL DELETE and INSERT queries and includes source and target graph parameters as well as the placeholder parameters from the Parameters Query.

   > **Note**
   >
   > By default, Anzo uses RDF encoding for parameters, meaning a parameter specified as `${param}` is translated as `${rdf.param}`. If you do not want to use RDF encoding, you can specify plain text by adding `text.` before the parameter name, for example, `${text.param}`.



7. Click **Save** to save the step configuration.

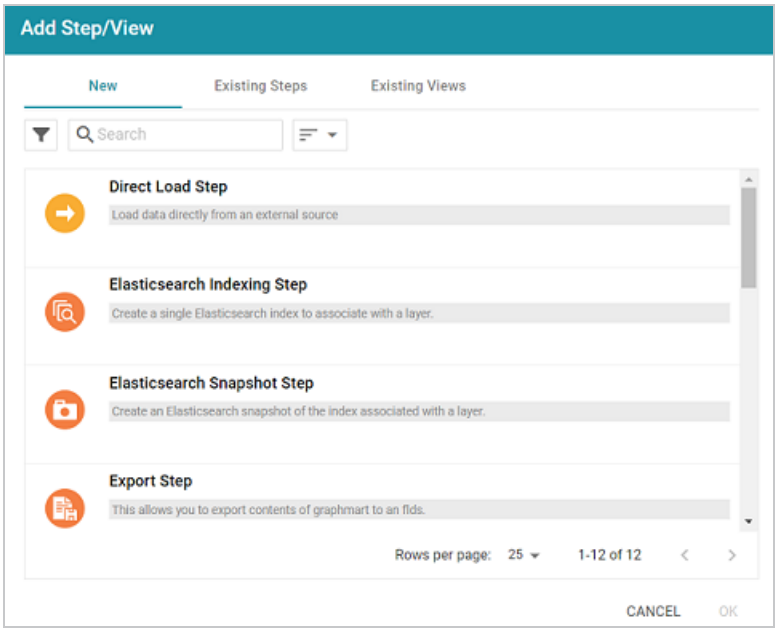
Anzo replaces the parameters at runtime. The query is executed *n* times, where *n* is the number of rows returned by the Parameters Query.

Once the Details tab is configured, the parameters query is in place, and the template is completed, the step can be run. Anzo replaces the parameters at runtime, and the query is executed *n* times, where *n* is the number of rows returned by the parameters query. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

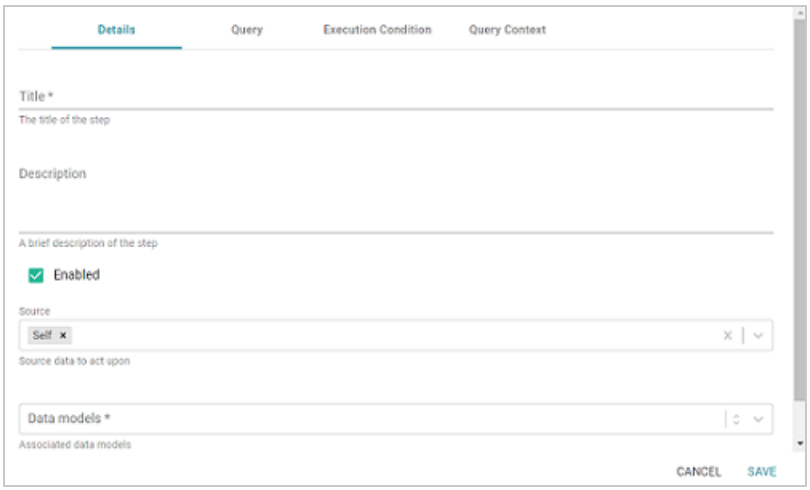
Run a Transformation Query (Query Step)

This topic provides guidance on configuring a Query Step that you can use for creating, cleaning, conforming, or transforming the data in a layer. Follow the steps below to create a Query Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new Query step, select **Query Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:

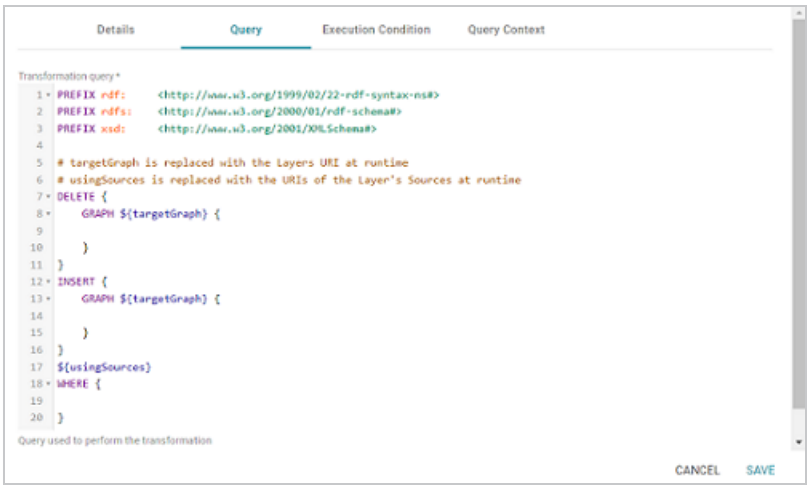


4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
     - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
     - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
     - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
     - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.

- **Data models**: This required field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (). When system data is included, the drop-down list displays the system models in addition to the user-generated models.
- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. When you have finished configuring the Details tab, click the **Query** tab. This tab defines the query that this step should run. The template includes the syntax for writing SPARQL INSERT and DELETE queries and includes the target and source graph parameters (`${targetGraph}` and `${usingSources}`). Anzo replaces the parameters with the appropriate URIs when the step runs. Edit the template as needed. You can click the **Preview in Query Builder** button to open the query in the Query Builder, where you can perform practice runs to see results without having to refresh the graphmart or layer.

> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.



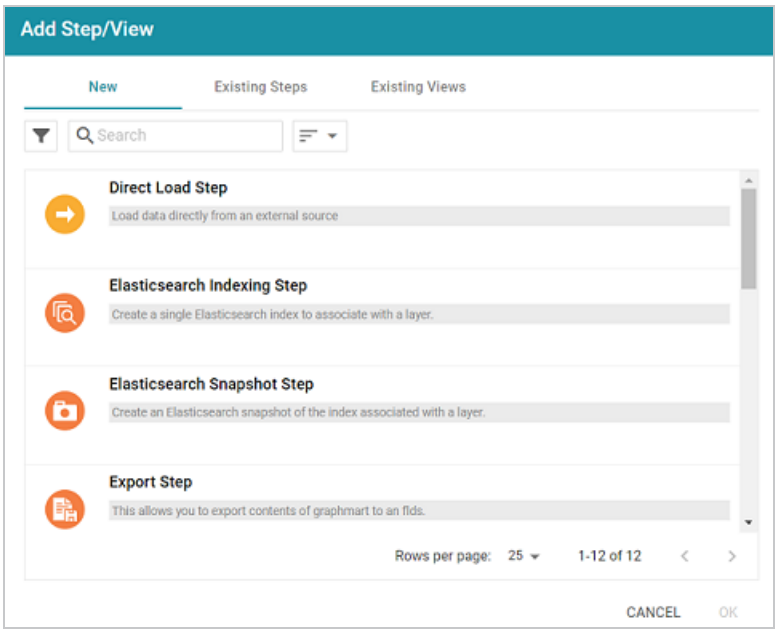
6. Click **Save** to save the step configuration.

Once the Details tab is configured and the query is defined, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

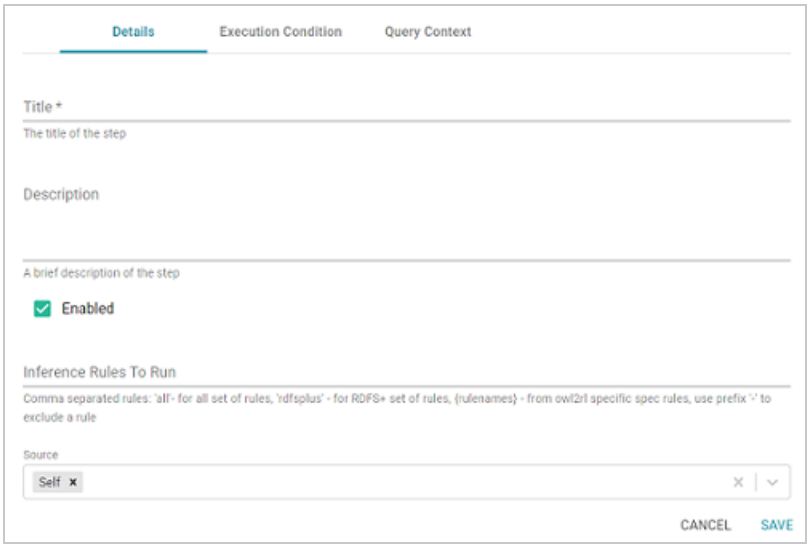
Infer New Data (RDFS+ Inference Step)

This topic provides guidance on configuring an RDFS+ Inference Step that uses RDFS+ and OWL rules to create new relationships based on the vocabularies in the existing data. Follow the steps below to create an RDFS+ Inference Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new RDFS+ Inference step, select **RDFS+ Inference Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:



4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.

- **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
- **Inference Rules to Run**: By default the step runs all of the RDFS-plus inference rules and a subset of the OWL 2 RL rules (see Inference Rule Reference below for specifics). If you want to customize the step to include or exclude certain rules, specify any combination of the following options in the **Inference Rules To Run** field. Specify multiple options in a comma-separated list:
  - **all**: Run all rules.
  - **rdfsplus**: Run only the RDFS-plus rules.
  - ***rule_names***: List specific rules to run only those rules. See Inference Rule Reference.
  - ***-rule_name***: Specify a hyphen (-) in front of a rule name to exclude that rule. For example, **-scm-svf2** excludes the scm-svf2 rule.

  For example, the following value runs all of the inference rules except prp-fp and prp-ifp:

```
all,-prp-fp,-prp-ifp
```

> **Note**
> Certain inference rules are coupled. Specifying either of the rules in the pair automatically runs the coupled rule. The list below describes the paired rules:
>
> - scm-dom1 and scm-rng1
> - scm-dom2 and scm-rng2
> - prp-inv1 and prp-inv2
>
> In addition, running scm-eqc1and cax-sco also runs cax-eqc1 and cax-eqc2. And running scm-eqp1 and prp-spo1 also runs prp-eqp1 and prp-eqp2.

- **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
  - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
  - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
  - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
  - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.
- **Data models**: This required field specifies the model or models to associate with this step. The list displays all of the available models. By default, the field is set to **Exclude System Data** (). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (). When system data is included, the drop-down list displays the system models in addition to the user-generated models.
- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running this step. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

5. Click **Save** to save the step configuration.

Once the Details tab is configured, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions.

Inference Rule Reference

This section provides reference information for the RDFS-plus rules and the subset of OWL 2 RL rules that inference steps run.

- RDFS+ Rules
- OWL 2 RL Rules

RDFS+ Rules

The tables below define the RDFS-plus inference rules.

- Semantics of Class Axioms
- Semantics of Axioms about Properties
- Semantics of Schema Vocabulary

Semantics of Class Axioms

> **Note**
> Because **cax-eqc1** and **cax-eqc2** (described in the table below) are implied rules that are coupled with scm-eqc1 and cax-sco, including **cax-eqc1** or **cax-eqc2** in the Inference Rules to Run field will result in an `invalid inference rule name` error. To run the cax-eqc1 and cax-eqc2 rules, specify **scm-eqc1** and **cax-sco** (**scm-eqc1,cax-sco**).

Rule	Description	IF	THEN
cax-eqc1	Two classes are synonymous.	T(?c1, owl:equivalentClass, ?c2) T(?x, rdf:type, ?c1)	T(?x, rdf:type, ?c2)
cax-eqc2	Two classes are synonymous.	T(?c1, owl:equivalentClass, ?c2) T(?x, rdf:type, ?c2)	T(?x, rdf:type, ?c1)

Rule	Description	IF	THEN
cax-sco	Members of a subclass are also members of the superclass.	T(?c1, rdfs:subClassOf, ?c2) T(?x, rdf:type, ?c1)	T(?x, rdf:type, ?c2)

Semantics of Axioms about Properties

> **Note**
> Because **prp-eqp1** and **prp-eqp2** (described in the table below) are implied rules that are coupled with scm-eqp1 and prp-spo1, including **prp-eqp1** or **prp-eqp2** in the Inference Rules to Run field will result in an `invalid inference rule name` error. To run the prp-eqp1 and prp-eqp2 rules, specify **scm-eqp1** and **prp-spo1** (**scm-eqp1,prp-spo1**).

Rule	Description	IF	THEN
prp-dom	Infer the subject's type from the predicate's domain.	T(?p, rdfs:domain, ?c) T(?x, ?p, ?y)	T(?x, rdf:type, ?c)
prp-eqp1	Two properties are synonymous.	T(?p1, owl:equivalentProperty, ?p2) T(?x, ?p1, ?y)	T(?x, ?p2, ?y)
prp-eqp2	Two properties are synonymous.	T(?p1, owl:equivalentProperty, ?p2) T(?x, ?p2, ?y)	T(?x, ?p1, ?y)
prp-fp	If predicate p is a functional property, then a subject can be related to only one specific object by p.	T(?p, rdf:type, owl:FunctionalProperty) T(?x, ?p, ?y1) T(?x, ?p, ?y2)	T(?y1, owl:sameAs, ?y2)
prp-ifp	If predicate p is an inverse	T(?p, rdf:type,	T(?x1,

Rule	Description	IF	THEN
	functional property, then a specific object can be related to only one subject by p.	owl:InverseFunctionalProperty) T(?x1, ?p, ?y) T(?x2, ?p, ?y)	owl:sameAs, ?x2)
prp-inv1	Two properties are the inverse of each other.	T(?p1, owl:inverseOf, ?p2) T(?x, ?p1, ?y)	T(?y, ?p2, ?x)
prp-inv2	Two properties are the inverse of each other.	T(?p1, owl:inverseOf, ?p2) T(?x, ?p2, ?y)	T(?y, ?p1, ?x)
prp-rng	Infer the object's type from the predicate's range.	T(?p, rdfs:range, ?c) T(?x, ?p, ?y)	T(?y, rdf:type, ?c)
prp-spo1	Relationships that are described by a subproperty also hold for the superproperty.	T(?p1, rdfs:subPropertyOf, ?p2) T(?x, ?p1, ?y)	T(?x, ?p2, ?y)
prp-symp	The inverse is true for a property.	T(?p, rdf:type, owl:SymmetricProperty) T(?x, ?p, ?y)	T(?y, ?p, ?x)
prp-trp	Chains of relationships collapse into a single relationship.	T(?p, rdf:type, owl:TransitiveProperty) T(?x, ?p, ?y) T(?y, ?p, ?z)	T(?x, ?p, ?z)

Semantics of Schema Vocabulary

Rule	Description	IF	THEN
scm-cls	Every class is its own	T(?c, rdf:type, owl:Class)	T(?c, rdfs:subClassOf, ?c)

Rule	Description	IF	THEN
	subclass and equivalent class, and it is a subclass of owl:Thing.		T(?c, owl:equivalentClass, ?c) T(?c, rdfs:subClassOf, owl:Thing) T(owl:Nothing, rdfs:subClassOf, ?c)
scm-dom1	A property with domain c also has domain c's superclasses.	T(?p, rdfs:domain, ?c1) T(?c1, rdfs:subClassOf, ?c2)	T(?p, rdfs:domain, ?c2)
scm-dom2	A subproperty inherits the domains of the superproperties.	T(?p2, rdfs:domain, ?c) T(?p1, rdfs:subPropertyOf, ?p2)	T(?p1, rdfs:domain, ?c)
scm-eqc1	Equivalent classes are subclasses of each other.	T(?c1, owl:equivalentClass, ?c2)	T(?c1, rdfs:subClassOf, ?c2) T(?c2, rdfs:subClassOf, ?c1)
scm-eqc2	If two classes are subclasses, they are also equivalent classes.	T(?c1, rdfs:subClassOf, ?c2) T(?c2, rdfs:subClassOf, ?c1)	T(?c1, owl:equivalentClass, ?c2)
scm-eqp1	Equivalent properties are subproperties of each other.	T(?p1, owl:equivalentProperty, ?p2)	T(?p1, rdfs:subPropertyOf, ?p2) T(?p2, rdfs:subPropertyOf, ?p1)
scm-eqp2	If two properties are subproperties, they are	T(?p1, rdfs:subPropertyOf, ?p2)	T(?p1, owl:equivalentProperty,

Rule	Description	IF	THEN
	also equivalent properties.	T(?p2, rdfs:subPropertyOf, ?p1)	?p2)
scm-rng1	A property with range c also has range c's superclasses.	T(?p, rdfs:range, ?c1) T(?c1, rdfs:subClassOf, ?c2)	T(?p, rdfs:range, ?c2)
scm-rng2	A subproperty inherits the ranges of its superproperties.	T(?p2, rdfs:range, ?c) T(?p1, rdfs:subPropertyOf, ?p2)	T(?p1, rdfs:range, ?c)
scm-sco	owl:subClassOf relationships are transitive	T(?c1, rdfs:subClassOf, ?c2) T(?c2, rdfs:subClassOf, ?c3)	T(?c1, rdfs:subClassOf, ?c3)
scm-spo	owl:subPropertyOf relationships are transitive.	T(?p1, rdfs:subPropertyOf, ?p2) T(?p2, rdfs:subPropertyOf, ?p3)	T(?p1, rdfs:subPropertyOf, ?p3)

Note

The scm-dp and scm-op schema vocabulary rules are not run. Those rules add significant compute overhead but do not result in meaningful inference results.

OWL 2 RL Rules

The tables below define the subset of OWL 2 RL inference rules that inference steps run.

- Semantics of Equality
- Semantics of Schema Vocabulary
- Semantics of Classes

Semantics of Equality

Rule	Description	IF	THEN
eq-rep-o	Describes the replacement property of the owl:sameAs axiom.	T(?o, owl:sameAs, ?o') T(?s, ?p, ?o)	T(?s, ?p, ?o')
eq-rep-p	Describes the replacement property of the owl:sameAs axiom.	T(?p, owl:sameAs, ?p') T(?s, ?p, ?o)	T(?s, ?p', ?o)
eq-rep-s	Describes the replacement property of the owl:sameAs axiom.	T(?s, owl:sameAs, ?s') T(?s, ?p, ?o)	T(?s', ?p, ?o)
eq-sym	Describes the symmetric property of the owl:sameAs axiom.	T(?x, owl:sameAs, ?y)	T(?y, owl:sameAs, ?x)
eq-trans	Describes the transitive property of the owl:sameAs axiom.	T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)	T(?x, owl:sameAs, ?z)

Semantics of Schema Vocabulary

Rule	Description	IF	THEN
scm-svf1	A property restriction c1 is a subclass of c2 if they are both someValuesFrom restrictions on the same property and c1's target class is a subclass of c2's target class.	T(?c1, owl:someValuesFrom, ?y1) T(?c1, owl:onProperty, ?p) T(?c2,	T(?c1, rdfs:subClassOf, ?c2)

Rule	Description	IF	THEN
		owl:someValuesFrom, ?y2) T(?c2, owl:onProperty, ?p) T(?y1, rdfs:subClassOf, ?y2)	
scm-svf2	A property restriction c1 is a subclass of c2 if they are both someValuesFrom restrictions on the same class where c1's target property is a subproperty of c2's target property.	T(?c1, owl:someValuesFrom, ?y) T(?c1, owl:onProperty, ?p1) T(?c2, owl:someValuesFrom, ?y) T(?c2, owl:onProperty, ?p2) T(?p1, rdfs:subPropertyOf, ?p2)	T(?c1, rdfs:subClassOf, ?c2)
scm-int		T(?c, owl:intersectionOf, ?x) LIST[?x, ?c1, ..., ?cn]	T(?c, rdfs:subClassOf, ?c1) T(?c, rdfs:subClassOf, ?c2) ... T(?c, rdfs:subClassOf, ?cn)

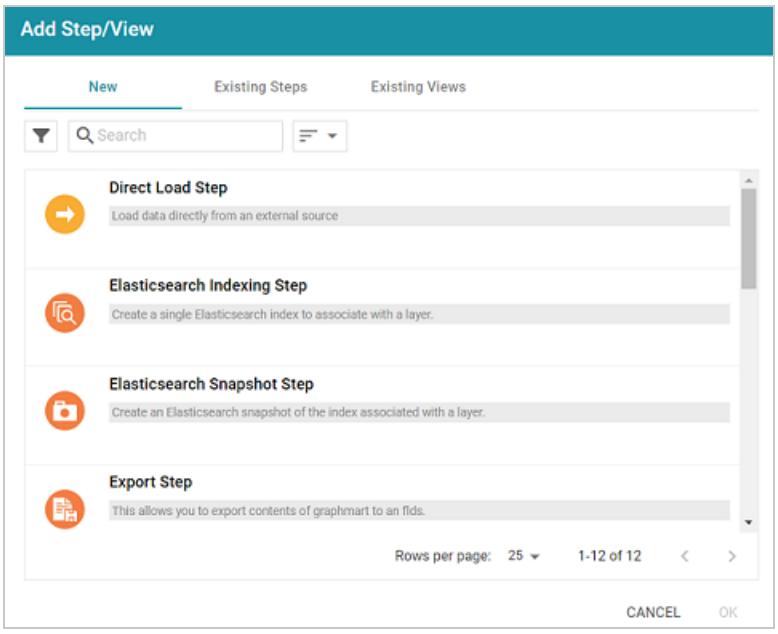
Semantics of Classes

Rule	Description	IF	THEN
cls-svf1	At least one object of a property is a member of the specified class.	T(?x, owl:someValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v) T(?v, rdf:type, ?y)	T(?u, rdf:type, ?x)
cls-int1	An instance belongs to every one of the specified classes.	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c1, ..., ?cn] T(?y, rdf:type, ?c1) T(?y, rdf:type, ?c2) ... T(?y, rdf:type, ?cn)	T(?y, rdf:type, ?c)

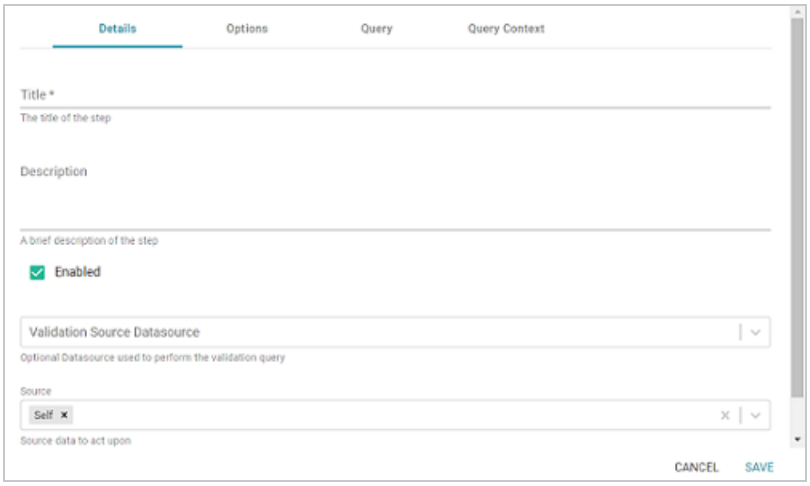
Validate the Data (Validation Step)

This topic provides guidance on configuring a Validation Step to use for validating the data in a layer and optionally setting up execution conditions. Follow the steps below to create a Validation Step.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



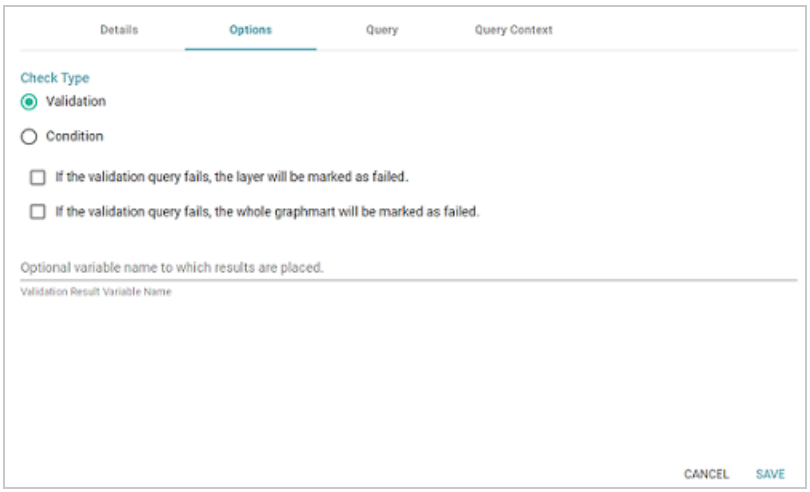
3. To create a new Validation step, select **Validation Step** and then click **OK**. If you want to clone an existing step, click the **Existing Steps** tab, select the step that you want to clone, and then click **OK**. Anzo creates or clones the step and displays the Details tab:



4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.

- **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
- **Validation Source Datasource**: This optional field enables you to select a data source (such as a system data source) to perform the validation against if you do not want the query to run against the graphmart that the step is in. When Validation Source Datasource is unset, the validation is performed against the graphmart.
- **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:
  - **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
  - **All Previous Layers Within Graphmart**: This option means that the step runs against the data in all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
  - **Previous Layer Within Graphmart**: This option means that the query runs against the data that is in the one layer that precedes the layer this step is in.
  - ***Layer Name***: The Source drop-down list also includes options for layer names. You can choose a specific layer to act upon the data in that layer.
- **Pre-Run Generate Statistics**: This option controls whether to initiate AnzoGraph's internal statistics gathering queries before running the query to pre-compile. The statistics gathering helps ensure that the AnzoGraph query planner generates ideal query execution plans for queries that are run against the graphmart.

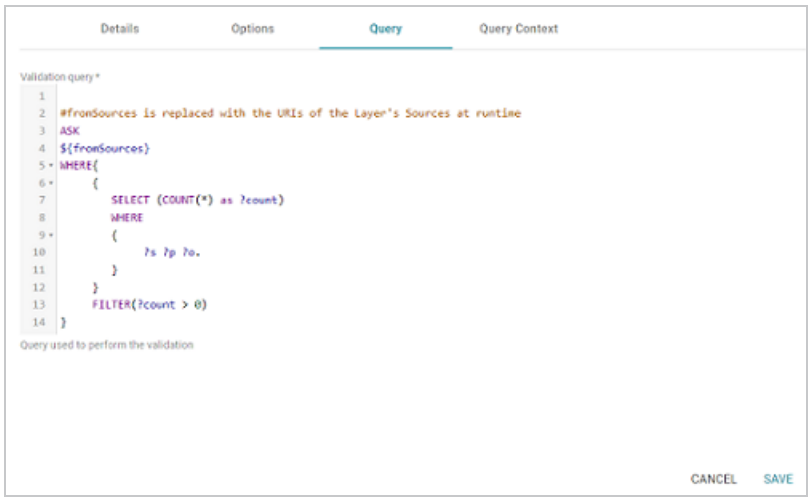
5. When you have finished configuring the Details tab, click the **Options** tab. This tab includes the settings that specify the type of check to perform on the data as well as instructions for what to do if the validation fails.



6. On the Options tab, determine which type of check to perform and select the appropriate radio button. There are two check types:
   - **Validation**: A Validation check validates the data according to the defined query (on the Query tab) and can be configured to take action depending on whether the validation passes or fails.
   - **Condition**: A Condition check takes the results of the query and associates it with the specified variable. That variable can then be used for setting up an execution condition at the layer or step level.
7. If you selected the Validation check type, you have the option to configure what to do if the validation query fails:
   - **If the validation query fails, the layer will be marked as failed**: Select this option if you want Anzo to abort the load of the layer if this step fails.
   - **If the validation query fails, the whole graphmart will be marked as failed**: Select this option if you want Anzo to abort the load of the entire graphmart if this step fails.
8. If you selected the Condition check type, you are required to specify the variable name that you want to use to store the result from the query. This variable becomes available as a choice when configuring an execution condition.
9. When you have finished configuring the Options tab, click the **Query** tab and compose the validation query that the step should run. The tab includes the syntax for writing a SPARQL

ASK query, which is useful for determining whether a certain pattern exists in the data. ASK queries return "true" or "false" to indicate whether a solution exists. The template includes a source graph parameter (`${fromSources}`). Using the configured Source options from the Details tab, Anzo automatically populates the query with the appropriate source graph URIs when the query runs.

> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.



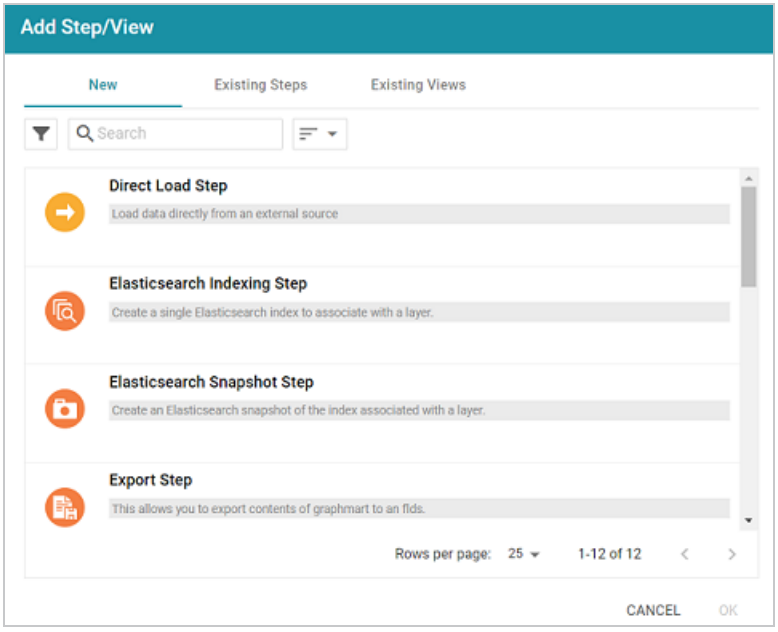
10. Click **Save** to save the step configuration.

Once the Details tab is configured and the validation options and query are defined, the step can be run. For information about setting up an execution condition that uses this step, see Defining Execution Conditions.

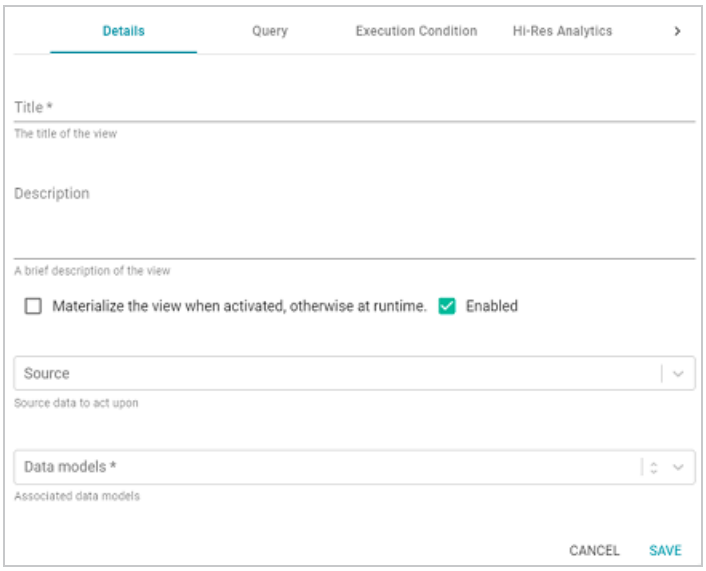
Construct a View of the Data (View Step)

This topic provides guidance on configuring a View Step to create a custom view of the data that does not change the graphmart or necessarily materialize any new data. View steps contain SPARQL CONSTRUCT queries to create a view definition in AnzoGraph. Follow the steps below to create a View.

1. Go to the graphmart for which you want to add a step and then click the **Data Layers** tab.
2. On the Data Layers tab, find the layer that you want to add the step to. Click the menu icon (⋮) for that layer and select **Add Step/View**. The Add Step/View dialog box is displayed with the New tab selected.



3. To create a new View step, select **View** and then click **OK**. If you want to clone an existing view, click the **Existing Views** tab, select the view that you want to clone, and then click **OK**. Anzo creates or clones the view and displays the Details tab:

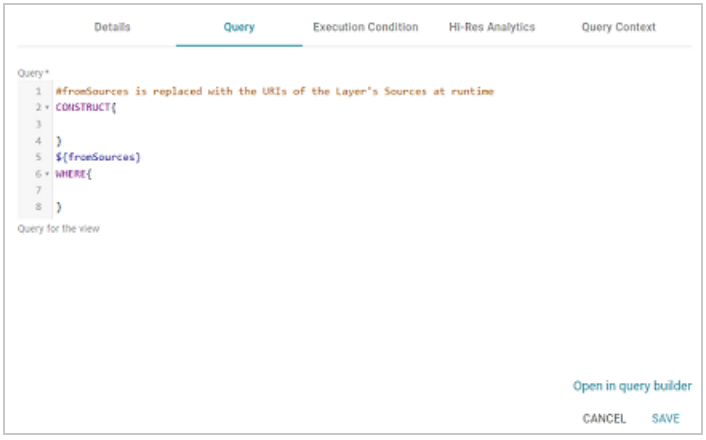


4. On the Details tab, configure the following options as needed:
   - **Title**: The required name of the step.
   - **Description**: An optional short description of the step.
   - **Materialize the view when activated, otherwise at runtime**: This option controls whether a copy of the data the view creates is saved in the graphmart (materialized) or whether this is a virtual view where the data is recreated each time the view runs. If you are creating a view against an extremely large data source or a source that changes often, typically the view should not be materialized. If you want to store a copy of the data that the view creates, select the **Materialize the view when activated...** check box. When this option is disabled Anzo creates a virtual view where only the view definition is stored in memory and not a copy of the data.
   - **Enabled**: When creating a new step, the Enabled option is selected by default, indicating that the step is enabled and will run when the layer is loaded or refreshed. If you want to disable the step so that it is not processed, clear the **Enabled** checkbox.
   - **Source**: The source data that this step should act upon. Steps can build upon the data generated by steps in other layers or can be self-contained, applying changes that relate only to the data defined in the layer that contains this step. You can select any number of the following options:

- **Self**: This option is selected by default and means that the step runs against the data that is in the parent layer.
- **All Previous Views Within Layer**: This options means that the step runs against the data that is generated by all of the previous views in the same layer.
- **Previous View Within Layer**: This options means that the step runs against the data that is generated by the previous view in the same layer.
- **All Previous Layers Within Graphmart**: This option means that the step runs against the data that is generated by all of the successful layers that precede the layer this step is in. Any failed layers are ignored.
- **Previous Layer Within Graphmart**: This option means that the query runs against only the data that is generated by the one layer that precedes the layer this step is in.

- **Data models**: This required field specifies the model or models that you want to create this view against. The list displays all of the available models. By default, the field is set to **Exclude System Data** (⇳). If you want to choose a system model, click the toggle button on the right side of the field to change it to **Include System Data** (⇵). When system data is included, the drop-down list displays the system models in addition to the user-generated models.

5. When you have finished configuring the Details tab, click the **Query** tab. This tab contains the query to use to create the view.



6. Edit the provided template to compose the CONSTRUCT query the step should run. For information about CONSTRUCT queries, see CONSTRUCT in the W3C SPARQL 1.1 Query Language specification.

> **Note**
> Do not include a GRAPH keyword in the CONSTRUCT clause. Anzo uses the view's URI as the graph URI for the constructed triples. In addition, Anzo uses the configured Source options from the Details tab to automatically replace the `${fromSources}` parameter with the appropriate FROM clauses when the query runs.

You can click the **Open in Query Builder** button to open the query in the Query Builder, where you can perform practice runs to see results without having to refresh the graphmart or layer.

> **Note**
> If your query connects to a source that requires input of connection and authorization information, Cambridge Semantics recommends that you do not include the connection and authorization values directly in the query. Instead, replace those values with Context Variables from a Query Context. You can access Context Providers for each data source from the step's Query Context tab. For detailed information about query contexts and referencing variables in a query, see Using Query Contexts.

> **Tip**
> If your view query employs the Graph Data Interface, be sure to use the following DataToolkitView service call in the query:
>
> ```
> SERVICE <http://cambridgesemantics.com/services/DataToolkitView>
> (${targetGraph})
> ```
>
> For more information, see GDI Query Syntax.

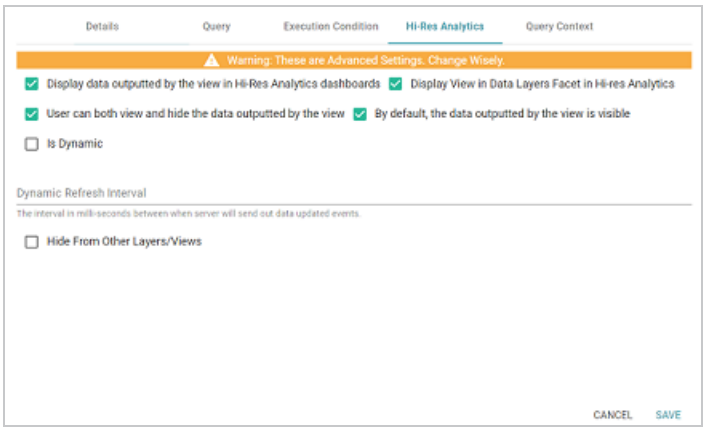
7. Click **Save** to save the step configuration.

Once the Details tab is configured and the query is written, the step can be run. For information about running this step conditionally by setting up an execution condition, see Defining Execution Conditions. For details about the advanced Hi-Res Analytics settings that control how the view affects dashboards, see Hi-Res Analytics Tab below.

Hi-Res Analytics Tab

The **Hi-Res Analytics** tab contains advanced settings that control how the layer is exposed to and affects Hi-Res Analytic dashboards.

> **Note**
> Changing these settings can have unexpected consequences, and Cambridge Semantics recommends that you do not modify them unless you understand the repercussions.



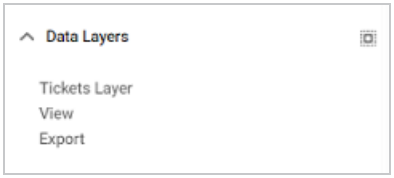
Display data outputted by the view in Hi-Res Analytics dashboards

This setting controls whether the data accessed by the view is available to query and display in dashboards:

- When the setting is **enabled** (the default value), the view data is available to dashboards.
- When the setting is **disabled**, other data layers in the graphmart can use the view's data, but the data is not available to use Hi-Res Analytics dashboards.

Display View in Data Layers Facet in Hi-res Analytics

This setting controls whether the view name is displayed in the Data Layers panel on dashboards. The image below shows an example Data Layers panel:



- When the setting is **enabled** (the default value), the view is listed in the Data Layers panel.
- When the setting is **disabled**, the view's data is always used in dashboards for this graphmart but users do not see the view listed in the Data Layers panel.

User can both view and hide the data outputted by the view

This setting controls whether users have the option to show and hide the view in the Data Layers panel on dashboards:

- When the setting is **enabled** (the default value), the view is listed in the Data Layers panel and users have the option to show and hide the layer.
- When the setting is **disabled**, whether the view shows up in the Data Layers panel depends on the **By default, the data outputted by the view is visible** setting. If the view is visible in the Data Layers panel ("By default, the data outputted by the view is visible" is enabled), users cannot toggle it on and off.

By default, the data outputted by the view is visible

This setting controls whether the data generated by the view is visible in dashboards:

- When the setting is **enabled** (the default value), the view is listed in the Data Layers panel in dashboards and is selected by default.
- When the setting is **disabled**, the view shows up in the Data Layers panel but is not selected. To include the view's data in a dashboard, the user must select the view.

Is Dynamic

Typically this option is used only for Graph Data Interface (GDI) connections where a remote data source is accessed and that source data changes dynamically. If the source is dynamic and you want Anzo to automatically refresh the view of the data at certain internals, select the **Is Dynamic** checkbox. Then set the Dynamic Refresh Interval (described below).

Dynamic Refresh Interval

If the **Is Dynamic** option is enabled, this setting configures the interval at which Anzo queries the data source to retrieve any updated view data. Specify the number of milliseconds to wait between refreshes of the data.

Hide from Other Layers/Views

This setting controls whether the other layers in the graphmart can act upon the data in this view.

- When the setting is **disabled** (the default value), this view is available as a choice in the **Source** drop-down list when a step is configured.
- When the setting is **enabled**, this view is not listed as a choice in the **Source** list.

Creating Data on Demand Endpoints

With the Anzo Data on Demand service you can generate Open Data Protocol (OData)-based feeds that can be used to access Graphmarts programmatically via a RESTful API or from third-party business intelligence applications such as TIBCO Spotfire, Tableau, and Microsoft Power BI. The OData protocol enables web clients to use simple HTTP messages to access resources that are identified using URLs. OData shares some similarities with JDBC and ODBC. Like ODBC, OData is not limited to relational databases. The Anzo Data on Demand service follows the OData Version 4.0 specification, which defines the standard URL conventions, query options, and metadata schema.

Anzo supports two types of Data on Demand endpoints. The first type is called an **Auto-Generated** endpoint. Auto-Generated endpoints are the quickest type to create. They simply make available as-is all of the data in the selected Data Layers. Any joins, filters, and other operations must be performed by the consumer of the data outside of Anzo and AnzoGraph.

The second type of endpoint is called a **Custom** endpoint (sometimes called a **Table** endpoint). Since queries that join data often perform very poorly when run in BI applications with a JDBC driver, Custom endpoints let you assemble custom queries that join classes and apply filters and formulas. The endpoint becomes a view in AnzoGraph and AnzoGraph executes the custom queries in memory. Results can then be viewed from the endpoint without having to run the complex analytic queries over JDBC.

The topics in this section provide instructions for creating both types of endpoints.

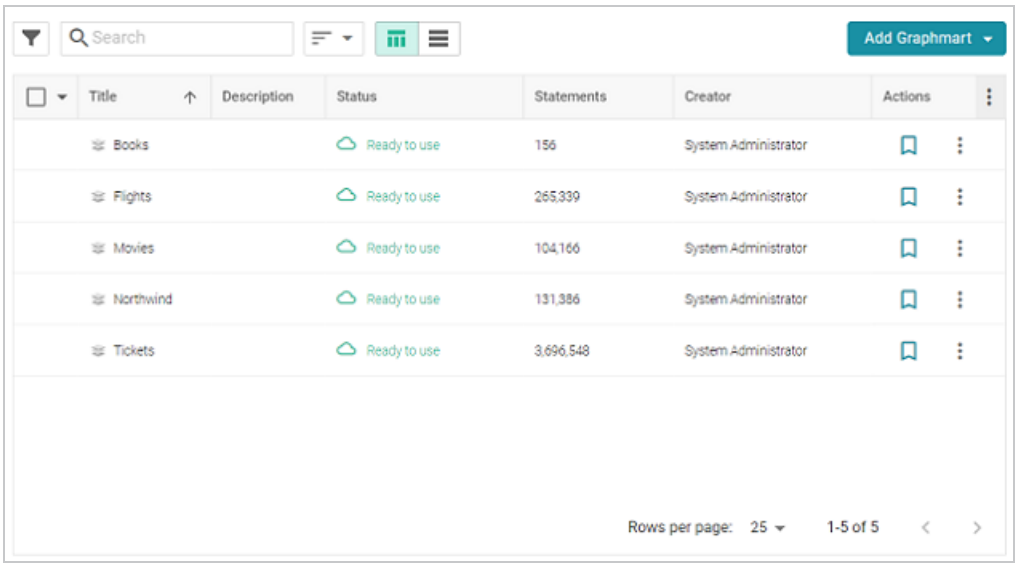
- Creating an Auto-Generated Endpoint
- Creating a Custom Endpoint

Creating an Auto-Generated Endpoint

Follow the instructions below to create an Auto-Generated Data on Demand endpoint. Auto-Generated endpoints can quickly be created to make available all of the data in the selected Data

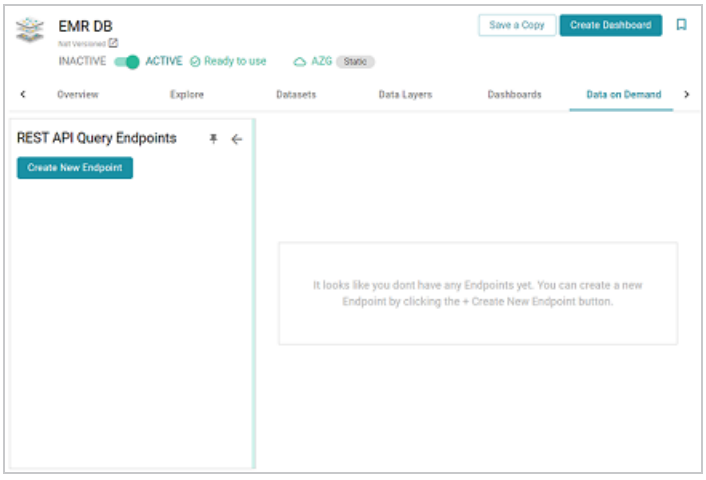
Layers. The data cannot be customized to exclude certain classes, join data across classes, or apply functions and formulas to properties. For instructions on creating an endpoint that can be customized, see Creating a Custom Endpoint.

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:

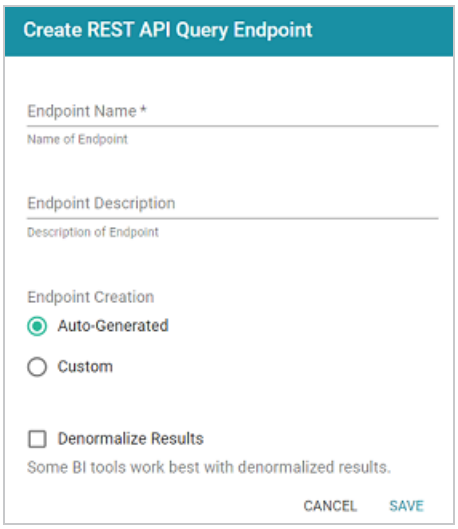


2. On the Graphmarts screen, click the name of the graphmart for which you want to create an endpoint.

3. Click the **Data on Demand** tab. Anzo displays the Data on Demand screen, which lists any existing endpoints. For example, the image below shows a graphmart that does not have any endpoints configured:



4. Click the **Create New Endpoint** button on the left side of the screen. Anzo displays the Create REST API Query Endpoint screen.



5. Configure the endpoint options on the screen as needed. The list below describes each setting:
   - **Endpoint Name**: Specify a name for the endpoint in this field. The endpoint name must be unique.
   - **Endpoint Description**: You can add an optional description for the endpoint in this field.
   - **Endpoint Creation**: This field specifies the type of endpoint to create. By default, the type is set to **Auto-Generated**. Leave the Auto-Generated radio button selected.
   - **Denormalize Results**: By default (when Denormalize Results is not selected), OData returns multi-valued properties as arrays. Certain BI tools, however, do not support arrays or multi-valued properties. If your data includes multi-valued properties and you plan to view the endpoint using a BI tool that does not support them, you can select the **Denormalize Results** setting to denormalize all multi-valued properties that are exposed in the endpoint. For JSON, XML, and CSV output formats, denormalization expands the properties into new rows so that they can be viewed in BI tools.

Tip

The following image shows an example of CSV output of multi-valued properties when **Denormalize Results** is disabled:

```
TeamId,TeamName,TeamToPlayer_PlayerName,league_tbal44_key
1,Al Thomas,"[Fred Wynn,Steve Jones,James Smith]",aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
2,Black Sox,"[Tim Hooper,Jared Bonds,Matt Butler]",aHR0cDovL2NzaS5jb20vVGVhbXMvMg
3,Braves,"[Billy Roper,Alex Granderson,Ted Sale]",aHR0cDovL2NzaS5jb20vVGVhbXMvMw
4,Somerville,"[Chris Underwood,Mike Magazine,Ted James]",aHR0cDovL2NzaS5jb20vVGVhbXMvNA
```

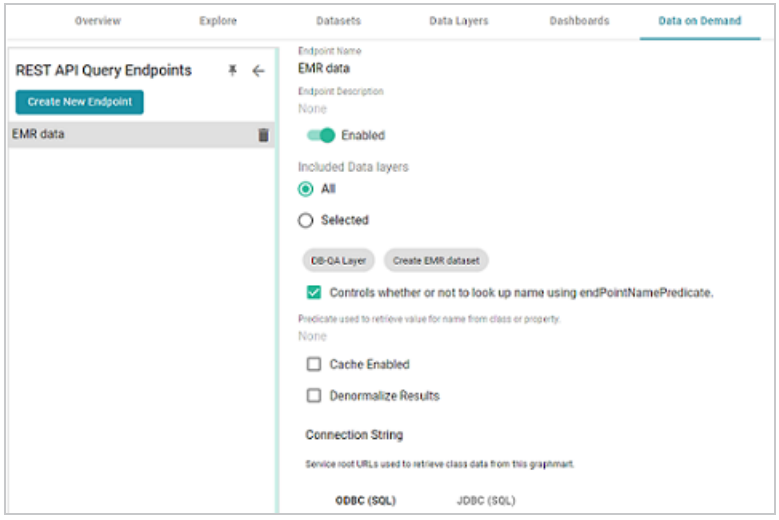
And the example below shows the output of multi-valued properties when **Denormalize Results** is enabled:

```
TeamId,TeamName,PlayerName,table51_key
1,Al Thomas,James Smith,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
1,Al Thomas,Fred Wynn,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
1,Al Thomas,Steve Jones,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
2,Black Sox,Matt Butler,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
2,Black Sox,Jared Bonds,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
2,Black Sox,Tim Hooper,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
3,Braves,Billy Roper,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
3,Braves,Alex Granderson,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
3,Braves,Ted Sale,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
4,Somerville,Mike Magazine,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
4,Somerville,Chris Underwood,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
4,Somerville,Ted James,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
```

Note

Keep in mind that denormalizing the results means that more rows are created and transferred. Depending on the number of multi-valued properties and how the data is set up, you may see slower performance when creating and querying denormalized endpoints.

6. Click **Save** to create the endpoint and view the configuration details. For example:



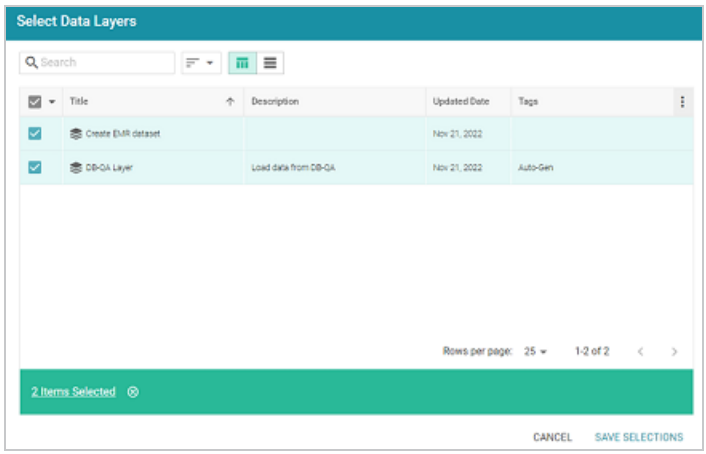
7. You can make changes to any of the following additional configuration options that become available after the endpoint is created:

   - **Enabled**: By default the endpoint is set to **Enabled**, indicating that the endpoint is active. If you want to disable the endpoint, slide the Enabled slider to the left.

     > **Note**
     > If a request is sent to a disabled endpoint, Anzo displays a `503: Service Unavailable` error with a message indicating that the endpoint is disabled. For example, "Unable to process request. The endpoint '<name>' is DISABLED."

   - **Included Data Layers**: By default the Included Data Layers option is set to **All**, indicating that all of the layers in the graphmart are available from the endpoint. The included layers are listed below the radio buttons.

     If you do not want to include all layers, click the **Selected** radio button. An **Edit Selections** link becomes available under the list of layers. Click **Edit Selections** to open the Select Data Layers dialog box. For example:

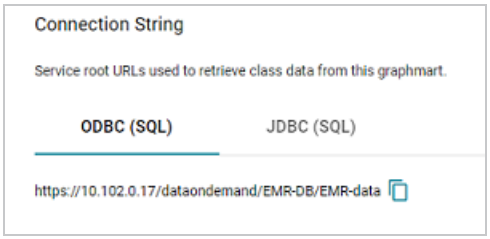


Clear the checkbox for any layer that you want to exclude from the endpoint, and then click **Save Selections** to save the change and return to the configuration screen.

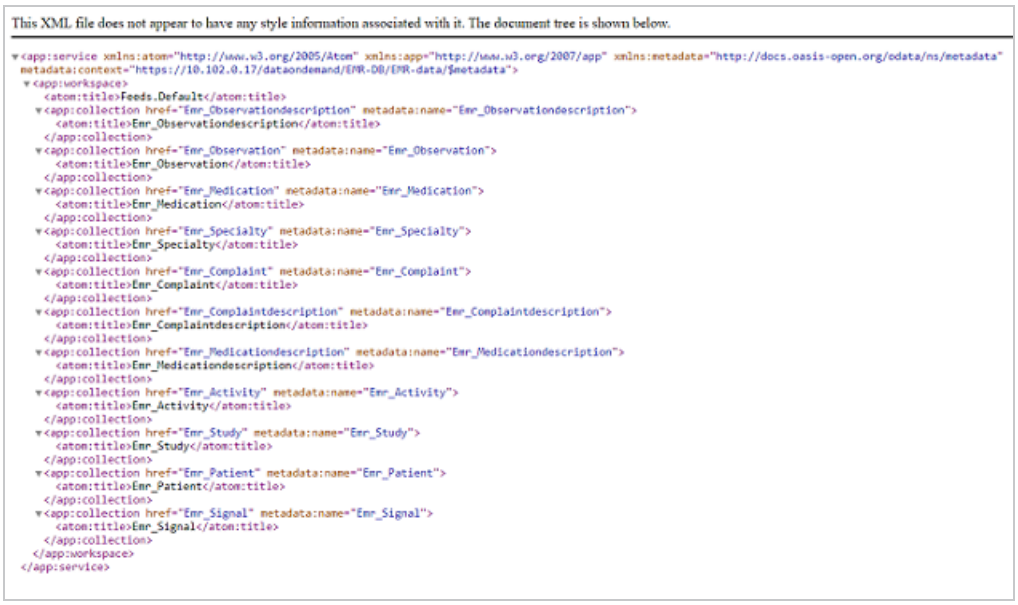
- **Controls whether or not to look up name using endPointNamePredicate**: This setting controls which predicate value from the related model is used for the class and property display names in the endpoint. By default, the setting is enabled and the **Predicate used to retrieve value for name from class or property** is blank. That means Anzo uses the **rdfs:label** (http://www.w3.org/2000/01/rdf-schema#label) value for each class and property name.
  - If you want the endpoint to use a different value for class and property names, you can edit **Predicate used to retrieve value for name from class or property** to specify the URI for another predicate from the model. For example, specifying `http://purl.org/dc/elements/1.1/description` would use each entity's **Description** value.
  - If you disable the **Controls whether or not to look up name using endPointNamePredicate** setting, each entity's local name is used.
- **Cache Enabled**: When the endpoint is accessed, Anzo translates the OData query to a SPARQL query and sends it to AnzoGraph for execution. The Cache Enabled setting controls whether the results of that AnzoGraph query are cached in Anzo so that subsequent endpoint requests can run against the cache in Anzo. When **Cache Enabled** is disabled (the default setting), Anzo does not store the cache, and endpoint requests are sent to AnzoGraph. When **Cache Enabled** is selected, Anzo stores the

cached results and AnzoGraph only gets queried if the cached results are invalidated and need to be refreshed.

Once you are satisfied with the configuration, this Data on Demand endpoint is ready for access via OData/ODBC or JDBC. At the bottom of the screen, retrieve the ODBC or JDBC service URL to use to access the endpoint. For example:



To test whether the endpoint is active, you can copy the ODBC service URL and paste it into a web browser. If the endpoint is active, the browser shows an XML feed of the schema. For example:



Note

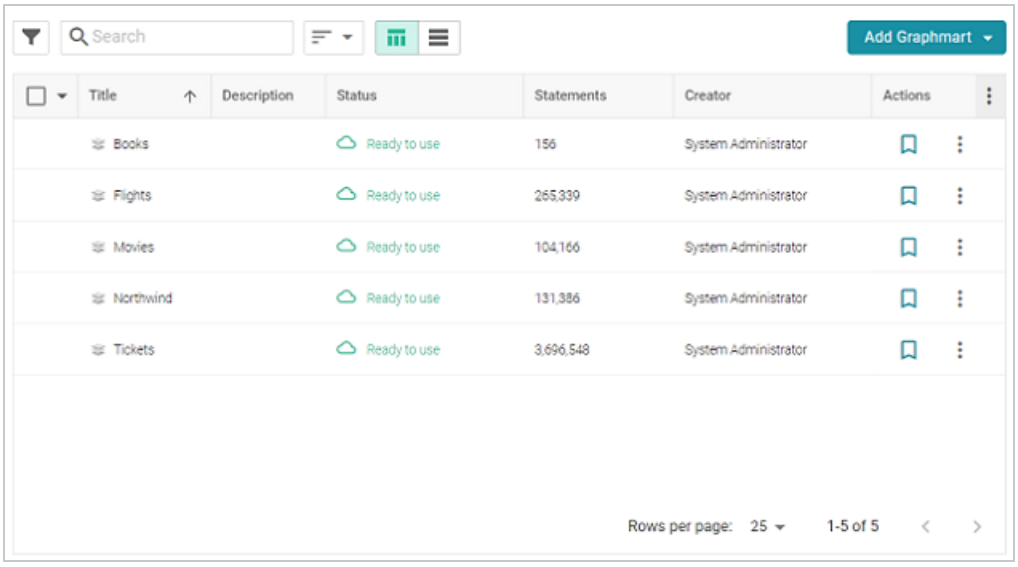
The endpoint is accessible only when it is **Enabled** and the associated graphmart is **Active**.

For information about accessing endpoints programmatically, see Accessing an Endpoint Programmatically. For information about accessing endpoints with third-party analytics tools, see Accessing an Endpoint from an Application. For information about the supported OData operators, output format, and query examples, see OData Reference.

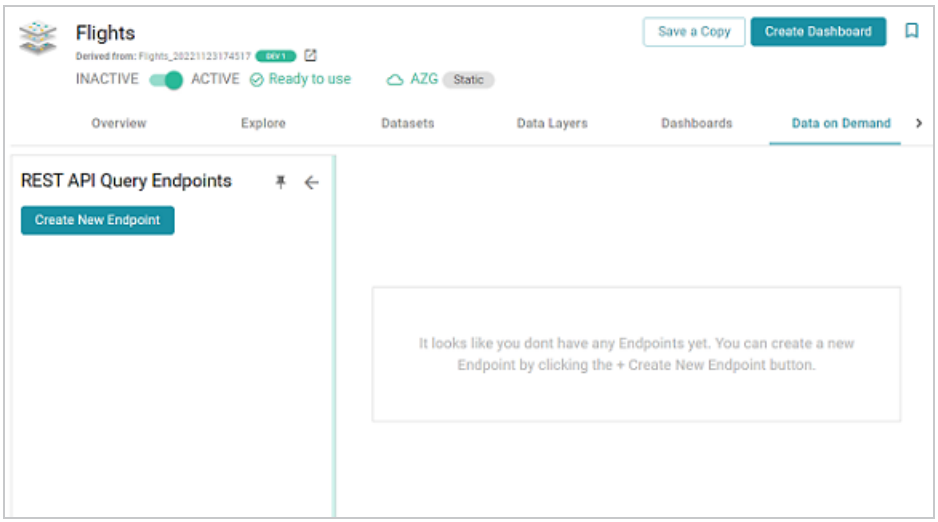
Creating a Custom Endpoint

Follow the instructions below to create a Custom Data on Demand endpoint (sometimes called a Table endpoint). Creating a custom endpoint is similar to creating a dashboard Table lens in that you build a table with the columns that you want to see. You can traverse the relationships and join classes, add filters, and apply functions to properties. The tables are translated to SPARQL queries that create views in AnzoGraph, allowing you to interact with the graph for complex analytics but generate results in the tabular format that BI tools expect.

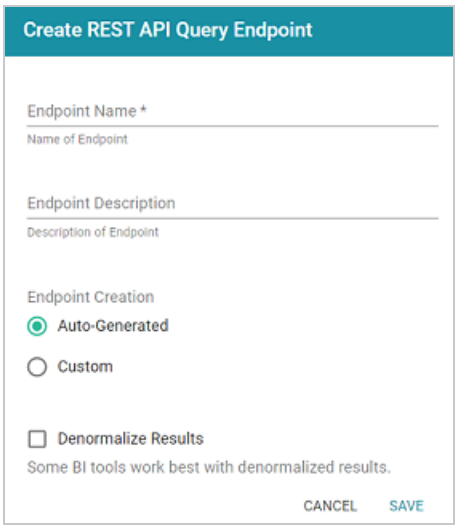
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



2. On the Graphmarts screen, click the name of the graphmart for which you want to create an endpoint.
3. Click the **Data on Demand** tab. Anzo displays the Data on Demand screen, which lists any existing endpoints. For example, the image below shows a graphmart without any endpoints:



4. Click the **Create New Endpoint** button on the left side of the screen. Anzo displays the Create REST API Query Endpoint screen:



5. Configure the endpoint options on the screen as needed. The list below describes each setting:
   - **Endpoint Name**: Specify a name for the endpoint in this field. The endpoint name must be unique.
   - **Endpoint Description**: You can add an optional description for the endpoint in this field.
   - **Endpoint Creation**: This field specifies the type of endpoint to create. By default, the type is set to **Auto-Generated**. Select the **Custom** radio button.

- **Denormalize Results**: By default (when Denormalize Results is not selected), OData returns multi-valued properties as arrays. Certain BI tools, however, do not support arrays or multi-valued properties. If your data includes multi-valued properties and you plan to view the endpoint using a BI tool that does not support them, you can select the **Denormalize Results** setting to denormalize all multi-valued properties that are exposed in the endpoint. For JSON, XML, and CSV output formats, denormalization expands the properties into new rows so that they can be viewed in BI tools.

Tip

The following image shows an example of CSV output of multi-valued properties when **Denormalize Results** is disabled:

```
TeamId,TeamName,TeamToPlayer_PlayerName,league_tbal44_key
1,Al Thomas,"[Fred Wynn,Steve Jones,James Smith]",aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
2,Black Sox,"[Tim Hooper,Jared Bonds,Matt Butler]",aHR0cDovL2NzaS5jb20vVGVhbXMvMg
3,Braves,"[Billy Roper,Alex Granderson,Ted Sale]",aHR0cDovL2NzaS5jb20vVGVhbXMvMw
4,Somerville,"[Chris Underwood,Mike Magazine,Ted James]",aHR0cDovL2NzaS5jb20vVGVhbXMvNA
```

And the example below shows the output of multi-valued properties when **Denormalize Results** is enabled:

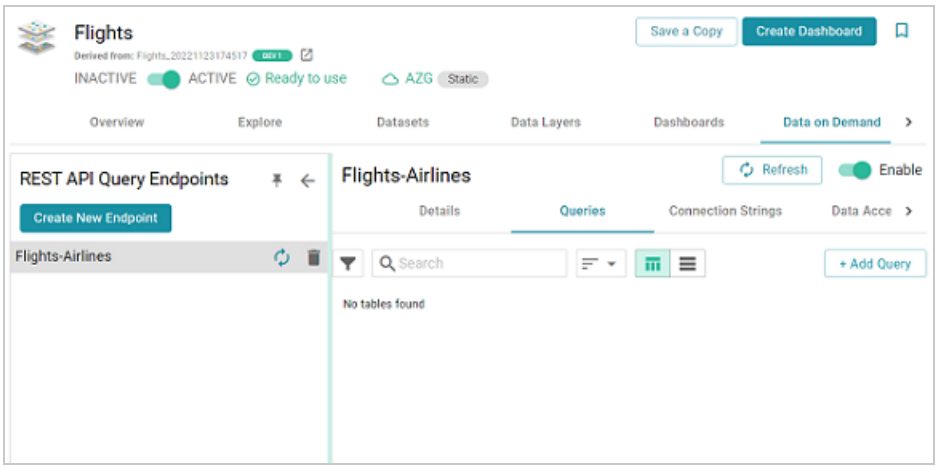
```
TeamId,TeamName,PlayerName,table51_key
1,Al Thomas,James Smith,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
1,Al Thomas,Fred Wynn,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
1,Al Thomas,Steve Jones,aHR0cDovL2NzaS5jb20vVGVhbXMvMQ
2,Black Sox,Matt Butler,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
2,Black Sox,Jared Bonds,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
2,Black Sox,Tim Hooper,aHR0cDovL2NzaS5jb20vVGVhbXMvMg
3,Braves,Billy Roper,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
3,Braves,Alex Granderson,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
3,Braves,Ted Sale,aHR0cDovL2NzaS5jb20vVGVhbXMvMw
4,Somerville,Mike Magazine,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
4,Somerville,Chris Underwood,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
4,Somerville,Ted James,aHR0cDovL2NzaS5jb20vVGVhbXMvNA
```

Note

For Custom endpoints, you also have the option to denormalize data on a per-column basis. If you do not want to denormalize all multi-valued properties, you can leave **Denormalize Results** disabled and then enable denormalization for specific columns when you build the endpoint views.

Also note that denormalizing all results means that more rows are created and transferred. Depending on the number of multi-valued properties and how the data is set up, you may see slower performance when creating and querying denormalized endpoints.

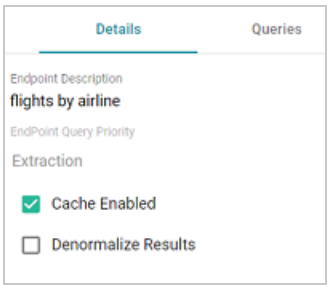
6. When you have finished configuring the endpoint options, click **Save**. The endpoint is created and is empty until you create queries. For example, the image below shows a new endpoint:



By default the endpoint is set to **Enable**, indicating that the endpoint is active. If you want to disable the endpoint, slide the **Enable** slider to the left.

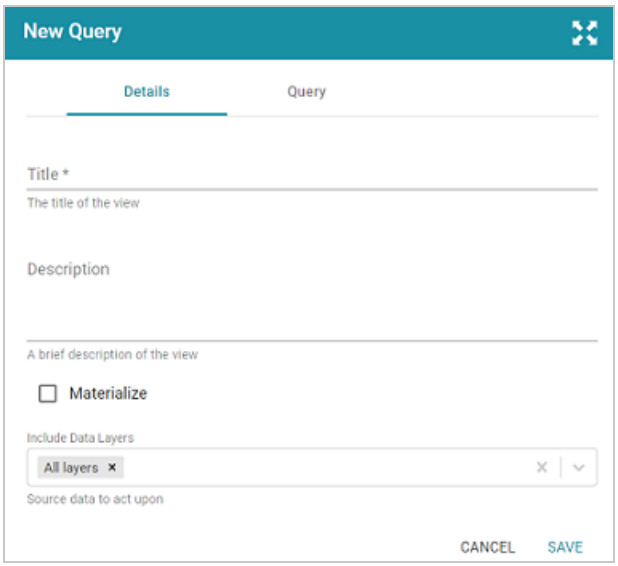
> **Note**
> If a request is sent to a disabled endpoint, Anzo displays a `503: Service Unavailable` error with a message indicating that the endpoint is disabled. For example, "Unable to process request. The endpoint '<name>' is DISABLED."

7. (Optional) You can click the **Details** tab to make changes to any of the following additional configuration options that become available after the endpoint is initially created:



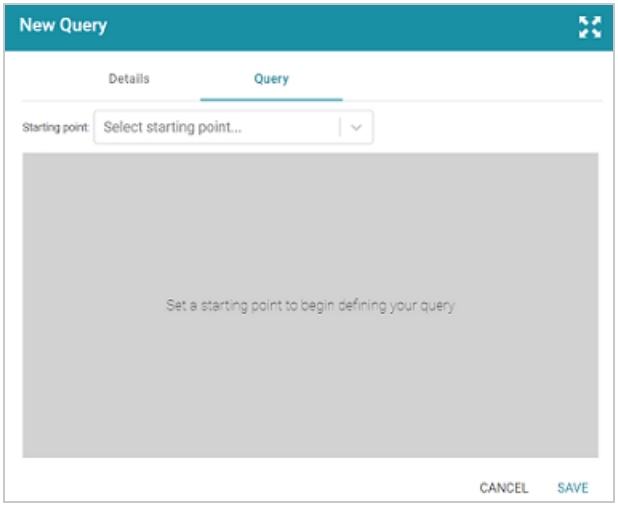
- **Endpoint Query Priority**: This setting controls where in Anzo's query queue the generated SPARQL queries for this endpoint are placed for processing, i.e., the priority for executing queries against this endpoint versus other types of queued queries like data layer and dashboard queries. By default, Endpoint Query Priority is set to **Extraction**, which is priority level 3 out of 8. The options are Interactive = 1, Extraction = 3, and Batch Process = 8.
- **Cache Enabled**: When the endpoint is accessed, Anzo translates the OData query to a SPARQL query and sends it to AnzoGraph for execution. The Cache Enabled setting controls whether the results of that AnzoGraph query are cached in Anzo so that subsequent endpoint requests can run against the cache in Anzo. When Cache Enabled is selected (the default setting), Anzo stores the cached results and AnzoGraph only gets queried if the cached results are invalidated and need to be refreshed. When Cache Enabled is disabled, Anzo does not store the cache, and endpoint requests are sent to AnzoGraph.

8. To start building a view, click the **Queries** tab (if necessary), and then click the **Add Query** button. The New Query dialog box is displayed.



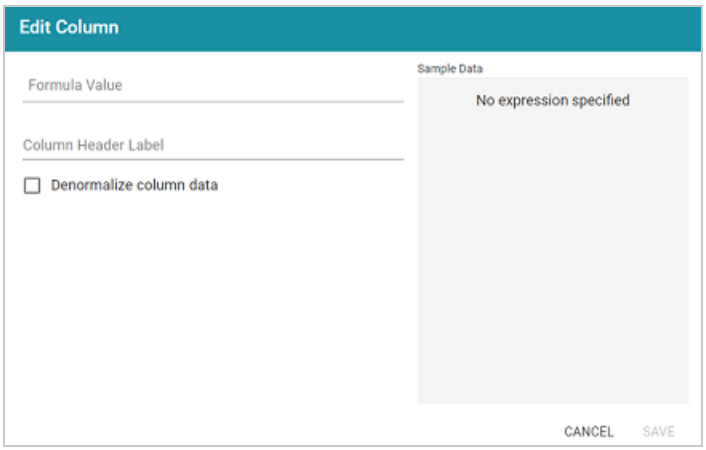
9. On the **Details** tab, configure the following options as needed:
   - **Title**: Type a name for the table in the Title field. The name must be unique for the endpoint.
   - **Description**: You can add an optional description for the view in this field.
   - **Materialize**: If you want to store a copy of the data that this view creates (materialize the data), select the **Materialize** check box. When this option is disabled AnzoGraph creates a virtual view where only the view definition is stored in memory and not a copy of the data. If a request is made against this view, AnzoGraph temporarily materializes the data in memory, performs the query operations, and then drops the temporary data.
   - **Include Data Layers**: By default, the Include Data Layers option is set to include **All Layers**. If you do not want the query to target the source data in all layers, select the field and choose alternate layers from the drop-down list.

10. Click the **Query** tab to start building a table.



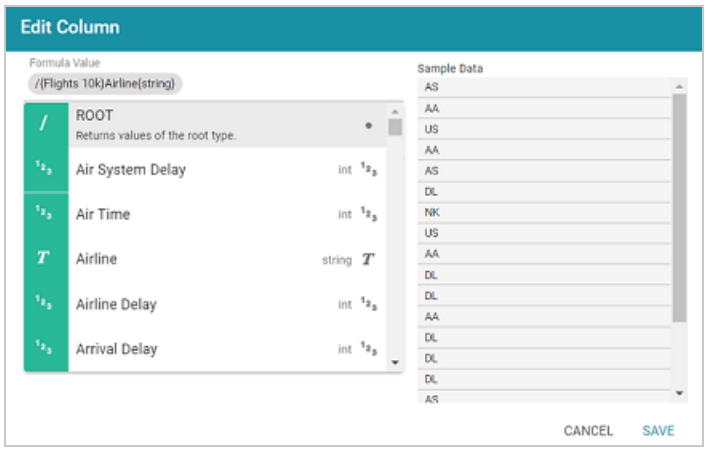
11. On the Query screen, get started by selecting a class to use as the starting point. Click the **Select Starting Point** drop-down list and select a class. Once you select the class, the **Add Column** button is displayed.

12. Click **Add Column** to create a column in the table. The Add Column dialog box is displayed:



13. On the Add Column screen, configure the following options as needed:

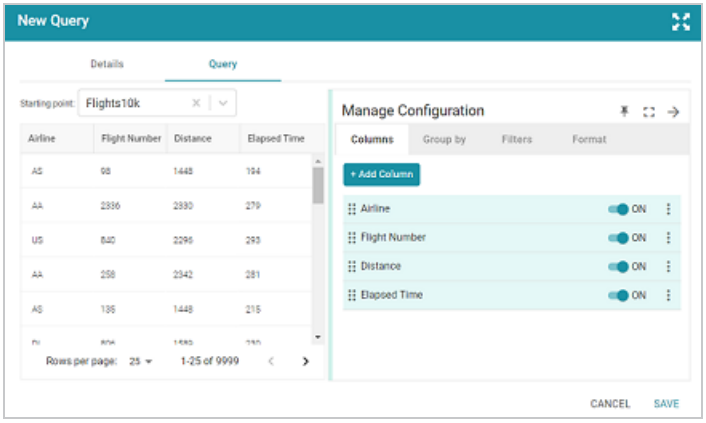
    - **Formula Value**: Click this field to choose the column header. Like building columns in a dashboard table, you can navigate the relationships to join data from different classes, and you can apply functions to the values. When you click the field and start to select paths and properties, sample data is shown on the right side of the screen. For example:



- **Column Header Label**: This is the label that you want to use for displaying the Formula Value.
- **Denormalize column data**: If you enabled **Denormalize Results** at the endpoint level, leave this setting disabled. If Denormalize Results is disabled at the endpoint level, you can enable this setting to denormalize the values for this column only.

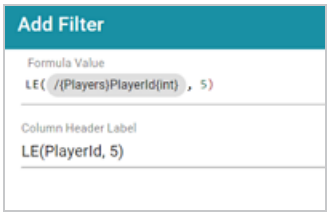
> **Tip**
> If you do not denormalize the data, you can use the **Format** tab to specify the character to use for separating the values in the arrays that are returned for multi-valued properties. By default, the Value Separator is set to comma (,). The Format tab becomes available after saving the column.

14. Click **Save** to add the column to the table.
15. Add new columns to the table by clicking **Add Column**. You can change the order of columns by dragging a column up or down, and you can enable or disable columns by sliding the slider for the column. Click the menu icon (⋮) for a column to edit or delete that column.



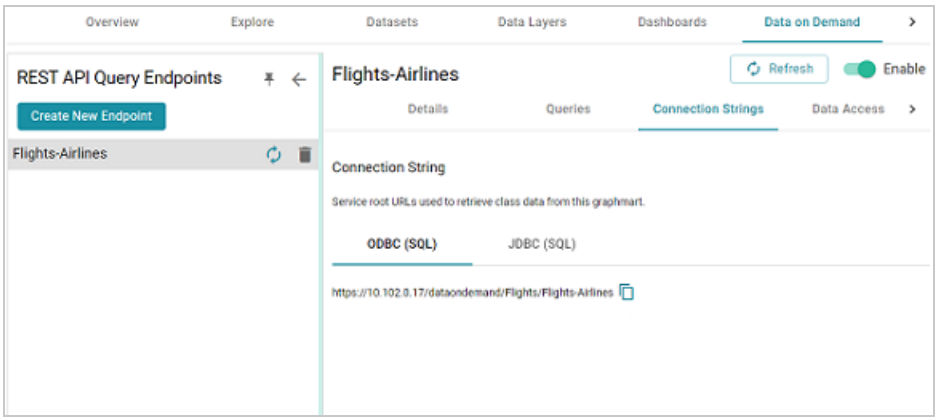
16. If you would like to group by clause, you can click the Group By tab and select the column or columns to group.

17. If you would like to filter out some data, you can add one or more filters to the overall query. To add a filter, click the **Filters** tab and then click **Add Filter**. In the **Formula Value** field, specify the formula to use to determine which values should be included in the results. Then specify a name for the column in the **Column Header Label**. For example, the filter below is configured to include only the results where the Player ID is less than or equal to 5.



18. Once you are satisfied with the configuration, click **Save** to create the endpoint.

This endpoint is ready for access via OData/ODBC or JDBC. On the **Connection Strings** tab for the endpoint, retrieve the ODBC or JDBC service URL to use to access the endpoint. For example:



To test whether the endpoint is active, you can copy the ODBC service URL and paste it into a web browser.

Important

After pasting the URL into the browser, **add the table name to the end of the string**. The URL for a Custom endpoint is `<ODBC_or_JDBC_URL>/<table_name>`. For example, if the ODBC URL is `https://10.10.0.11/dataondemand/Flights/Flights-Airlines`, and the table name is `Airlines`, the connection string is `https://10.10.0.11/dataondemand/Flights/Flights-Airlines/Airlines`.

If the endpoint is active, the browser shows an XML feed of the data. For example:



Note

The endpoint is accessible only when it is **Enabled** and the associated graphmart is **Active**.

For information about accessing endpoints programmatically, see Accessing an Endpoint Programmatically. For information about accessing endpoints with third-party analytics tools, see Accessing an Endpoint from an Application. For information about the supported OData operators, output format, and query examples, see OData Reference.

Sharing Access to Graphmarts

This topic introduces the concepts to know when working with graphmart and data layer access control and provides instructions for configuring permissions.

- Sharing Concepts
- Changing Configuration-Level Access
- Changing Data-Level Access

Sharing Concepts

This section describes the concepts that are helpful to know when working with graphmart and data layer permissions. It also gives of overview of the graphmart sharing settings and the predefined permission sets and associated privileges.

- Default Access Configuration
- Configuration vs. Data Access Control
- Permission Inheritance
- Configuration Permissions

Default Access Configuration

When a new graphmart is created, the access control configuration of that Graphmart is defined by the **Graphmarts Registry** Default Access Policy that is configured by your administrator (see Managing Default Access Policies in the Administration Guide for information). The graphmart also inherits permissions from other artifacts in the onboarding workflow. For example, when a graphmart is created from a data source, the graphmart inherits permissions from the source schema (which inherits permissions from the data source). Users who have permission to modify graphmart access can share that graphmart with other users and groups.

Configuration vs. Data Access Control

Graphmart and data layer sharing is managed on two levels: **Configuration** and **Data Access**. When managing access at the **Configuration** level, you are controlling who can view or modify the *configuration* of the graphmart, such as who can edit the graphmart settings on the Overview tab, who can enable, disable, modify, or add layers, and who can view or modify the graphmart permissions. The **Data Access** configuration controls who can view the data that is contained within the graphmart.

Permission Inheritance

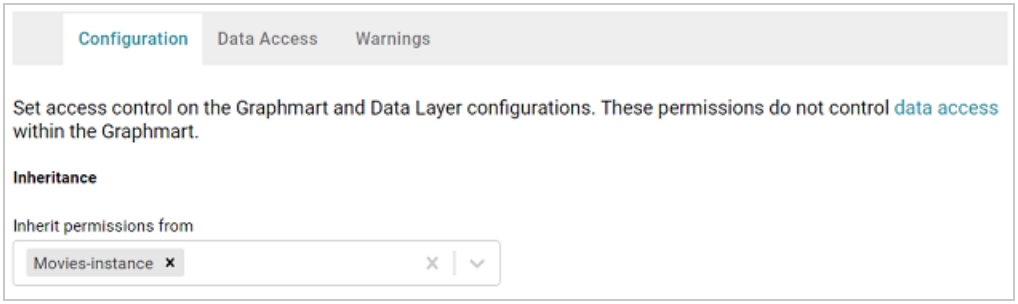
When assigning Configuration and Data Access permissions at the graphmart level, you can configure the graphmart to inherit the permissions from another artifact and/or pass on its permissions to additional artifacts. For example, you can configure one graphmart to pass its permissions to other graphmarts. Inheritance transmits all of the artifact's permissions for all users and groups.

> **Note**
> Since data layers are created in graphmarts, they inherit their permissions from the graphmart by default—with one exception: Layers with Load Dataset Steps inherit their Data Access permissions from the dataset. Data on Demand endpoints also inherit their permissions from the parent graphmart by default.

The following inheritance settings are displayed at the top of the Configuration and Data Access tabs on the graphmart Sharing screen.

Configuration Inheritance

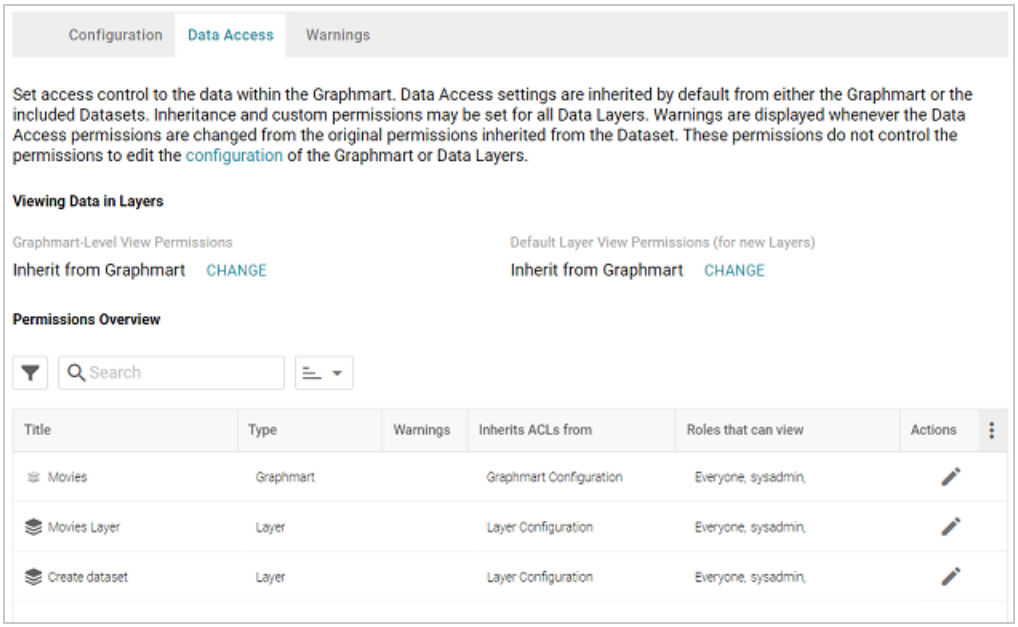
The image below shows a graphmart Configuration tab with the default inheritance settings. The **Inherit permissions from** field shows that the graphmart inherits permissions from the schema instance that the graphmart was created from.



Data Access Inheritance

The image below shows the Data Access tab for the same graphmart. The **Graphmart Level View Permissions** are set to **Inherit from Graphmart** by default. And **Default Layer View Permissions (for new Layers)** is also set to **Inherit from Graphmart**.

Below the inheritance settings, the **Permissions Overview** provides a detailed view of the permission inheritance for each layer, view, and Data on Demand endpoint in the graphmart.



Configuration Permissions

Graphmart Configuration permissions control who can view or modify the graphmart settings, who can enable, disable, modify, or add data layers, and who can view or modify the graphmart permissions. There are three predefined permission sets that can be applied to a user or group. The permission sets include a combination of six permissions. You also have the option to customize the set of permissions that are applied to a user or group.

The tables below list the predefined permission sets and describe the privileges that are granted for each permission that is part of the set:

View

The following table describes the permissions in the **View** set.

Permission	Allows a user to:
View	• See the graphmart in the Anzo application. • Copy the graphmart URI from the Overview tab. • Copy data layer URIs from the data layers tab. • See the existing Data on Demand endpoints on the Data on Demand tab. • View and clone the dataset editions that are included in the graphmart. • Reload and refresh the graphmart. • Create and import graphmart versions.
Meta View	This permission relates only to the graphmart Sharing tab. A user with this permission can see the Sharing tab, but they cannot modify, add, or remove permissions.

Modify

In addition to the **View** and **Meta View** permissions described above, the **Modify** set includes the **Add/Edit** and **Delete** permissions described below.

Permission	Allows a user to:
Add/Edit	• Rename the graphmart and edit the description. • Create Data on Demand endpoints. • Add datasets and data sources to the

Permission	Allows a user to:
	graphmart. • Enable, disable, add, or edit layers and steps. • Activate and deactivate the graphmart.
Delete	• Remove datasets from the graphmart. • Delete data layers and steps from the graphmart. • Cannot delete the graphmart.

Admin

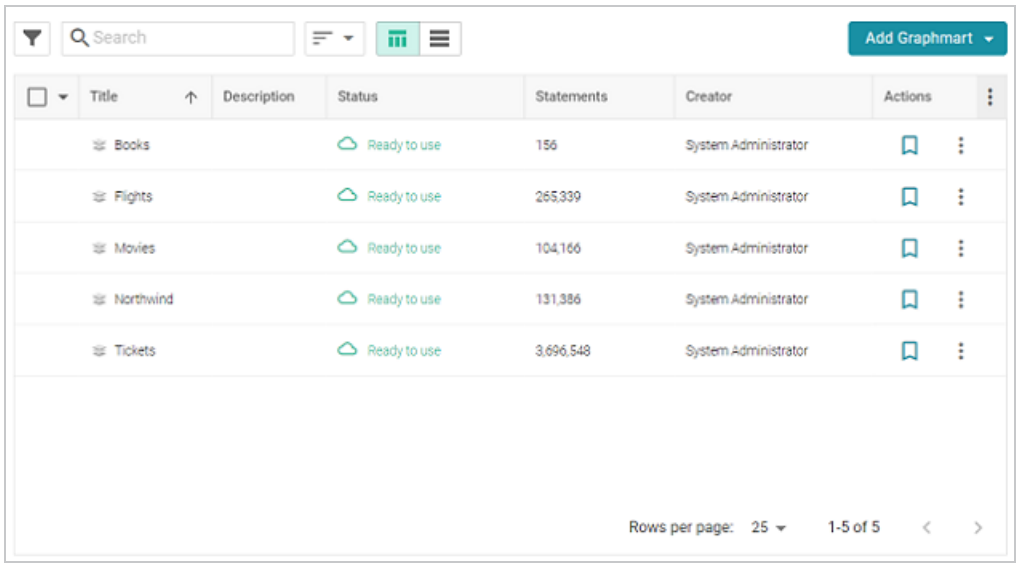
In addition to the **View**, **Meta View**, **Add/Edit**, and **Delete** permissions described above, the **Admin** set includes the **Meta Add/Edit** and **Meta Delete** permissions described below.

Permission	Allows a user to:
Meta Add/Edit	This permission relates only to the graphmart Sharing tab. A user with this permission can modify the sharing settings by adding permissions to a user or group.
Meta Delete	• Modify the sharing settings by removing permissions from a user or group. • Delete the graphmart.

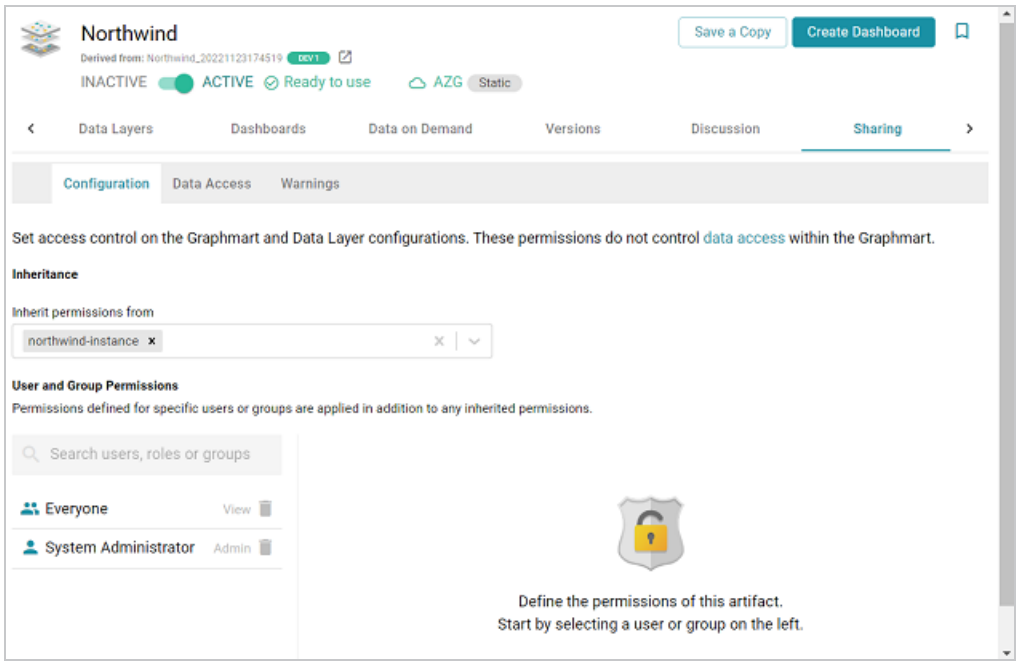
Changing Configuration-Level Access

Follow the steps below if you want to modify the configuration-level access for a graphmart.

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



2. Click the name of the graphmart for which you want to configure permissions. Then click the **Sharing** tab. The Sharing screen is displayed and the **Configuration** tab is selected. For example:

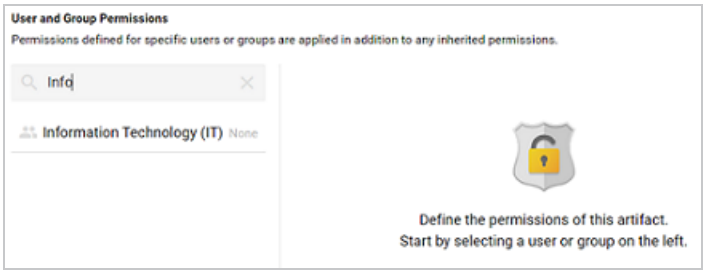


3. If you want to change how the Configuration permissions are inherited, use the **Inherit permissions from** field at the top of the screen. To apply all of the permissions from another

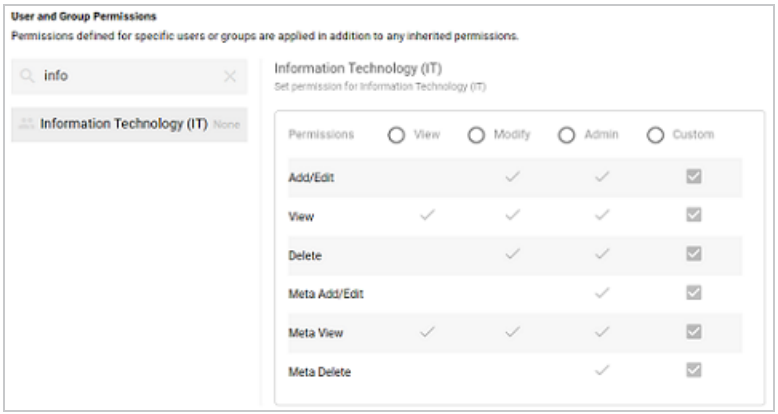
artifact to this one, select the artifact to inherit from in the **Inherit permissions from** field.

> **Tip**
> For more information about permission inheritance at the graphmart level, see Permission Inheritance.

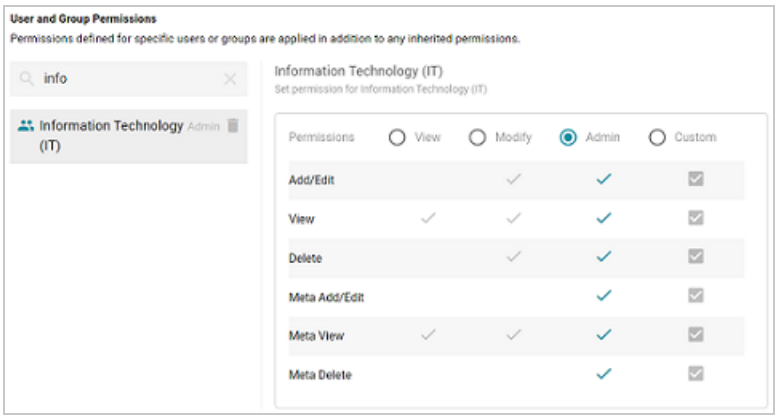
4. To modify Configuration access to this graphmart with a particular user or group, type a value in the **Search users, roles or groups** field to find and display the user or group. The resulting list shows the current permission level that is set for each user or group in the search results. For example, the image below shows the current permissions for the IT group (None):



5. Select the user or group for which you want to configure permissions. The permissions settings are displayed on the right side of the screen. For example:



6. To assign a predefined set of permissions, click the **View**, **Modify**, or **Admin** radio button to assign that level of access to the selected user or group. Refer to Configuration Permissions for details about the permission sets. For example, the image below gives **Admin** permissions to users in the IT group:

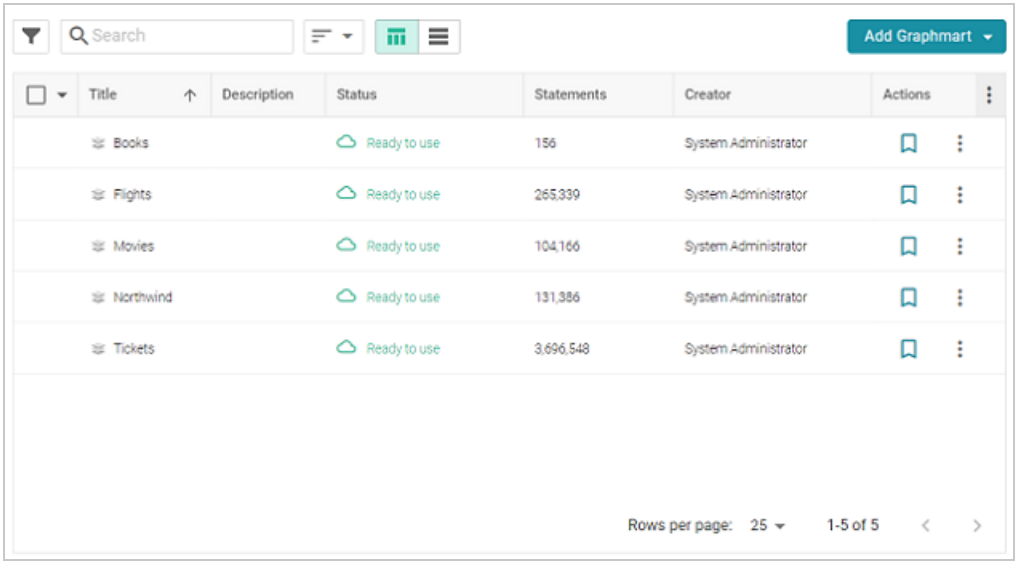


If you want to customize the permissions, click the **Custom** radio button and then select or deselect the permissions checkboxes. To clear permissions for a user or group, click the trashcan icon ( ) next to the name.

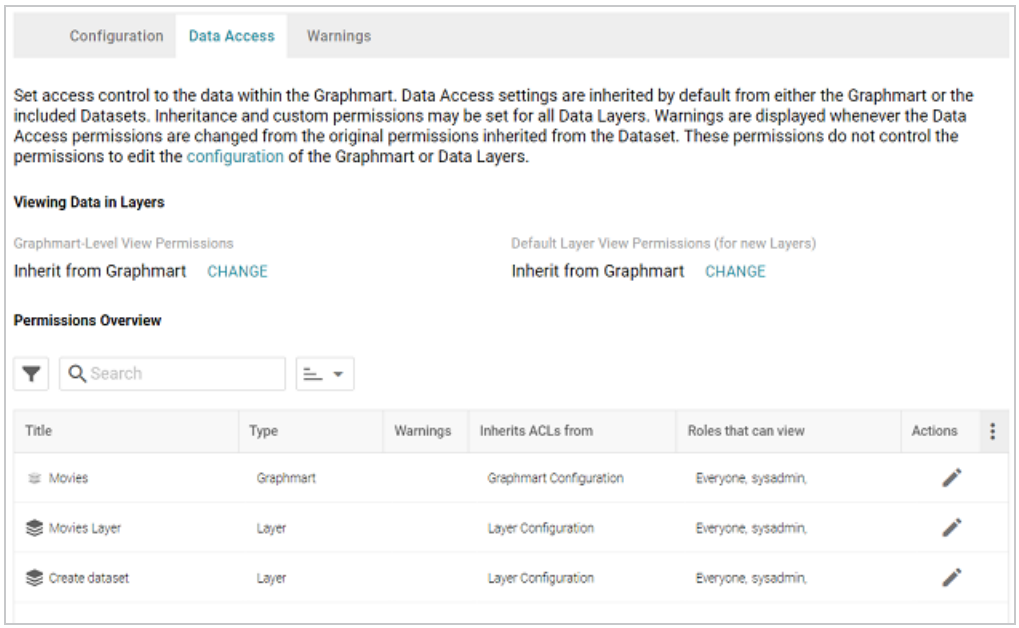
Changing Data-Level Access

Follow the steps below if you want to modify permissions at the Data Access level for a Graphmart.

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



2. Click the name of the graphmart for which you want to configure permissions. Then click the **Sharing** tab. The Sharing screen is displayed and the **Configuration** tab is selected. Click the **Data Access** tab. For example:



3. If you want to change how the Data Access permissions are inherited, use the fields at the top of the screen:
   - **Graphmart-Level View Permissions** controls who can view the data within the entire graphmart.
   - **Default Layer View Permissions (for new Layers)** controls who can view the data within the data layers.

   > **Tip**
   > For more information about permission inheritance at the graphmart level, see Permission Inheritance.

4. To change the permissions for an individual layer, Data on Demand endpoint, or another graphmart component that is listed in the Permissions Overview, click the Edit icon (✎) in the Actions column in the in the row for that component.

Changes to graphmart and layer permissions take effect immediately. Users do not need to log out and log back in, and affected graphmarts do not need to be reloaded or refreshed.

Graphmart FAQ

This topic provides answers to frequently asked questions about graphmarts.

- What is the difference between graphmart reload vs. refresh?
- How do I find the URI for a graphmart?
- How do I find the URI for a layer?
- How do I see the models in a graphmart?

What is the difference between graphmart reload vs. refresh?

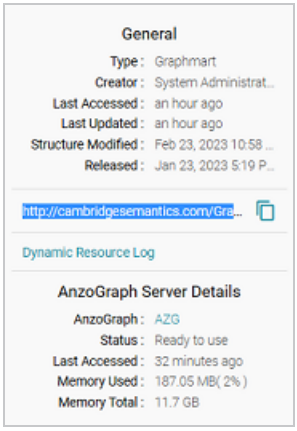
When you make modifications to data layers in a graphmart, Anzo displays **Reload** and **Refresh** buttons on the top of the screen. For example:



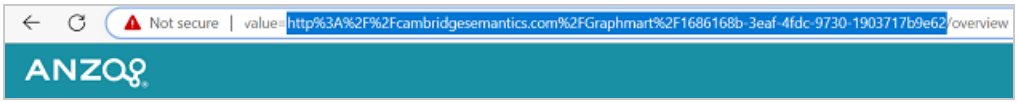
The Refresh option becomes available when changes have been made to one or more data layers. Clicking **Refresh** resets (deletes from AnzoGraph) and reloads only the data layers that have changed. Clicking **Reload** resets and reloads the entire graphmart to AnzoGraph, including the data layers that have not changed.

How do I find the URI for a graphmart?

Anzo displays graphmart details on the Overview screen for the graphmart. To view and copy a graphmart URI, go to the Overview tab for the graphmart. The URI is under General information on the right side of the screen. You can click the clipboard icon ( ) to copy the URI.

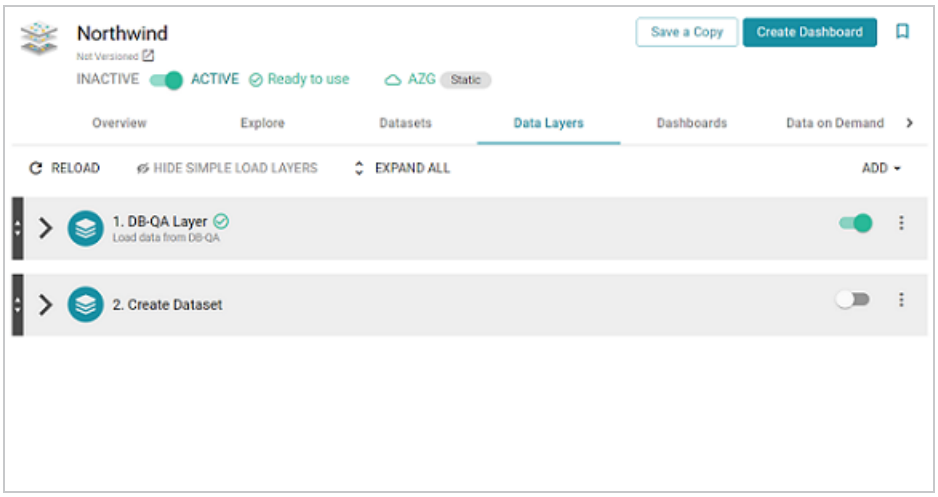


You can also copy a URL-encoded version of the graphmart URI from the address bar in the browser when viewing the graphmart. For example:

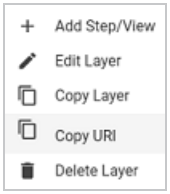


How do I find the URI for a layer?

You can retrieve a data layer URI on the Data Layers screen for a graphmart. To view and copy a layer URI, go to the Data Layers tab for the graphmart. Anzo displays the layers. Each layer is a graph. For example:



Click the menu icon (⋮) for the layer whose URI you want to copy and click **Copy URI**:



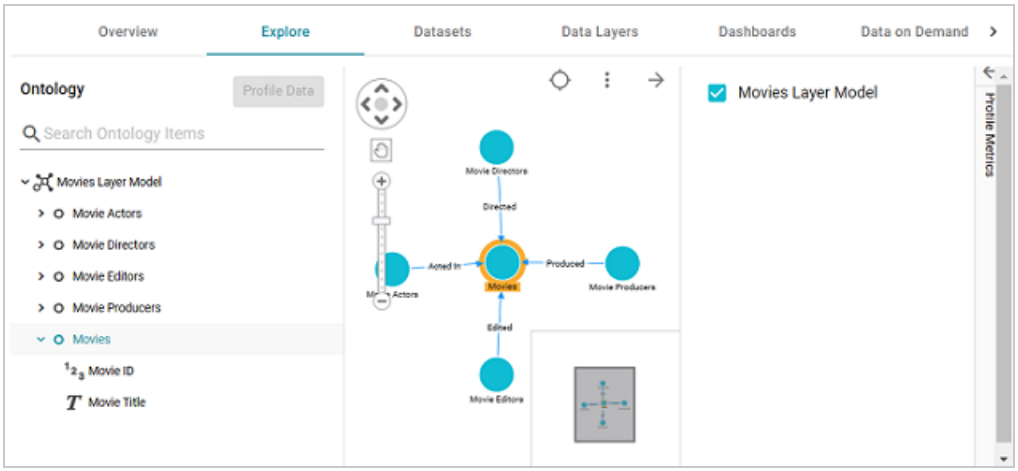
How do I see the models in a graphmart?

Anzo displays the list of included models on the Explore screen for a graphmart. To view the list of models:

1. Go to the Explore tab for the graphmart for which you want to view models.
2. In the top right corner of the graph view in the center of the screen, there are three icons:

3. To view the associated models, click the contents icon (☰) on the right.
4. For example, the image below shows a graphmart with one model:

   You can click the arrow icon (→) to close the model panel.

Profiling Datasets and Graphmarts

To help you explore your data and assess its quality, Anzo provides the option to generate a data profile for datasets and graphmarts. Creating a profile runs several metrics against the data and reports statistics at the class, property, and instance levels. Data profile metrics measure data quality, perform data discovery, and can help you decide on the types of analytics to run. The topics in this section provide instructions for generating data profiles and describe each of the metrics that Anzo runs.

In this section:

Generating a Dataset Data Profile 605

Generating a Graphmart Data Profile 610

Data Profiling Metrics 614

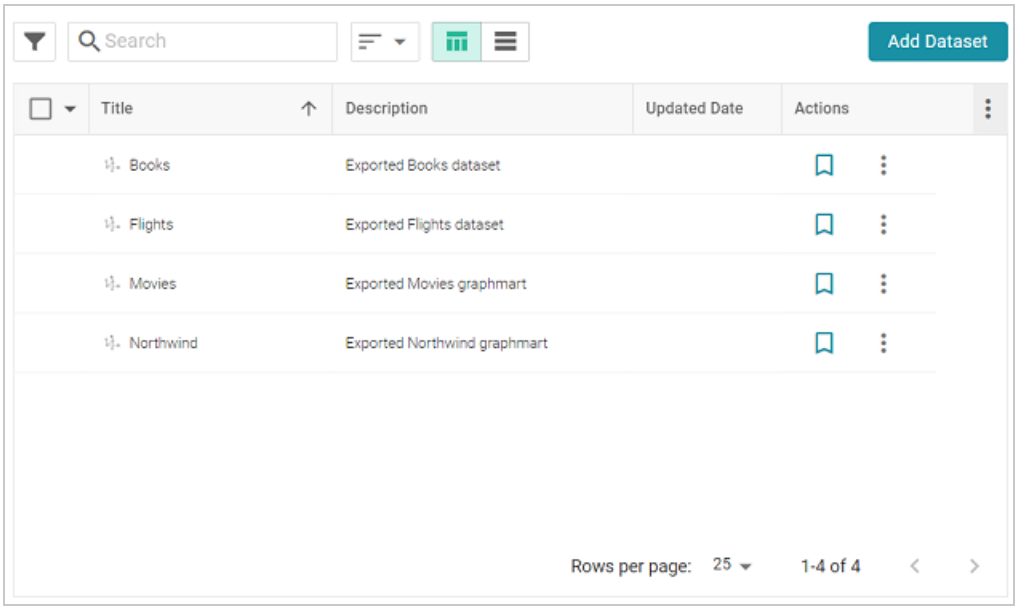
Generating a Dataset Data Profile

Similar to generating a profile for a graphmart, you can generate a data profile for a dataset in the Datasets catalog. Generating a dataset profile helps users perform data discovery, assess the quality of the onboarded data, and decide whether to use the dataset in a graphmart. The reports can also assist users in determining the types of data layer steps to create and writing the queries to include in the steps.

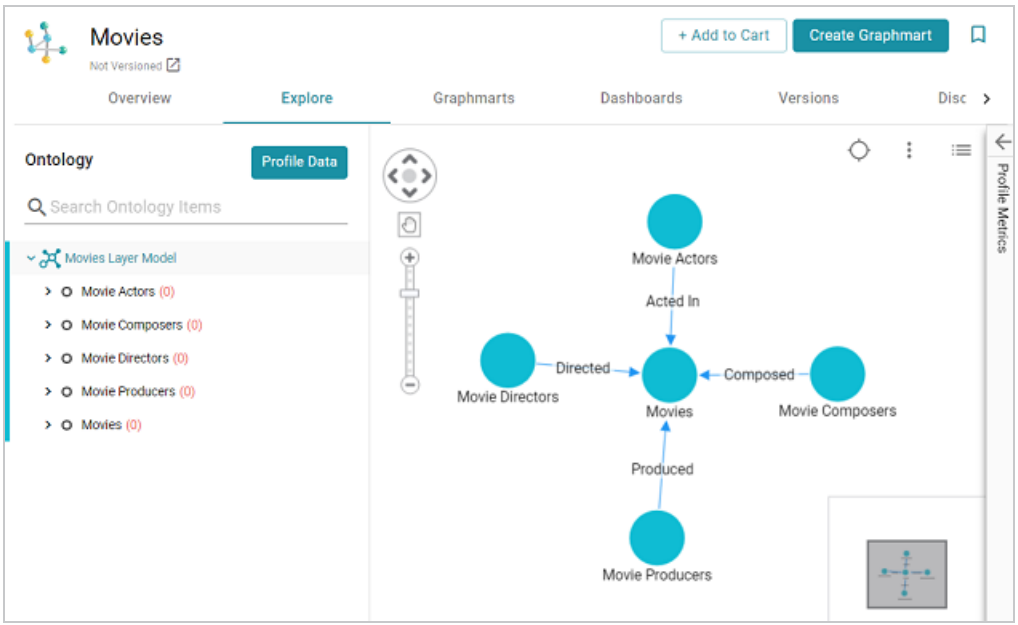
Important

To generate a dataset data profile, AnzoGraph must be online. If you have dynamic AnzoGraph deployments enabled, AnzoGraph will be provisioned automatically when the profile is generated.

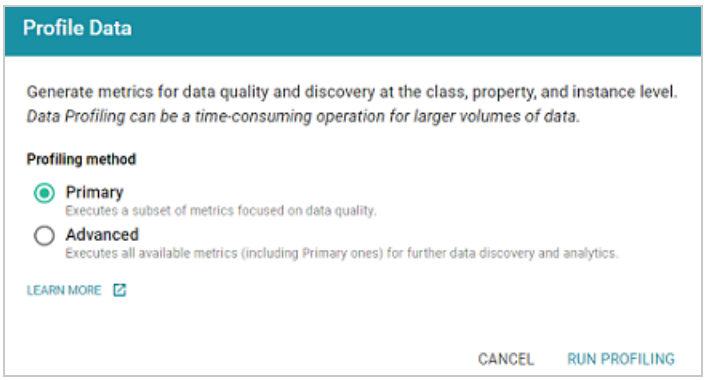
1. In the Anzo application, expand the **Blend** menu and click **Datasets**. Anzo displays the Datasets screen, which lists the catalog of datasets. For example:



2. On the Datasets screen, click the name of the dataset that you want to generate a profile for. Anzo displays the Explore tab for the dataset. For example, the image below shows a dataset for which a data profile has not been generated and the class and property counts are 0:



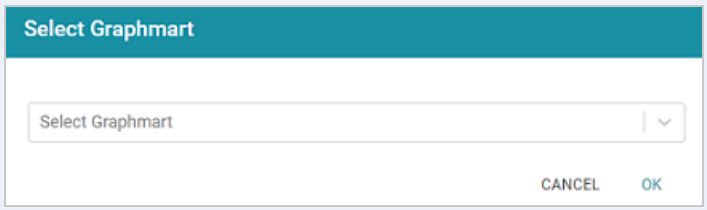
3. Click the **Profile Data** button on the left side of the screen. The Data Profile dialog box is displayed::



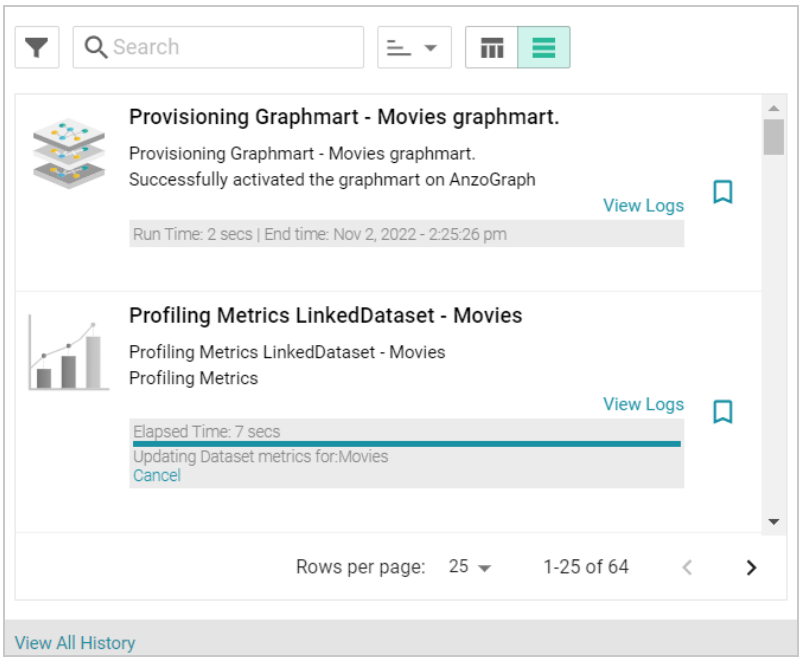
4. On the Data Profile screen, choose the Profiling Method to use. The **Primary** method focuses on data quality type analysis. The **Advanced** method includes the Primary data quality analytics plus several advanced metrics for deeper data discovery and analysis. For details about the Primary and Advanced metrics, see Data Profiling Metrics.

> **Note**
>
> To run the Advanced metrics, the optional AnzoGraph C++ extensions and dependencies must be installed. If you use dynamic, K8s-based deployments of AnzoGraph, the extensions are included. If you installed a static AnzoGraph instance with the installer, the C++ extensions are optional and are only installed if **yes** was specified for the `Do you want to install C++ UDXs packaged with AnzoGraph DB?` prompt. For information about the C++ dependencies, see Install the Optional C++ Extension Dependencies in the Deployment Guide.

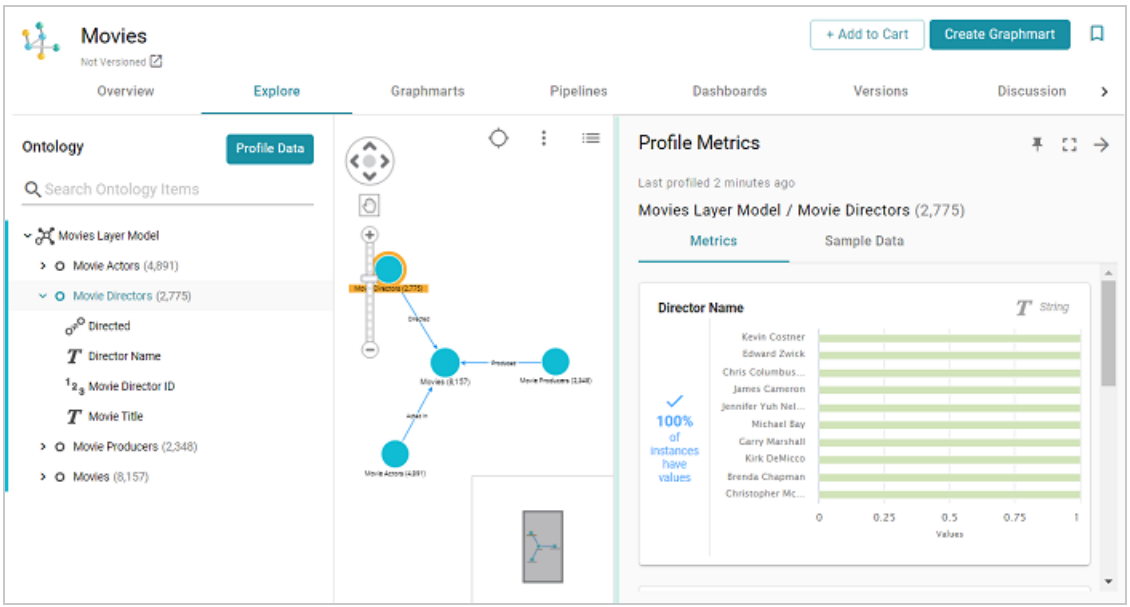
5. Click **Run Profiling** to start generating the profile.

> **Note**
>
> If the dataset is used in a graphmart that is active, Anzo displays the Select Graphmart dialog box (shown below), which prompts you to choose whether the online dataset can be used for running the profiling queries or whether to provision another temporary graphmart for the dataset.
>
> 
>
>
> To use the dataset in the graphmart that is online, click the **Select Graphmart** drop-down list and select the graphmart name. If you want Anzo to provision a temporary graphmart instead, select **Don't reuse the Graphmart**.

The profiling process may take several minutes, especially for large volumes of data. You can check the status of the process in the Activity Log. The Activity Log also presents the option to stop the profiling process by clicking **Cancel** under the progress bar for the task. For example:



6. Once the profiling is complete, the Profile Metrics panel is expanded on the Explore tab. To populate the panel, click a class or property in the ontology or a class in the graph view in the middle of the screen. For example:



You can click the Expand icon (⛶) on the right side of the screen to collapse the graph view and expand the metrics view.

Select any class or property to view its metrics. For details about each of the metrics that are run, see Data Profiling Metrics.

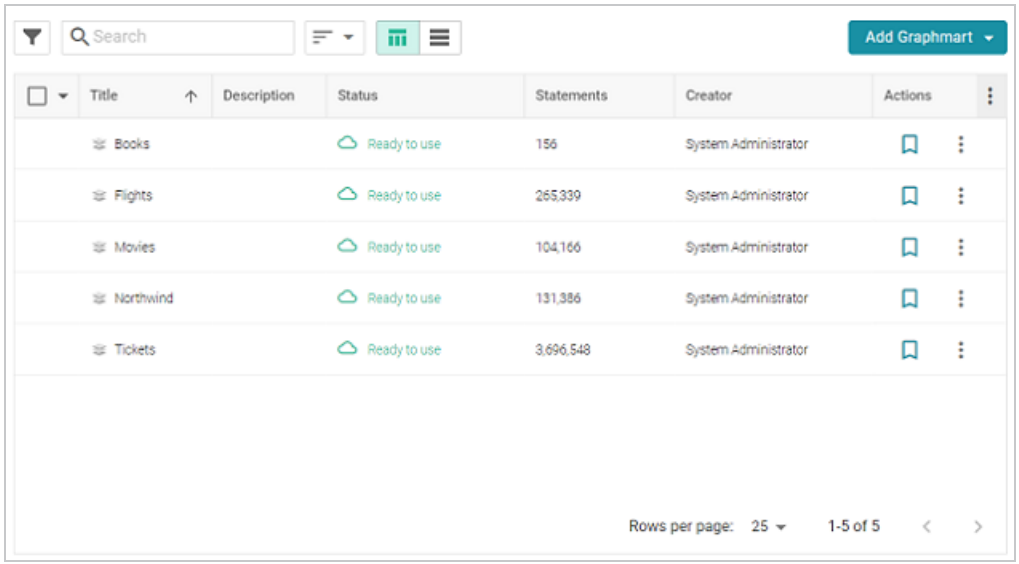
Generating a Graphmart Data Profile

Similar to generating a data profile for a dataset, you can profile a graphmart, which may include multiple datasets. When metrics are generated for graphmarts, Anzo profiles the data that results from all of the enabled layers and reports metrics for the classes and properties in the model as well as statistics about the values for the properties. Generating a graphmart profile helps users perform data discovery, assess the quality of the data, and decide on the types of analytics to perform.

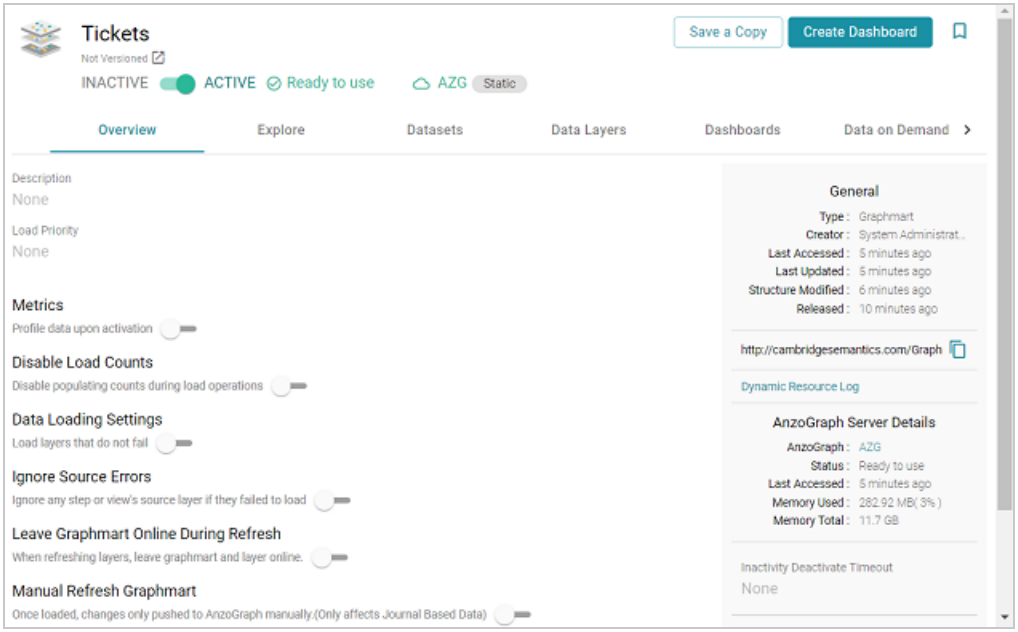
Important

To generate a graphmart data profile, AnzoGraph must be online. If you have dynamic AnzoGraph deployments enabled, AnzoGraph will be provisioned automatically when the profile is generated.

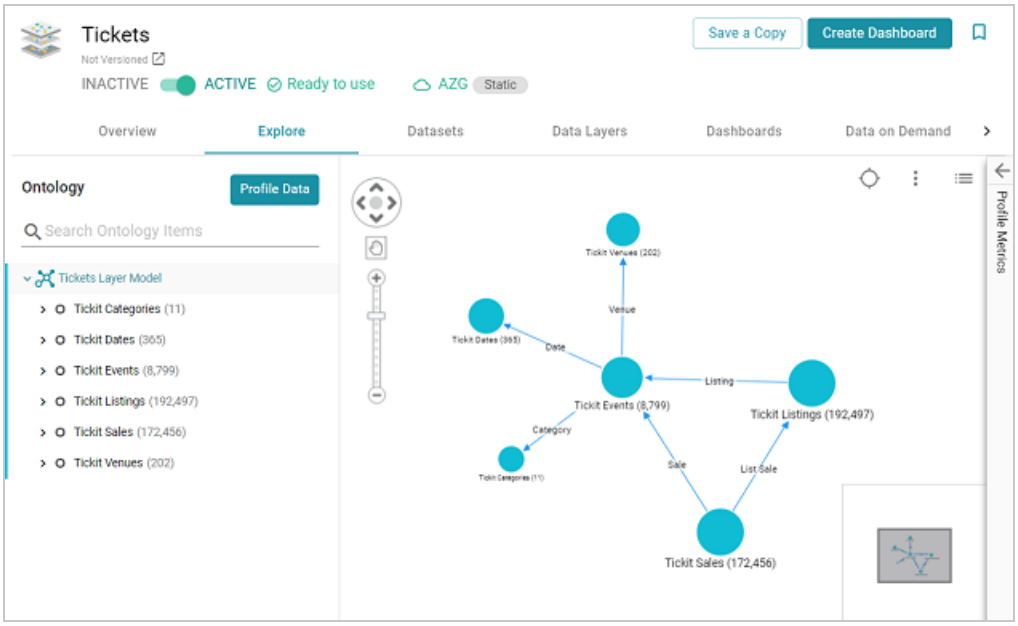
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



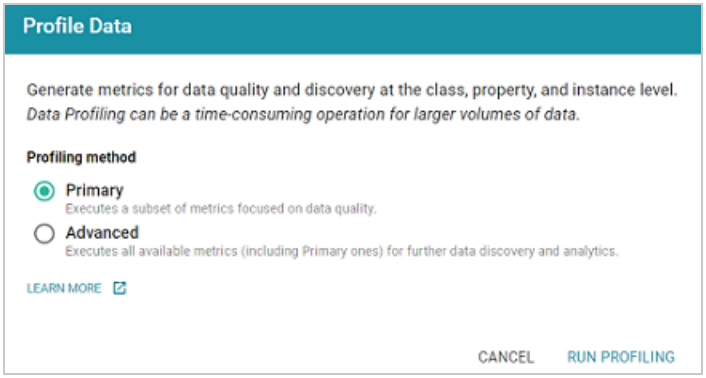
2. On the Graphmarts screen, click the name of the graphmart for which you want to generate metrics. Anzo displays the Overview for that graphmart. For example:



3. Click the **Explore** tab.



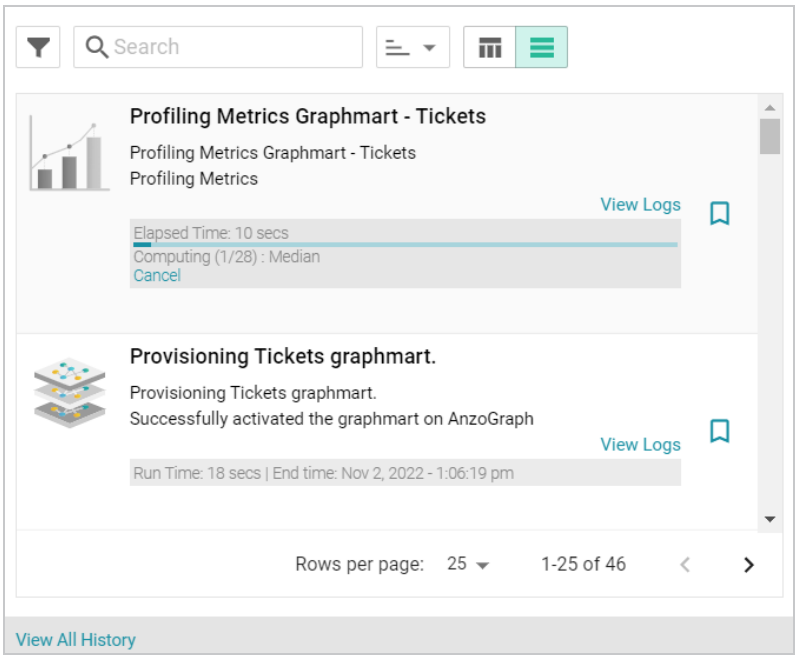
4. If necessary, activate the graphmart, and then click the **Profile Data** button on the left side of the screen. The Data Profile dialog box is displayed:



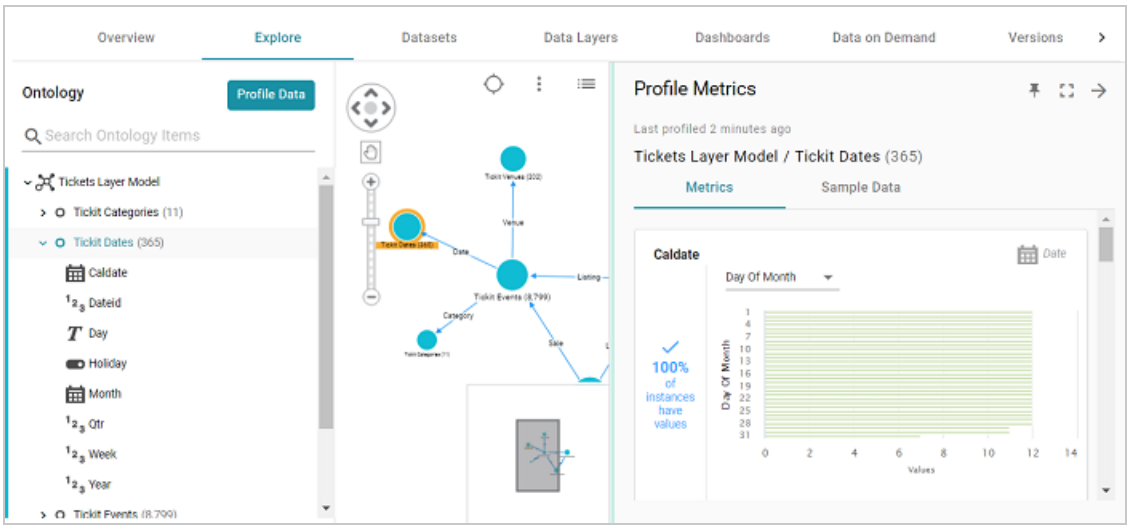
5. On the Data Profile screen, choose the Profiling Method to use. The **Primary** method focuses on data quality type analysis. The **Advanced** method includes the Primary data quality analytics plus several advanced metrics for deeper data discovery and analysis. For details about the Primary and Advanced metrics, see Data Profiling Metrics.

   > **Note**
   > To run the Advanced metrics, the optional AnzoGraph C++ extensions and dependencies must be installed. If you use dynamic, K8s-based deployments of AnzoGraph, the extensions are included. If you installed a static AnzoGraph instance with the installer, the C++ extensions are optional and are only installed if **yes** was specified for the `Do you want to install C++ UDXs packaged with AnzoGraph DB?` prompt. For information about the C++ dependencies, see Install the Optional C++ Extension Dependencies in the Deployment Guide.

6. Click **Run Profiling** to start generating the profile. The process may take several minutes, especially for large volumes of data. You can check the status of the process in the Activity Log. The Activity Log also presents the option to stop the profiling process by clicking **Cancel** under the progress bar for the task. For example:



7. Once the profiling is complete, the Profile Metrics panel is expanded on the Explore tab. To populate the panel, click a class or property in the Ontology or a class in the graph view in the middle of the screen. For example:



You can click the Expand icon ( ) on the right side of the screen to collapse the graph view and expand the metrics view.

Select any class or property to view its metrics. For details about each of the metrics that are run, see Data Profiling Metrics.

Data Profiling Metrics

When a data profile is generated for a dataset or graphmart, Anzo runs several metrics that can help users measure data quality and perform data discovery at the class, property, and instance level. The metrics are grouped into two categories: a **Primary** category that focuses on data quality type analysis, and an **Advanced** category that includes the Primary data quality analytics plus several advanced metrics for deeper data discovery and analysis.

> **Note**
> To run the Advanced metrics, the optional AnzoGraph C++ extensions and dependencies must be installed.

The lists below give a summary of the Primary and Advanced metrics. For more information and sample images of the visualizations that are generated, click a metric name.

Primary

- Row Count: Reports the total row (instance) count per class.
- Absent Property: For each property, reports the total number of instances that do not have a value.
- Empty Property: For each **string** property, reports the total number of empty strings.
- Sample Values: Returns sample values for each property.
- Average: For each **numeric** property, computes the average of all values.
- Sum: For each **numeric** property, computes the sum of all values.
- Average String Length: For each **string** property, computes the average length of the strings.
- Value Present: For each property, computes the percentage of instances that have at least one value.
- Unique Pattern Count: For each property, counts the total number of unique value patterns.
- Unique Values Count: For each property, counts the total number of unique values.

- Median: For each **numeric** property, computes the median of all values.
- Standard Deviation: For each **numeric** property, computes the standard deviation of all values.
- Mode: For each **numeric** property, computes the mode of all of the values.
- Presence: For each property in a class, reports the percentage of instances that have values vs. do not have values.
- Top Value Counts: For each property, computes the top *N* most occurring values.
- Bottom Value Counts: For each property, computes the *N* least occurring values.
- Top Pattern Counts: For each property, computes the top *N* most common value patterns.
- Bottom Pattern Counts: For each property, computes the *N* least common value patterns.
- Range: For **numeric** properties, reports the total range of values.
- Value Types: For each property, returns the data types for the instances.
- DateTime Distribution By Year/Month/Day: For **dateTime** properties, computes a histogram that shows the distribution of values by year, month, and day.

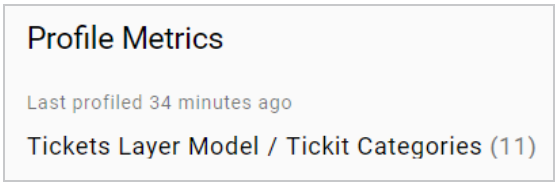
Advanced

- Pearson Skewness: For each **numeric** property, computes the Pearson coefficient of skewness.
- Geometric Mean: For each **numeric** property, computes the geometric mean of all values.
- Variance: For each **numeric** property, computes the variance of all values.
- Discrete Entropy: For each property, computes the discrete entropy of all values.
- Discrete Probability: For each property, computes the discrete probability of all values.
- String Length Range: For each **string** property, reports the range of string lengths.
- Unique Values: For each property, computes the percentage of unique values.

- Lower Case Strings: For each **string** property, computes the percentage of values with all lower case characters.
- Upper Case Strings: For each **string** property, computes the percentage of values with all upper case characters.
- Trivial Values: For each **string** property, computes the percentage of instances that have one of the following values: **NA**, **N/A**, **NONE**, or **NULL**.

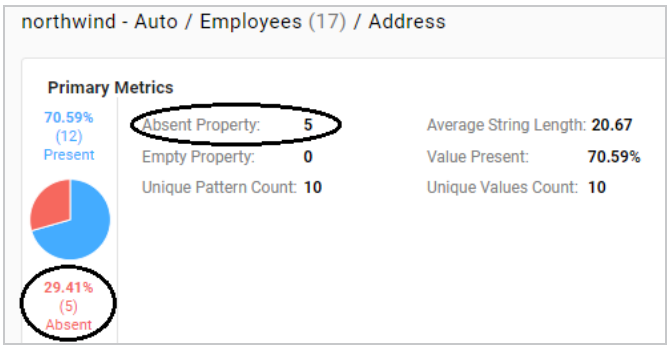
Row Count

This metric computes the total row count per class. The row count is in parenthesis at the top of the screen next to the class name, as shown in the image below.



Absent Property

For each property, this metric reports on the total number of instances that do not have a value for that property.



Empty Property

For string properties, this metric reports the total number of empty strings.



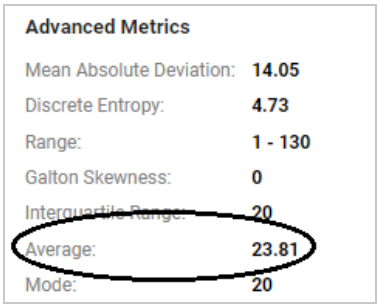
Sample Values

This metric returns sample values for each property.



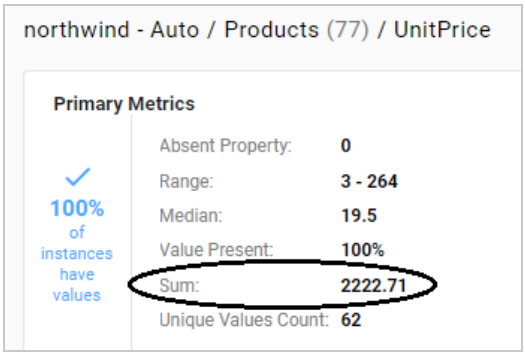
Average

For each numeric property, this metric computes the average of all values.



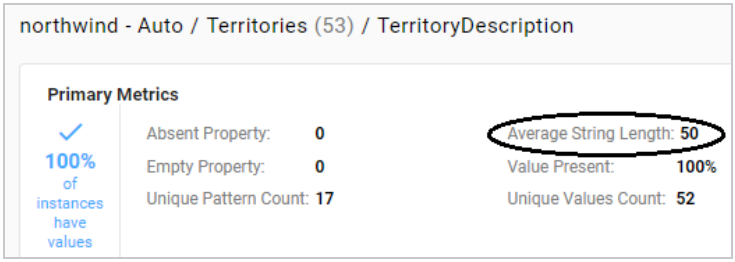
Sum

For each numeric property, this metric computes the sum of all values.



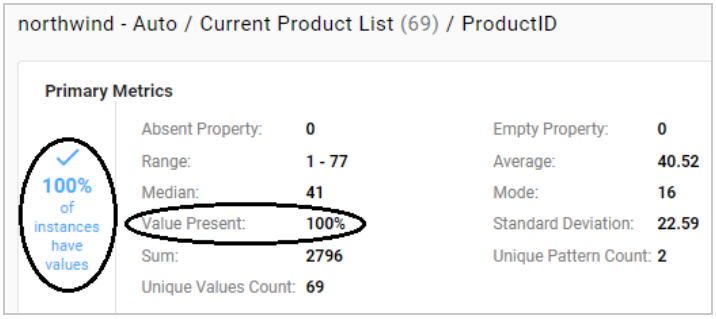
Average String Length

For each string property, this metric computes the average length of the values.



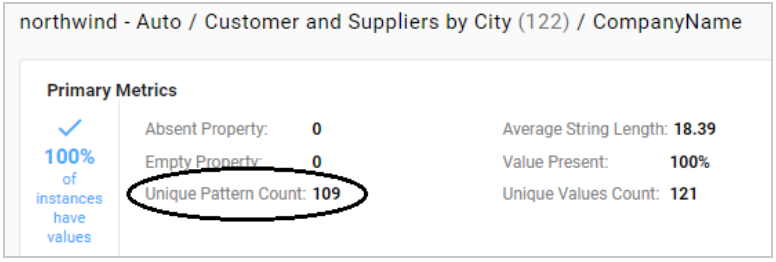
Value Present

For each property, this metric computes the percentage of instances that have at least one value.



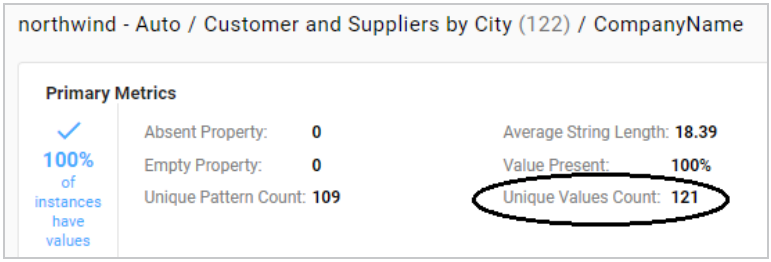
Unique Pattern Count

For each property, this metric counts the total number of unique value patterns.



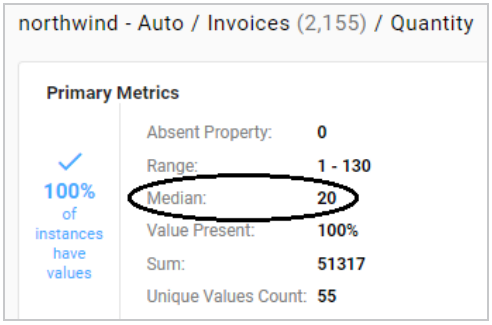
Unique Values Count

For each property, this metric counts the total number of unique values.



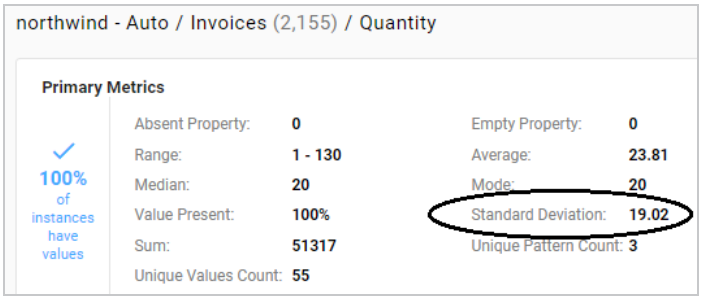
Median

For each numeric property, this metric computes the median of all values.



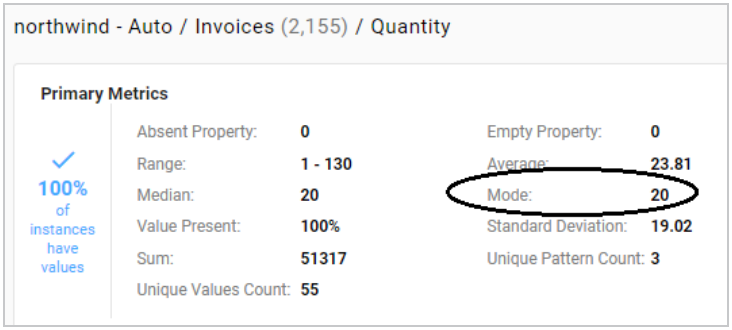
Standard Deviation

For each numeric property, this metric computes the standard deviation of all values.



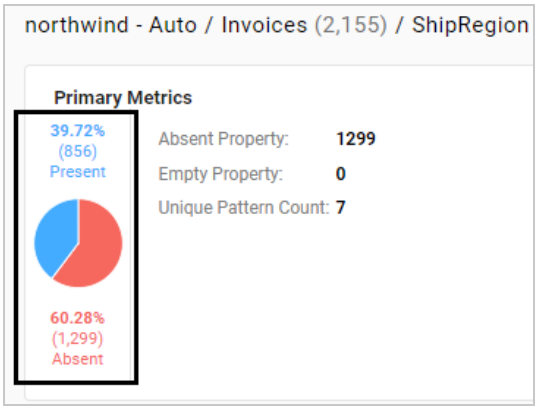
Mode

For each numeric property, this metric computes the mode of all of the values.



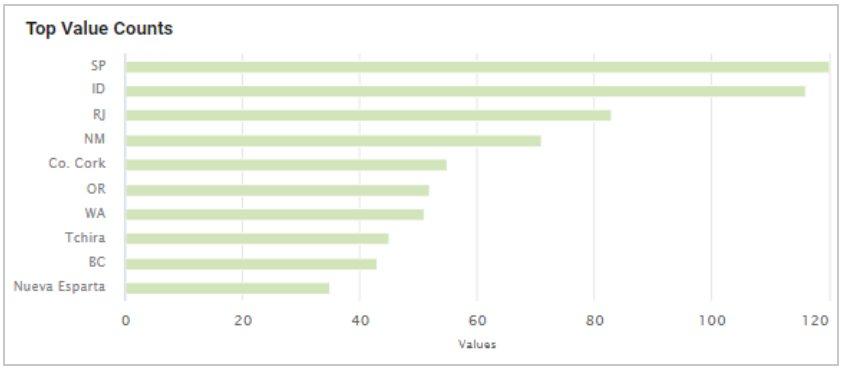
Presence

The metric is available when viewing a class. For each property in the class, this metric reports on the percentage of instances that have values and the percentage of instances that do not have values.



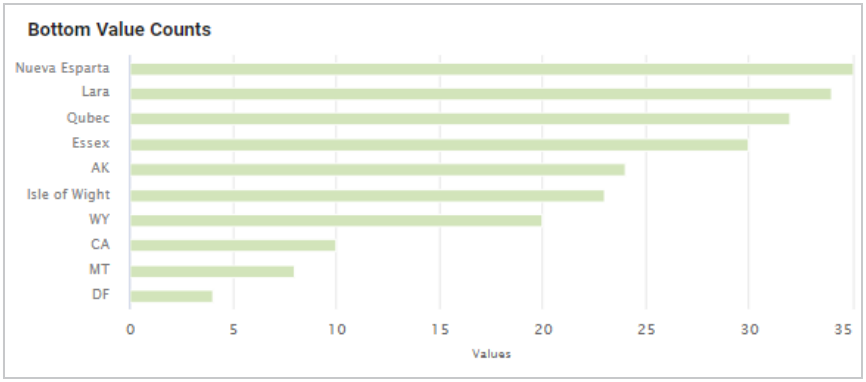
Top Value Counts

For each property, this metric computes the top *N* most occurring values.



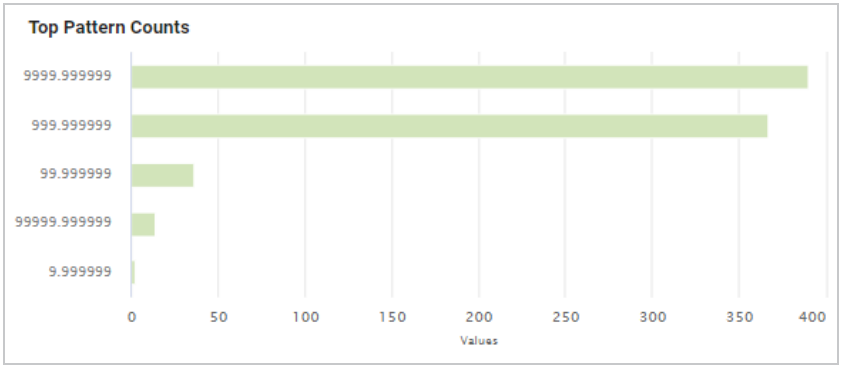
Bottom Value Counts

For each property, this metric computes the *N* least occurring values.



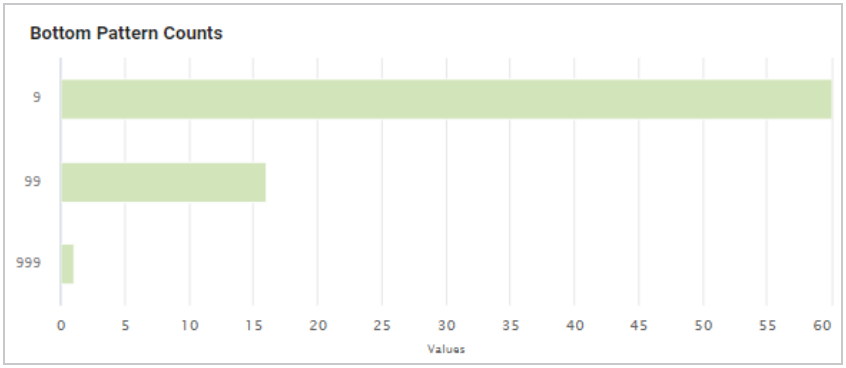
Top Pattern Counts

For each property, this metric computes the top *N* most common value patterns.



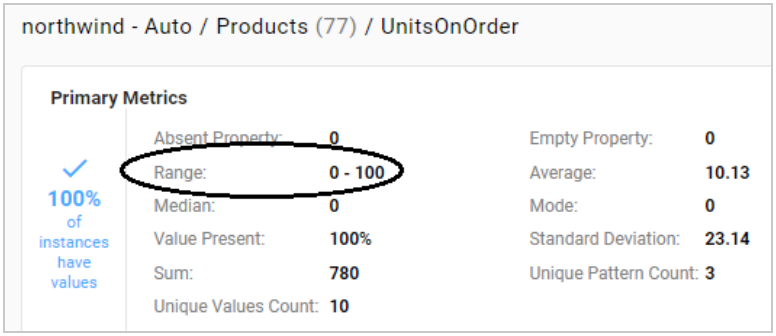
Bottom Pattern Counts

For each property, this metric computes the *N* least common value patterns.



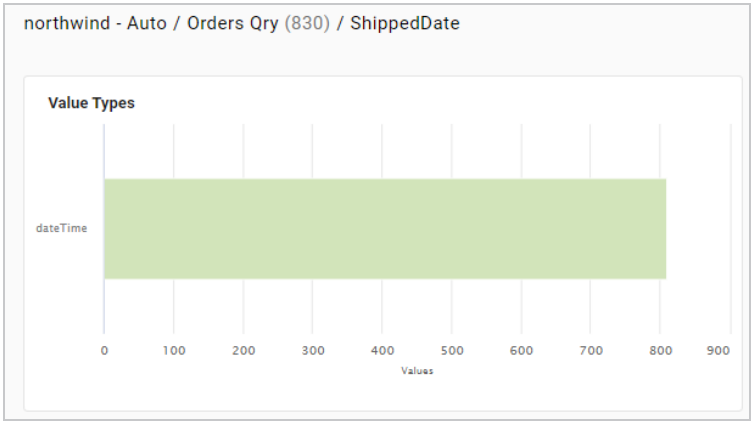
Range

For numeric properties, this metric computes the range of all values.



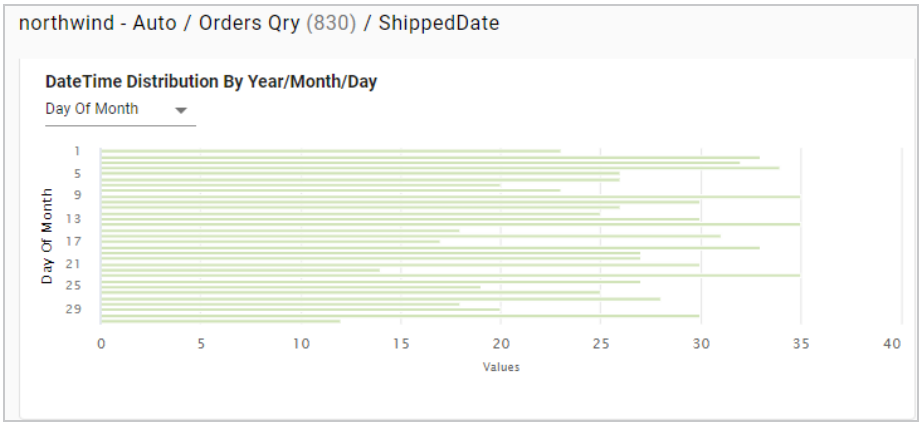
Value Types

For each property, this metric returns the data types for the instances.



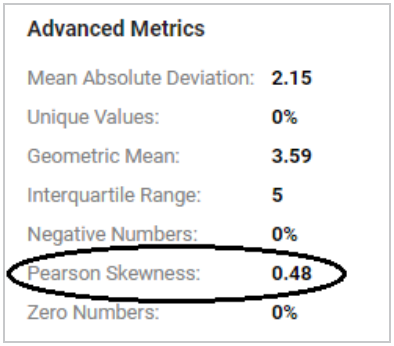
DateTime Distribution By Year/Month/Day

For each dateTime property, this metric computes a histogram that shows the distribution of values by year, month, and day.



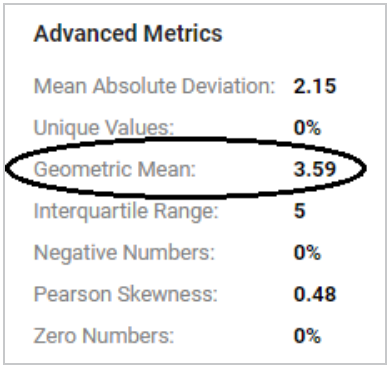
Pearson Skewness

For each numeric property, this metric computes the Pearson coefficient of skewness to show the distribution of values. A value of 0 indicates no skew, a positive number indicates positive skew, and a negative number indicates negative skew.



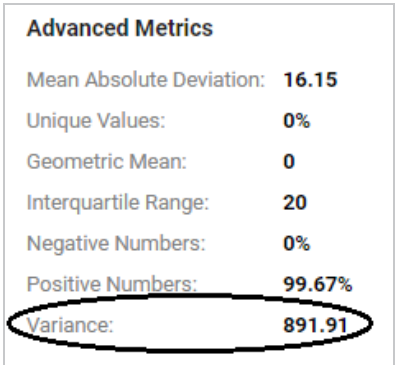
Geometric Mean

For each numeric property, this metric computes the geometric mean of all of the values.



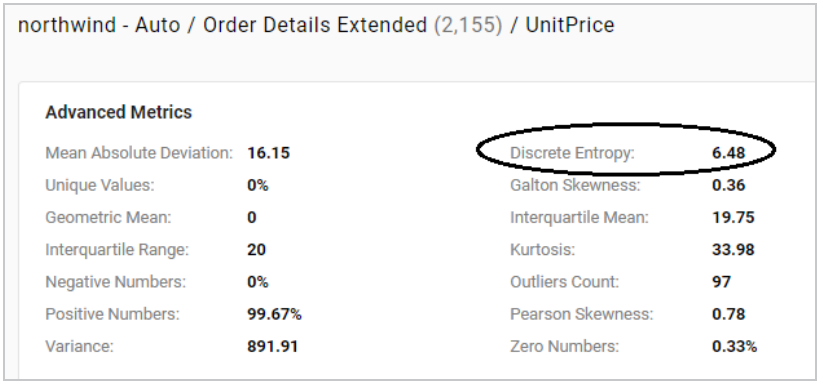
Variance

For each numeric property, this metric computes the variance of all values.



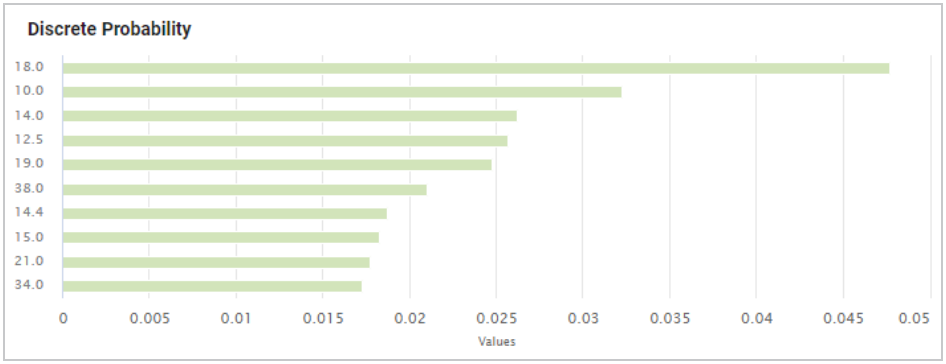
Discrete Entropy

For each property, this metric computes the discrete entropy of all values.



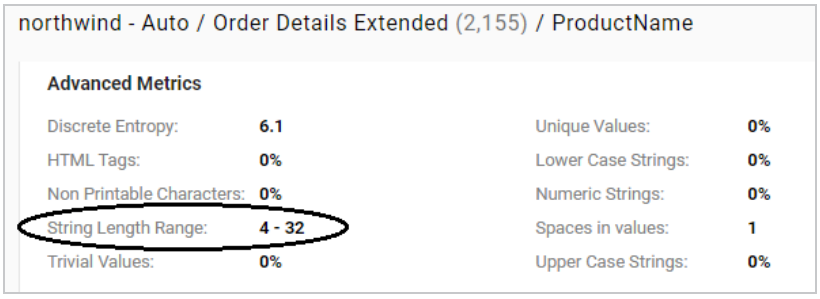
Discrete Probability

For each property, this metric computes the discrete probability of all values.



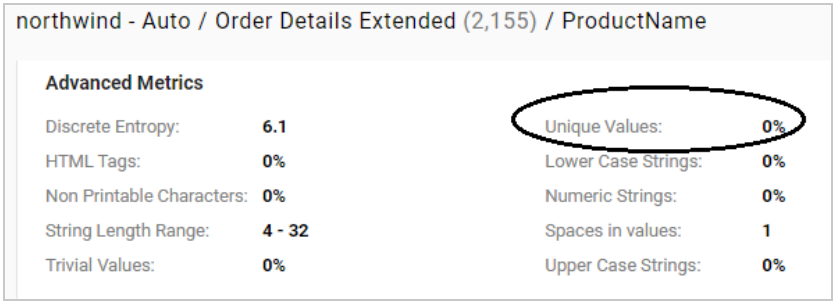
String Length Range

For each string property, this metric reports the range of string value lengths.



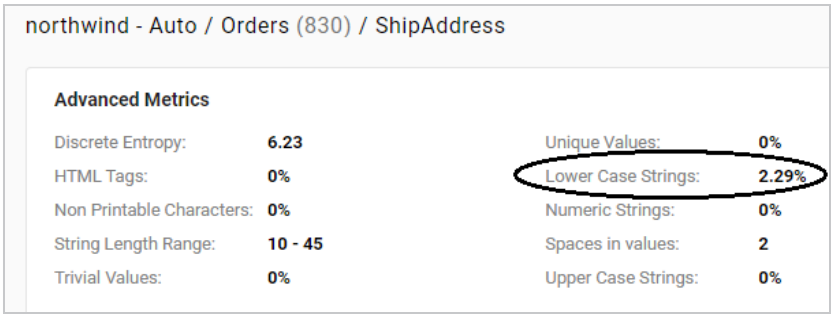
Unique Values

For each property, this metric computes the percentage of unique values.



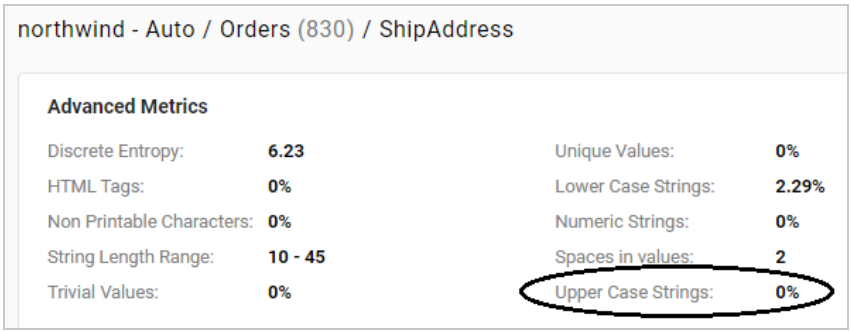
Lower Case Strings

For each string property, this metric computes the percentage of values that have all lower case characters.



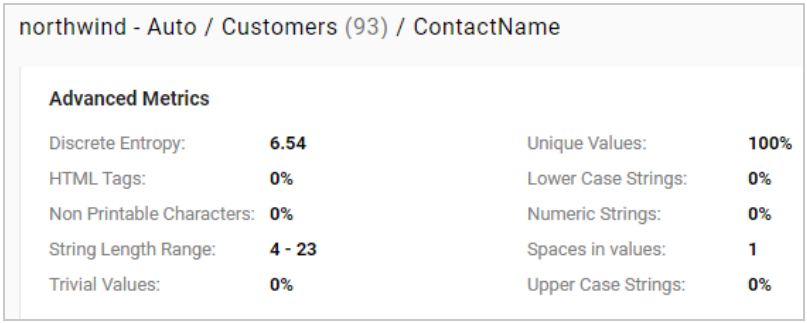
Upper Case Strings

For each string property, this metric computes the percentage of values that have all upper case characters.



Trivial Values

For each string property, this metric computes the percentage of instances that have one of the following values: **NA**, **N/A**, **NONE**, or **NULL**.



For additional metrics based on the type of data quality checks needed, contact Cambridge Semantics.

Access & Analyze

Once data has been onboarded, users have several options for accessing and analyzing the data. Anzo includes the Hi-Res Analytics application where users can create dashboards for exploring and visualizing the data without needing to have specialized query knowledge. The Query Builder enables users to find specific statements or run SPARQL queries. Users can also access data from the SPARQL endpoint or by using the Data on Demand service to generate data feeds for third-party business intelligence tools. The topics in this section provide information about the ways to access data in Anzo as well as information about sharing, versioning, and migrating artifacts.

In this section:

Access Data with Hi-Res Analytics Dashboards 628

Access Data with the Query Builder 906

Access Data on Demand Endpoints 921

Access the SPARQL Endpoint 953

Access the HTTP Client Interface 965

Share Access to Artifacts 973

Version and Migrate Artifacts 980

SPARQL Best Practices and Query Templates 996

Function and Formula Reference 1010

Access Data with Hi-Res Analytics Dashboards

The Anzo Hi-Res Analytics application enables users to answer both ad-hoc and pre-determined questions using custom dashboards. Automated query generation eliminates the need to have specialized query knowledge, and users can traverse complex, multi-dimensional data by building exploratory charts, filters, tables, and network views.

The topics in this section provide guidance on getting started with Hi-Res Analytics dashboards and include instructions for creating and modifying dashboards and dashboard components.

In this section:

Introduction to Hi-Res Analytics 629

Getting Started: Explore and Visualize Your Data 637

Working with Dashboards 648

Working with Lenses 697

Working with Filters 827

Calculating Values in Lenses and Filters 881

Combining Data from Multiple Classes 887

Searching for Text in Unstructured Documents 892

Sharing Access to Dashboards and Lenses 899

Introduction to Hi-Res Analytics

Hi-Res Analytics dashboards enable you to create visual representations of your data using the latest in powerful web technologies. This introduction defines the fundamental concepts of working with dashboards and provides an overview of the Hi-Res Analytics user interface.

Tip
To fully leverage the advanced capabilities of Hi-Res Analytics, it helps to have skills working with Excel functions and formulas, SPARQL, and JavaScript and HTML. You can create dashboards without these skills but may not be able to take advantage of all functions.

- Concepts and Vocabulary
- Application Overview

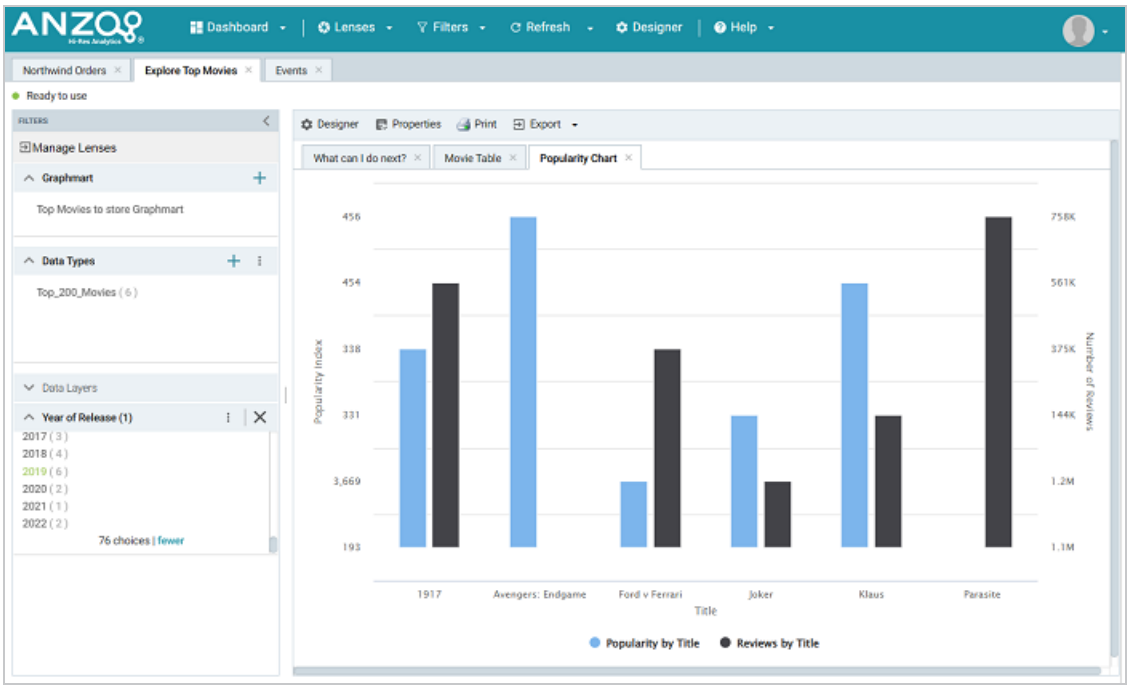
Concepts and Vocabulary

Term	Description
Dashboard	Dashboards are containers for the elements that enable you to visualize, analyze, and share your data. Data is viewed through lenses, such as tables and charts, that format the data for display. You can also apply filters to the data to refine the results. There are two types of dashboards that you can create: a **Graphmart Dashboard** that offers several choices of lens and filter types, and a **Network Navigator Dashboard**, which is an interactive graph visualization tool for exploring the relationships in a graph.
Data Layer	Since graphmarts typically have multiple data layers, users can include or exclude the data from certain layers when creating or viewing dashboards.
Lens	Lenses are the structures that display your data. You must have at least one lens in a dashboard. You can reuse existing lenses or create new ones. For more information, see Creating a Lens.

Term	Description
Filter	Filters narrow and further define the data to display. Dashboard-level filters apply globally to all lenses in a dashboard. Lens-level filters apply only to a specific lens and are not displayed on the lens. They are shown only in the lens designer. For more information, see Working with Filters.
Property	A property is a predicate that contains the instance data to display. The data type of a property determines the functional aspects within a dashboard. For example, certain filters act only on dates or numbers.
Path	Paths are relationships. They are transitional elements that allow you to connect data across classes.
Functions and formulas	Functions and formulas can be applied to properties to modify the data that is presented. Available functions depend on the property's data type. For more information, see Calculating Values in Lenses and Filters.

Application Overview

This section gives an overview of the user interface. The images below show an administrator view. Some options are not available to users with lower permission levels.



Main Toolbar

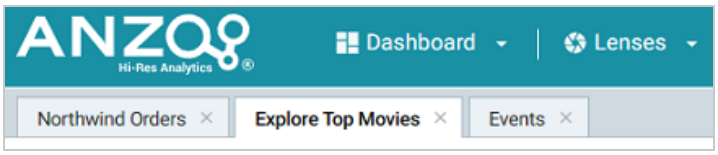
The toolbar at the top of the screen provides the following options:

- **Dashboard**: This menu includes options to save or reset the dashboard and create a new dashboard or open an existing one. This menu also provides access to the permission settings, properties, and the option to delete the dashboard.
- **Lenses**: This menu includes options to create a new lens or open an existing one.
- **Filters**: This menu includes options to create a new filter, clear selected filters, and show all active filters.
- **Refresh**: This menu includes an **Automatic** option that controls whether the dashboard is automatically updated when the underlying graphmart data changes. In addition, the menu includes a **Show Update Controls** option that, when enabled, adds pause icons to lenses and filters so that users can pause the refresh of individual components and then manually update them when desired.
- **Designer**: This menu opens the dashboard designer, which includes settings that control dashboard layout and design as well as the update method to use for the overall dashboard.

- **Help**: This menu includes options to open the Progress window as well as the Query Manager and the documentation.
- **User**: This menu presents the option to log out of the application.

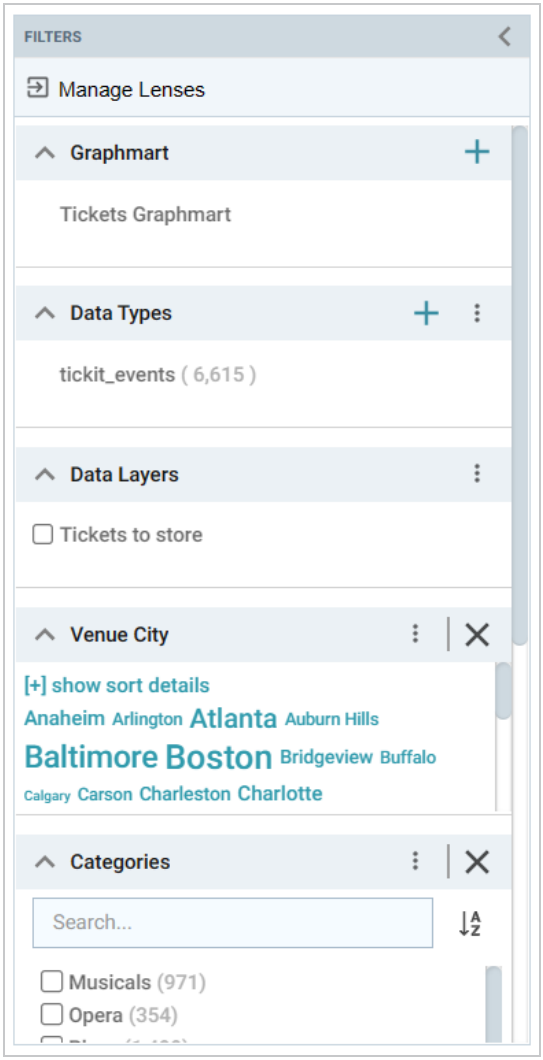
Dashboard Tabs

The dashboard tabs under the main toolbar display the open dashboards and enable you to navigate between dashboards. When you change a dashboard, an asterisk appears on the dashboard tab. Save the dashboard to preserve the changes.



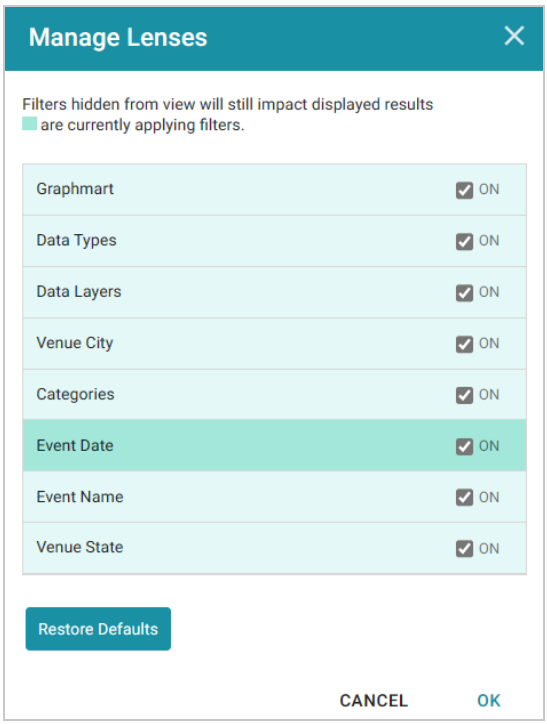
Filters Panel

The left panel (shown below) contains the dashboard filters. Each of the panels are described below.



Manage Lenses

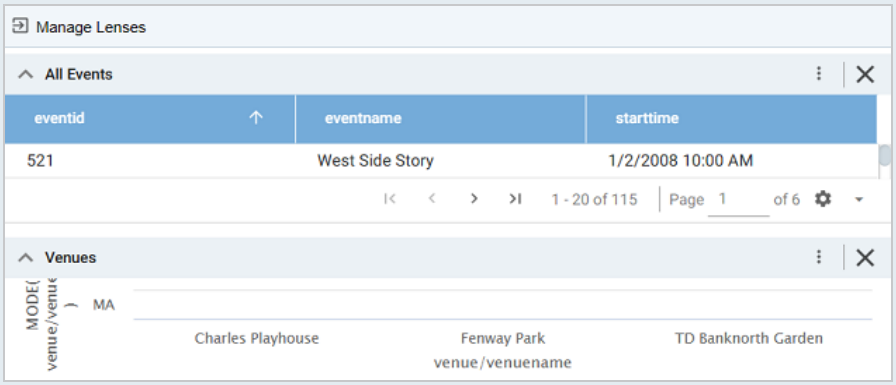
Clicking **Manage Lenses** opens the Manage Lenses dialog box, which enables you to show and hide the items in the panel. For example:



Clearing an **ON** checkbox hides that item in the panel. You can select a checkbox to show an item or click **Restore Defaults** to show all items.

Tip

When the dashboard layout is a **Vertical List Container**, there is also a **Manage Lenses** button available for showing and hiding lenses. For example:



Graphmart

The Graphmart panel displays the selected graphmart for the dashboard:



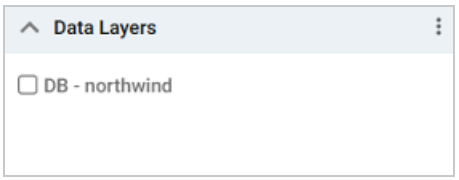
Data Types

The Data Types panel displays the selected data type from the graphmart.



Data Layers

The Data Layers panel displays the layers in the graphmart.



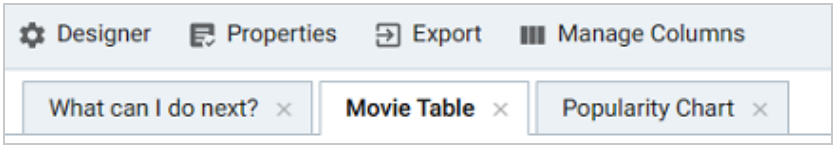
Filters

By default, filters that you create appear in the left column of the dashboard.



Object Toolbar and Tabs

The object toolbar and tabs enable you to manage the lenses and filters in the selected dashboard. The tabs display the open lenses, and the toolbar enables you work with the lens configuration.


Designer
Properties
Export
Manage Columns
What can I do next?
Movie Table
Popularity Chart

Getting Started: Explore and Visualize Your Data

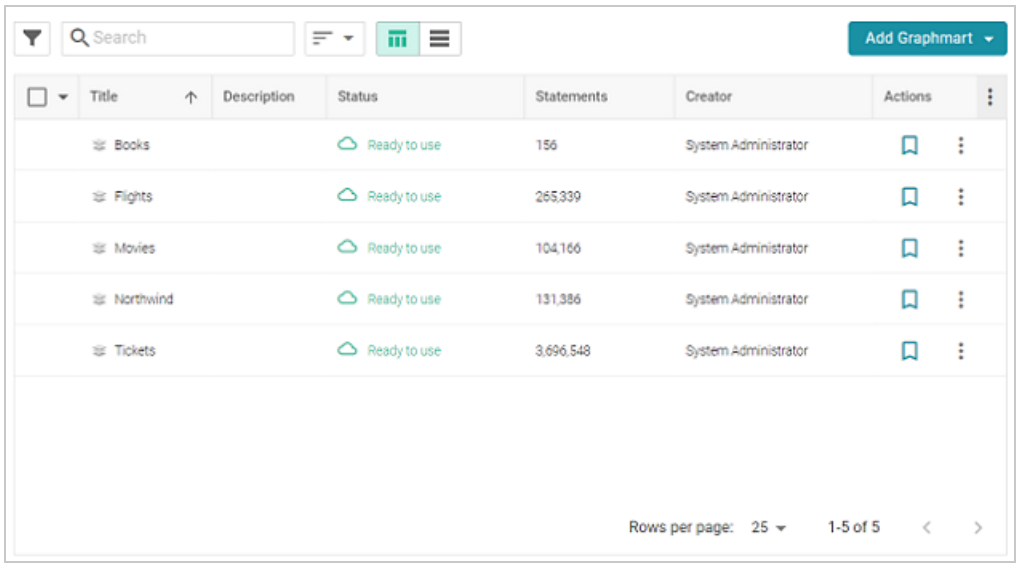
When you start to build a new dashboard, you might not know what data exists in the knowledge graph, which values in graph you ultimately want to display, and the most pertinent way to visualize the results. This topic introduces the available lenses and filters and provides guidance on getting started by using the Hi-Res Analytics tools to perform data discovery. By experimenting with simple objects, you can explore the data, determine which questions you want to answer, and start to visualize the end result.

To get started:

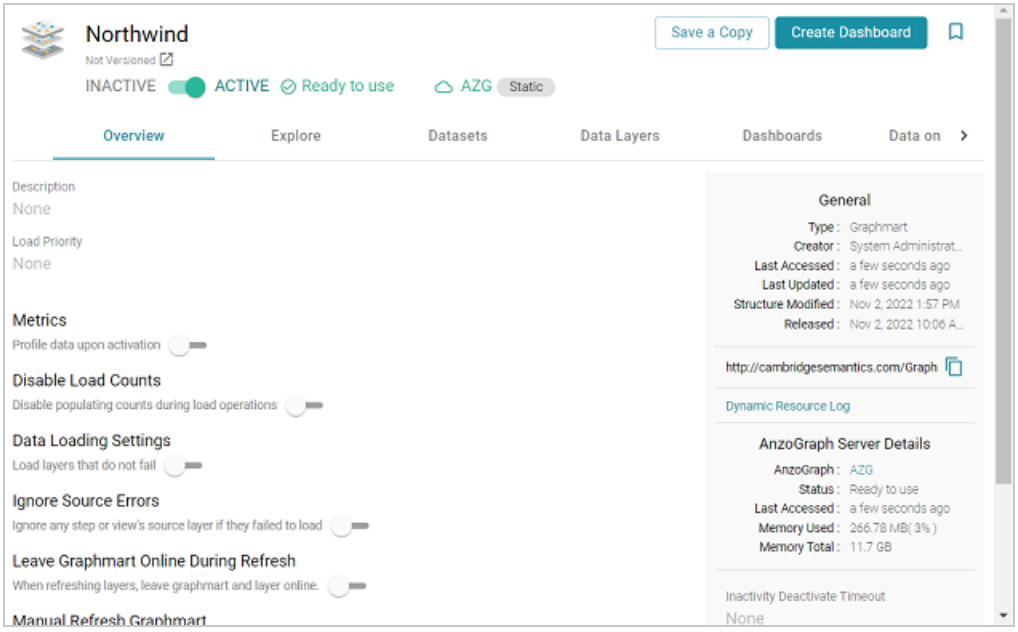
1. Create a New Dashboard
2. Explore the Data
3. Create Visualizations of the Data

Create a New Dashboard

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:

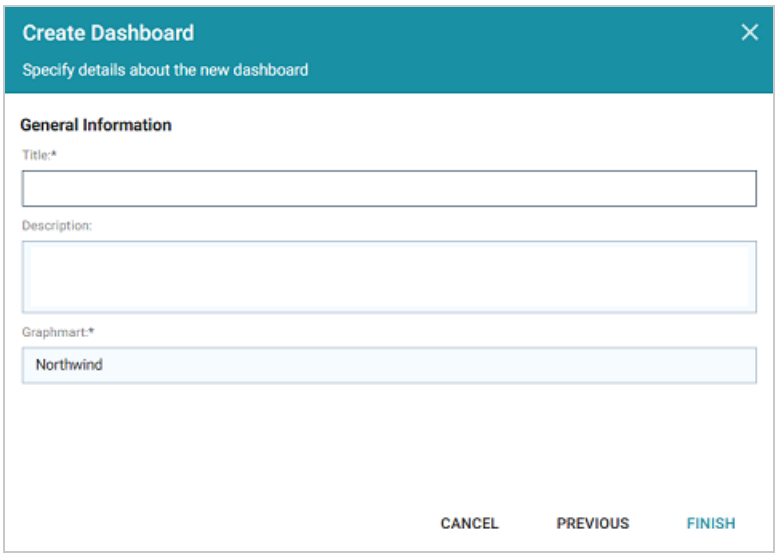


2. On the Graphmarts screen, click the name of the graphmart for which you want to create a dashboard. The Overview screen is displayed. For example:

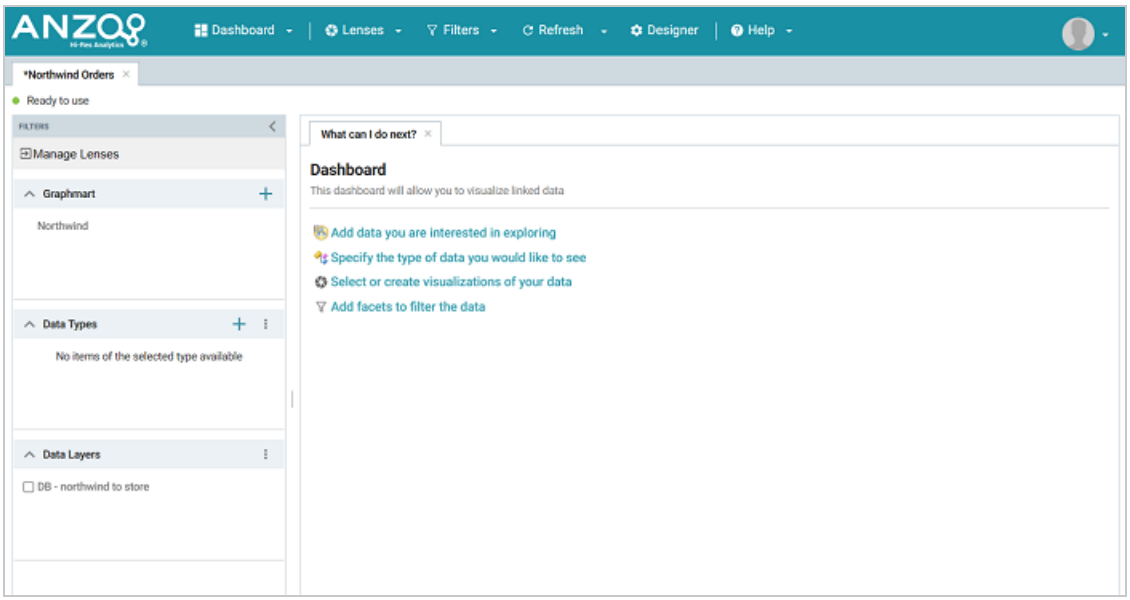


3. Click the **Create Dashboard** button. The Hi-Res Analytics application opens and displays the Create Dashboard dialog box. Leave **Graphmart Dashboard** selected and click **Next**.

4. Next, type a name for the dashboard in the **Title** field and enter an optional **Description**.



5. Click **Finish** to create the dashboard. The new dashboard appears as a new tab and contains a sub-tab titled **What can I do next?**. This tab acts as a wizard to guide you through the initial dashboard creation. For example:



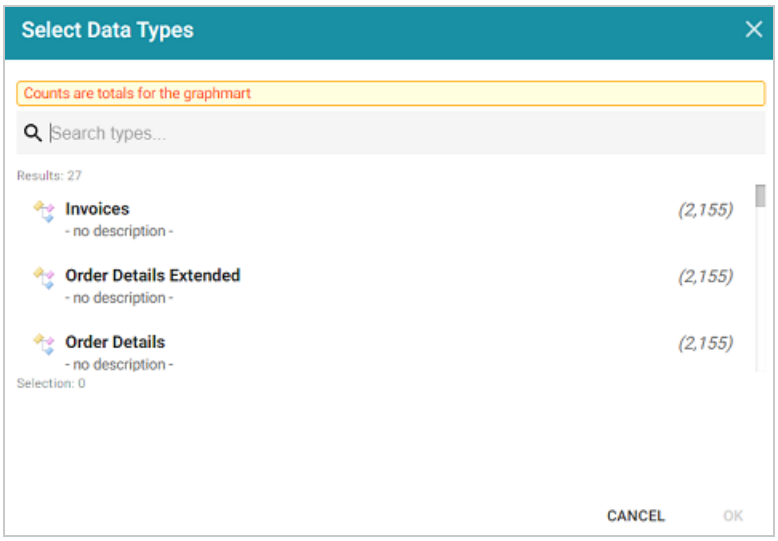
6. In the main toolbar, click the **Dashboard** button and select **Save**. Proceed to Explore the Data below for guidance on next steps.

Explore the Data

Once you create a new dashboard, you can experiment with Hi-Res Analytics tools to get to know the data and decide the best way to display it.

Decide What Type (Class) of Data You Want to See

1. First, review the types of data or classes that exist in the data. On the What can I do next? tab, click **Specify the types of data you would like to see**. The Select Data Types dialog box displays the available data types. The value in parentheses shows the total number of instances of that type exist in the data set:



2. Select one data type. The property that you choose determines the fields that become available to filter on.

   > **Tip**
   > Though you must choose one base data type for a dashboard, you can leverage the relationships in the graph to access and integrate data from additional classes. See Combining Data from Multiple Classes for more information.

   Click **OK** to close the Select Data Types dialog box. The data type is added to the Data Types panel on the left side of the screen. Proceed to Create Filters to See the Values for Properties below for next steps.

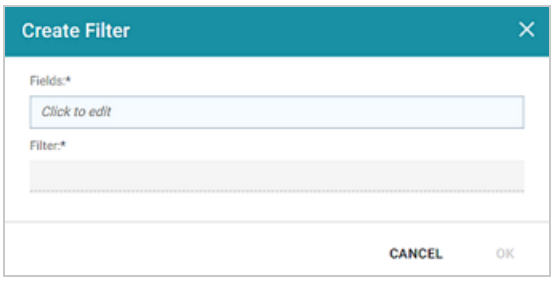
> **Tip**
> You might want to create multiple dashboards so that you can click between dashboards and view multiple classes of data at the same time.

Create Filters to See the Values for Properties

To dive deeper into the data and quickly determine what values exist for the class of properties you selected, you can start adding filters to the dashboard. Filters reveal the values associated with fields and help you learn the data set specifics such as whether data exists for certain properties

and whether the data includes many duplicate or unique values. Learning more about the details enables you to start making decisions about what properties to group on, what properties have relationships, and what results you want to visualize on the dashboard.

1. To create a filter, click **Add facets to filter the data** on the What can I do next tab. The Create Filter dialog box opens.



2. In the Create Filter dialog box, click the **Fields** field and select the property or property path to filter on.
3. Then click the **Filter** field to select the filter type. The list of available choices depends on the data type of the property you selected in Fields. The table below describes each filter type.

Filter Types

Filter	Description
Cloud	Cloud filters display values in term clouds where each term is written in a font size that represents the number of results for that value. Unlike list filters, which enable you to select and filter on multiple values at once, cloud filters allow you to filter on one value at a time. The Cloud filter is available for all data types.
List	List filters display values in a list and allow you to filter on multiple values at the same time. List filters are available for all data types.
Single Select List	Single Select List filters are similar to List filters but only allow you to filter one value from the list at a time. This type of filter is available for properties of all data types but is not available for paths.

Filter	Description
Limit	Limit filters are used to limit the results on the dashboard to a specified number of either the largest or smallest values. The Limit filter is available for any data type. For strings, results are ordered alphabetically. "Largest" orders by the last letters in the alphabet and "Smallest" orders by the first letters in the alphabet.
Date Range	Date Range filters are used to limit the results on a dashboard to data that falls in (or outside of) certain date and time groupings. Date Range filters are available for properties with date, dateTime, and time data types.
Numeric Range	Numeric Range filters are used to limit the results on a dashboard to data that falls in (or outside of) certain numeric groupings. Numeric Range filters are available for properties with integer and double data types.
Range Slider	Range Slider filters display a slider control that enables you to filter dashboard results by setting one range that you can adjust as needed. This type of filter is available for properties with integer, double, date, time, and dateTime data types. It is not available for paths.
Relative Time	Relative Time filters are used to filter for records that fall into a specified increment of time relative to the current time. This type of filter is available for date, time, and dateTime data types.
Search	Search filters are used to search for values of a property that contain a partial match, exact match, or do not equal the text that you specify. The search is case-insensitive. This type of filter is available for all data types. It is not available for use with paths.
Presence	Presence filters group results based on whether the value exists or does not exist. This type of filter is useful for testing whether there are records that are missing a particular value. Presence filters are available for paths and properties of all data types.

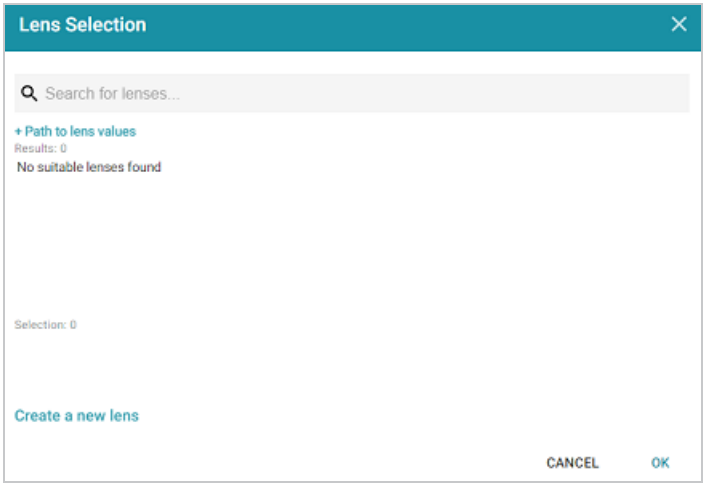
Filter	Description
Quartile	Quartile filters group and rank the values for a property into four equal ranges. This filter is available for properties with integer, double, date, time, and dateTime data types. It is not available for paths.
Hierarchy	If hierarchies exist in your knowledge graph, you can create a Hierarchy filter to explore the parent and child relationships and filter the dashboard based on the relationships. Unlike the majority of dashboard filters, where you select a property to filter on, Hierarchy filters operate on relationships and are only available as a filter type when you select a path to filter on.
Types	Types filters are used to filter data according to the types of data (classes) that are connected by a specified path. This type of filter is available only for paths and not properties.

For information about configuring each type of filter, see Working with Filters.

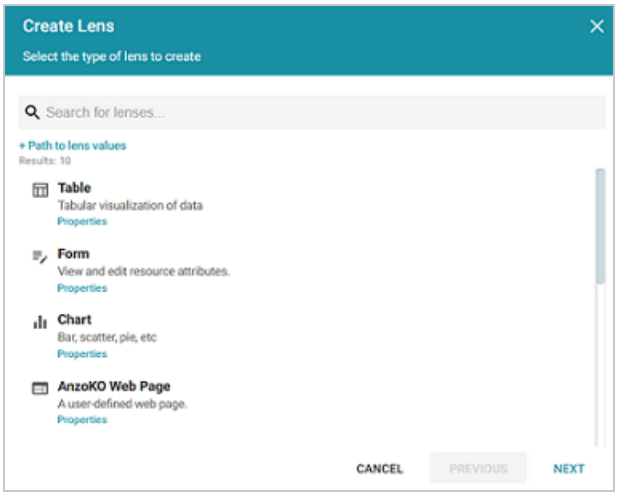
Create Visualizations of the Data

Once you have a good understanding of the values and relationships that exist in the data , you can experiment with the Hi-Res Analytics lenses and decide on the most appropriate way to display the data. Creating a Table lens is a quick way to view the data that you filtered. This section provides instructions for creating a table lens and describes each of the lenses available in Hi-Res Analytics.

1. To create a Table lens, click **Select or create visualizations of your data** in the What can I do next tab. Anzo displays the Lens Selection dialog box.

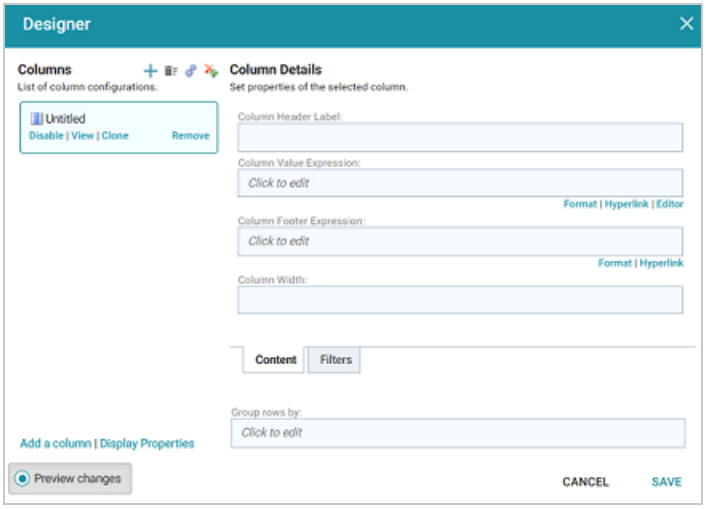


2. In the dialog box, click **Create a new lens**. Anzo displays the Create Lens dialog box.



3. Select the **Table** lens and click **Next**.

4. Type a **Title** for the lens, and then click **Finish**. Anzo opens the Table Designer:



5. In the Designer, click the **Auto-generate columns** icon ( ) to add all available columns to the table. Then click **Save**.

The new lens displays as a new sub-tab on the dashboard and displays the data according to the data type and filter or filters that you created. Now that you can view a summary of the data in a table, it can help you determine how to further narrow or expand the results by adding, changing, or removing filters. In addition, you can experiment by adding other lenses to the dashboard to find the ideal way to display the data to answer the questions that you have. The table below describes each type of lens. For more information about each lens, see Creating a Lens.

Lens	Description
AnzoKO Web Page	AnzoKO Web Page lenses include the Knockout JavaScript framework and enable you to display data on a web page that you create using HTML, CSS, and JavaScript.
Chart	Anzo offers several types of chart lenses. These lenses are useful for displaying large amounts of complex data and have the widest format range of any lens type. The ability to add an axis enables you to compare data, such as for comparing monthly sales data for multiple stores.
Drill Down	Drill down lenses combine other lenses into a hierarchical interface. Clicking an

Lens	Description
	object in one lens opens the next lens in the hierarchy and can present the data in a different view.
Form	Form lenses enable you to create an editable or read-only form on the dashboard. Creating forms can be useful for displaying many details about each record instead of using a table where the large number of columns makes the data hard to read. **Note** By default, only the sysadmin user has access to create Form lenses. In addition, Form lenses are valid in only in Linked Dataset dashboards. For more information, see Creating a Form Lens.
List	List lenses display results in a list layout, similar to the Microsoft Windows® Explorer interface. The lens enables you to add icons for each data value, and results are grouped onto pages according to the given page size value.
Query	Query lenses enable you to retrieve data using a custom SPARQL query and display the results by writing basic HTML and CSS. You can use a Query lens to access data from external sources. Query lenses do not bind directly to the linked data set, data type, or filters defined on the dashboard.
Resource Tree Navigator	Resource Tree Navigator lenses display results in a hierarchical tree view. You can click parent data points to open the successive child data points. This lens is useful for presenting small amounts of data; each discrete group appears on a separate page in the dashboard. You can also click certain objects to view the object's data properties in the left panel. **Note** By default, the only user who has permission to create a Resource Tree Navigator lens is the **sysadmin** user.

Lens	Description
Table	Table lenses present results in a basic table grid consisting of rows and columns. Table lenses are useful for presenting data aggregates or summaries.
Web Page	Web Page lenses enable you to display results on a web page that you create using HTML, CSS, and JavaScript.

Working with Dashboards

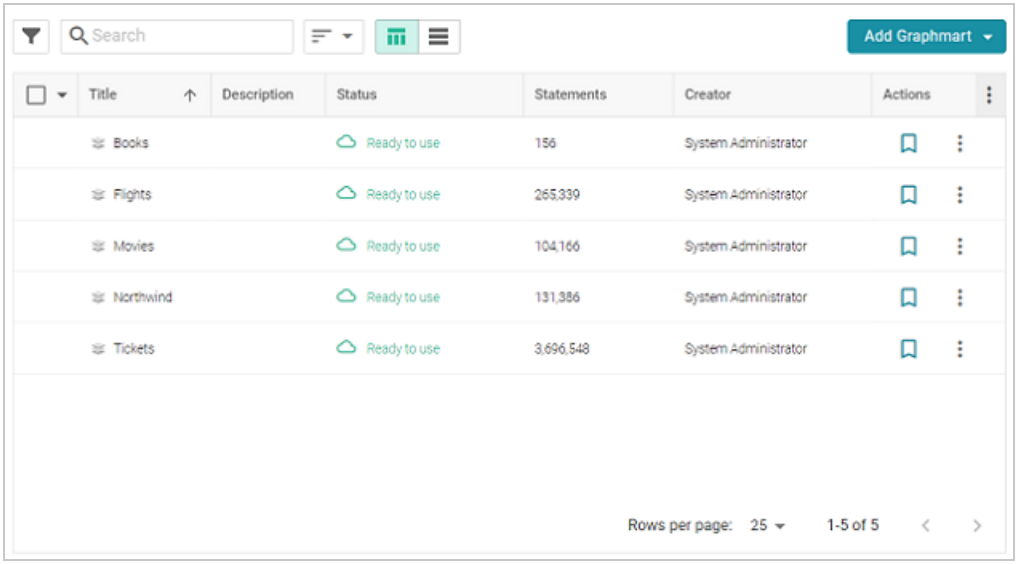
The topics in this section provide instructions on creating and configuring dashboards.

- Creating a Graphmart Dashboard
- Creating a Network Navigator Dashboard
- Configuring a Dashboard to Update in Batch Reporting vs. Interactive Mode
- Capturing User-Defined Values in Dashboards

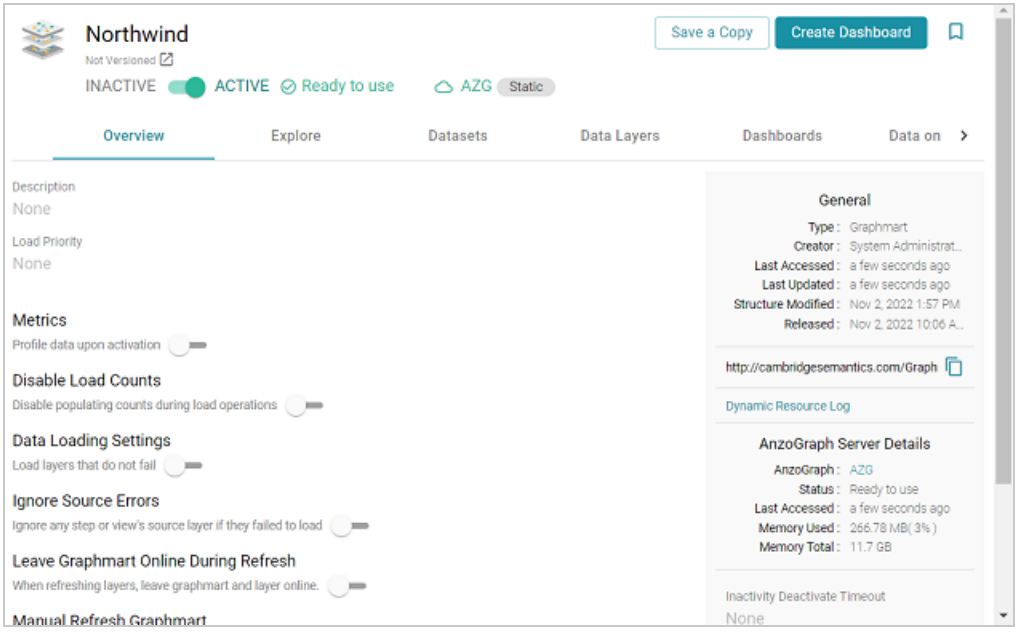
Creating a Graphmart Dashboard

Follow the instructions below to create a new dashboard for a graphmart. For instructions on creating a Network Navigator dashboard, see Creating a Network Navigator Dashboard.

1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:

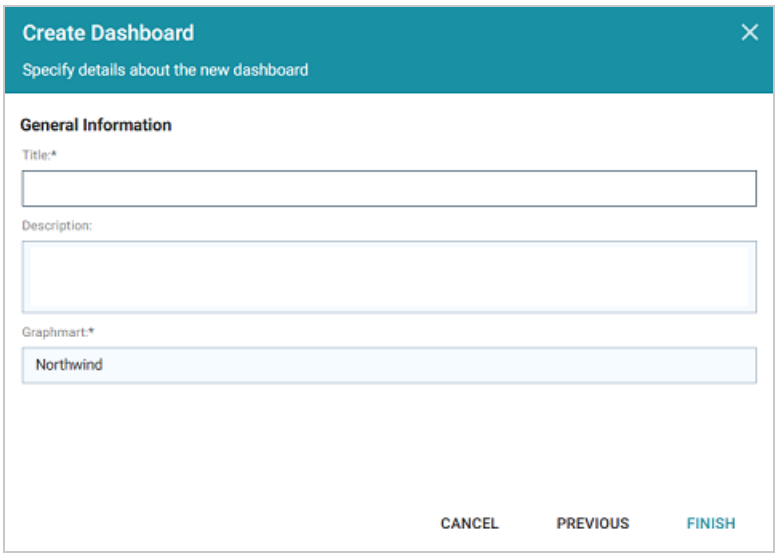


2. On the Graphmarts screen, click the name of the graphmart for which you want to create a dashboard. The Overview screen is displayed. For example:

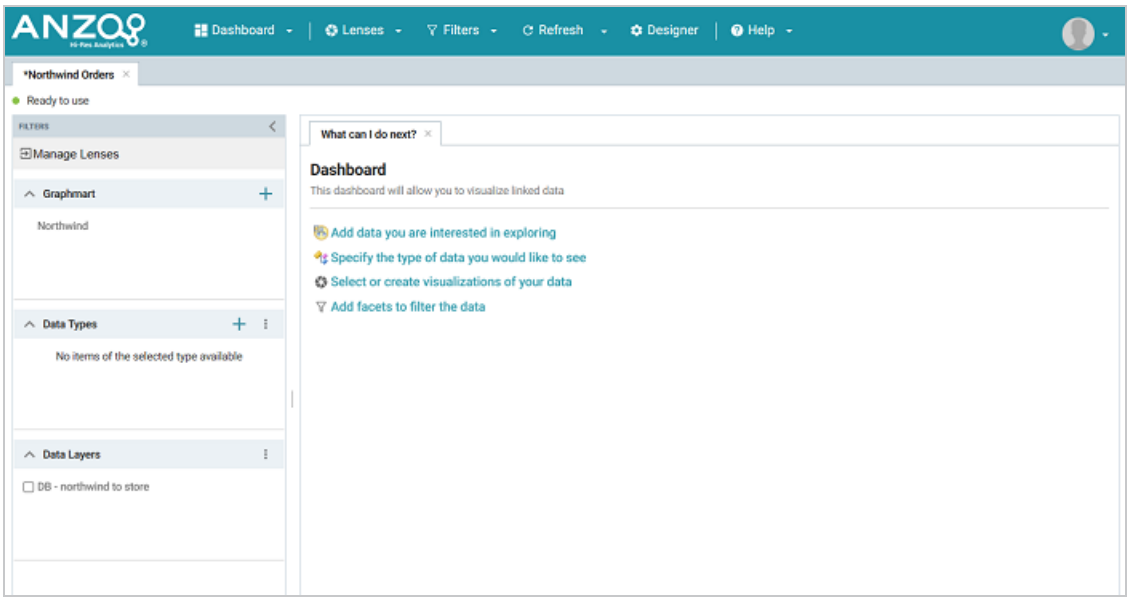


3. Click the **Create Dashboard** button. The Hi-Res Analytics application opens and displays the Create Dashboard dialog box. Leave **Graphmart Dashboard** selected and click **Next**.

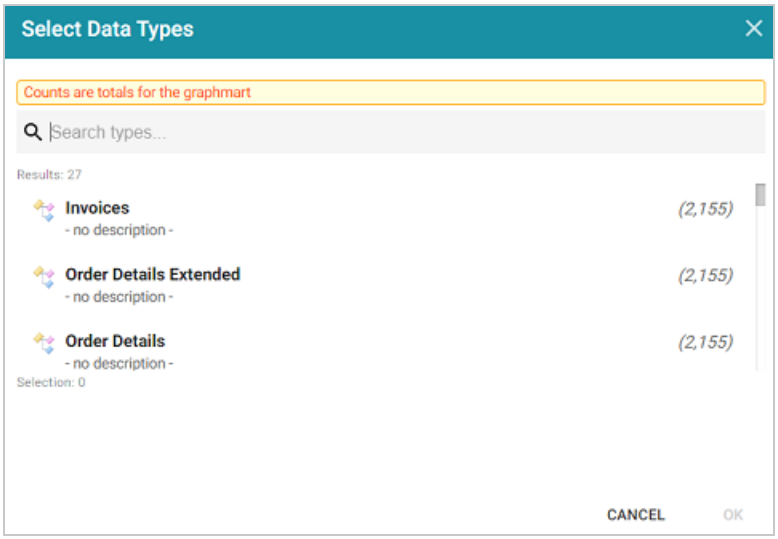
4. Next, type a name for the dashboard in the **Title** field and enter an optional **Description**.



5. Click **Finish** to create the dashboard. The new dashboard appears as a new tab and contains a sub-tab titled **What can I do next?**. This tab acts as a wizard to guide you through the initial dashboard creation. For example:



6. On the What can I do next? tab, click **Specify the types of data you would like to see**. The Select Data Types dialog box displays the available data types. The value in parentheses shows the total number of instances of that type exist in the graphmart:



7. In the Select Data Types dialog box, select the data type or class of data that you want to display on the dashboard. Anzo uses the type, along with any filters, to populate the visualizations (lenses) that you add to the dashboard.

> **Tip**
> Though you must choose one base data type for a dashboard, you can leverage the relationships in the graph to access and integrate data from additional classes. See Combining Data from Multiple Classes for more information.

8. Click **OK** to close the Select Data Types dialog box. The data type is added to the Data Types panel on the left side of the dashboard.
9. In the main toolbar, click the **Dashboard** button and select **Save** to save your progress.

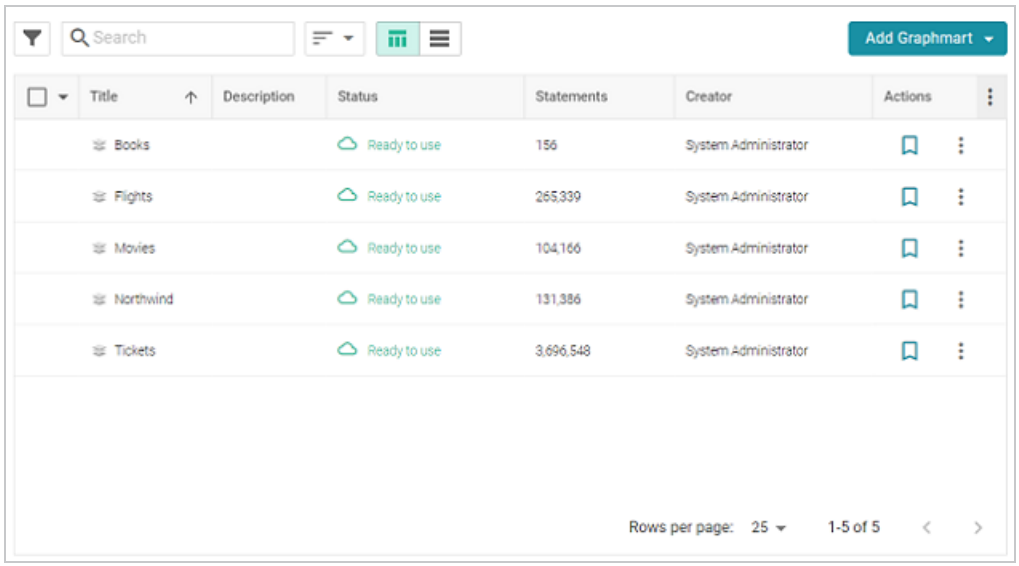
Now that the dashboard basics are defined, see Creating a Lens and Working with Filters for instructions on adding lenses and filters to the dashboard.

Creating a Network Navigator Dashboard

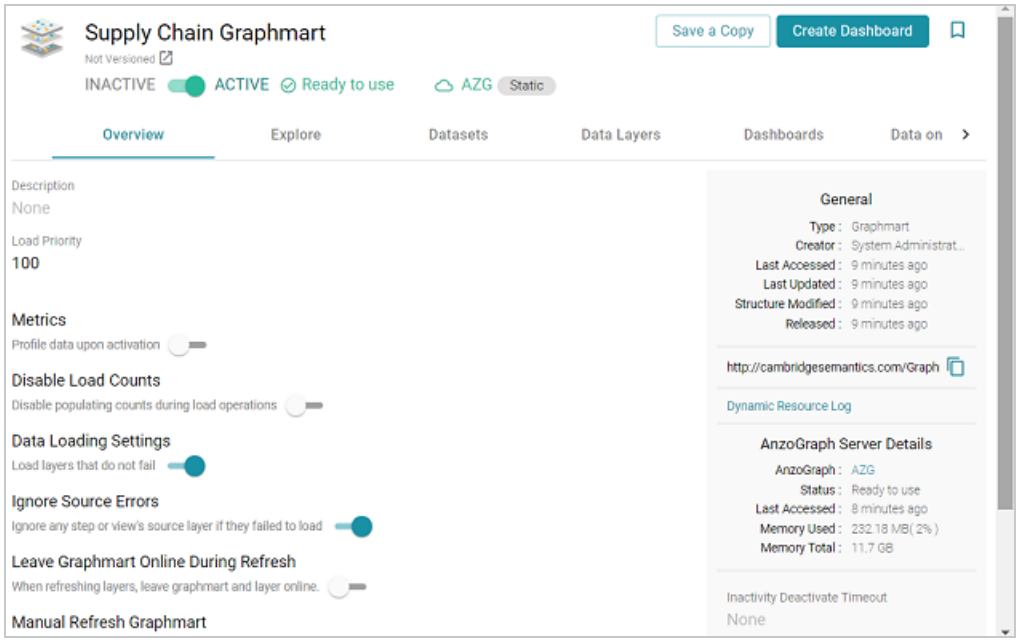
The Network Navigator Dashboard is an interactive graph visualization tool that enables you to find relationships in the knowledge graph and explore the paths to build out a Network View. This topic helps you get started building a Network Navigator Dashboard by covering the basic steps and functionality that is presented by default. Additional topics describe the functionality in further detail and cover more advanced dashboard configuration options. For instructions on creating a Graphmart dashboard, see Creating a Graphmart Dashboard.

> **Note**
> By default, only the sysadmin user has permission to create a Network Navigator Dashboard. However, the sysadmin user can share created dashboards with other users and groups (see Sharing Access to Dashboards and Lenses).

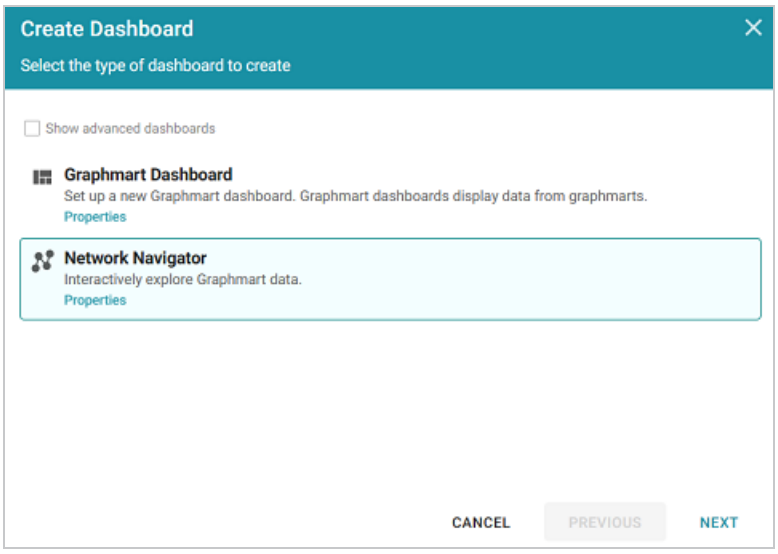
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



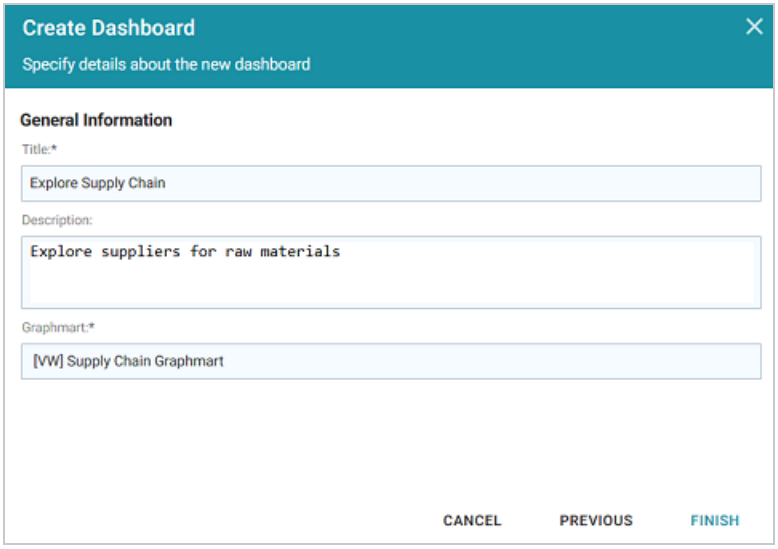
2. On the Graphmarts screen, click the name of the graphmart for which you want to create a dashboard. The Overview screen is displayed. For example:



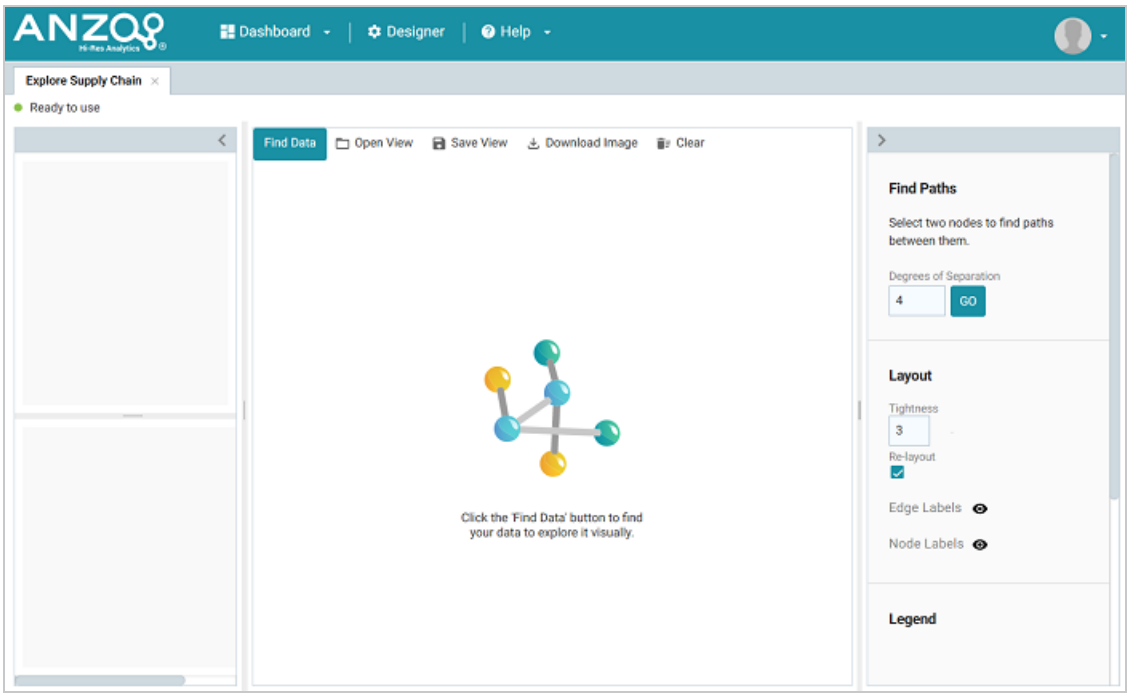
3. Click the **Create Dashboard** button. The Hi-Res Analytics application opens and displays the Create Dashboard dialog box. Select **Network Navigator Dashboard** and click **Next**. Remember that only the **sysadmin** user has the option to create this type of dashboard. If you do not see the option and think you should, check with your system administrator.



4. Next, type a name for the dashboard in the **Title** field and enter an optional **Description**. For example:



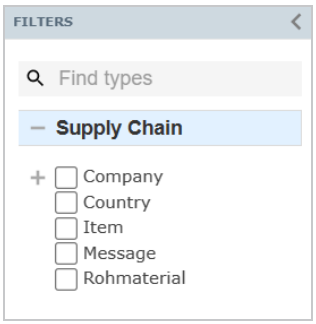
5. Click **Finish** to create the dashboard. The dashboard appears as a new tab in the application. For example:



6. The first step in rendering the Network View is to choose a node (or group of nodes) as the starting point. A node is an object in the data. In order to be able to traverse the network, start with an object for a property that has one or more relationships with other properties. To select a value, click the **Find Data** button at the top of the dashboard. The Find Data dialog box is displayed and lists all of the values for each of the properties in the data model. Select one or more objects in the list.

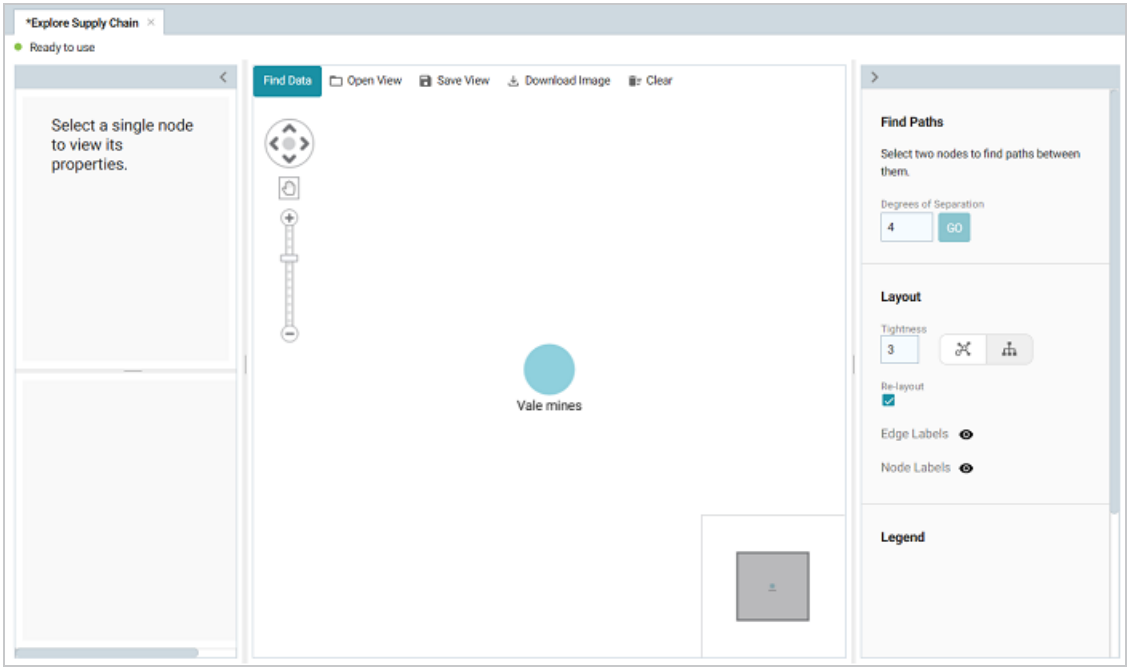


To narrow down the list of values by class or type of data, you can click **Filter** to open the Filter panel on the left side of the screen. The Filter panel lists the classes and subclasses from the model. For example, the image below shows the classes for a knowledge graph that contains supply chain data for an automobile manufacturer:



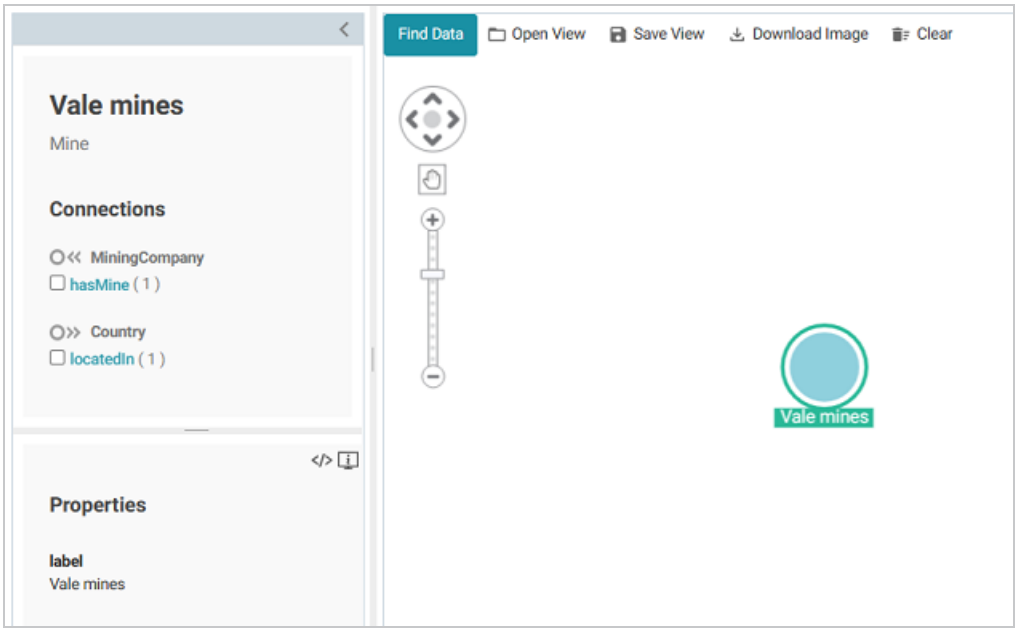
You can expand or collapse the classes to display or hide subclasses. And you can start typing text in the **Find Types** field to search for specific classes. Select the checkbox next to any of the classes from which you want to display data. The list in the main part of the screen is refreshed to display only the values for the properties in the selected classes.

7. Once you have selected the starting node in the Find Data dialog box, click **OK** to add the node to the Network View. For example, in the image below the value **Vale mines** was selected as the starting point, and the Vale mines node is added to the View:



> **Tip**
> You can configure the dashboard to **Auto-Expand** by a certain number of degrees or hops so that adding a node to a View automatically adds the specified number of related nodes and paths. For information, see Auto-Expanding a Network View.

8. To start building out the network, select a node to view its connections and properties on the left side of the screen. For example, selecting the **Vale mines** node shows its incoming and outgoing connections as well as properties like the label.



The image shows that Vale mines has an incoming connection (or backward path o«) from **Mining Company** via **hasMine** and an outgoing connection (or forward path o») to **Country** via **locatedIn**.

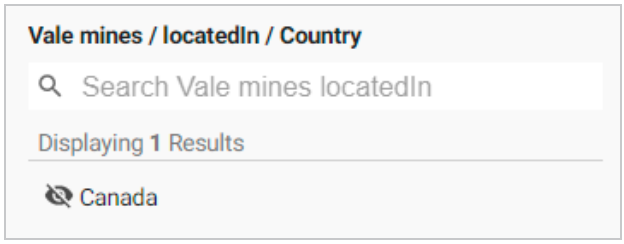
9. There are two options for exploring the connections:

See a list of connected nodes without adding them to the View

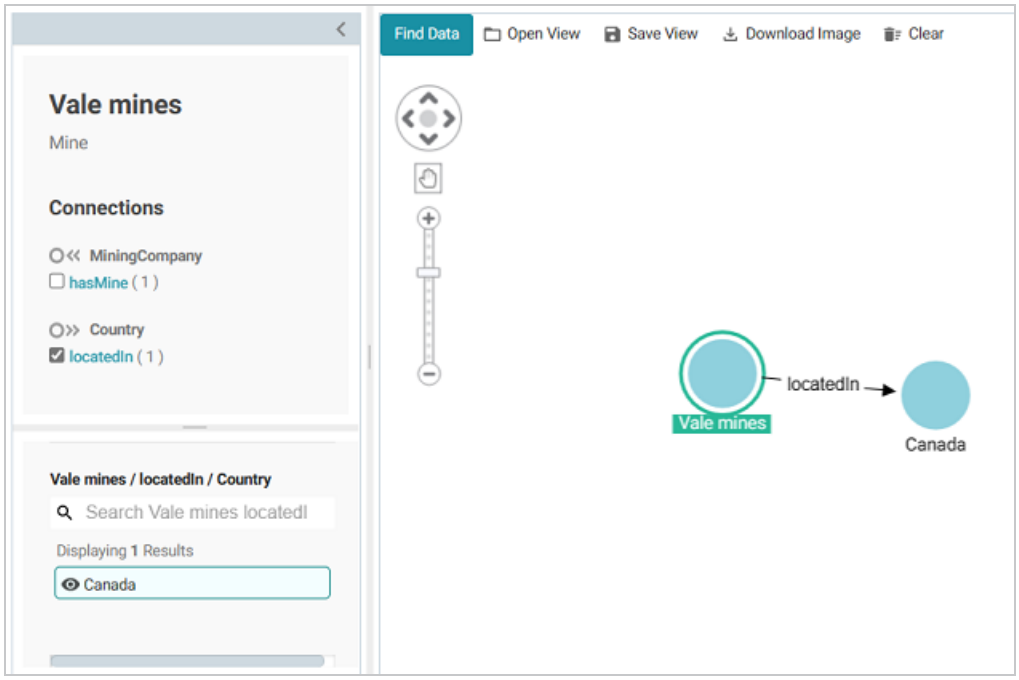
To see a list of nodes that are connected by a path without adding the nodes and path to the View, you can click the path name link.



Details are shown in the lower panel, but the nodes are hidden from the network View by default. For example, clicking the **locatedIn** path from the image above, shows that Vale mines is located in Canada.



If you decide you want a node and path to be added to the View, you can click the hidden node to make it visible. For example, clicking **Canada** adds the path and node to the View:

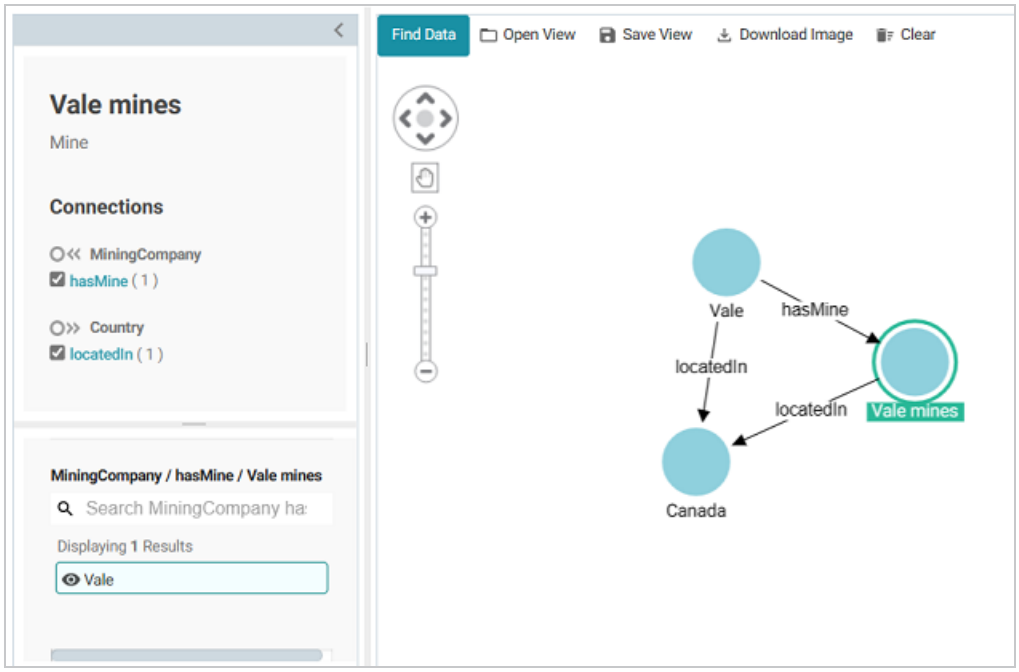


Add a path and node directly to the View

To add a path and node directly to the network View, you can select the checkbox next to the path.



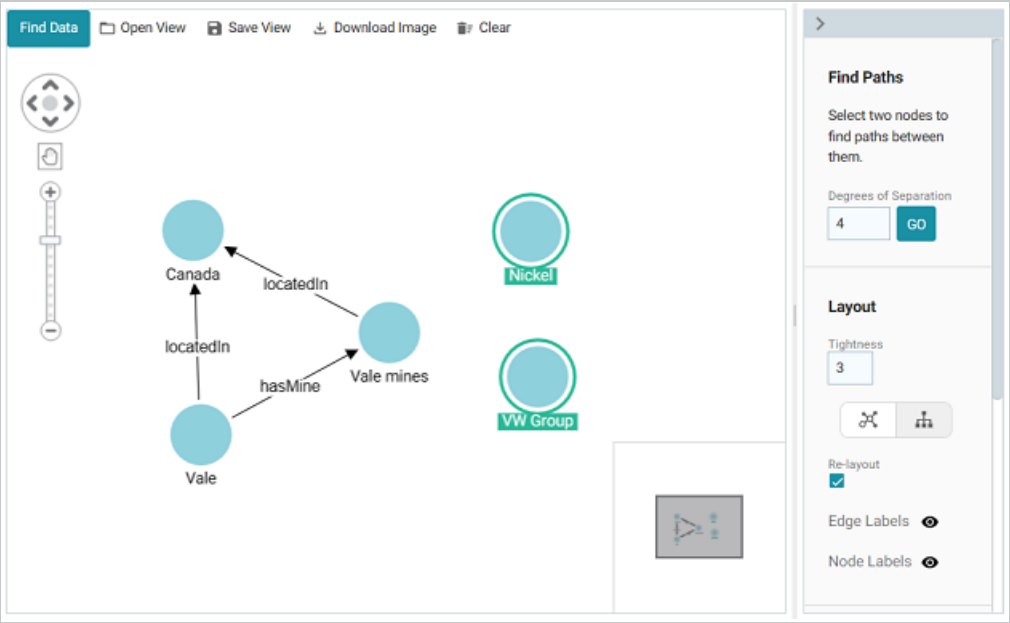
For example, selecting the **hasMine** checkbox adds the path and the MiningCompany value (**Vale**) to the View:



> **Tip**
>
> Any time you want to clear the View and start again, you can click **Clear** at the top of the screen. To toggle between showing and hiding a path or node, you can click the path or node in the panel on the left side of the screen to select or deselect the item.
>
> You can save the View by clicking **Save View** and specifying a Title and optional Description in the dialog box. When you save the dashboard, the View that is visible at the time it is saved is presented when the dashboard is reopened.

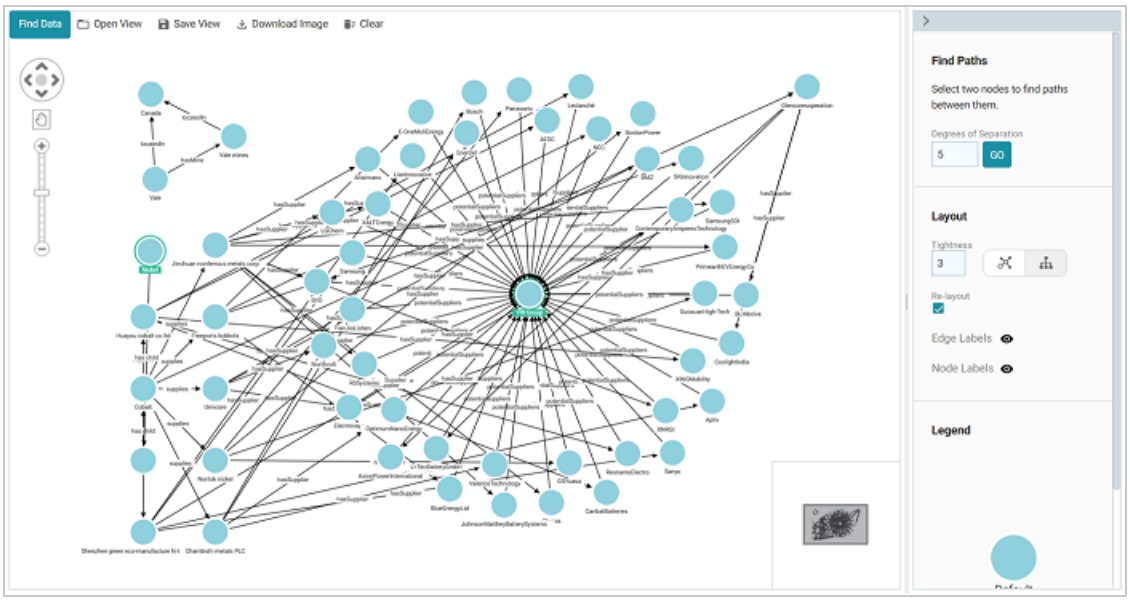
10. To continue to explore and expand the Network View, you can select another node and repeat the previous step to identify additional paths to follow. Or you also have the option to select two nodes and do a search for paths between them. To find paths, find and add to the view, if necessary, the two nodes whose relationships you want to explore. Then hold the **Ctrl** button and select the two nodes.

For example, in the image below, two nodes, an OEM called VW Group, and a raw material, Nickel, were added to the View, and both nodes are selected:



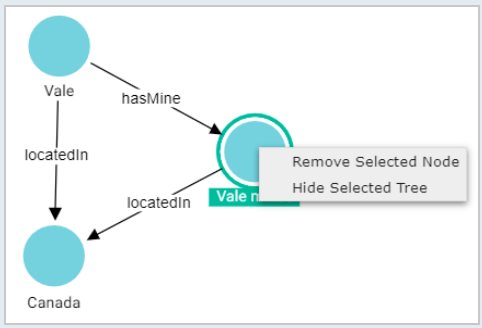
11. On the right side of the screen under **Find Paths**, adjust the **Degrees of Separation** value as needed (the default is 4) and then click **Go**. Anzo runs an All Paths graph algorithm to find all of the paths that exist between the two nodes—where the maximum number of hops is the value in the **Degrees of Separation** field. If a "No paths found" message is returned, that means there is no path between the two nodes that is *N* or fewer hops away, where *N* is the value in Degrees of Separation. You can change the value and try again.

    In the example for VW Group and Nickel, finding paths with the default value of 4 returned "No paths found." However, increasing Degrees of Separation to 5 renders several nodes and relationships.



Tip

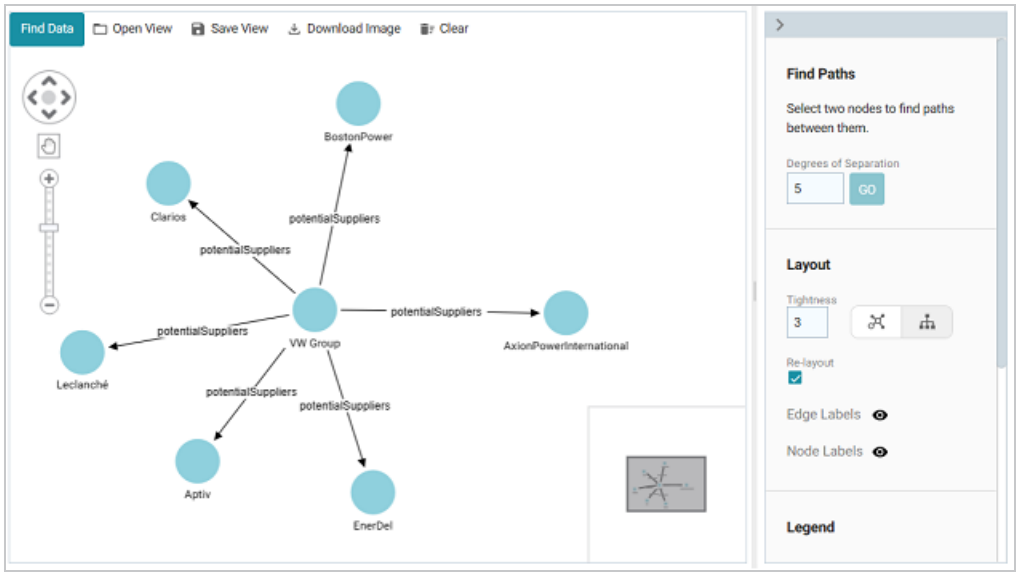
You can remove a single node or a hide a group of nodes by right-clicking a node and selecting an option from the pop-up menu (shown below). **Remove Selected Node** deletes the node and its incoming and outgoing connections. **Hide Selected Tree** hides the selected node and all of its connected nodes and paths.



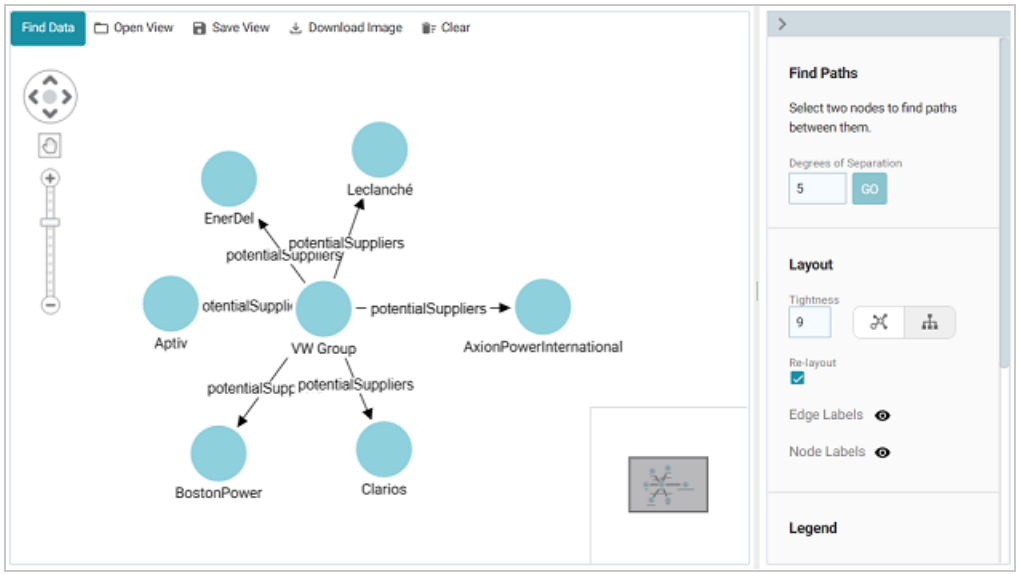
12. You can continue to explore the network using the methods described above. In addition, you can adjust the layout of the View using the options under **Layout** on the right side of the screen:

Tightness

The **Tightness** setting controls how close or far apart the nodes are. Valid values are **1** – **9**. The lower the Tightness value, the looser the nodes are. Increasing Tightness brings the nodes closer together. Type a number in the field and then press **Enter** to apply it. In the simple example below, the default Tightness value of **3** shows nodes that are far apart.

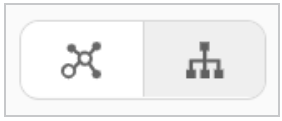


Increasing Tightness to **9** attracts the nodes closer together:

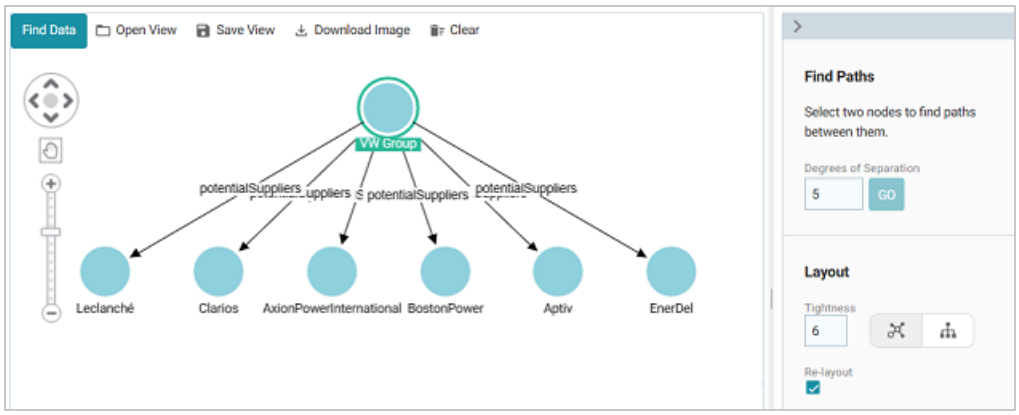


Standard vs. Hierarchical Layout

The layout buttons (shown below) enable you to switch between a **Standard** (⁂) layout (the default setting) and a **Hierarchical** (⁂) layout.



To change to a Hierarchical layout from a Standard layout, select the node in the View that is the root node and then click the **Hierarchy** (⁂) button. For example, in the image below, **VW Group** was selected as the root node. Changing to the Hierarchical layout changes the View to a hierarchy:



Re-Layout

The **Re-Layout** setting controls whether the entire View is refreshed and rearranged when a node is added. When Re-Layout is enabled, the View is rearranged when a node is added. If Re-Layout is disabled, new nodes are added to the View and existing nodes remain in place.

Show or Hide Edge and Node Labels

The Eye (👁) icons next to **Edge Labels** and **Node Labels** enable you to toggle between showing or hiding the edge and node labels in the View.

You can save the View any time by clicking **Save View**. You are prompted to specify a **Title** and optional **Description**. To avoid creating multiple Views with the same name, make sure that you specify a unique Title. When you save the dashboard, the View that is visible at the time it is saved is presented when the dashboard is reopened. Other Views can be opened by clicking **Open View** and selecting a View. Only one View is displayed at time per dashboard. You also have the option to export a PNG version of the Network View by clicking **Download Image**.

Adding Icons to a Network Navigator Dashboard

In Network Navigator Dashboards, you can use custom icons to represent different types (classes) of data in the knowledge graph. The steps below guide you through identifying the types in the Network View, if necessary, so that you can choose icons, uploading the icons the library, and applying the icons to specific classes.

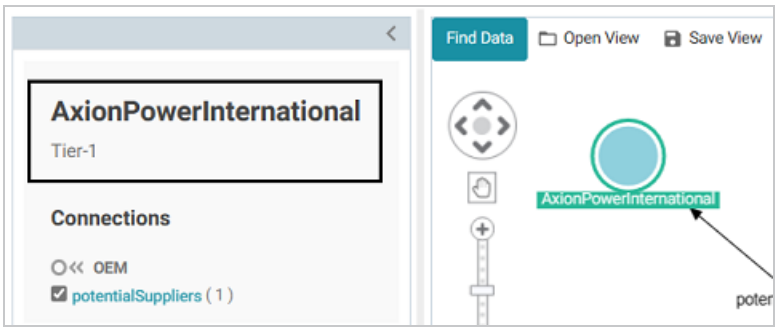
- Identifying Types and Preparing Icons
- Uploading and Applying Icons to a Dashboard

> **Note**
> To be able to upload and apply icons, a Network Navigator Dashboard must exist and have a Network View that displays at least one node. For instructions on creating a dashboard, see Creating a Network Navigator Dashboard.

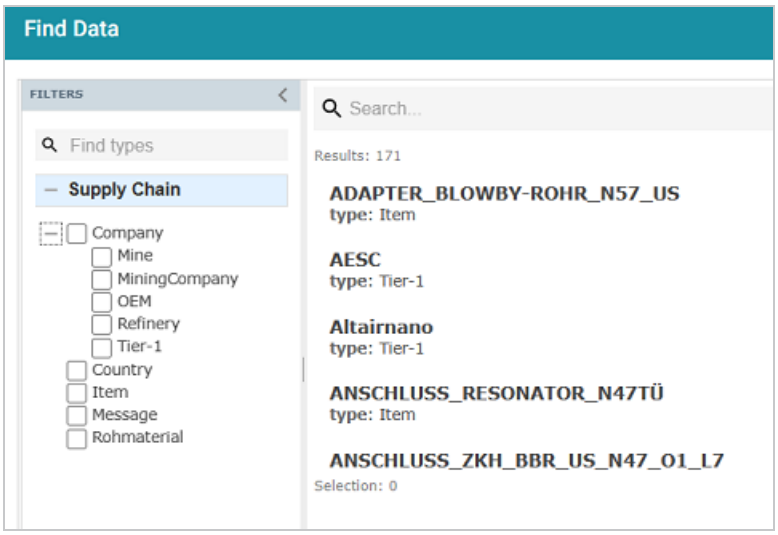
Identifying Types and Preparing Icons

When configuring a dashboard to display icons, the icons are applied at the class (type) level. That way each property of the same type is represented by the same icon. If you are unfamiliar with the data that is displayed in a View, there are a couple of ways you can identify the classes:

- You can select a node in the View and see the type below the node label on the left side of the screen. For example, the image below shows that the **AxionPowerInternational** node is in the **Tier-1** class.

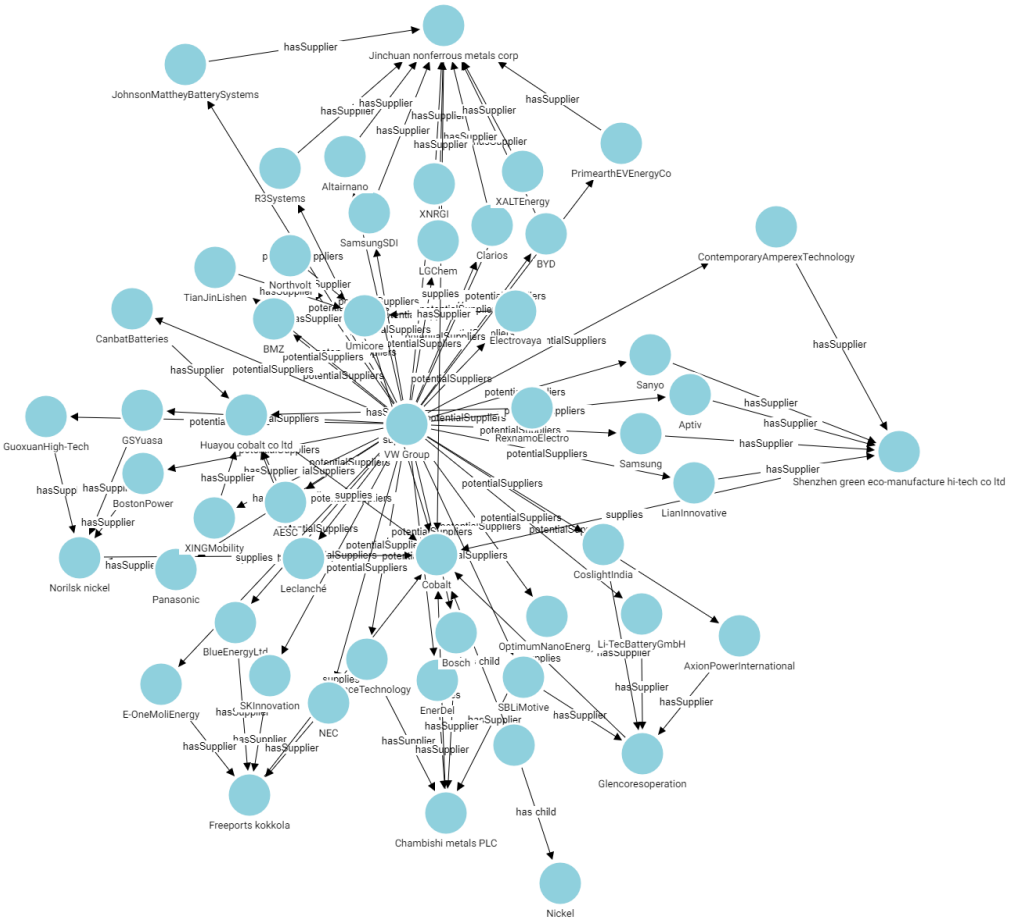


- You can click **Find Data** and open the **Filter** panel. Expand the contents to see the classes and subclasses. For example:



Once you know the types of the nodes in the View, you can determine the appropriate icon to use for each type. Download the icons to your computer, if necessary. When uploading icons to the library, you will browse your computer to select them. The application supports standard image formats like PNG, JPEG, and SVG.

The examples from the list above show a portion of a View and a list of the classes and subclasses for an automobile supply chain knowledge graph. The entire View (shown below) shows companies that are potential suppliers of Cobalt to the VW Group. In the example, adding icons would hep users distinguish between the first tier companies, mining companies, refineries, mines, etc.

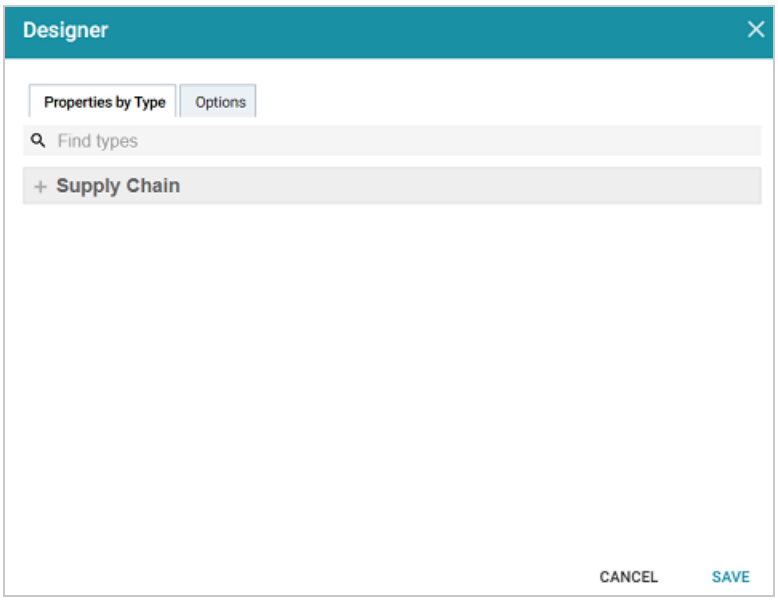


Uploading and Applying Icons to a Dashboard

Follow the instructions below to upload images to the icon library and then apply them to the classes in your Network View.

> **Note**
> When you apply icons to the classes (types) in a dashboard, those icons will apply to all Network Views that include those types.

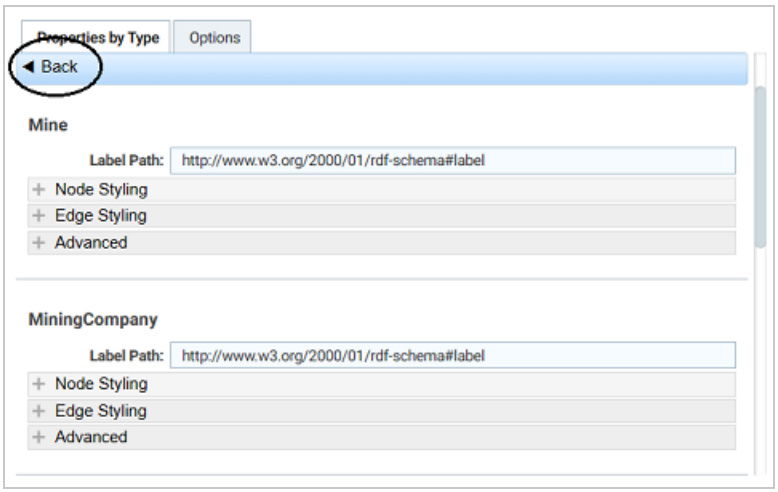
1. Open in the Hi-Res Analytics application the Network Navigator Dashboard that you want to add icons to.
2. In the main toolbar, click the **Designer** button. The **Properties by Type** tab is displayed in the Designer. For example:



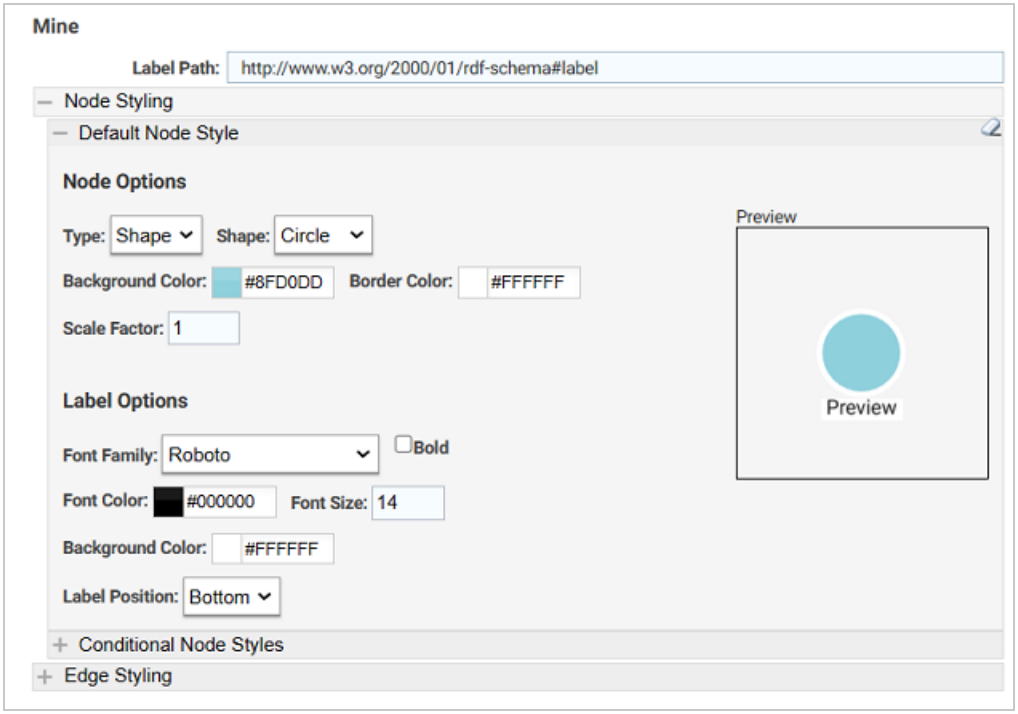
3. Expand the view of the model to display each of the types (classes). If the model contains subclasses, you can view them by clicking the **View Sub Types** link next to the parent class (shown below).



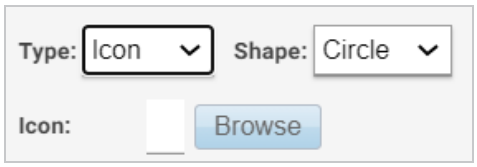
When viewing the subtypes, you can click **Back** to return to the parent level. For example:



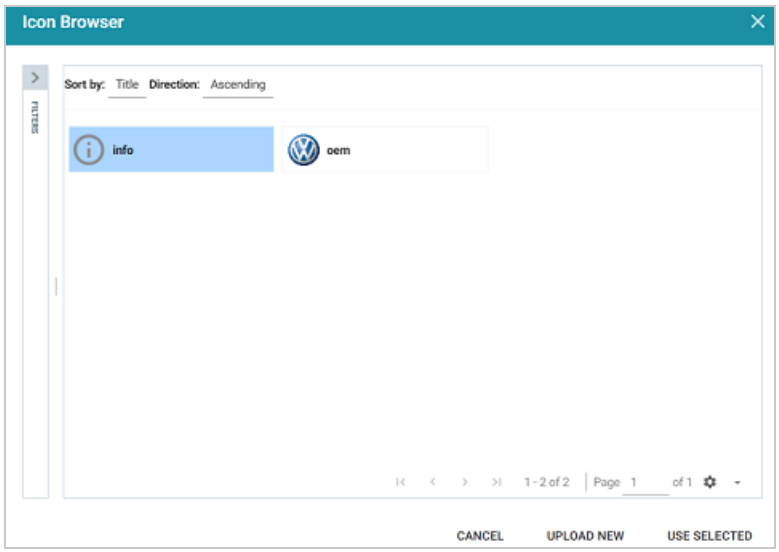
4. Under the class that you want to apply an icon to, expand **Node Styling**. Then expand **Default Node Style**. For example, the image below shows the default node style configuration for the **Mine** type.



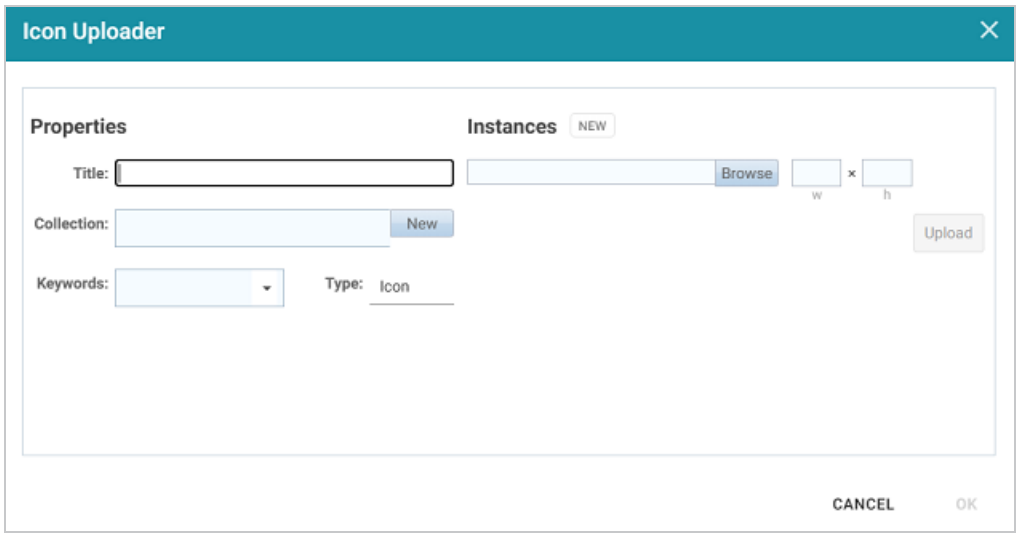
5. Under Node Options, click the **Type** drop-down list and select **Icon**. An Icon field with a **Browse** button is displayed below Type:



6. Click **Browse** to open the Icon Browser. The Browser lists the icons that have been uploaded to the application and gives you the option to upload new icons. For example, the image below shows that two icons have been uploaded previously and are in the library.

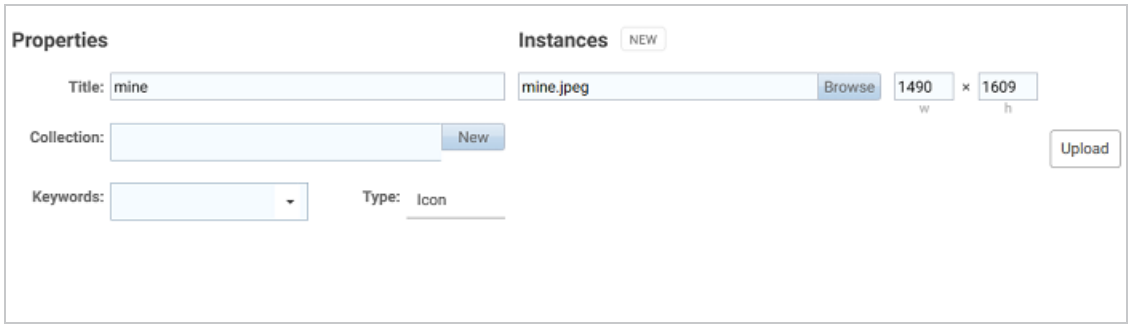


7. To apply an existing icon, select the icon and click **Use Selected**. To add new icons, click **Upload New**. The Icon Uploader dialog box is displayed:



8. In the Icon Uploader dialog box, complete the following fields as needed:
   - **Title**: Required field that specifies the name of the icon.
   - **Instances**: Required field that specifies the icon to upload. Only one icon can be selected at a time. Click **Browse** to navigate to the icon on your computer and select it. When an icon is selected, the width and height fields are populated to show the size of the selected image. You can adjust the size but it is not required. The images are automatically downsized when displayed in the View.
   - **Collection**: Optional field that specifies the collection to add the icon to. If you want to be able to filter icons by collection, you can select an existing collection from the drop-down list or click **New** and type a new name to add to the list.
   - **Keywords**: Optional field that specifies any keywords to tag the icon with. Keywords also enable users to filter icons.
   - **Type**: Required field that specifies the type of image you are adding to the library. **Icon** is the default value and should not be changed.

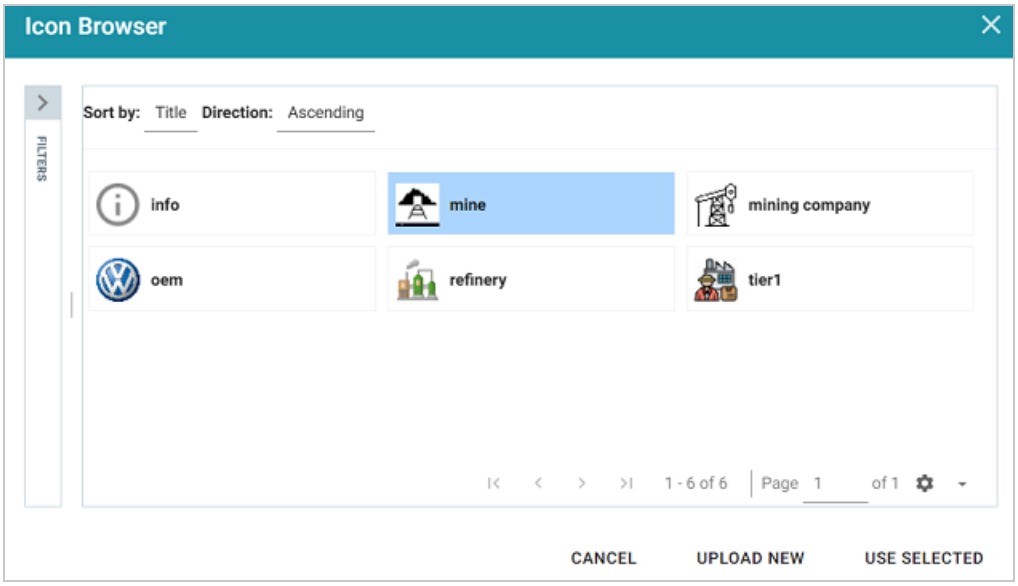
   The image below shows an example of the properties set for an icon that will be used to represent the nodes for mines.



9. When you have finished configuring the icon properties, click **Upload** to upload the icon. The dialog box shows a preview of the new icon. Click **OK** to close the dialog box and add the icon to the Icon Browser.

> **Tip**
> It is helpful to repeat Steps 7 – 9 until you have uploaded all of the icons that you want to use for the dashboard. That way you can more efficiently apply the icons to the types in the Designer.

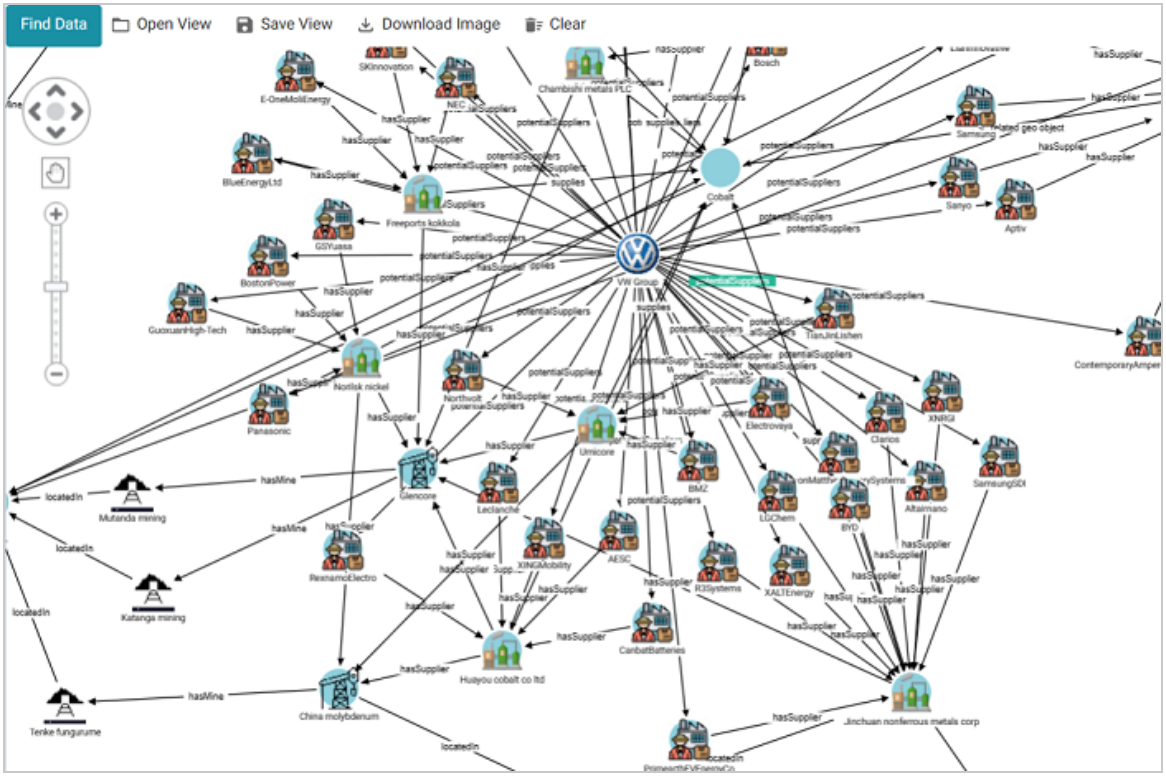
10. When you have finished uploading icons to the Icon Browser, the next step is to apply them to the classes. First, select the icon that you want to apply to the class you selected in Step 4. For example, in the image below, the **mine** icon is selected for the **Mine** class.



11. Click **Use Selected** to apply the icon and return to the Designer where you can preview the icon and configure additional node options.



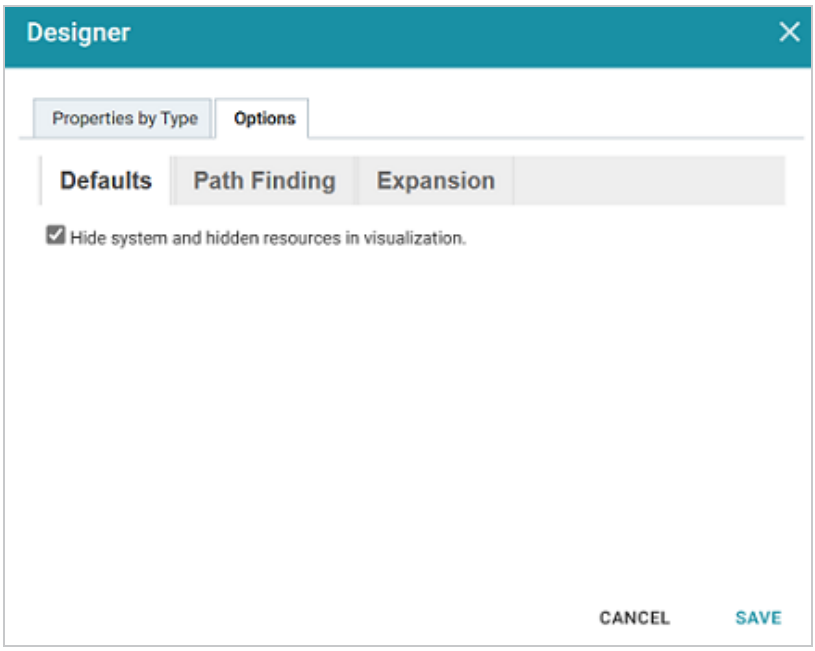
12. If you want to change the shape, size, and background or border color for the icon, you can adjust the following properties:
    - **Shape**: Enables you to set the icon shape to Circle or Square.
    - **Background Color**: Controls the color of the background if the icon is transparent or is smaller than the chosen shape. You can click the colored square to use the color picker or type the hex value for the desired color in the text box.
    - **Border Color**: Controls the color of the border around the icon. You can click the colored square to use the color picker or type the hex value for the desired color in the text box.
    - **Auto-Trim Images**: This setting trims the image to fit in the chosen shape if it is larger than the shape.
    - **Scale Factor**: Enables you to scale the icon size up or down.
13. When you have finished configuring the icon for the selected class, navigate to the other classes and repeat the steps above to apply and configure additional icons. When you have finished applying icons, click **Save** to apply the changes and close the Designer. The new icons will be applied to the nodes in each Network View for the dashboard. For example, the image below shows the revised View of the sample automobile supply chain knowledge graph.



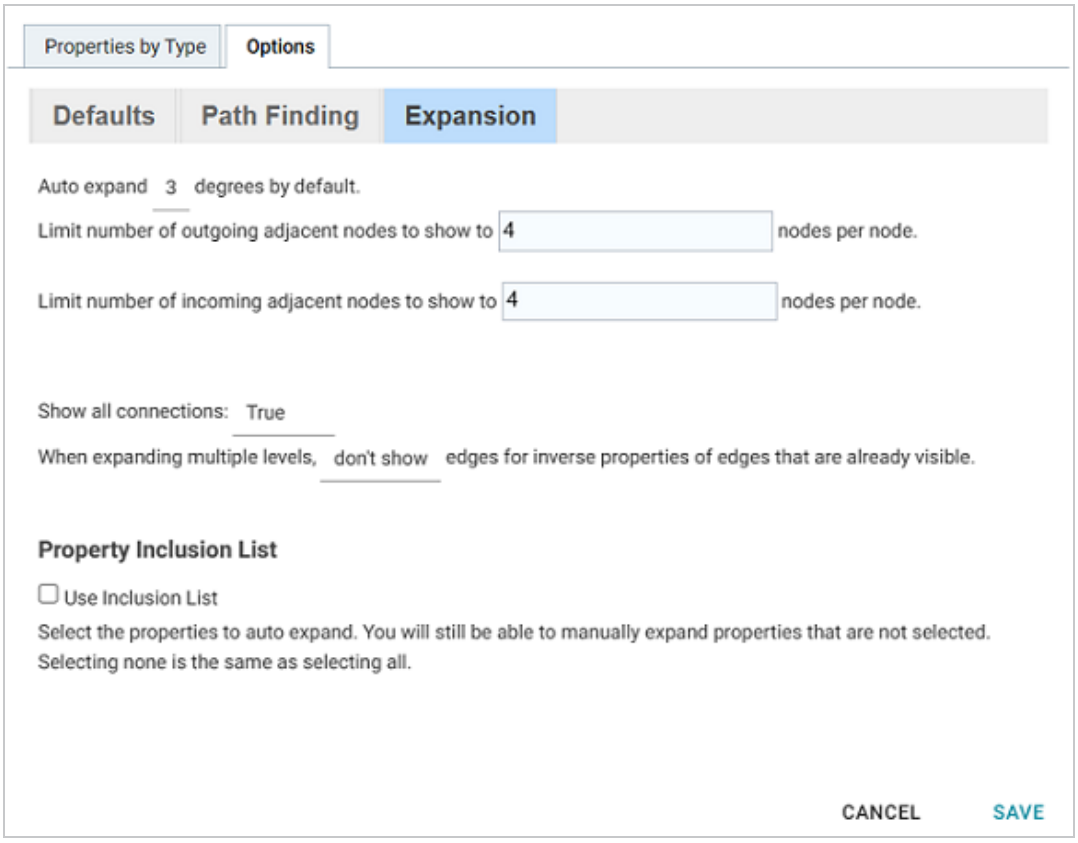
Auto-Expanding a Network View

When you add nodes to a Network View, the Network Navigator Dashboard is configured by default to add only the selected nodes and paths to the View without automatically expanding the paths to show related nodes. You have the option to enable the **Auto-Expand** feature in the Dashboard Designer, however, so that the network is automatically expanded by the specified number of degrees any time a new node or path is added to the View. To configure a dashboard to auto-expand, follow the steps below.

1. Open in the Hi-Res Analytics application the Network Navigator Dashboard that you want to configure.
2. In the main toolbar, click the **Designer** button. The Designer opens and the **Properties by Type** tab is displayed. Click the **Options** tab:



3. Click the **Expansion** tab to view the expansion options. For example, the image below shows the default configuration.



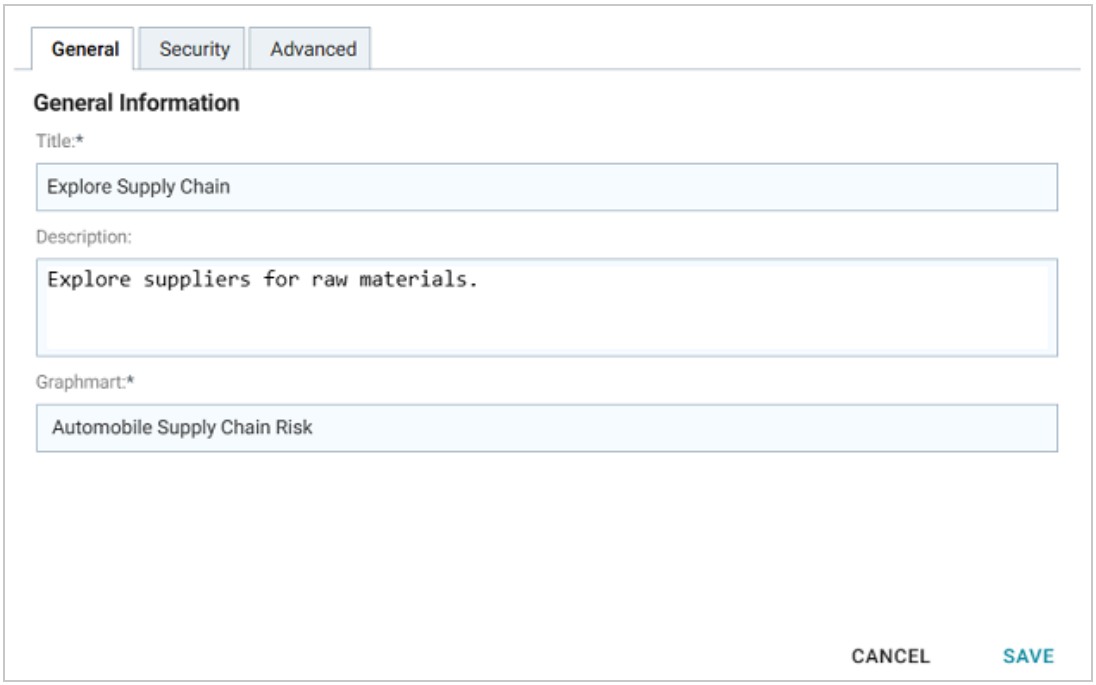
4. On the Expansion tab, configure the options as needed:
    - **Number of Degrees to Expand**: At the top of the screen, select the default number of degrees (hops) to auto-expand. The options are 0 – 6. When this option is set to a value greater than 0 and a node is added to a View in the dashboard, Anzo automatically populates the View with the nodes and paths that are *N* or fewer hops from the added node, where *N* is the specified value. If the View does not auto-expand when a node is added, that means there are no paths to or from that node that are *N* or fewer hops.
    - **Show All Connections**: This option controls whether or not all connections are shown for all nodes.
    - **Show or Hide Inverse Edges**: When auto-expanding the network multiple levels, this option controls whether to show or hide the inverse edges, i.e. whether or not to show both the incoming and outgoing connections to the same nodes. When this option is set to **don't show**, only one edge (path) will be shown between nodes. When this option is set to **show**, both the incoming and outgoing paths will be displayed.
    - **Limit the Number of Adjacent Nodes**: This option controls the number of adjacent nodes to show per node.
    - **Property Inclusion List**: If you would like auto-expansion to apply only to particular properties, you can select the **Use Inclusion List** checkbox. When Use Inclusion List is enabled, the dashboard's properties are listed at the bottom of the screen. Select each of the properties that you want to include.
5. When you have finished configuring the expansion options, click **Save** to apply the changes and close the Designer.

Once Auto-Expand is enabled, adding a node or path to a Network View in this dashboard will also automatically add to the View the related nodes and paths that meet the specified criteria.

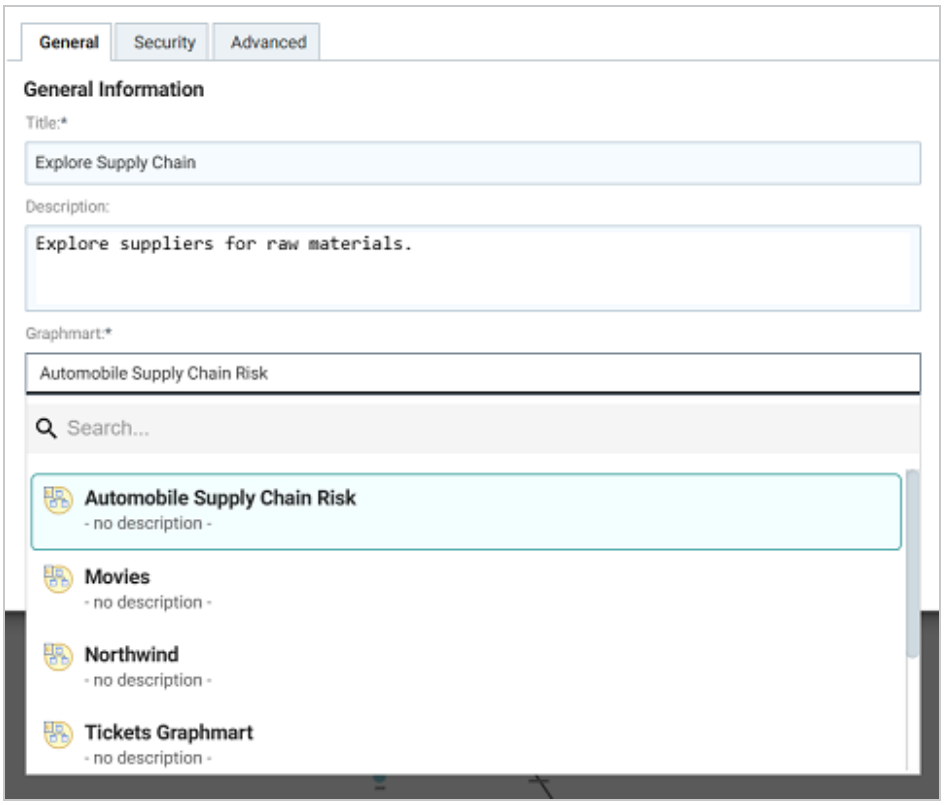
Swapping the Graphmart in a Network Navigator Dashboard

Follow the instructions below if you want to change a Network Navigator dashboard to access a different graphmart.

1. Open in the Hi-Res Analytics application the Network Navigator Dashboard that you want to configure.
2. In the main toolbar, click the **Dashboard** menu and select **Properties**. The Properties dialog box for the dashboard is displayed. For example:



3. On the General tab, click the Graphmart drop-down list and select the graphmart that you want to use in place of the current selection.



4. Click **Save** to save the selection and close the Properties dialog box.

Configuring a Dashboard to Update in Batch Reporting vs. Interactive Mode

Dashboards are configured to run in interactive mode by default. In interactive mode, Anzo updates dashboards automatically each time a user adds, selects, or modifies a filter or lens. Each change generates one or more SPARQL queries that are run against the graphmart data in AnzoGraph. Depending on conditions such as the number of concurrent users, the size of the data, the number of filters, and/or the complexity of the calculations, running in interactive mode can result in significant server workloads, as many complex queries are executed simultaneously. If dashboard performance becomes unsatisfactory, there are two options for changing how and when a dashboard is updated:

- You can configure a dashboard to run in "Reporting" mode. Dashboards in reporting mode are not refreshed immediately with every change. Instead, update queries are paused until a user completes their changes and manually initiates the update process. The update refreshes all of the lenses and filters on the dashboard.

- For finer-grained control over which lenses and filters are updated automatically and which are not, you can enable update controls on each lens and filter. Adding update controls does not change the update method for the entire dashboard, but certain components can be paused and then updated individually.

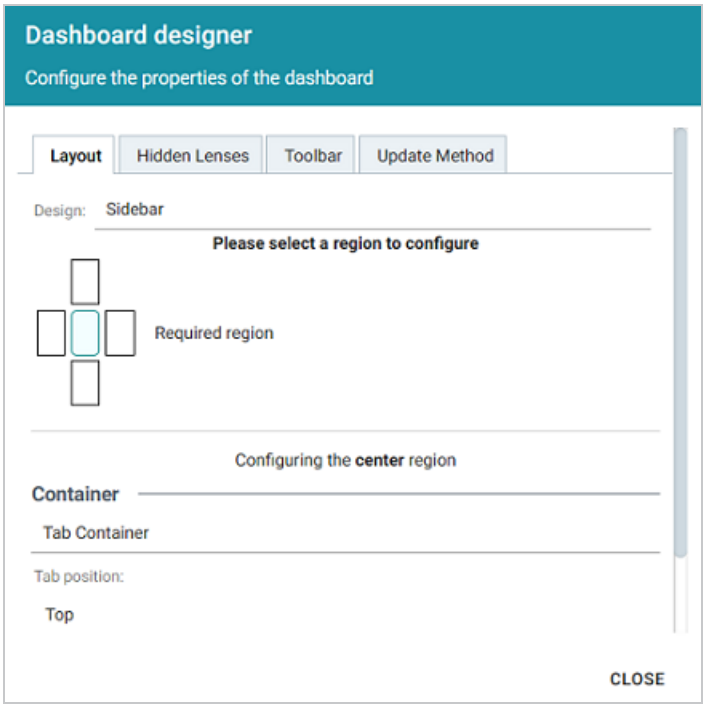
> **Tip**
> By default, dashboards are also automatically refreshed when the data in the backing graphmart changes. The **Automatic** setting in the Refresh menu controls this behavior. When Automatic is disabled and the graphmart is updated, the user is informed that the data changed and they can manually refresh the dashboard to get the updates.

This topic provides instructions for changing the update method at the dashboard level and enabling update controls so that lenses can be updated individually.

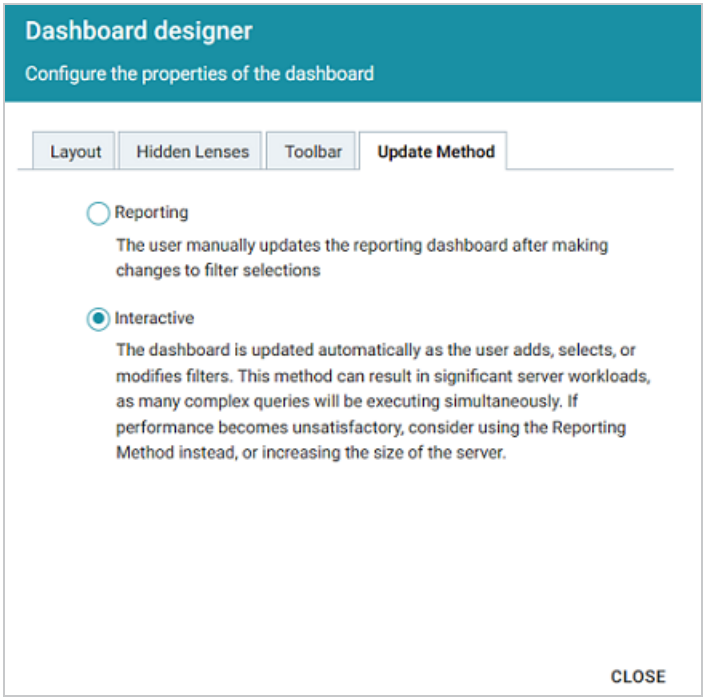
- Changing the Update Method for a Dashboard
- Adding Manual Update Controls to Lenses and Filters

Changing the Update Method for a Dashboard

1. Open in the Hi-Res Analytics application the dashboard that you want to change.
2. In the main toolbar, click **Designer** to open the dashboard designer. For example:



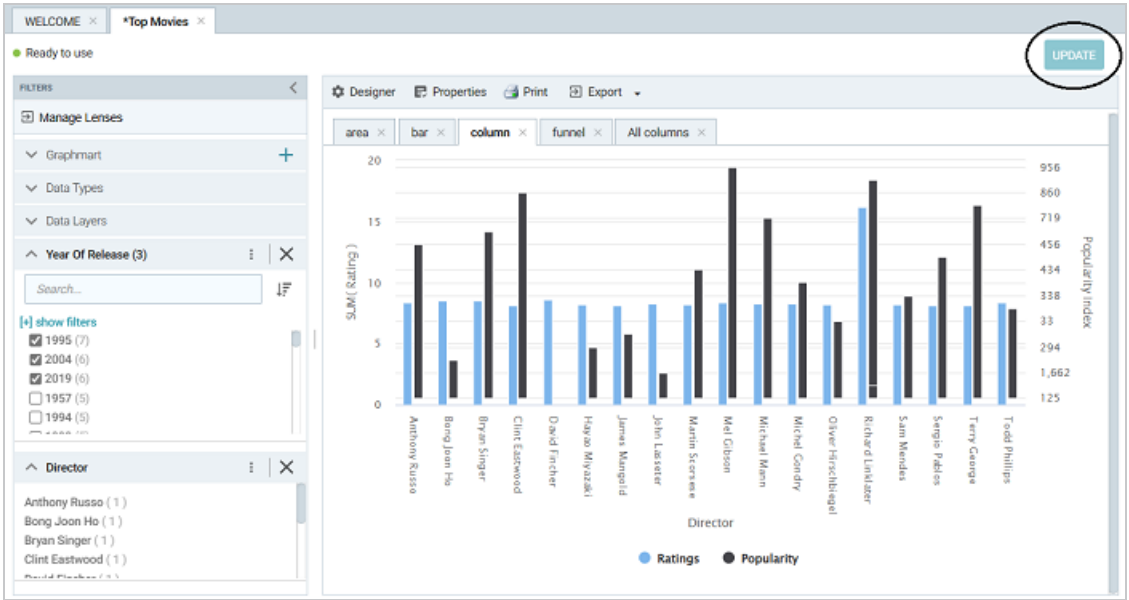
3. In the designer, click the **Update Method** tab. The image below shows the Update Method tab for a dashboard that is in Interactive mode.



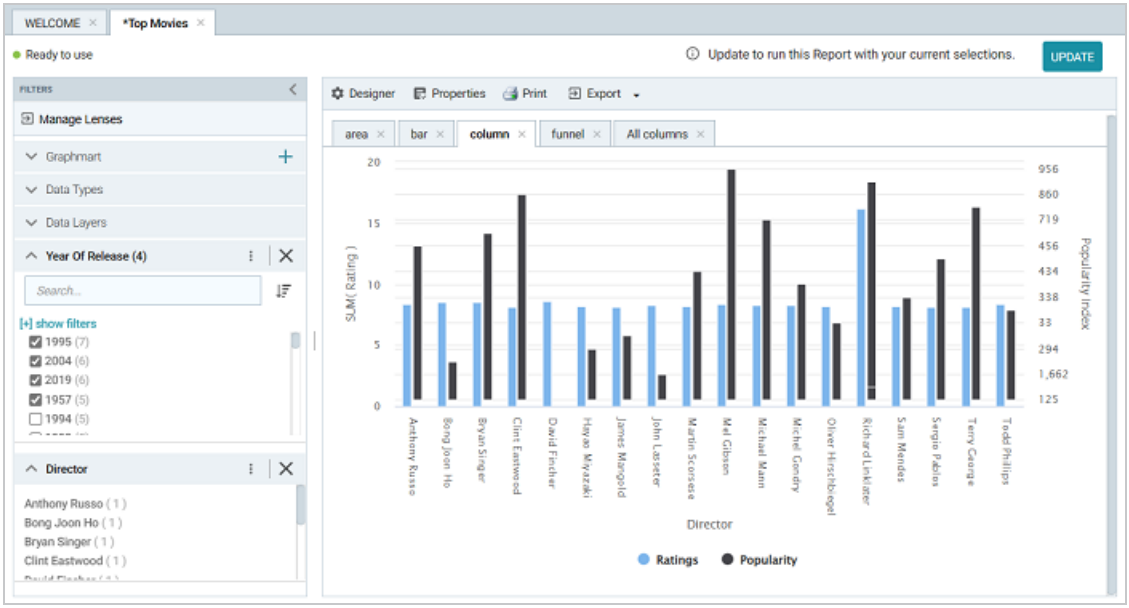
4. Click the radio button for the update mode that you want the dashboard to use. Then click **Close** to close the designer.

5. Save the dashboard to save the configuration change.

When a dashboard is in Reporting mode, an Update button is displayed on the top right corner of the dashboard. The button is disabled when the dashboard is up to date. The image below shows the disabled button.

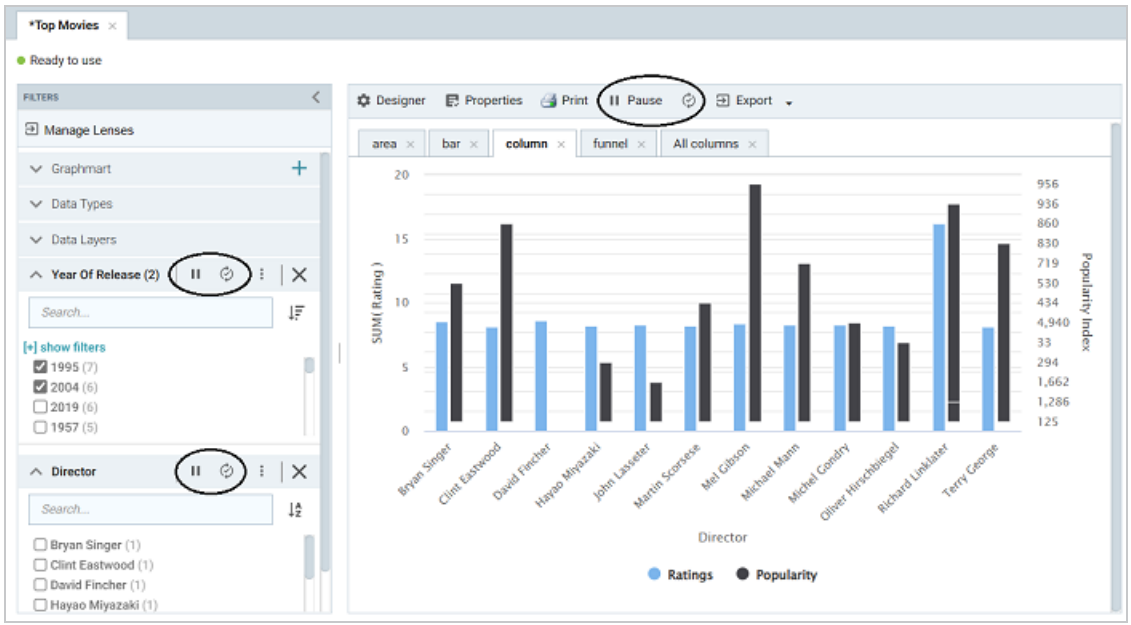


When a user makes a change to a component, the Update button is enabled (as shown in the image below) and the dashboard can be refreshed as needed. When the update is initiated, all lenses are refreshed.



Adding Manual Update Controls to Lenses and Filters

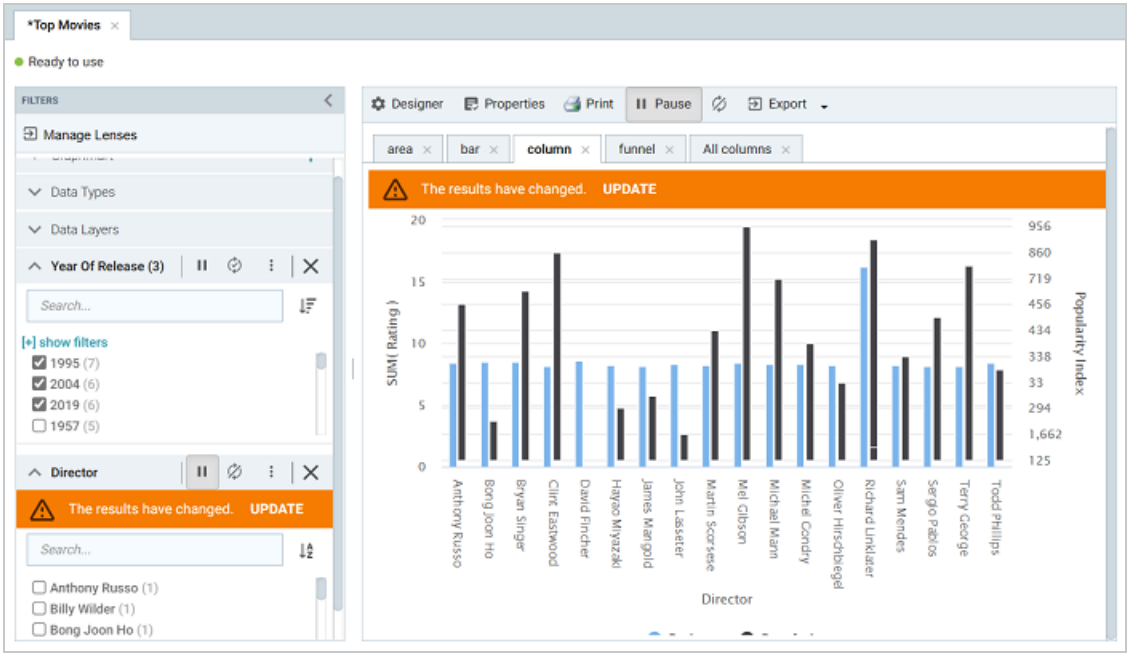
1. Open in the Hi-Res Analytics application the dashboard that you want to change.
2. In the main toolbar, click the Refresh menu and select **Show Update Controls**. A pause icon and status icon are added to each filter and lens on the dashboard, as shown in the image below.



The pause icon ( ) is used to pause or re-enable automatic updates of the component, and the status icon indicates whether the component is up-to-date ( ) or out of sync ( ).

3. Save the dashboard to save the configuration change.
4. In the dashboard, click the **pause** icon for any filters and lenses that you do not want to be updated automatically.

When a filter or lens is paused and a user makes an update that affects that component, an update message is displayed. For example, the image below shows a lens and filter that are out of sync. Clicking **UPDATE** in the message refreshes that component.



Capturing User-Defined Values in Dashboards

You can configure a Table lens in a dashboard to allow users to input values and save those values to a linked dataset (LDS) in a graphmart. This topic provides instructions for configuring a graphmart, LDS, and dashboard to enable and save user-defined input. There are several steps required to configure the environment:

> **Note**
> The procedures below must be performed by the sysadmin user or a user with Anzo Administrator permissions. Once the setup is complete, the components can be shared with other users and groups.

1. Create a Volume for Storing User Input
2. Configure the Graphmart
3. Add User Input Properties to the Model
4. Create and Configure the Dashboard
5. Share the LDS and Add it to the Graphmart

Create a Volume for Storing User Input

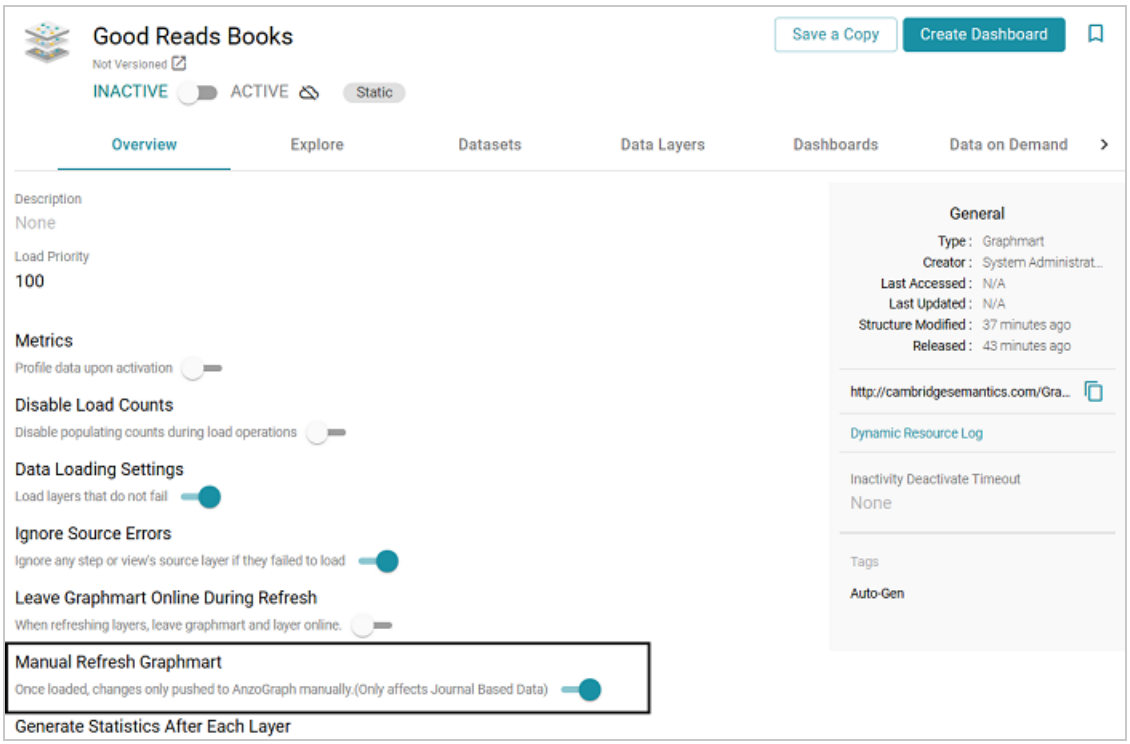
When you configure a Table lens to save user-entered values to an LDS, you must choose the volume in which to save the LDS. Cambridge Semantics strongly recommends that you do not write to the system journal (`<install_path>/Server/data/journal/anzo.jnl` by default). Instead, use a separate volume that is dedicated to storing user-entered values. If necessary, create a new volume before proceeding. See Creating a New Volume in the Administration Guide for instructions.

> **Note**
> Cambridge Semantics also strongly recommends that you create a separate volume for each graphmart that will include user-defined values.

Configure the Graphmart

The first step is to deactivate the graphmart that will contain the user-entered values and disable manual refresh. When manual refresh is disabled, the graphmart is refreshed automatically any time a user updates the LDS by adding values in the dashboard. Follow the steps below to configure the graphmart.

1. In the graphmart for which you want to add user-defined text, go to the Overview tab.
2. From the Overview tab, deactivate the graphmart if it is online, and then find the **Manual Refresh Graphmart** setting at the bottom of the screen:



3. If **Manual Refresh Graphmart** is enabled, slide the slider to the left to disable it.
4. Activate the graphmart and proceed to the next step to update the model.

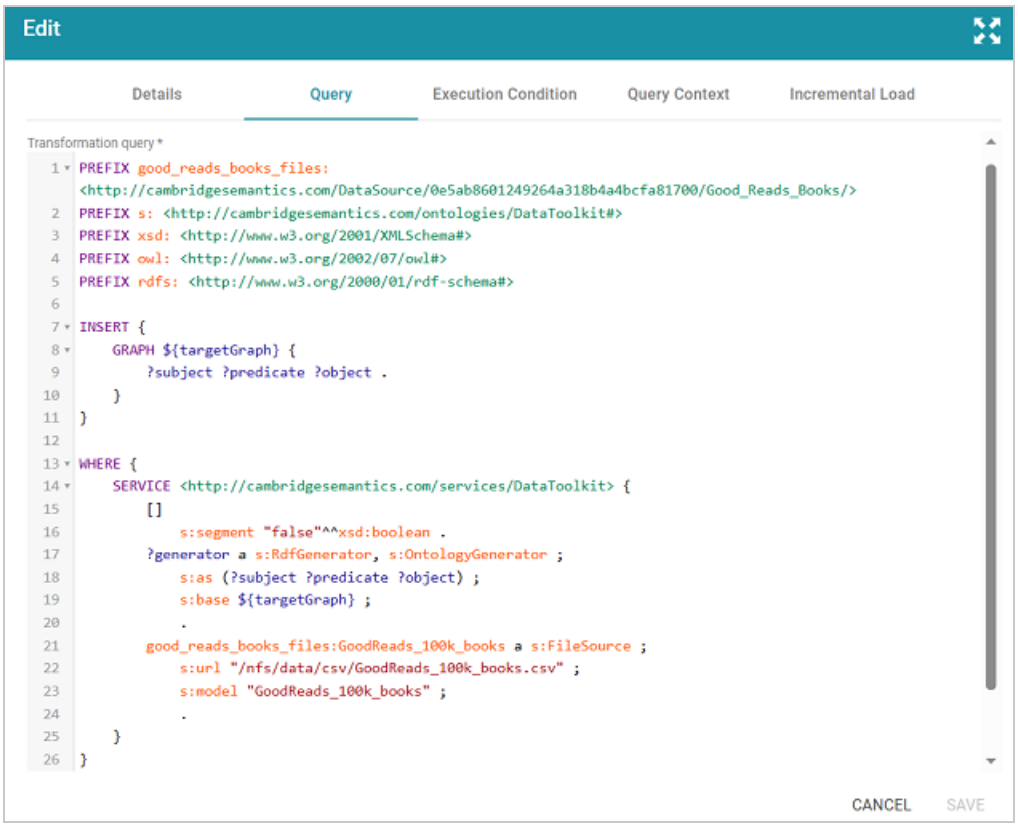
Add User Input Properties to the Model

The next step is to update the model that was generated when the data was onboarded. Any new properties that you want to add as columns for text entry in a Table lens need to be added to the model. Follow the steps below to update the Direct Load Step query to add new properties to the model.

1. Click the **Data Layers** tab. Expand the layer that was generated when you created the graphmart and find the Direct Load Step that inserts the data.

   > **Tip**
   > You might want to start another session of Anzo so that you can view the layer's model in the Model editor and the graphmart query at the same time.

2. Open the Direct Load Step for editing and click the **Query** tab. For example, the image below shows the query that was generated to onboard data about books from a CSV file:



3. Make sure the following prefixes are declared in the PREFIX clause at the top of the query. If any are missing, add them to the query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

4. Next, add the following GRAPH clause statements under the SERVICE clause:

```
GRAPH <model_URI> {
  <property_URI> a owl:DatatypeProperty;
    rdfs:comment "comment" ;
    rdfs:label "property_label" ;
    rdfs:domain <class_URI> ;
    rdfs:range <datatype> .
[ <property2_URI> a owl:DatatypeProperty;
    rdfs:comment "comment" ;
    rdfs:label "property2_label" ;
    rdfs:domain <class_URI> ;
    rdfs:range <datatype> . ]
```

```
[ ... ]
}
```

Value	Data Type	Description
model_URI	URI	The URI of the model that was generated by the Direct Load Step. You can copy the URI from the Model editor from the Details tab for the model. For example, `<http://cambridgesemantics.com/Layer/698.../Model>`.
property_URI	URI	The URI to use for the new property. You can look at the URIs for other properties in the model and follow the same scheme. For example, `<http://cambridgesemantics.com/Layer/698.../Model#GoodReads100kBooks.Notes>`.
comment	string	A comment that describes the property. For example, `"Notes from users"`.
property_label	string	The label to give the property. For example, `"Notes"`.
class_URI	URI	The URI of the class that the property should be added to. You can copy the URI from the Model editor from the Details tab for the class. For example, `<http://cambridgesemantics.com/Layer/698.../Model#GoodReads100kBooks>`.
datatype	URI	The datatype of the new property. For example, `xsd:string`.

For example, the following query adds one new property to the model that is generated by the Direct Load Step query shown above:

```
PREFIX good_reads_books_files:
<http://cambridgesemantics.com/DataSource/0e5ab8601249264a318b4a4bcfa81700/Goo
d_Reads_Books/>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT {
  GRAPH ${targetGraph} {
    ?subject ?predicate ?object .
  }
}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    GRAPH
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model> {

<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model#Goo
dReads100kBooks.Notes> a owl:DatatypeProperty;
      rdfs:comment "Notes from users" ;
      rdfs:label "Notes" ;
      rdfs:domain
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model#Goo
dReads100kBooks> ;
      rdfs:range xsd:string .
  }
    []
      s:segment "false"^^xsd:boolean .
    ?generator a s:RdfGenerator, s:OntologyGenerator ;
      s:as (?subject ?predicate ?object) ;
      s:base ${targetGraph} .

    good_reads_books_files:GoodReads_100k_books a s:FileSource ;
      s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
      s:model "GoodReads_100k_books" .
  }
}
```

5. Next, **if the class for any of the new properties does not have a primary key defined, you must create a key for that class**. If all of the classes you referenced in the query have primary keys, you can continue to the next step. If one or more of the classes do not have primary keys, follow the instructions below:

   a. Locate in the query the statement block for each class that needs a key defintion. For example, in the query above, there is only one class, `s:model "GoodReads_100k_Books"`. If you have multiple classes, the query has several blocks, such as this example:

```
...
emrdbsmall:emr_complaint a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
  s:table "emrdbsmall.emr_complaint" ;
  s:model "emr_complaint" .

emrdbsmall:emr_patient a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
  s:table "emrdbsmall.emr_patient" ;
  s:model "emr_patient" .

emrdbsmall:emr_complaintdescription a s:DbSource ;
  s:using mysql_db:MySQL_DB ;
  s:table "emrdbsmall.emr_complaintdescription" ;
  s:model "emr_complaintdescription" .
...
```

   b. At the end of the block for the class you want to add a key to, change the period (.) after `s:model` to a semicolon (;).

   c. Next, add the following line below `s:model`:

```
s:key ("key_property" [, "key_property2" ] [, ... ]) .
```

   Where `key_property` is the label of the property to use as a key for the class. The property that you choose must have unique values. If there is not a property in the class with unique values, you can specify a combination of properties that would create a

unique value. Make sure that the value of `key_property` matches the label for that property in the model. For example, for the query in step 4, the `Isbn` property can be used as a unique key for the `GoodReads_100k_Books` class:

```
good_reads_books_files:GoodReads_100k_books a s:FileSource ;
  s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
  s:model "GoodReads_100k_books" ;
  s:key ("Isbn") .
```

The final, completed query is shown below:

```
PREFIX good_reads_books_files:
<http://cambridgesemantics.com/DataSource/0e5ab8601249264a318b4a4bcfa81700
/Good_Reads_Books/>
PREFIX s: <http://cambridgesemantics.com/ontologies/DataToolkit#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT {
  GRAPH ${targetGraph} {
    ?subject ?predicate ?object .
  }
}
WHERE {
  SERVICE <http://cambridgesemantics.com/services/DataToolkit> {
    GRAPH
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l> {

<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l#GoodReads100kBooks.Notes> a owl:DatatypeProperty;
      rdfs:comment "Notes from users" ;
      rdfs:label "Notes" ;
      rdfs:domain
<http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Mode
l#GoodReads100kBooks> ;
      rdfs:range xsd:string .
  }
    []
      s:segment "false"^^xsd:boolean .
    ?generator a s:RdfGenerator, s:OntologyGenerator ;
```

```
      s:as (?subject ?predicate ?object) ;
      s:base ${targetGraph} .

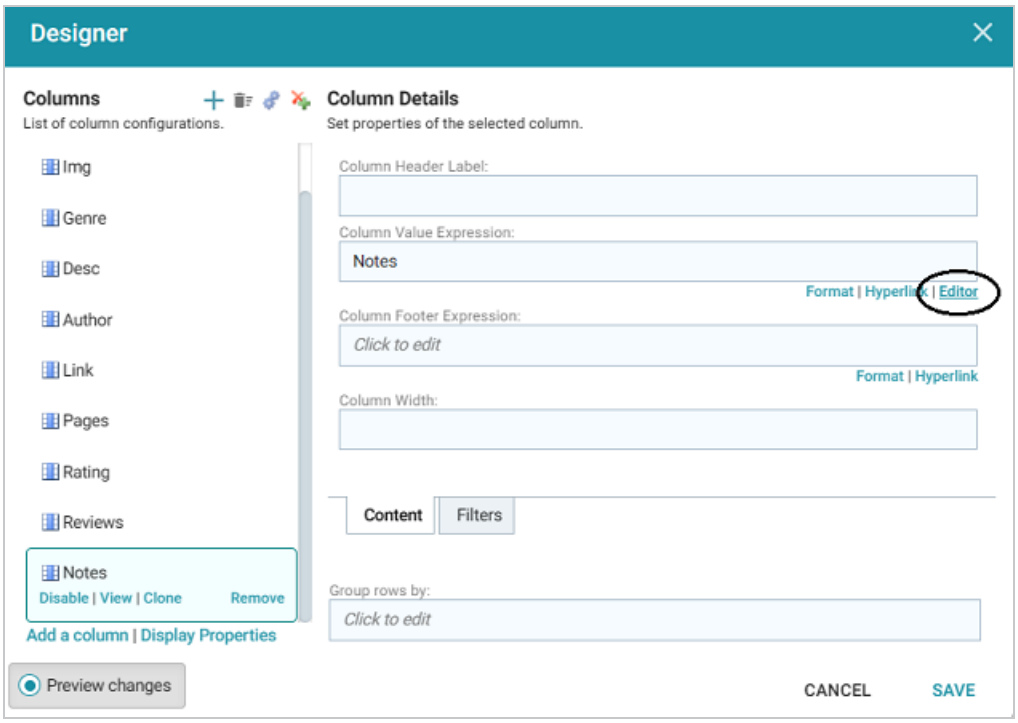
    good_reads_books_files:GoodReads_100k_books a s:FileSource ;
      s:url "/nfs/data/csv/GoodReads_100k_books.csv" ;
      s:model "GoodReads_100k_books" ;
      s:key ("Isbn") .
  }
}
```

6. Save the step and then refresh or reload the graphmart to update the model with the new properties.

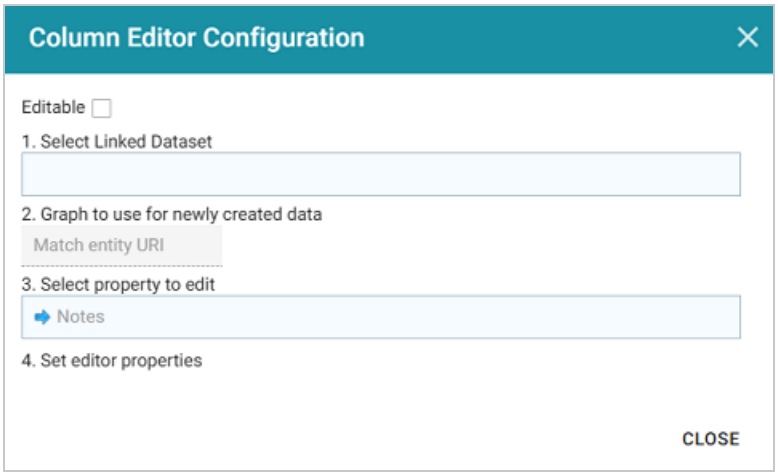
Create and Configure the Dashboard

Now that the graphmart is updated, the next step is to create and configure the dashboard that will enable input for the new properties.

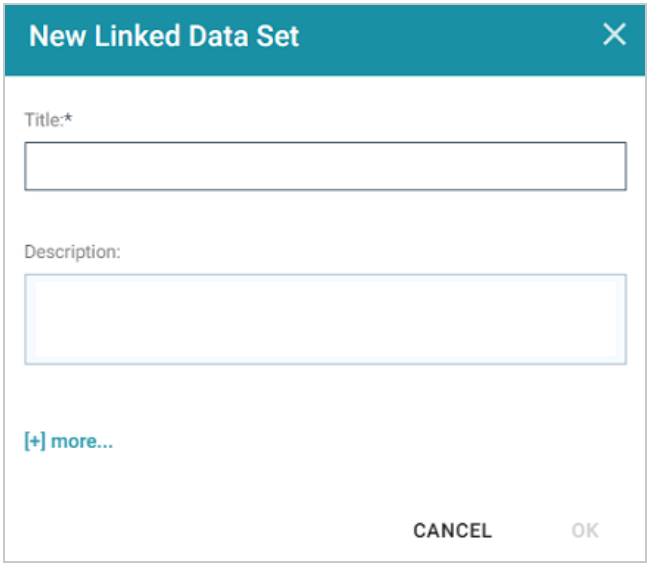
1. Create a dashboard for the graphmart that was updated in the previous task.
2. Add a Table lens to the dashboard. In the lens, add any columns that you would like to see, including columns for the user input properties that you added to the model.
3. In the lens Designer, select a column that will allow input and click the **Editor** link under **Column Value Expression** (shown in the image below).



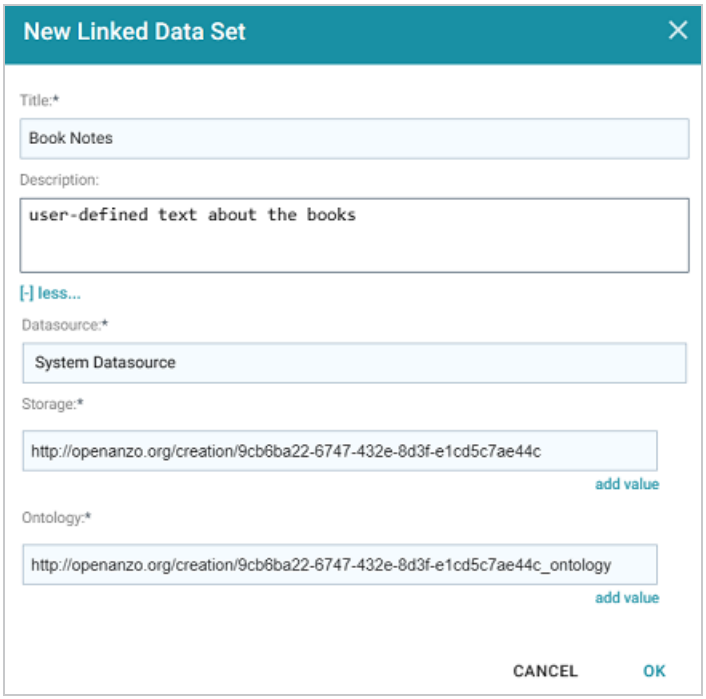
The Column Editor Configuration dialog box is displayed and populated with the name of the chosen property:



4. In the dialog box, select the **Editable** checkbox. Then click the **Select Linked Dataset** field and select **Create a linked dataset**. The New Linked Data Set dialog box is displayed:



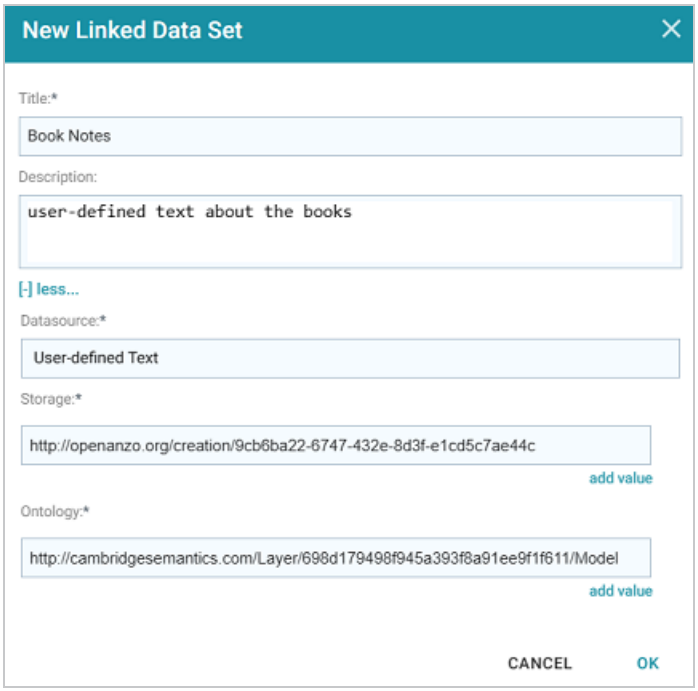
5. Specify a title for the new LDS and add an optional description. Then click the **[+] more** link to expose the following additional settings.



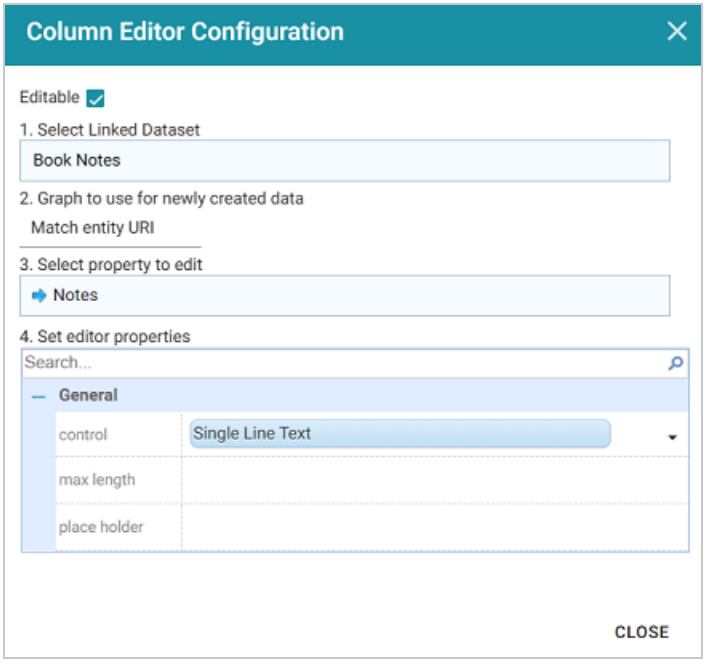
6. Complete the required fields:
   - **Datasource**: Click this field and select the volume to save the LDS in. This is the volume that you created in Create a Volume for Storing User Input.

- **Storage**: This field lists the system-generated URI for the LDS and dataset storage graph. Cambridge Semantics recommends that you do not change this value.
- **Ontology**: Replace this value with the URI of the model that contains the writable properties. For the example shown above, the value is `http://cambridgesemantics.com/Layer/698d179498f945a393f8a91ee9f1f611/Model`.

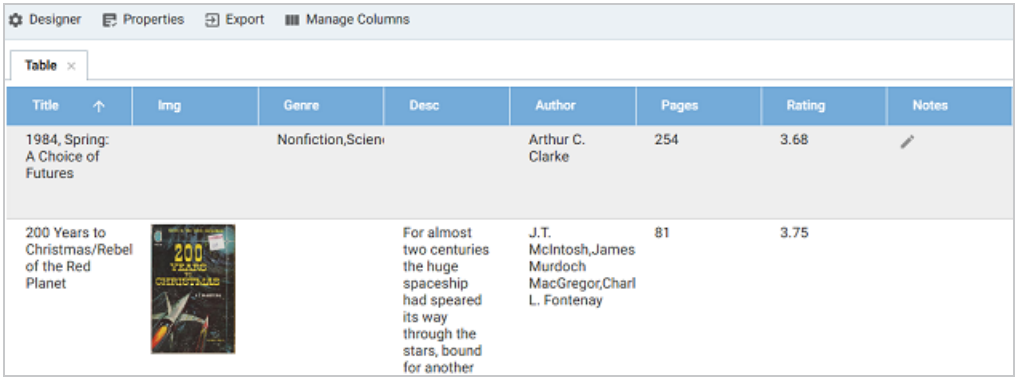
The image below shows the completed screen:



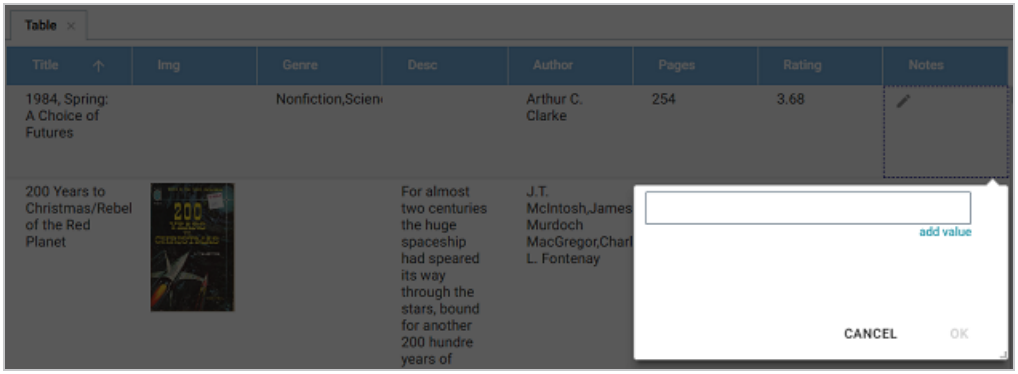
7. Click **OK** to save the LDS configuration and create the LDS. The LDS is added to the Datasets catalog and it is shown on the Column Editor Configuration screen. Additional editor properties are made available on the screen. For example:



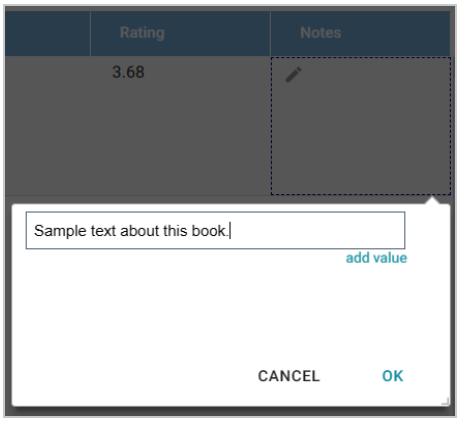
8. Under **Set editor properties**, you have the option to modify the constraints for the editable property.
9. When you are finished setting any constraints, click **Close** and then click **Save** in the Designer. The lens is now configured to display the editable field. Hovering over a row in the table shows an edit icon (✎) in the field. For example:



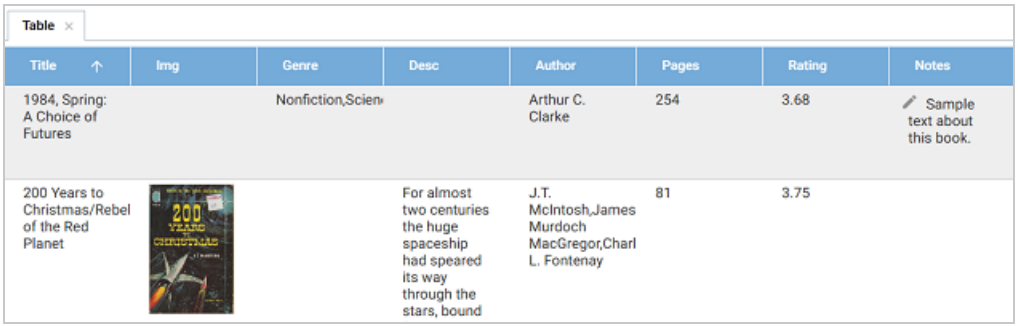
10. Repeat steps 3 – 9 for any additional properties that you added to the model and want to make editable.
11. To add text in the lens, double-click an editable field to open the text editor, as shown in the image below.



Add text in the provided field, and then click **OK** to add the text to the dashboard.



The text is displayed in the table:



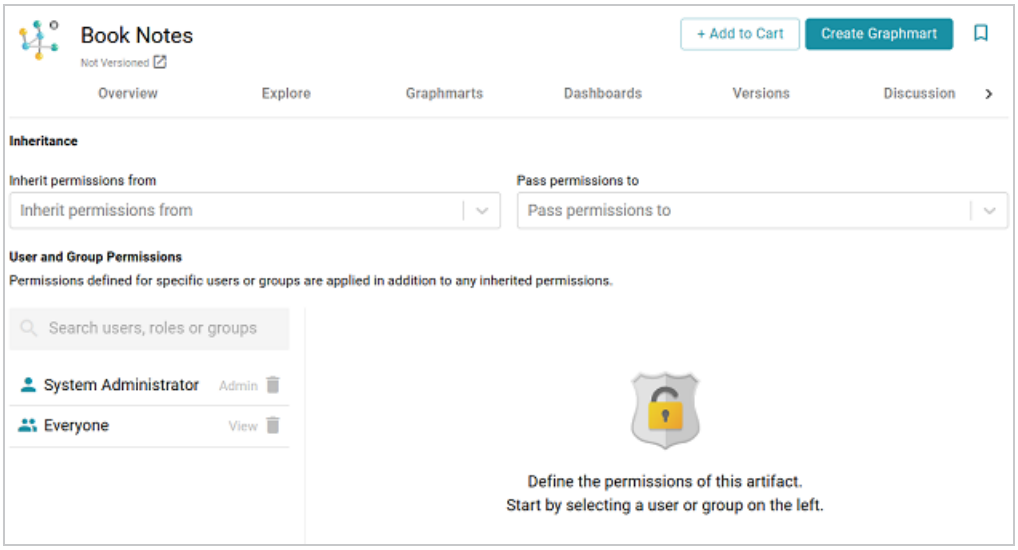
You can edit the text by double-clicking the field. You can also add another text entry by opening the text editor and clicking **add value**. Click **OK** after making changes or adding values.

12. Finally, save the dashboard and proceed to the next task.

Share the LDS and Add it to the Graphmart

In order for other users to be able to add values to the dashboard, they need to have permission to modify the LDS. Follow the steps below to share the LDS and then add it to the graphmart.

1. In the Datasets catalog, open the LDS that is associated with the dashboard you configured in the previous task. Then click the **Sharing** tab to configure the permissions.



2. On the Sharing tab, share the dataset with the appropriate users and/or groups. For information about configuring permissions, see Share Access to Artifacts.
3. Next go to the graphmart and click the **Datasets** tab.
4. On the Datasets tab, click **Add Dataset**. Then select the dataset that you shared and click **Add** to add the dataset to the graphmart. For more information about adding a dataset to a graphmart, see Adding a Dataset to a Graphmart.
5. Refresh the graphmart to load the dataset to AnzoGraph.

The graphmart, linked dataset, and dashboard are now configured to allow and save user-defined values.

Working with Lenses

The topics in this section provide instructions on creating, exporting, and deleting lenses.

- Creating a Lens
- Cloning a Lens
- Exporting a Lens
- Deleting a Lens

Creating a Lens

Lenses define the data's visual presentation. Each type of lens represents a unique method for displaying data. For instance, in a column chart, you can present multiple data series for comparison. You can also apply custom formats such as fonts and colors to any lens. The topics in this section provide instructions for creating lenses.

- Creating a Chart Lens
- Creating a Drill Down Lens
- Creating a Form Lens
- Creating a List Lens
- Creating a Table Lens
- Advanced Lenses

Creating a Chart Lens

Anzo Hi-Res Analytics employs the Highcharts API to provide interactive chart lenses. The topics in this section provide information about creating each type of chart.

- Create an Area Chart
- Create a Bar Chart
- Create a Bubble Chart
- Create a Column Chart

- Create a Funnel Chart
- Create a Heat Map
- Create a Line Chart
- Create a Polar Chart
- Create a Scatter Chart

Create an Area Chart

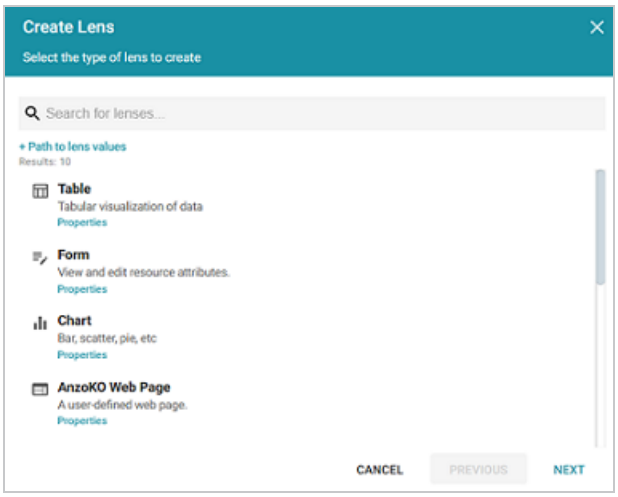
Area charts are useful for emphasizing trends in your data. They are similar to line charts but include options for displaying stacked data series. This topic provides instructions for creating an area chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

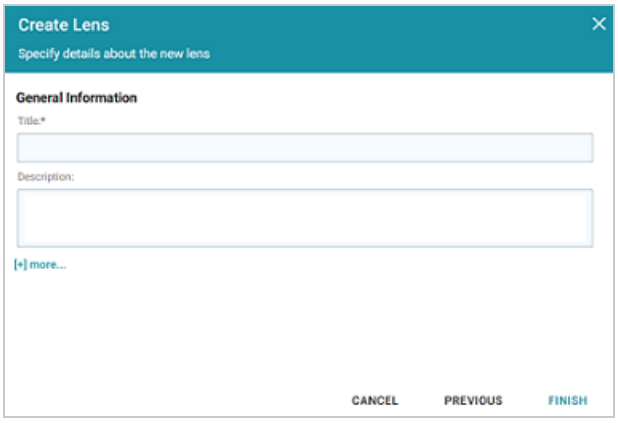
Complete the Minimum Configuration

Follow the instructions below to create an area chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

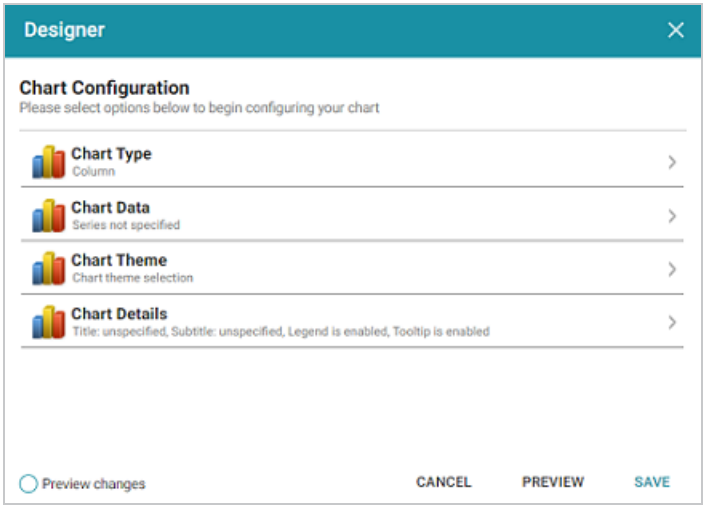


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

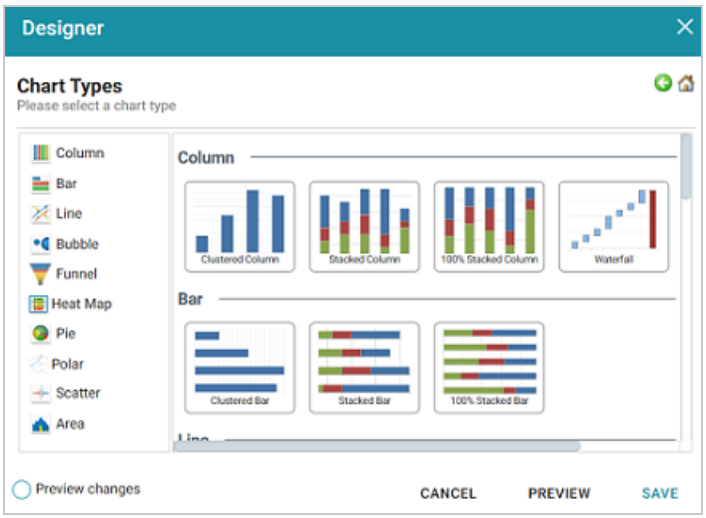


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:



5. Click **Chart Type** to open the Chart Types dialog box:

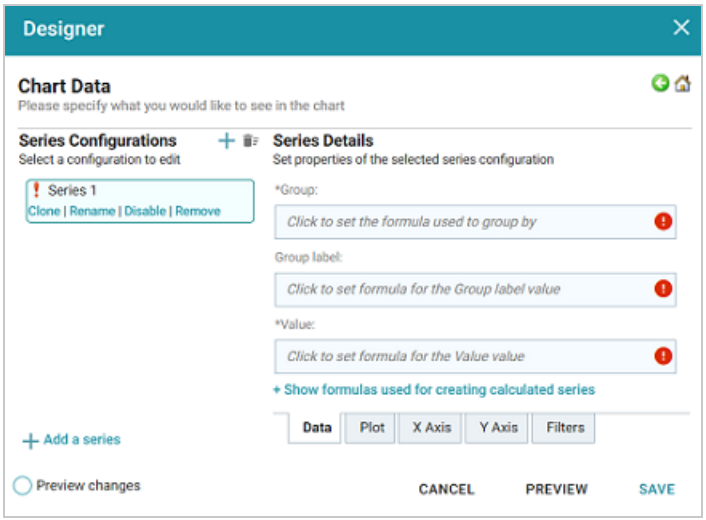


6. On the left side of the screen, select **Area**. On the right side of the screen, select the icon for the type of area chart that you want to create. There are five types:
   - **Area**: Connects value points on the chart with straight lines and shades the area below the lines.
   - **Step Area**: Connects value points on the chart with short horizontal steps and shades below the lines. This chart emphasizes the extent of value change by expanding the data points across the X axis.

- **Area Spline**: Connects value points on the chart with curved lines and shades the area below the lines.
- **Stacked Area**: Connects value points on the chart with straight lines and shades the area below the lines. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.
- **100% Stacked Area**: Compares each value as a percentage of the total and shades the area below each series. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.

Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (🗑). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.
9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. These are the values for the X axis. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:
   - The **Root Value** (⌂) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

     > **Tip**
     > To view the root values, you can use the STR function to show a string representation of the URIs.

   - Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
   - When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
   - After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the Y axis values. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.
    - **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
    - **Series Sort Direction**: Specifies the sort direction for the series groups.
13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

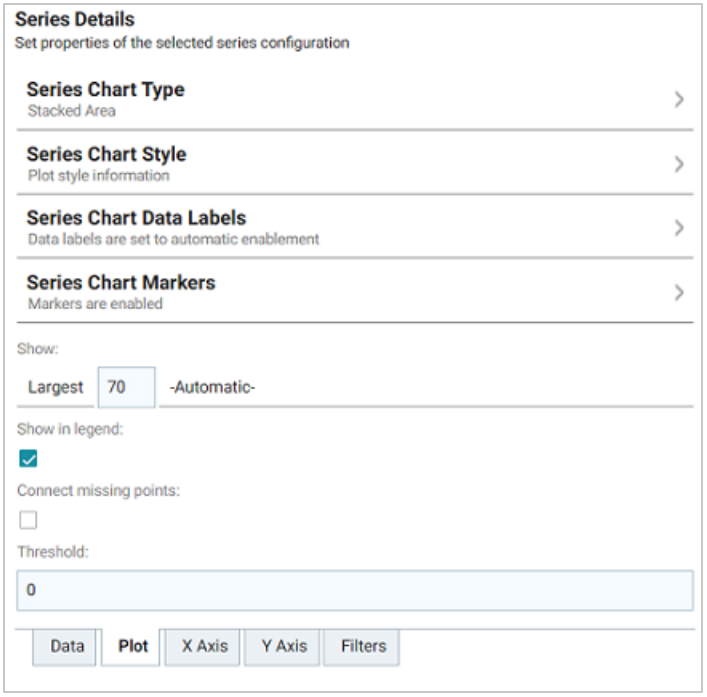
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing an area chart.

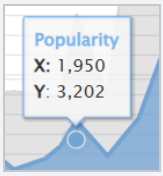
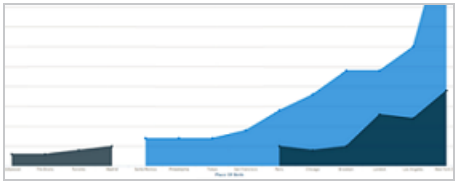
- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings. These settings control the thickness and color of lines for the series as well as the color and opacity of the fill area below the lines.
Series Chart Data Labels	This option contains the data label settings, which control the font, alignment, and font effects of the labels for the series.
Series Chart	This option contains the data marker settings. These settings control the

Field	Description
Markers	placement and format of the data points for a series that appear when a user hovers the pointer over a point in the line. For example, the image below shows a data marker in an area chart: 
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.
Connect missing points	Selecting this option connects the series line across any missing points. For example, if this option was enabled for the chart below, the two dark blue areas would be connected. 
Threshold	Defines the Y axis value to use as a base (starting point) for the shaded area. For example, a threshold of 0 begins all shading at the value 0. A threshold of 10 begins the shading at 10 and draws the area chart above or below the threshold as required. The image below shows an area chart with a threshold of 30.

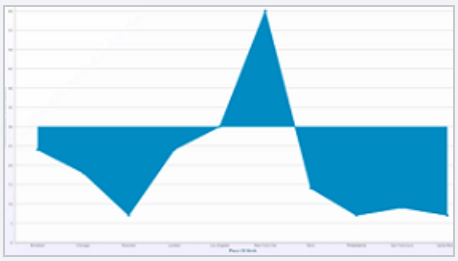
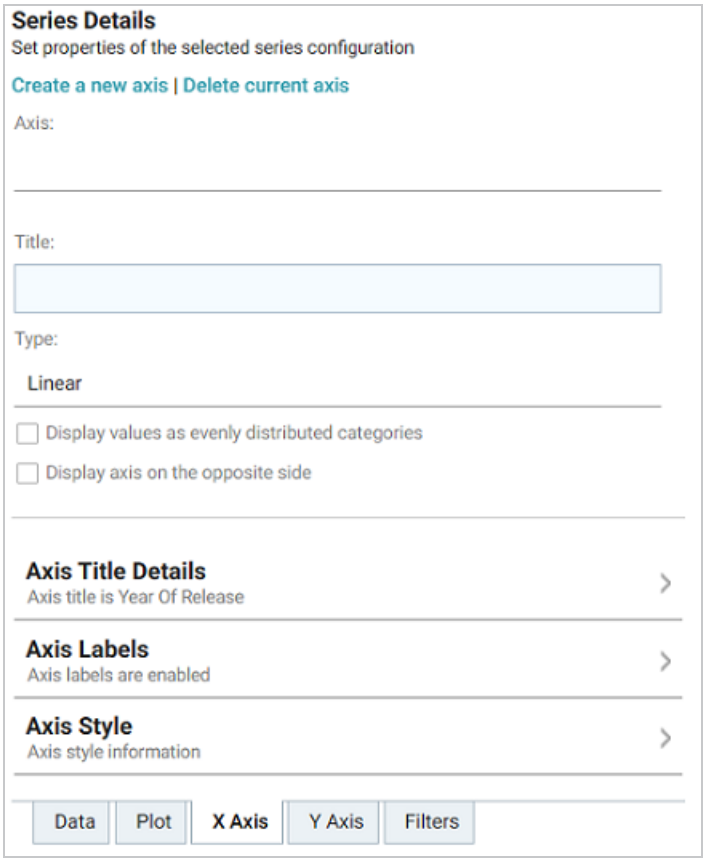
Field	Description
	

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

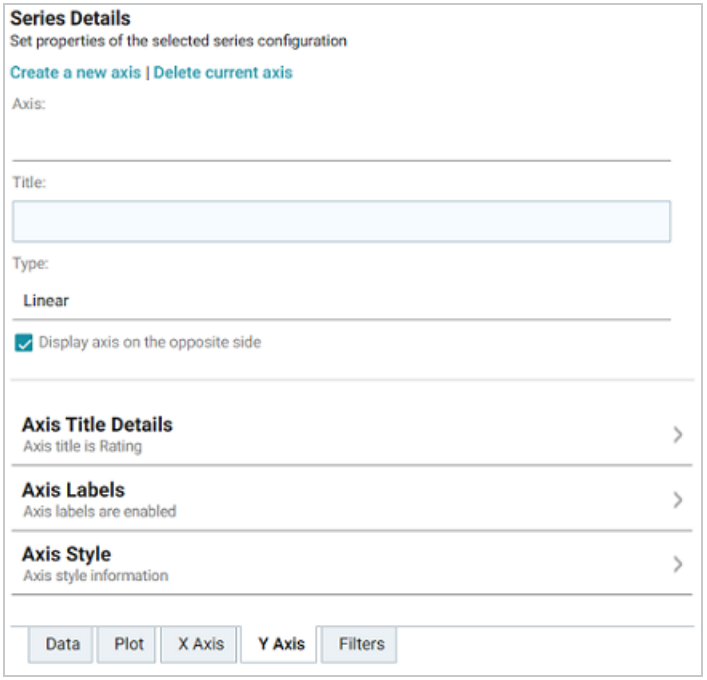


Field	Description
Axis	The Axis and Title settings are populated with the values from the `Group` and `Group Label` values from the Data tab. If multiple axes exist, you can

Field	Description
	selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Type	This setting controls the scale for the X axis, linear or logarithmic.
Display values as evenly distributed categories	This setting controls whether to evenly distribute the X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.
Axis Title Details	This option contains the X axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

This tab defines the formats and labels for the Y axis values in the series.

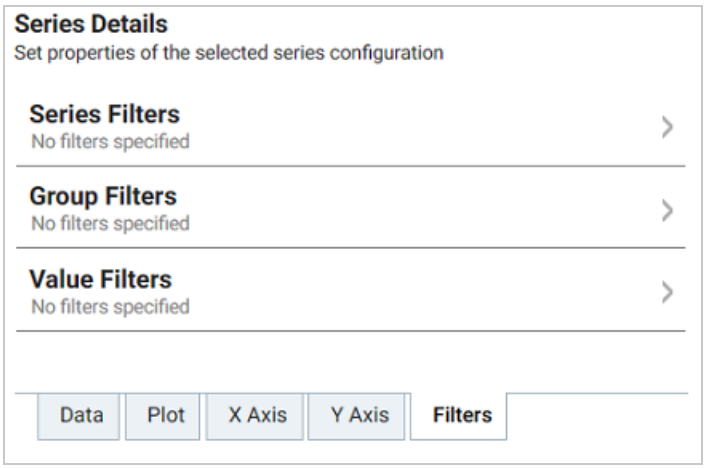


Field	Description
Axis	The Axis and Title settings are populated with the `Value` from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.

Field	Description
Axis Style	This option contains the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Group Filters	This option can be used to define filters that apply only to the `Group` values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

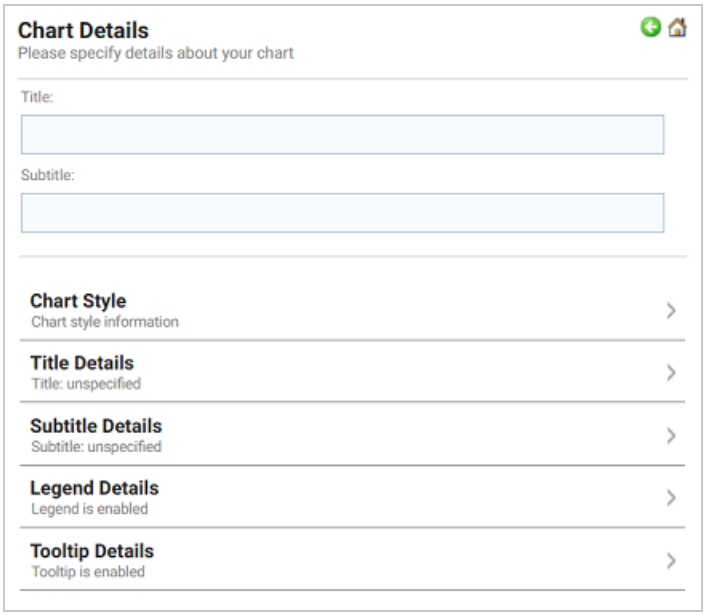
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Bar Chart

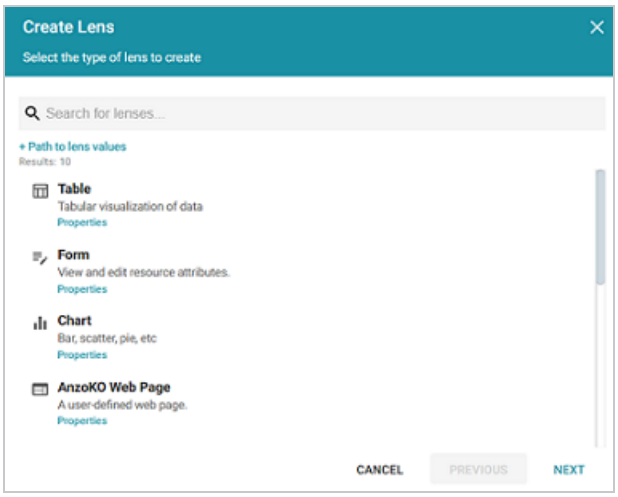
Bar charts are useful for comparing different sets of data or emphasizing drastic changes in a data set over time. This topic provides instructions for creating a bar chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

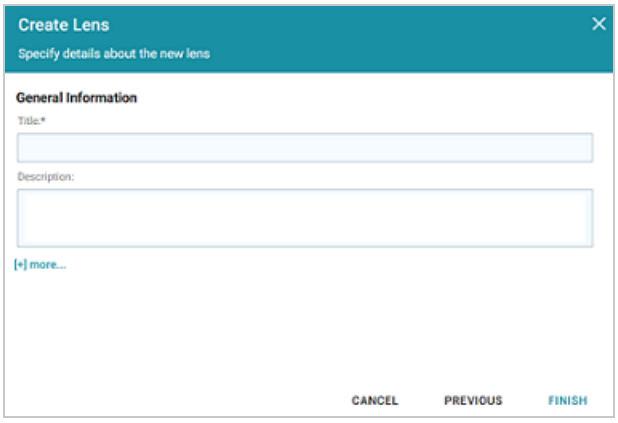
Complete the Minimum Configuration

Follow the instructions below to create a bar chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

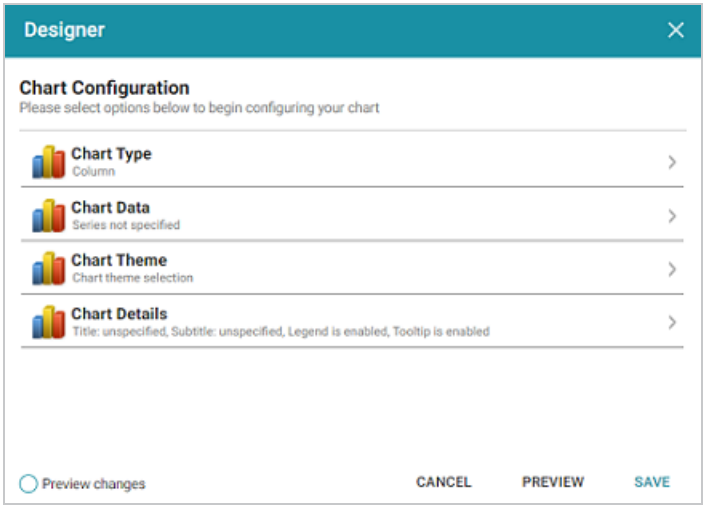


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

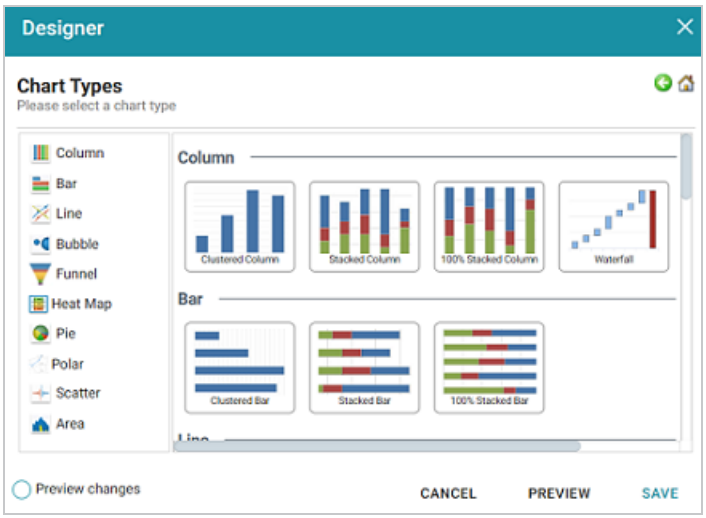


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:



5. Click **Chart Type** to open the Chart Types dialog box:

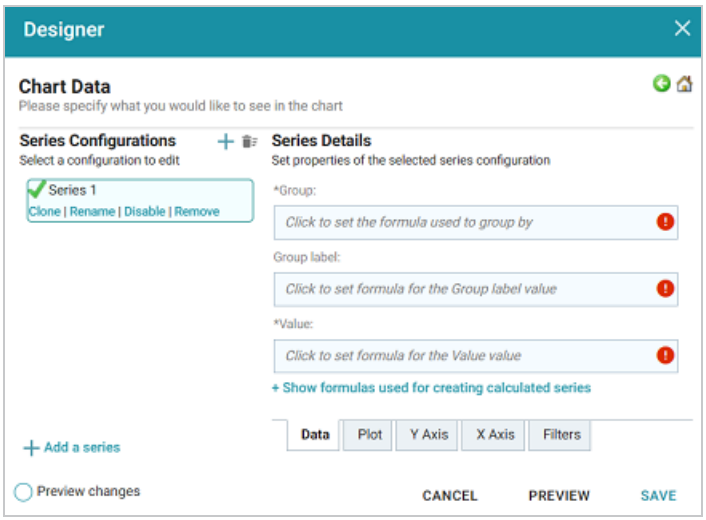


6. On the left side of the screen, select **Bar**. On the right side of the screen, select the icon for the type of bar chart that you want to create. There are three types:
   - **Clustered Bar**: Compares values across categories.
   - **Stacked Bar**: Compares the contribution of each value to a total across categories. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.

- **100% Stacked Bar**: Compares each value as a percentage of the total. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.

Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. These are the values for the X axis. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when

choosing a property:

- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the Y axis values. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:

- **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.
- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

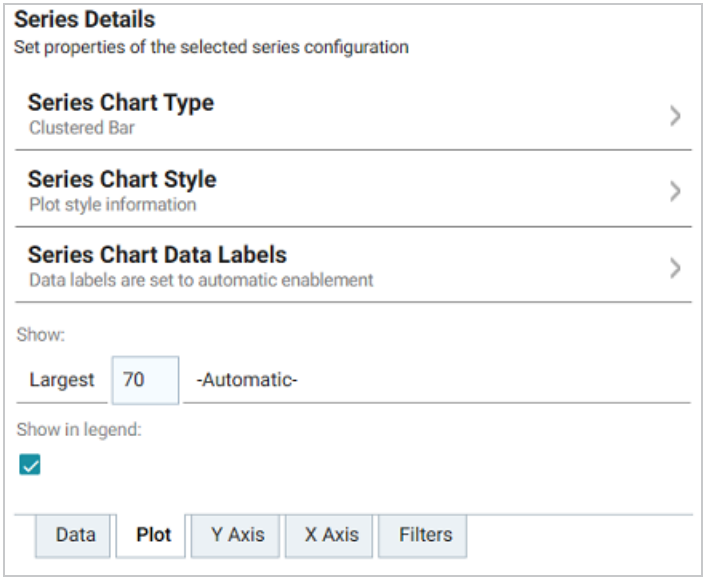
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a bar chart.

- Chart Data: Plot Tab
- Chart Data: Y Axis Tab
- Chart Data: X Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

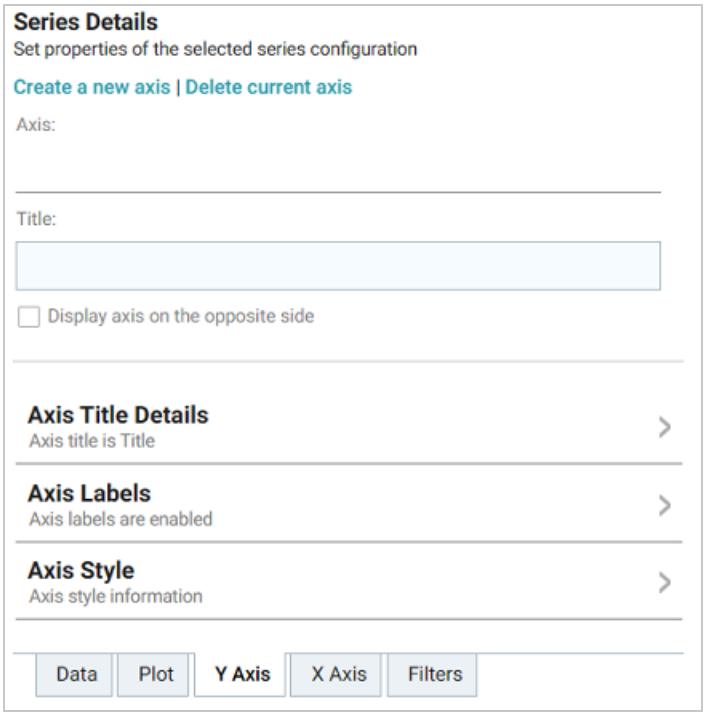
This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings, which control the thickness and color of the bars for the series.
Series Chart Data Labels	This option contains the data label settings, which control the font, alignment, and font effects of the labels for the series.
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: Y Axis Tab

This tab defines the formats and labels for the Y axis values in the series.

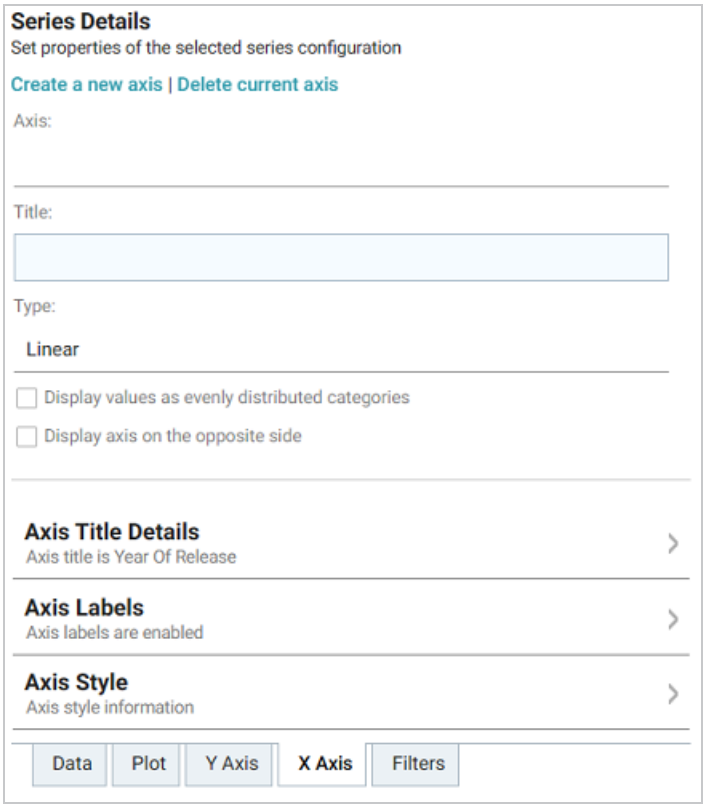


Field	Description
Axis	The Axis and Title settings are populated with the `Value` from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option opens the Y axis title settings, which control the font, alignment, and font effects of the axis title.

Field	Description
Axis Labels	This option opens the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

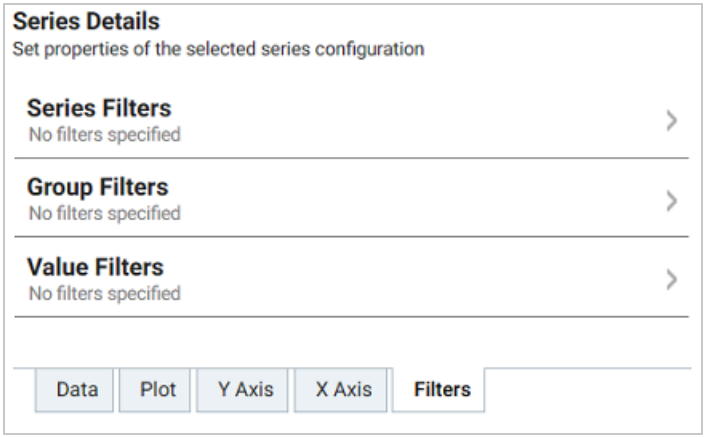


Field	Description
Axis	The Axis and Title settings are populated with the values from the `Group` and `Group Label` values from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.

Field	Description
Title	This setting defines the title for the X axis.
Type	This setting controls the scale for the X axis, linear or logarithmic.
Display values as evenly distributed categories	This setting controls whether to evenly distribute X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.
Axis Title Details	This option accesses the X axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option accesses the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Group Filters	This option can be used to define filters that apply only to the `Group` values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

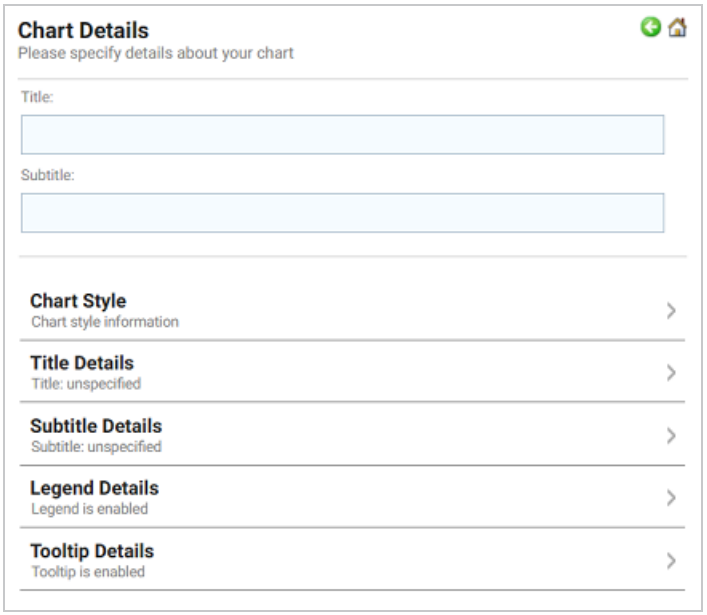
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Bubble Chart

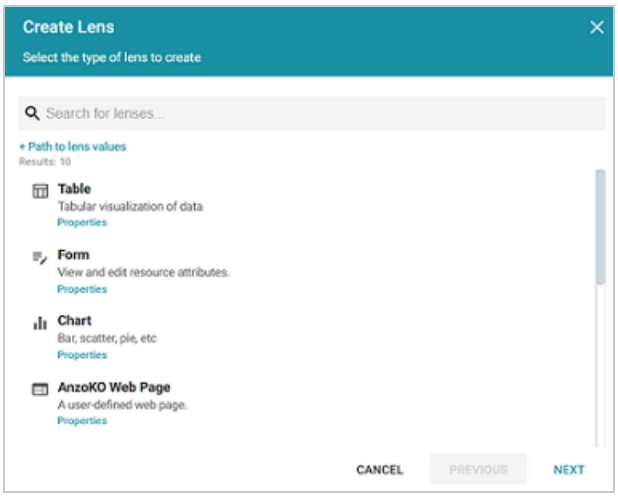
A bubble chart is similar to a scatter chart but has a third dimension. In addition to the X and Y axes, there is a size axis that determines the size of the bubbles in the chart. This topic provides instructions for creating a bubble chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

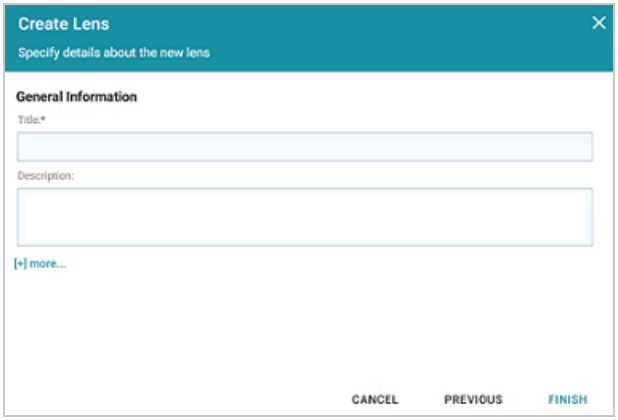
Complete the Minimum Configuration

Follow the instructions below to create a bubble chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

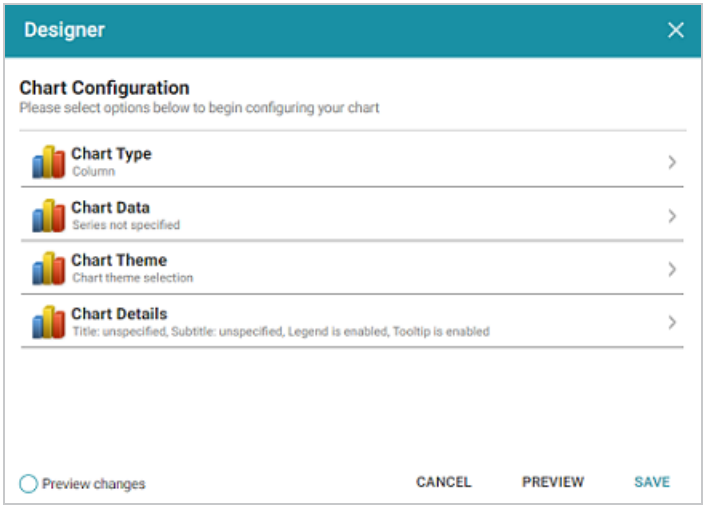


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

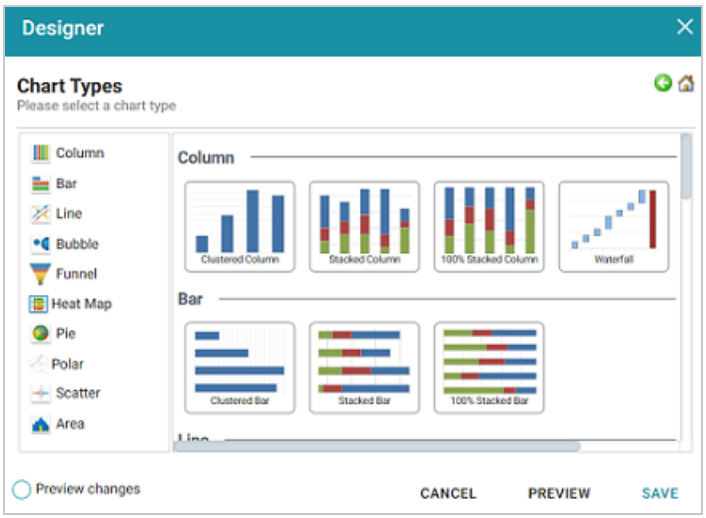


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:

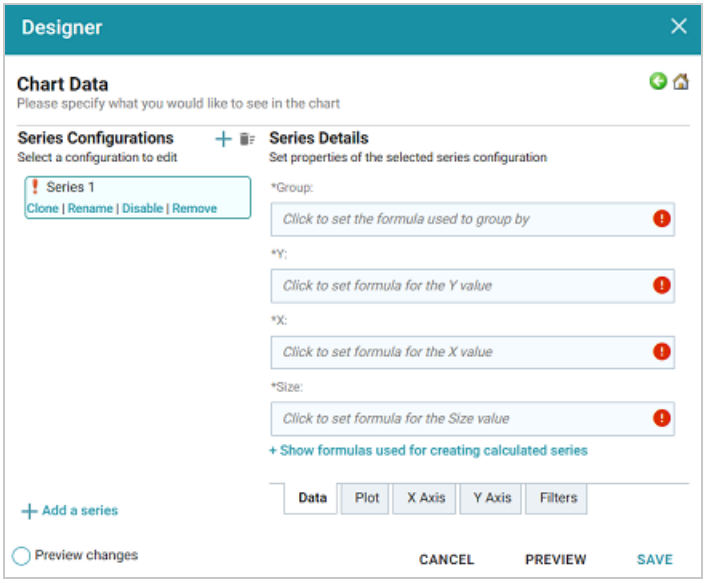


5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Bubble**, and then select the Bubble icon in the main part of the screen. Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:
   - The **Root Value** () is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Next, click the **Y** field and select the property to use for the Y axis values in the series. You can include functions to derive the values.
11. Then select the property to use for the X axis values by clicking the **X** field and choosing a property. You can also include functions.
12. Next, click the **Size** field and select the property to use for the size axis, which determines the sizes of the bubbles for the series.
13. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.

- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

14. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

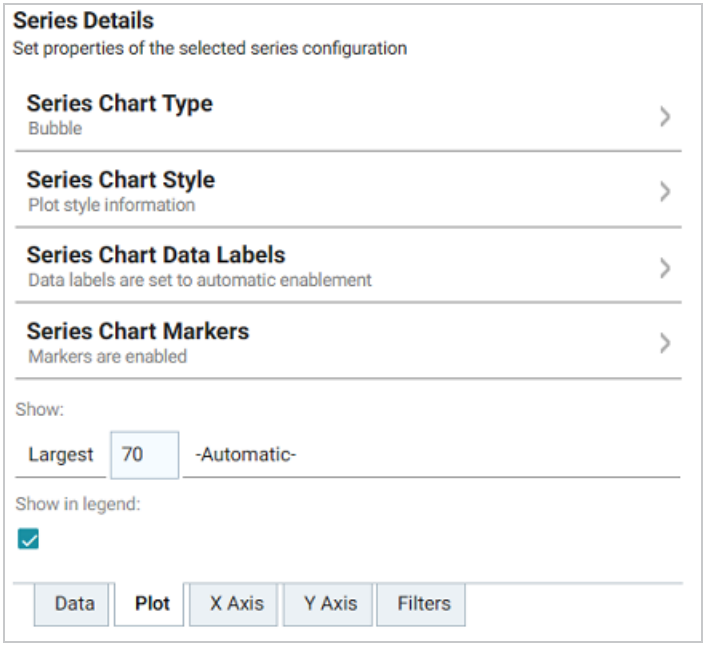
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a bubble chart.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the color of the bubbles.
Series Chart Data Labels	This option accesses the data label settings, which control the font, alignment, and font effects of the labels for the series.
Series Chart Markers	This option accesses the data marker settings. These settings control the placement and format of the data points for a series that appear when a user hovers the pointer over a bubble. For example, the image below shows a data marker:

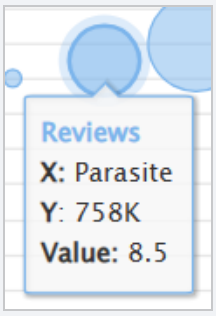
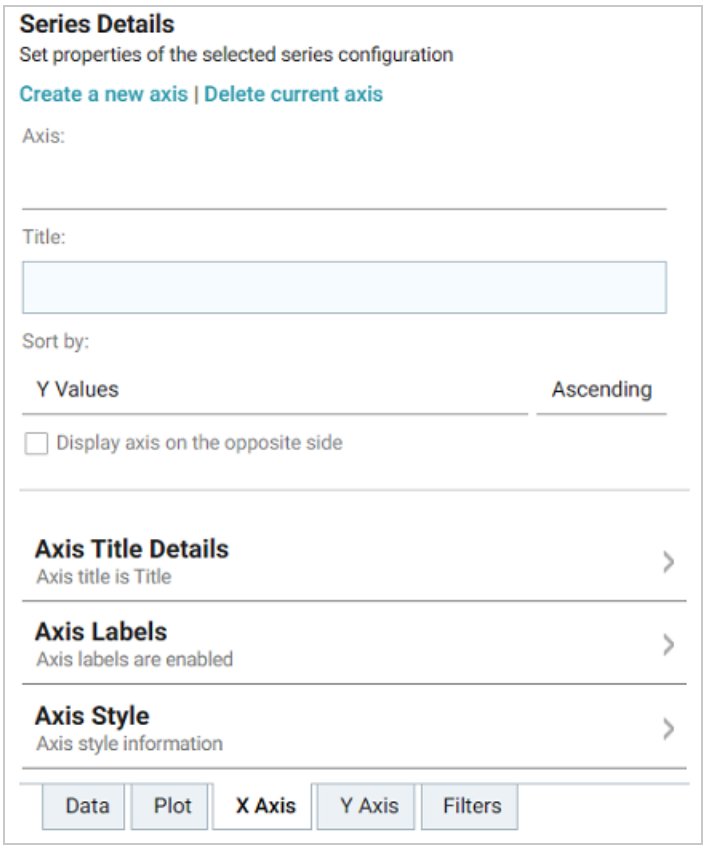
Field	Description
	
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

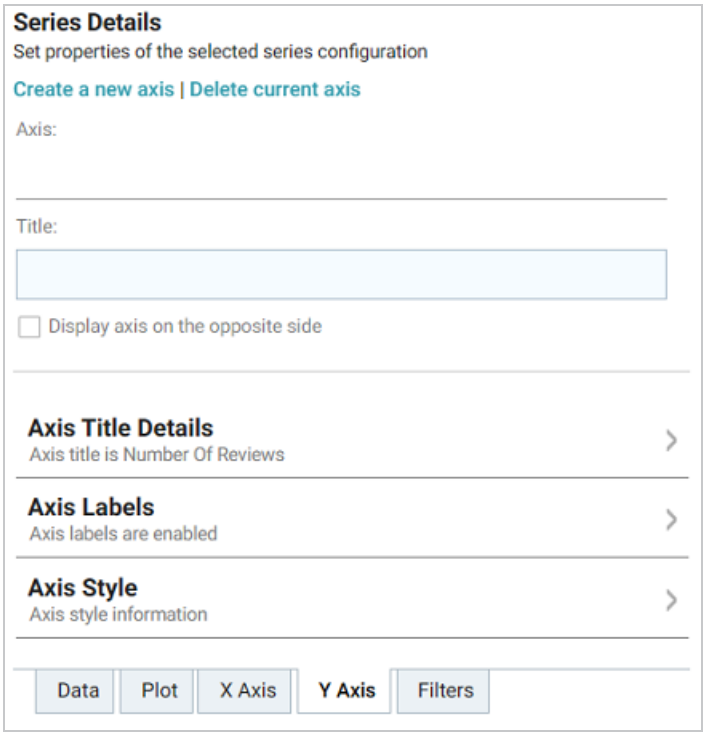


Field	Description
Axis	The Axis and Title settings are populated with the X axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Sort by	This setting controls how to sort the bubbles, by X, Y, or Size axis.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.
Axis Title Details	This option contains the X axis title settings, which control the font, alignment, and font effects of the axis title.

Field	Description
Axis Labels	This option contains the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

This tab defines the formats and labels for the Y axis values in the series.

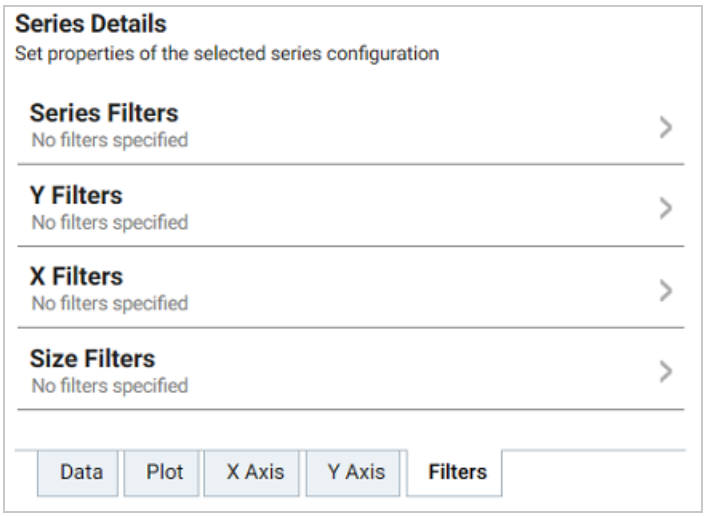


Field	Description
Axis	The Axis and Title settings are populated with the Y axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.

Field	Description
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option accesses the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.

Field	Description
Y Filters	This option can be used to define filters that apply only to the `Y` axis values for the series.
X Filters	This option can be used to define filters that apply only to the `X` axis values for the series.
Size Filters	This option can be used to define filters that apply only to the `Size` axis values for the series.

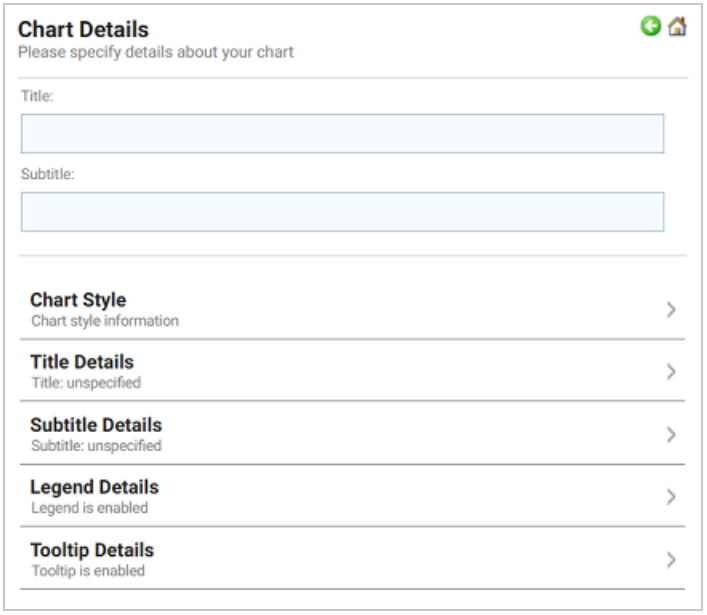
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Column Chart

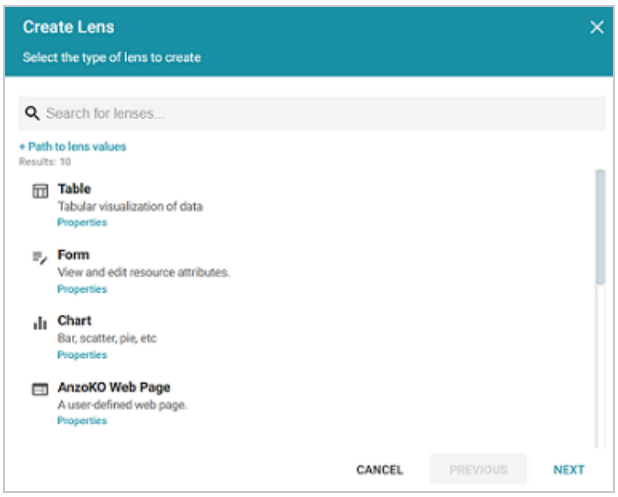
Like bar charts, column charts are also useful for comparing different sets of data or emphasizing drastic changes in a data set over time. This topic provides instructions for creating a column chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

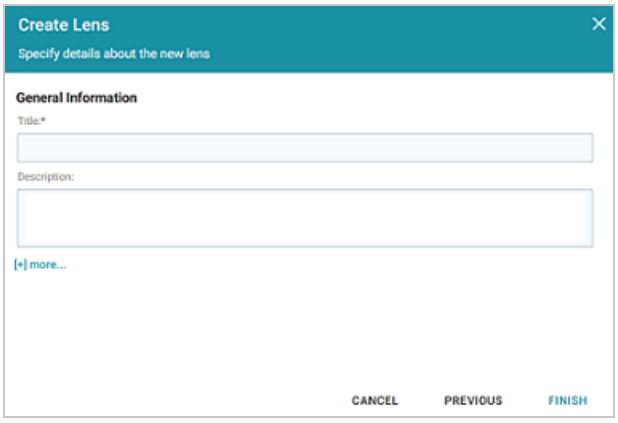
Complete the Minimum Configuration

Follow the instructions below to create a column chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

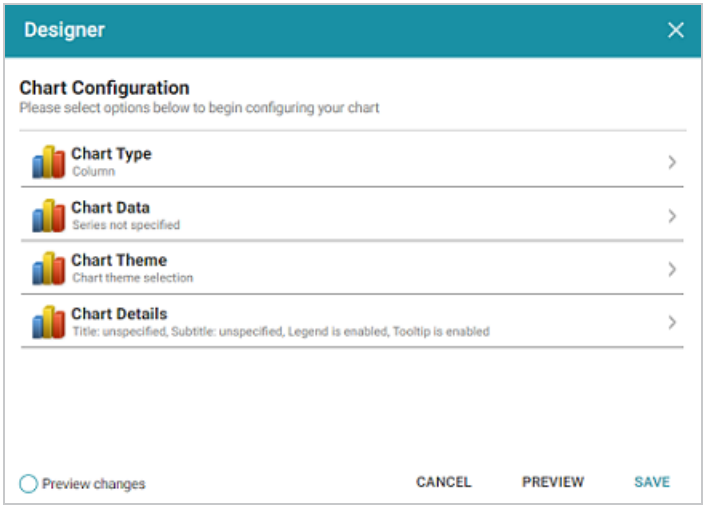


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

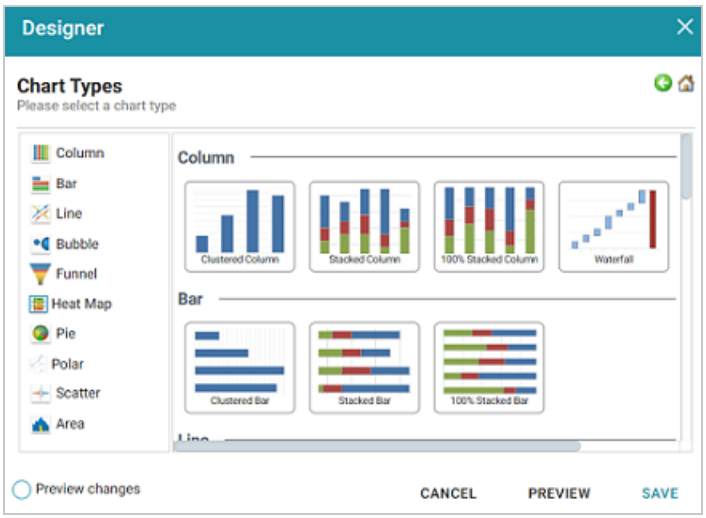


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:



5. Click **Chart Type** to open the Chart Types dialog box:

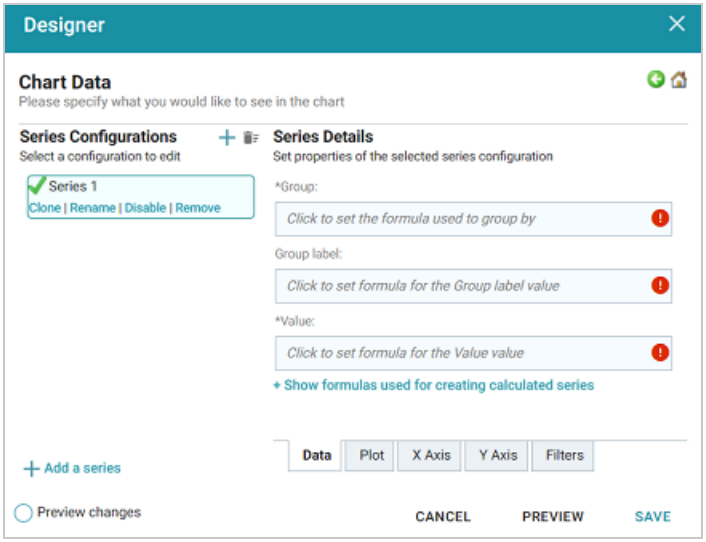


6. On the left side of the screen, select **Column**. On the right side of the screen, select the icon for the type of bar chart that you want to create. There are three types:
   - **Clustered Column**: Compares values across categories.
   - **Stacked Column**: Compares the contribution of each value to a total across categories. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.

- **100% Stacked Column**: Compares each value as a percentage of the total. Includes the option to configure multiple groups within a series to distinguish between groups of values inside the total value.

Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. These are the values for the X axis. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when

choosing a property:

- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the Y axis values. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:

- **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.
- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

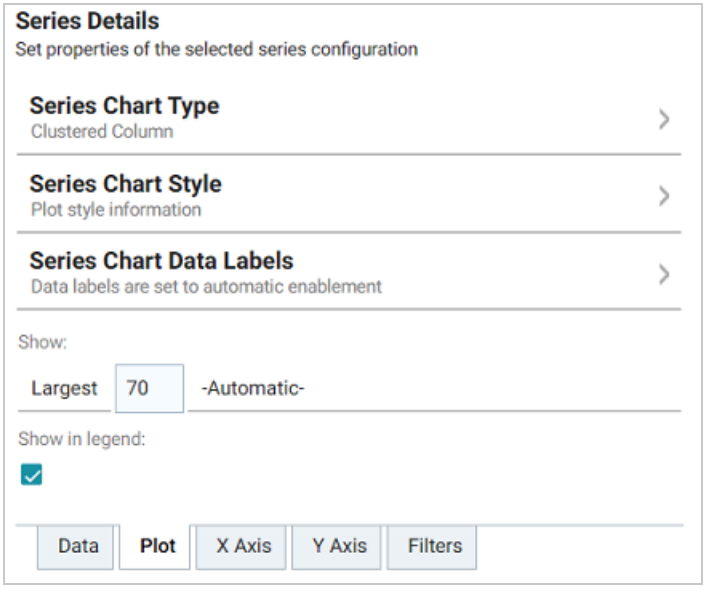
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a column chart.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

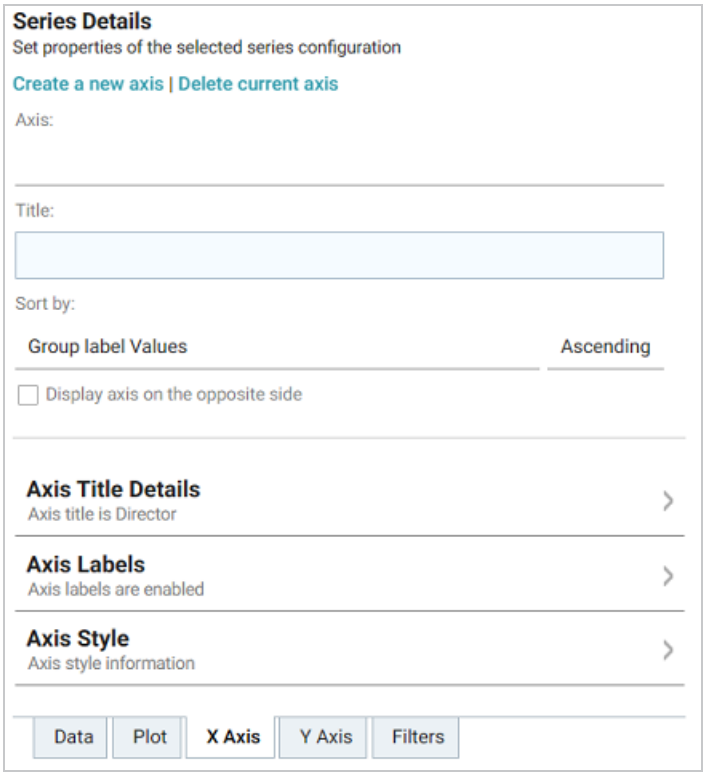
This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the thickness and color of the columns for the series.
Series Chart Data Labels	This option accesses the data label settings, which control the font, alignment, and font effects of the labels for the series.
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

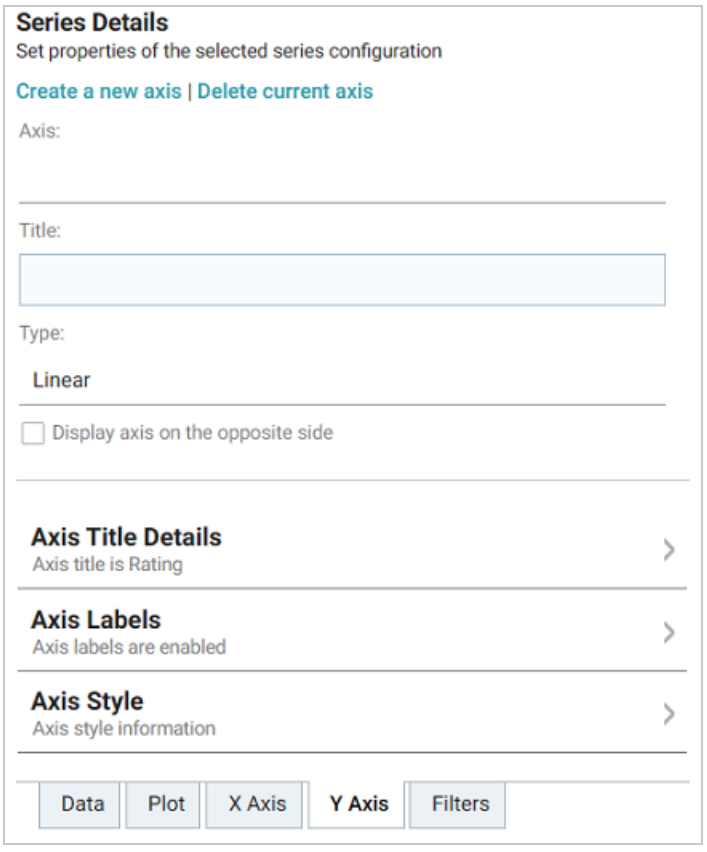


Field	Description
Axis	The Axis and Title settings are populated with the values from the `Group` and `Group Label` values from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Sort by	This setting controls how to sort the data for the X axis, by the `Group Label` values or the `Value` values.
Display axis on the opposite	This setting can be enabled to move the X axis to the opposite side of the chart.

Field	Description
side	
Axis Title Details	This option contains the X axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option accesses the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

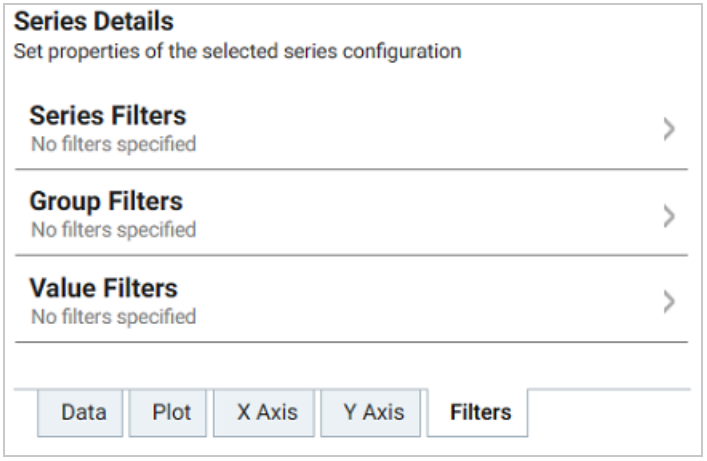
This tab defines the formats and labels for the Y axis values in the series.



Field	Description
Axis	The Axis and Title settings are populated with the `Value` from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Group Filters	This option can be used to define filters that apply only to the `Group` values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

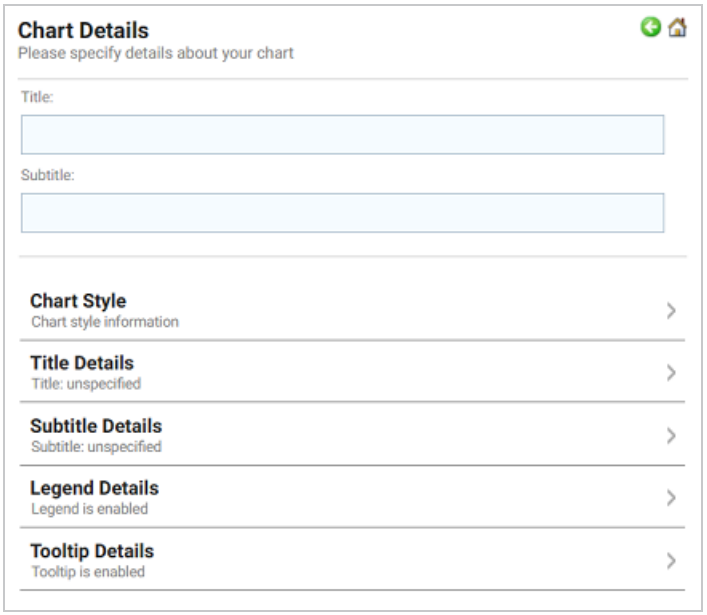
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Funnel Chart

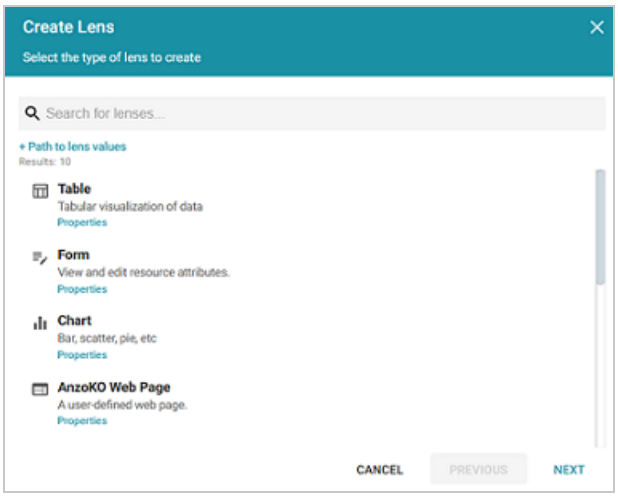
A funnel chart is useful for showing progress through stages in which the data values typically decrease. This topic provides instructions for creating a funnel chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

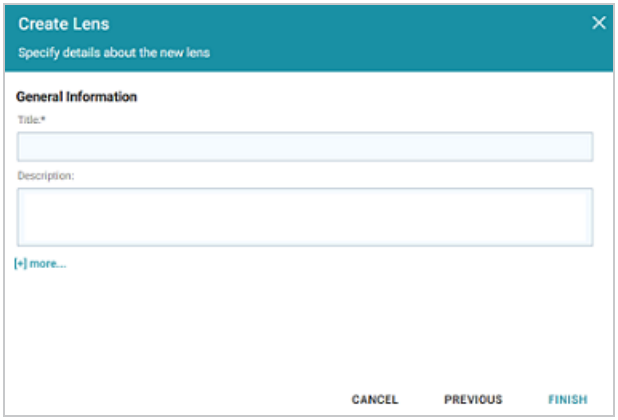
Complete the Minimum Configuration

Follow the instructions below to create a funnel chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

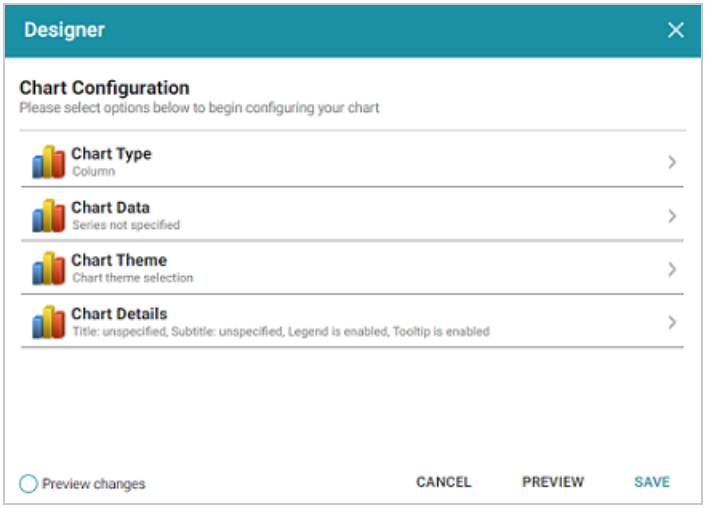


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

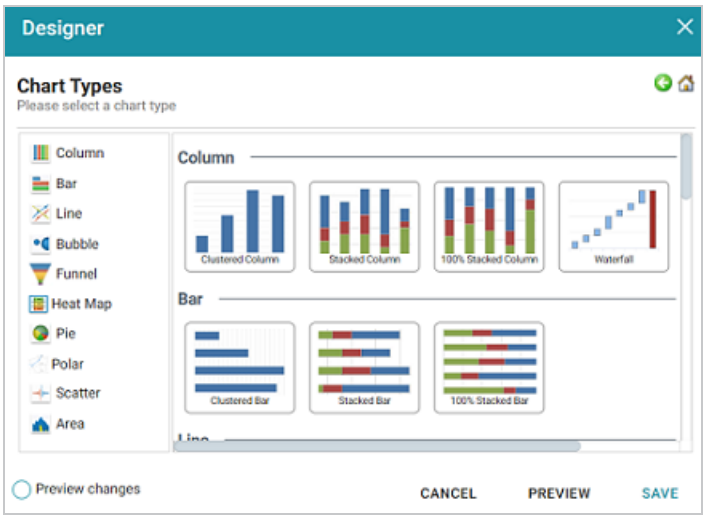


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:

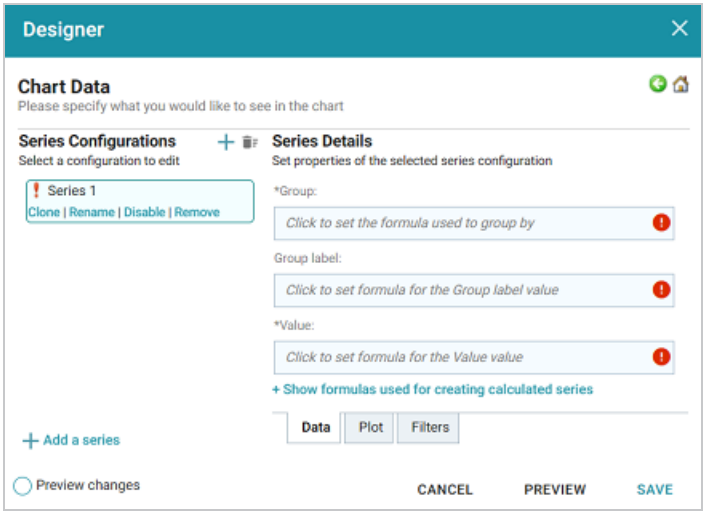


5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Funnel**, and then select the Funnel icon in the main part of the screen. Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** () is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the values in the groups. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.

- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

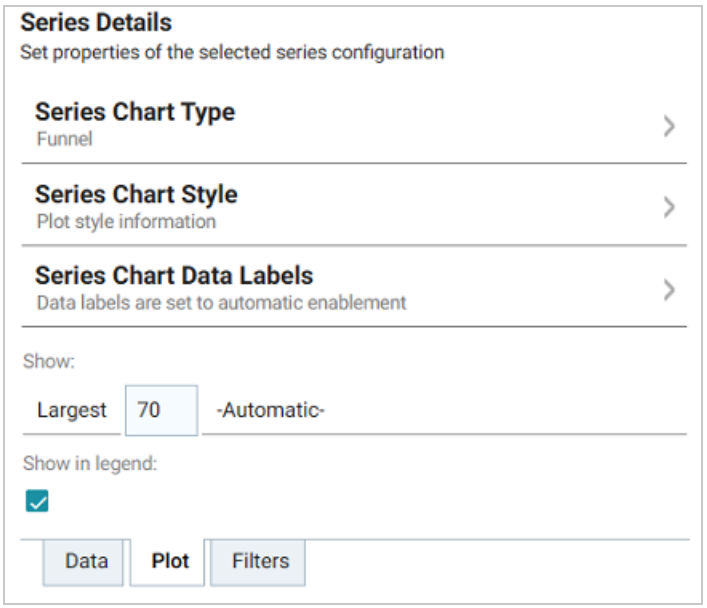
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a funnel chart.

- Chart Data: Plot Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

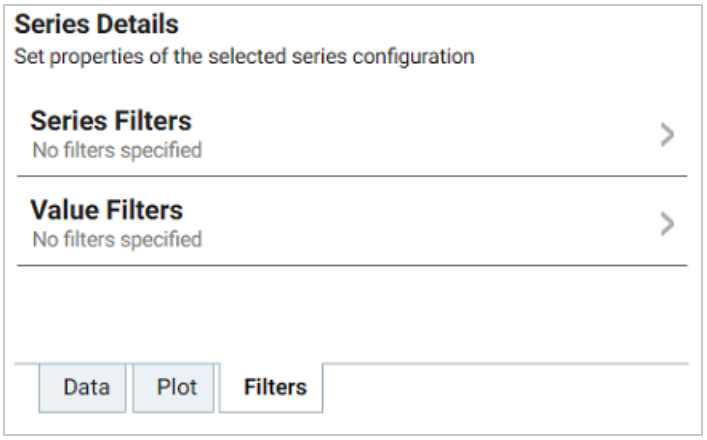
This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the color of the groups in the series.
Series Chart Data Labels	This option accesses the data label settings, which control the font, alignment, and font effects of the labels for the series.
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

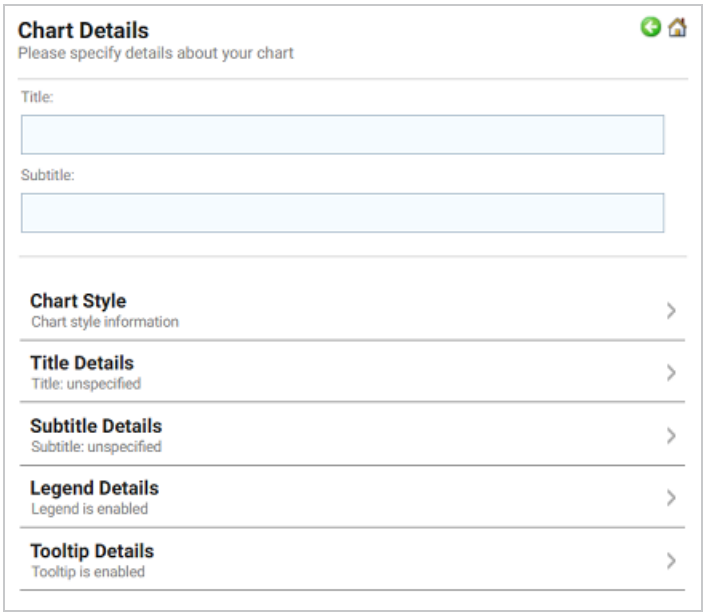
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Heat Map

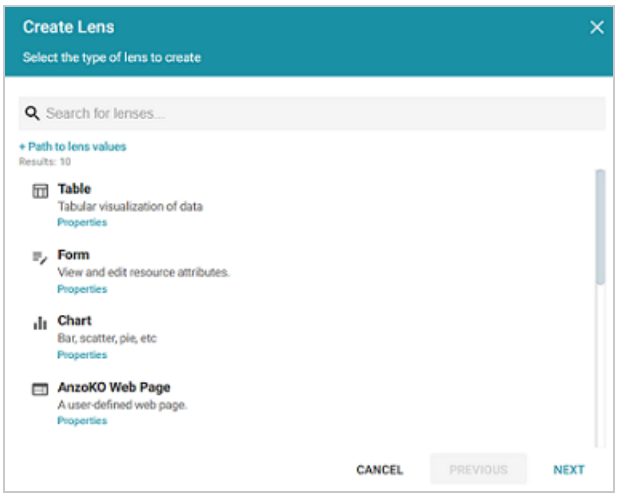
A heat map depicts values of interest across two axes. The axis variables are divided into ranges like a bar chart, and each cell's color indicates the value in the corresponding cell range. This topic provides instructions for creating a heat map with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

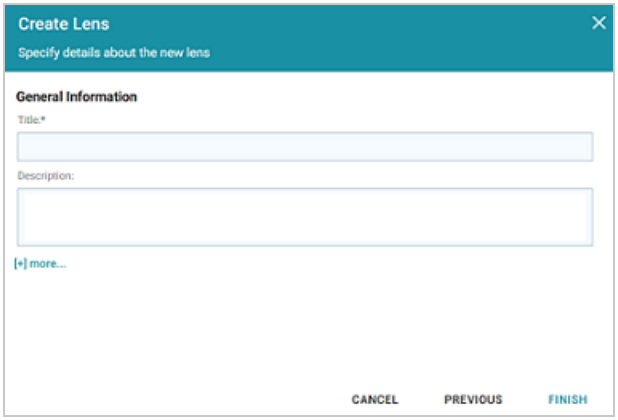
Complete the Minimum Configuration

Follow the instructions below to create a heat map. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

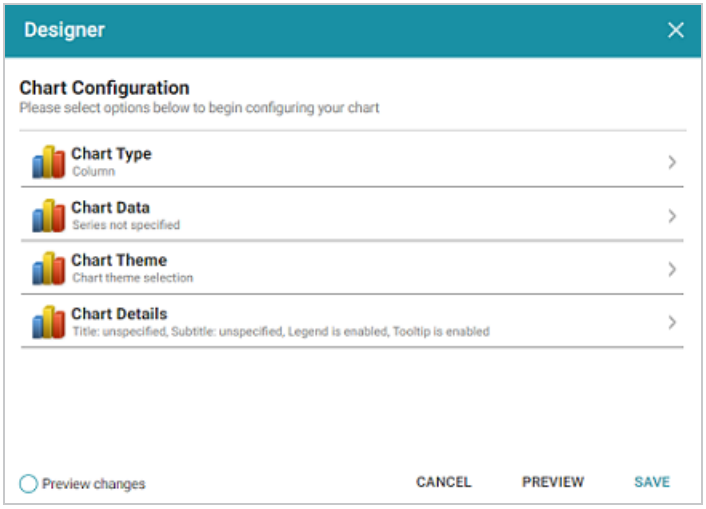


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

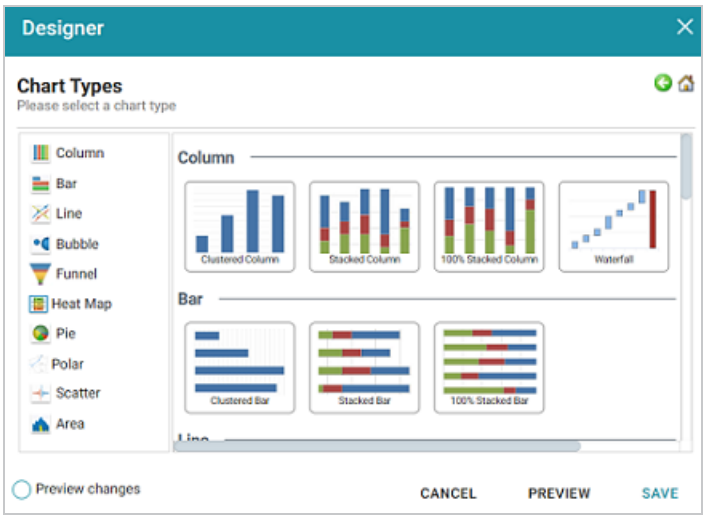


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:

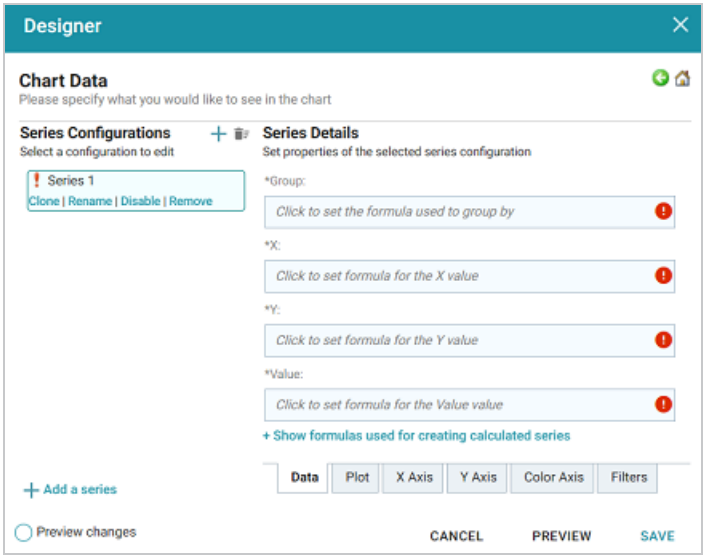


5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Heat Map**, and then select the Heat Map icon in the main part of the screen. Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:
   - The **Root Value** () is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Next, click the **X** field and select the property to use for the X axis values in the series. You can include functions to derive the values.
11. Then select the property to use for the Y axis values by clicking the **Y** field and choosing a property. You can also include functions.
12. Next, click the **Value** field and select the property to use for the value range in the series.
13. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.

- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

14. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

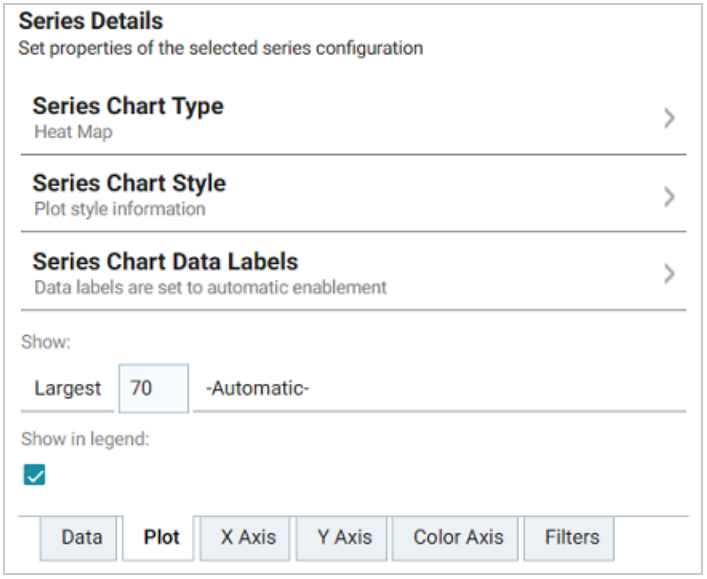
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a heat map.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Color Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

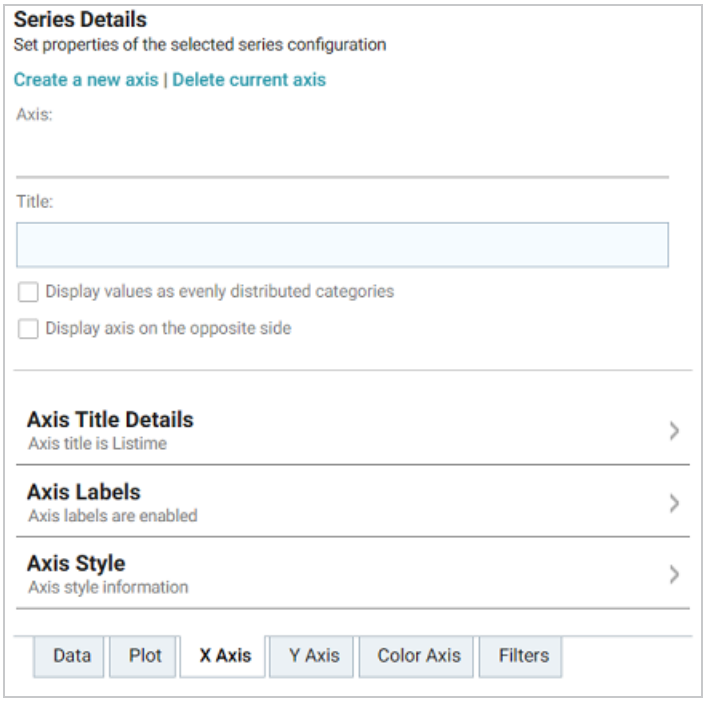
This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the color of the ranges.
Series Chart Data Labels	This option contains the data label settings, which control the font, alignment, and font effects of the labels for the series.
Show	This setting can be used to limit the data included in the series. The limit is defined based on the largest or smallest `X` axis, `Y` axis, or `Value` values.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

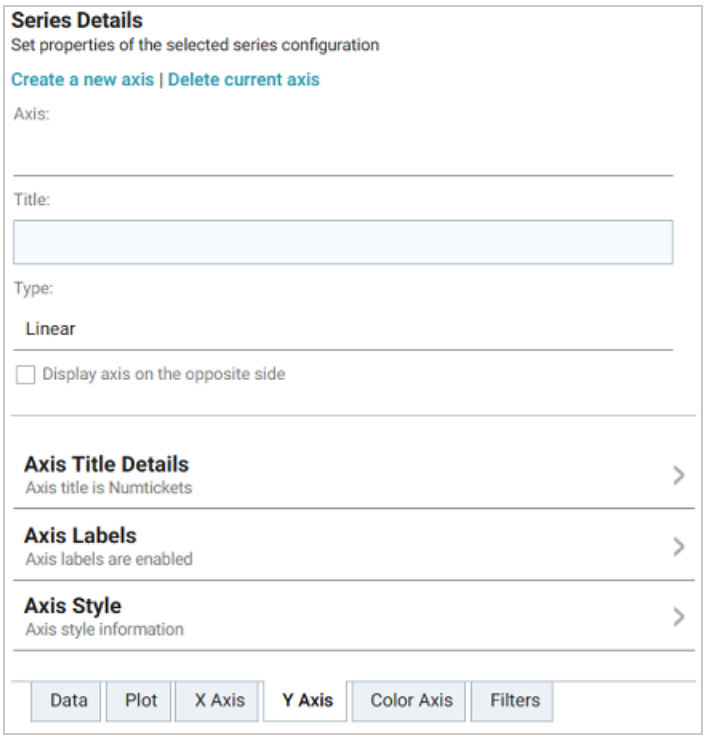


Field	Description
Axis	The Axis and Title settings are populated with the X axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Display values as evenly distributed categories	This setting controls whether to evenly distribute X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.
Axis Title Details	This option contains the X axis title settings, which control the font,

Field	Description
	alignment, and font effects of the axis title.
Axis Labels	This option accesses the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

This tab defines the formats and labels for the Y axis values in the series.

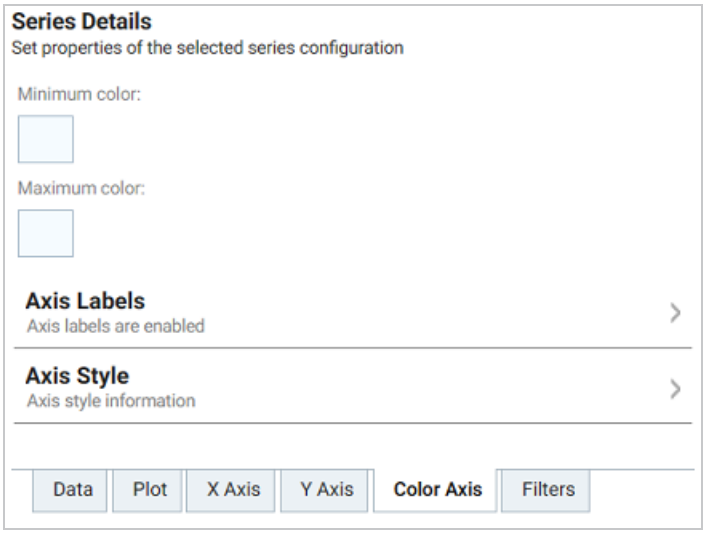


Field	Description
Axis	The Axis and Title settings are populated with the Y axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.

Field	Description
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option accesses the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Color Axis Tab

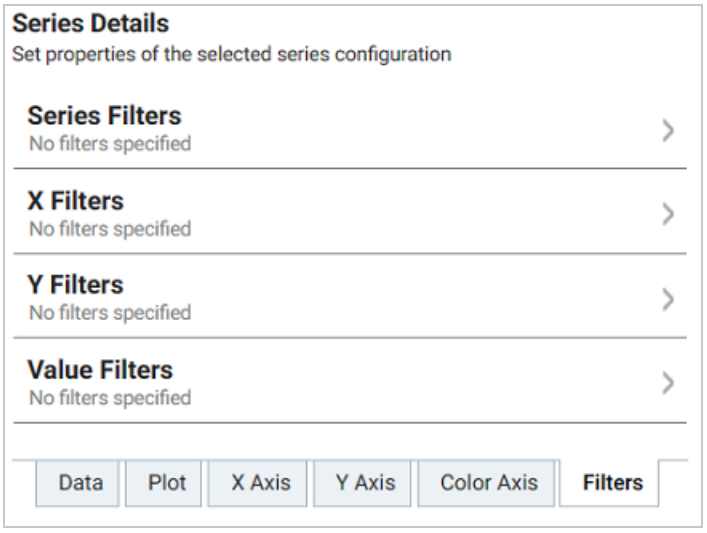
This tab defines the value range block colors and axis labels and styles.



Field	Description
Minimum color	This option defines the color for the minimum range of values.
Maximum color	This option defines the color for the maximum range of values.
Axis Labels	This option accesses the color axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the color axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.

Field	Description
X Filters	This option can be used to define filters that apply only to the `X` axis values for the series.
Y Filters	This option can be used to define filters that apply only to the `Y` axis values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

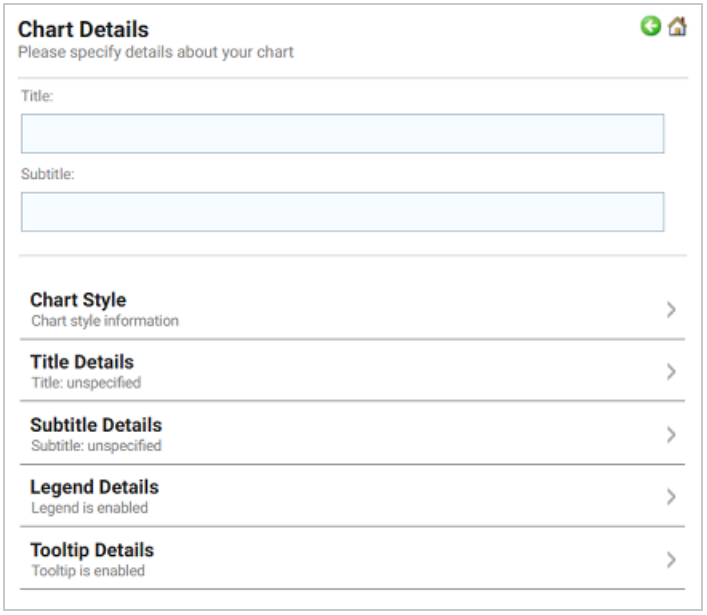
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Line Chart

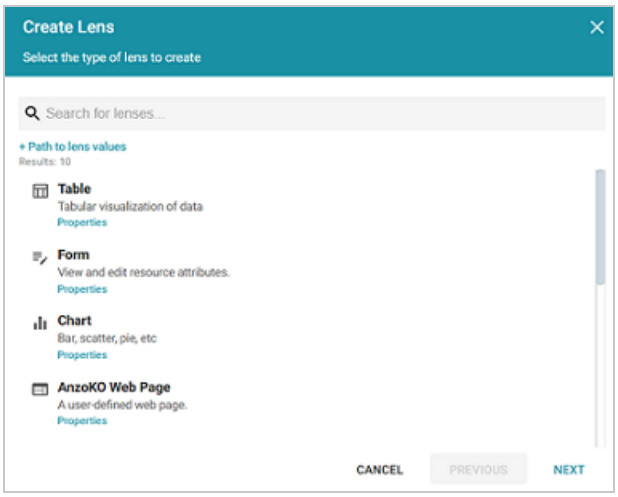
Line charts are useful for emphasizing trends in your data. This topic provides instructions for creating a line chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

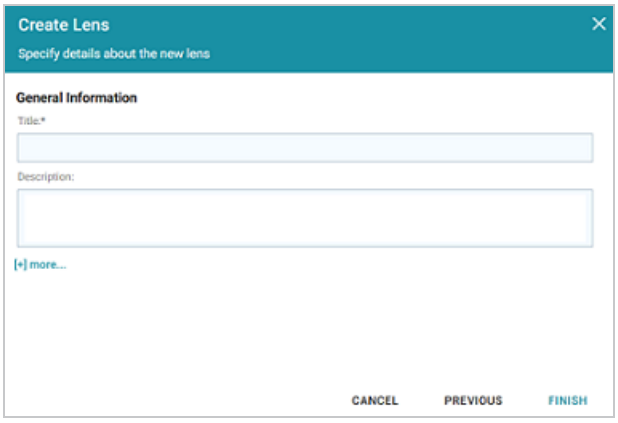
Complete the Minimum Configuration

Follow the instructions below to create a line chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

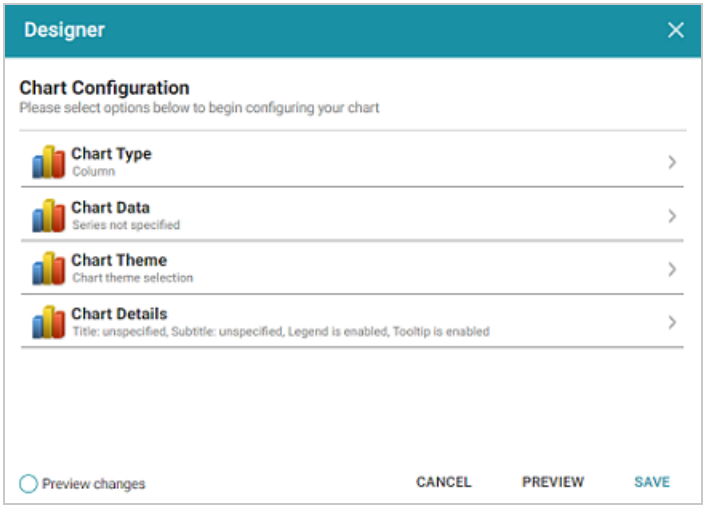


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

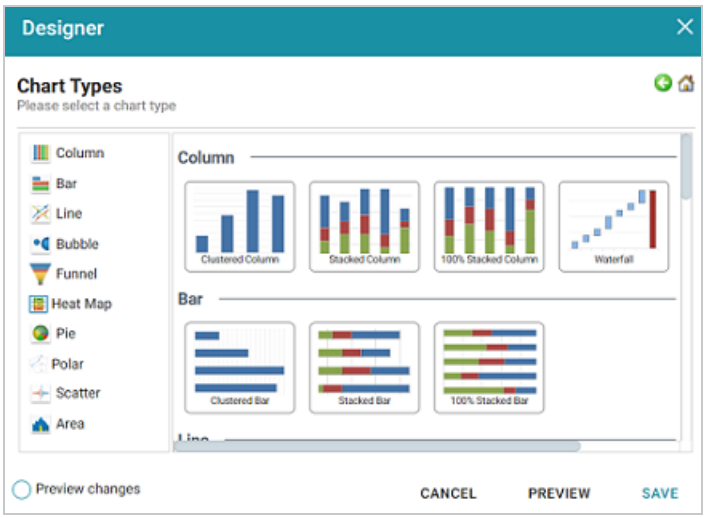


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:



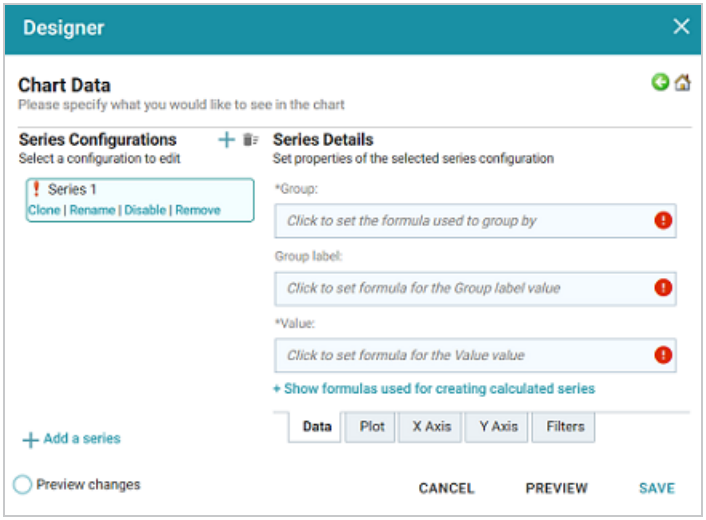
5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Line**. On the right side of the screen, select the icon for the type of line chart that you want to create. There are three types:
   - **Line**: Connects value points on the chart with straight lines.
   - **Spline**: Connects value points on the chart with curved lines.
   - **Step Line**: Connects value points on the chart with horizontal lines.

   Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



   The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon ( ). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. These are the values for the X axis. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** ( ) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the Y axis values. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.

- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

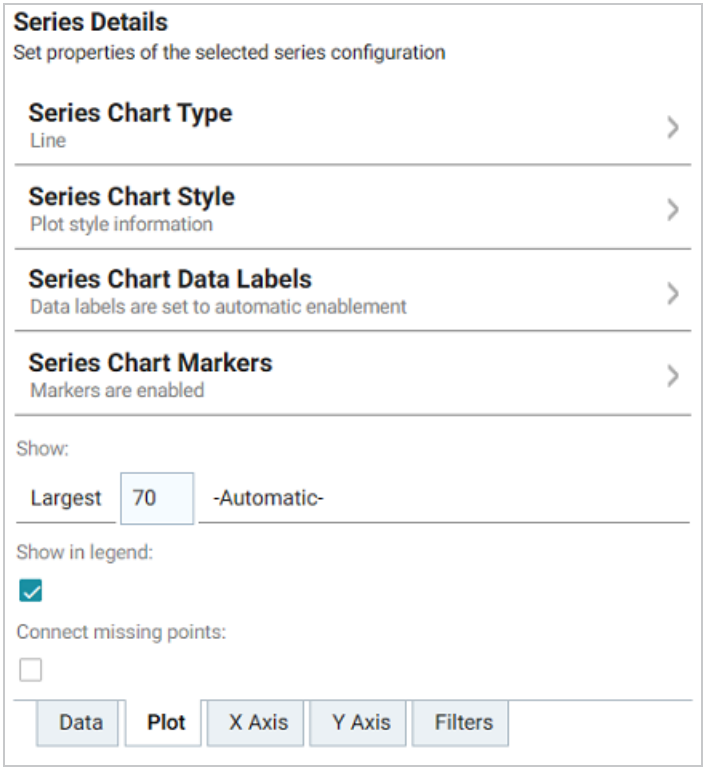
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a line chart.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the thickness and color of lines for the series.
Series Chart Data Labels	This option contains the data label settings, which control the font, alignment, and font effects of the labels for the series.
Series Chart Markers	This option contains the data marker settings. These settings control the placement and format of the data points for a series that appear when a user hovers the pointer over a point in the line. For example, the image below shows a data marker:

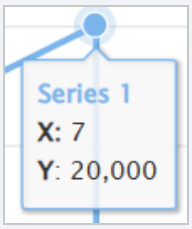
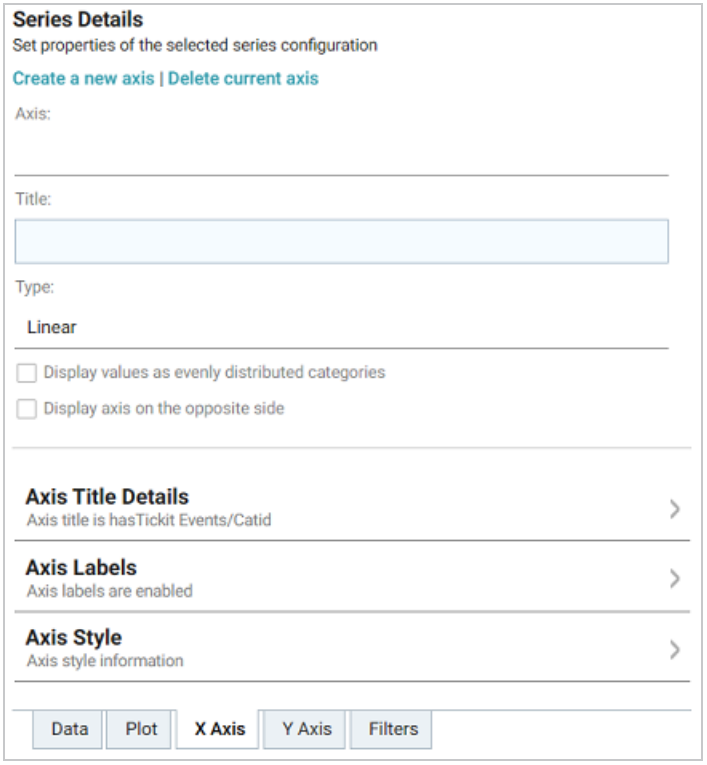
Field	Description
	
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.
Connect missing points	Selecting this option connects the series line across any missing points.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

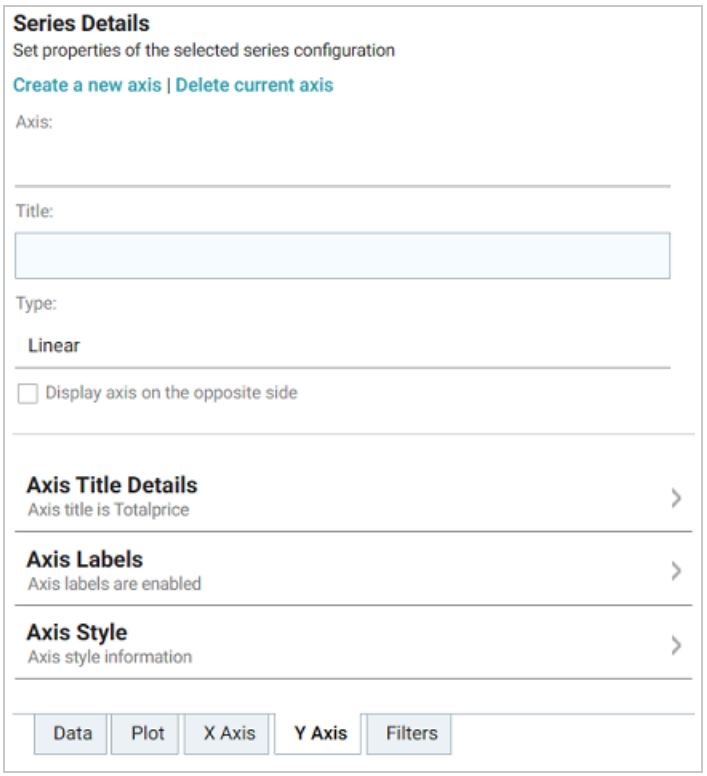


Field	Description
Axis	The Axis and Title settings are populated with the `Group` and `Group Label` values from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Type	This setting controls the scale for the X axis, linear or logarithmic.
Display values as evenly distributed categories	This setting controls whether to evenly distribute X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.
Axis Title Details	This option contains the X axis title settings, which control the font,

Field	Description
	alignment, and font effects of the axis title.
Axis Labels	This option accesses the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

This tab defines the formats and labels for the Y axis values in the series.

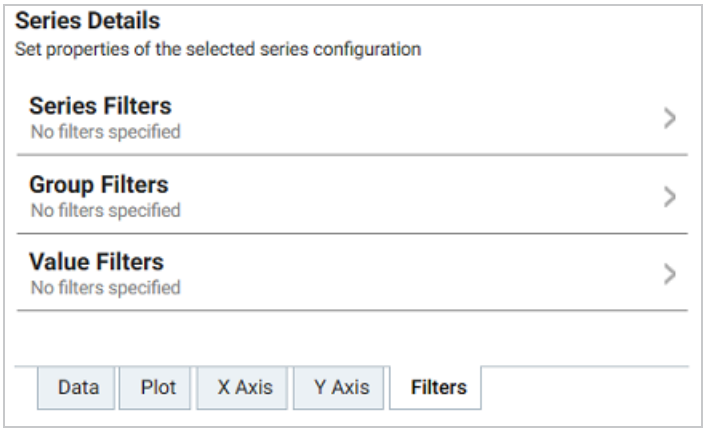


Field	Description
Axis	The Axis and Title settings are populated with the `Value` from the Data tab. If

Field	Description
	multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option accesses the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option contains the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Group Filters	This option can be used to define filters that apply only to the `Group` values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

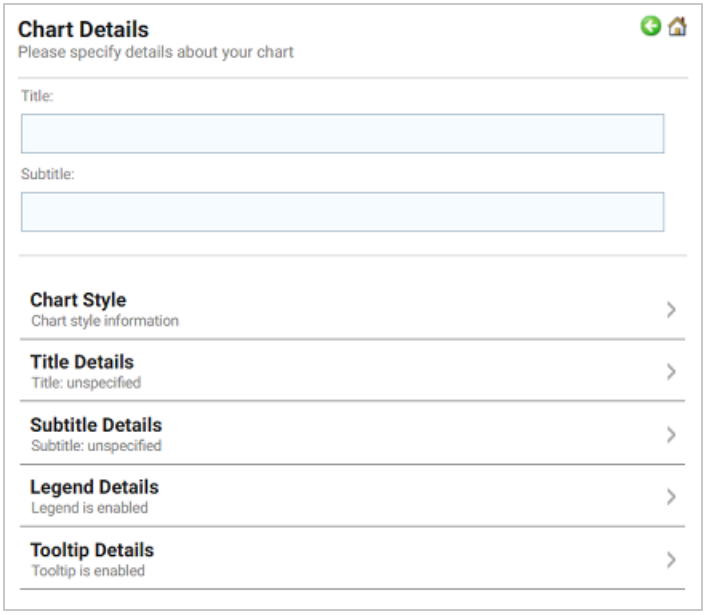
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Polar Chart

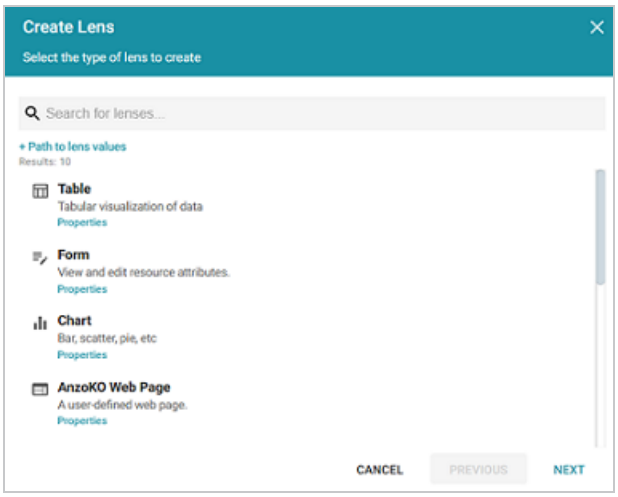
Polar or spider charts can be useful for showing similarities, differences, and outliers in your data. These charts plot one or more groups of values over common variables. The variables are axes that are arranged radially around a central point. This topic provides instructions for creating a polar chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

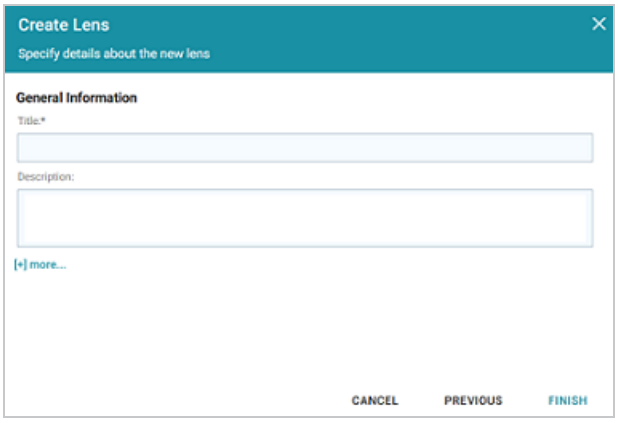
Complete the Minimum Configuration

Follow the instructions below to create a polar chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

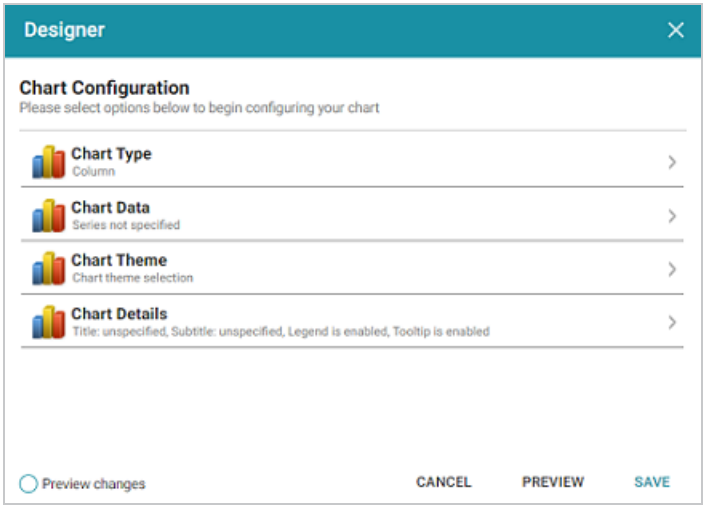


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

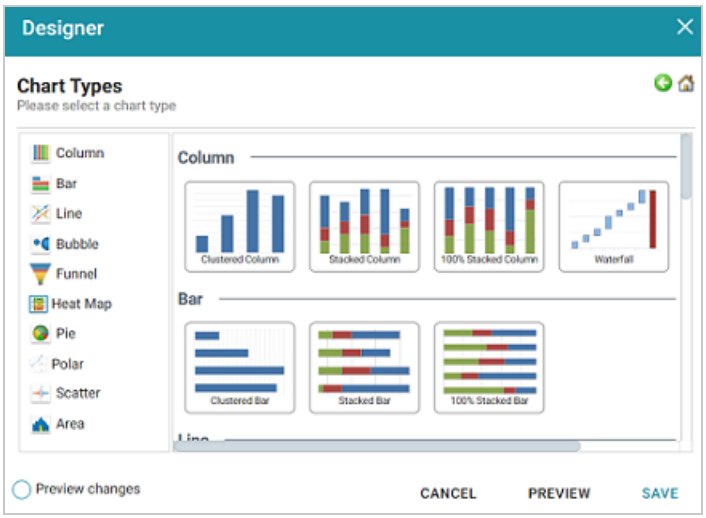


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:

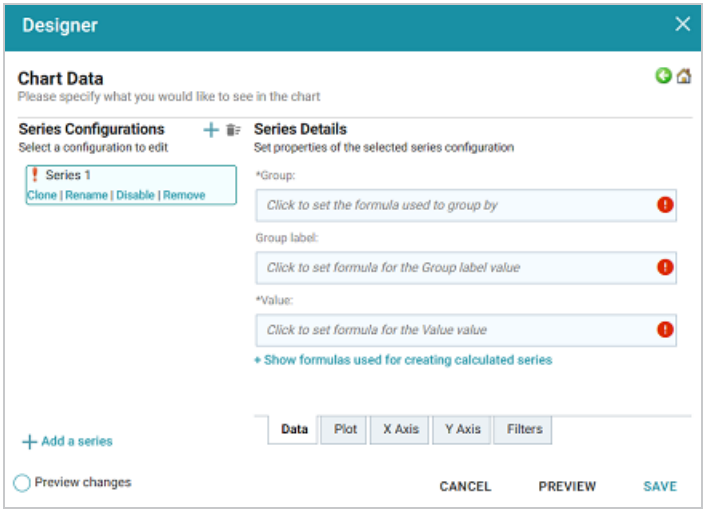


5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Polar**, and then select the Spider icon in the main part of the screen. Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon (). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. These are the values for the X axis. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** () is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Once you select the Group, the **Group Label** field is populated with the same value. This setting configures the property whose values should serve as the group label. You can edit the value if necessary.
11. Next, click the **Value** field and select the property to use for the Y axis values. You can include functions to derive the values.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.

- **Series Label**: Specifies the property whose values should serve as the Series Group label. It is typically set to the same value as `Series Group`.
- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

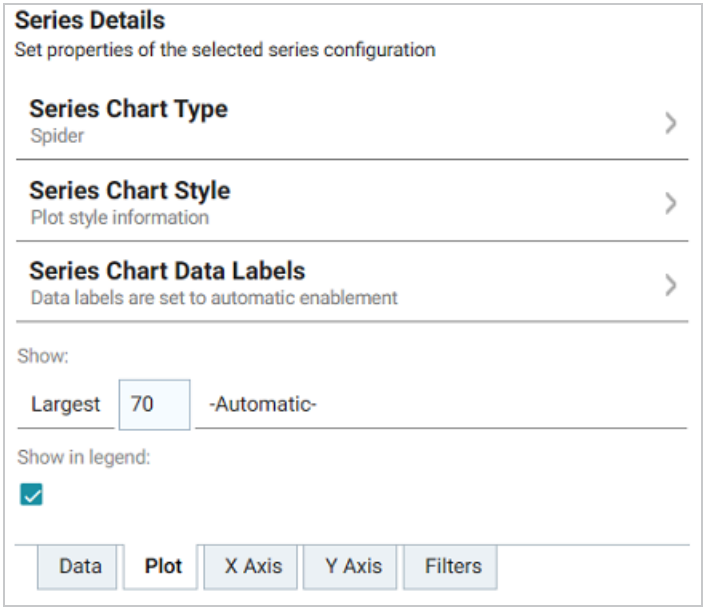
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a polar chart.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

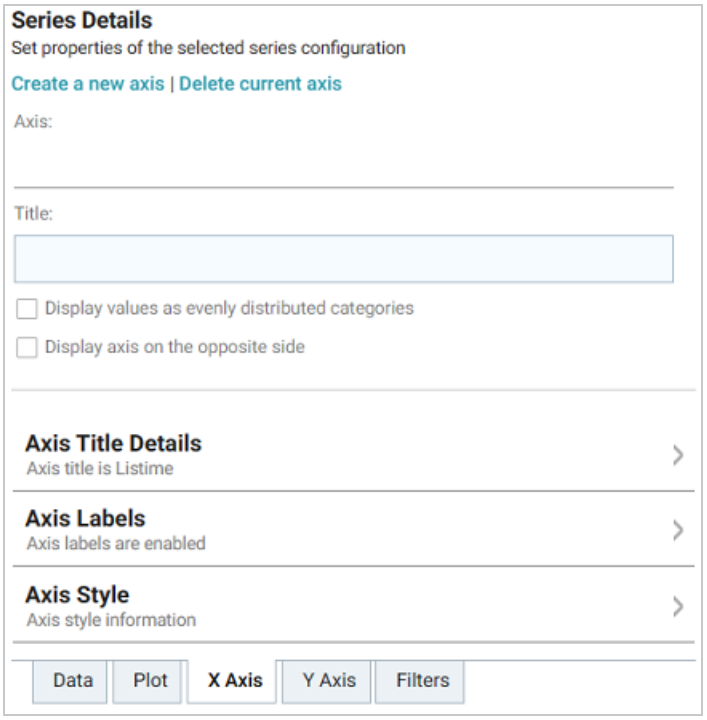
This tab contains settings that control series formatting, such as data labels, legends, and other display options.



Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style setting that controls whether the plot lines have shadows.
Series Chart Data Labels	This option accesses the data label settings, which control the font, alignment, and font effects of the labels for the series.
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest `Group Label` or `Value`.
Show in legend	This setting controls whether to show the name of the series in the legend.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

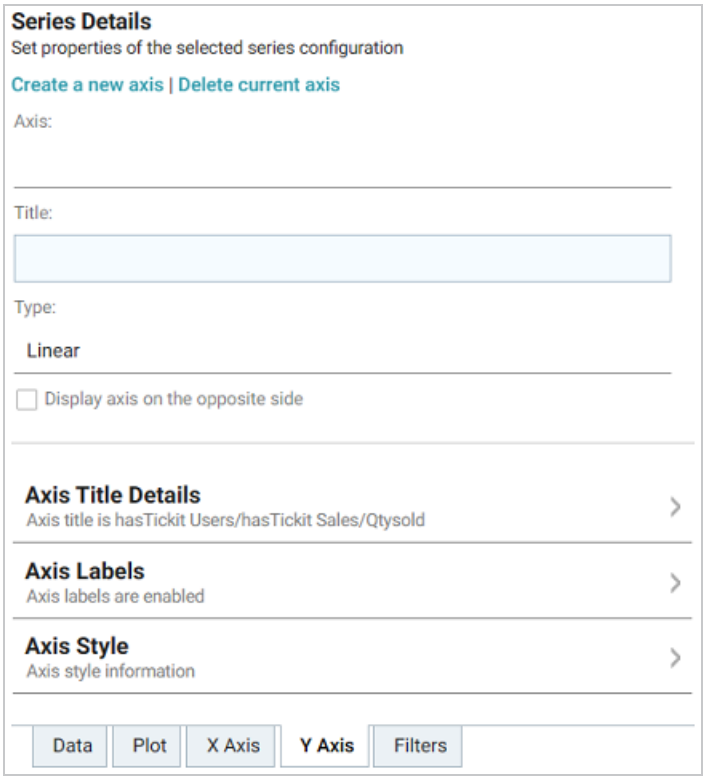


Field	Description
Axis	The Axis and Title settings are populated with the `Group` and `Group label` values from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Display values as evenly distributed categories	This setting controls whether to evenly distribute X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.

Field	Description
Axis Title Details	This option contains the X axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

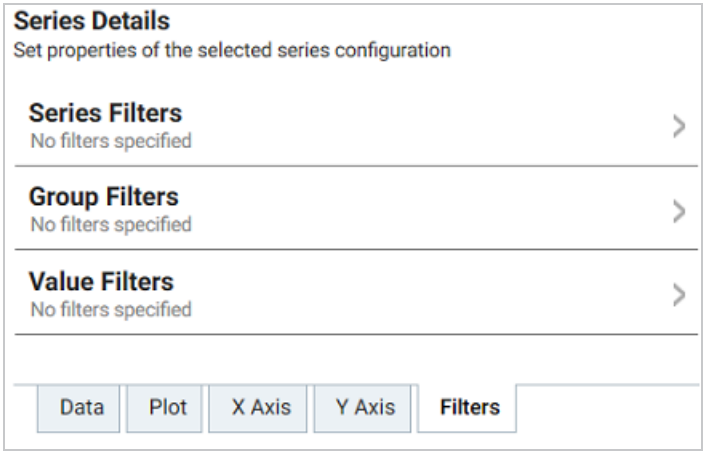
This tab defines the formats and labels for the Y axis values in the series.



Field	Description
Axis	The Axis and Title settings are populated with the `Value` from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axes, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
Group Filters	This option can be used to define filters that apply only to the `Group` values for the series.
Value Filters	This option can be used to define filters that apply only to the `Value` values for the series.

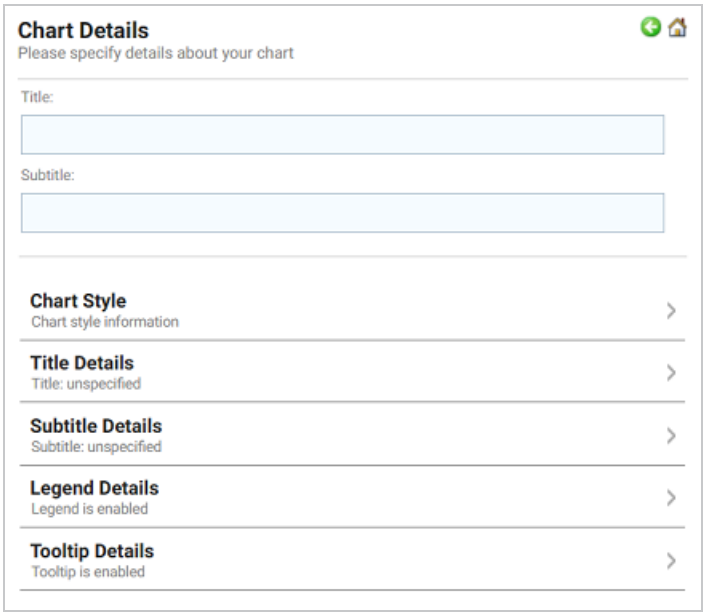
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



Create a Scatter Chart

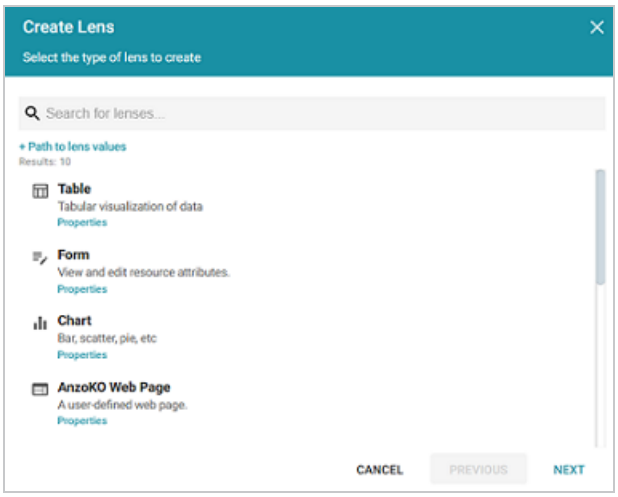
Scatter charts are also useful for observing relationships between variables, patterns, or outliers in your data. The position of dots on the horizontal and vertical axes represent values for individual data points. This topic provides instructions for creating a scatter chart with minimal configuration. Descriptions of all of the available configuration options are included below the steps.

- Complete the Minimum Configuration
- Optional Configuration Settings

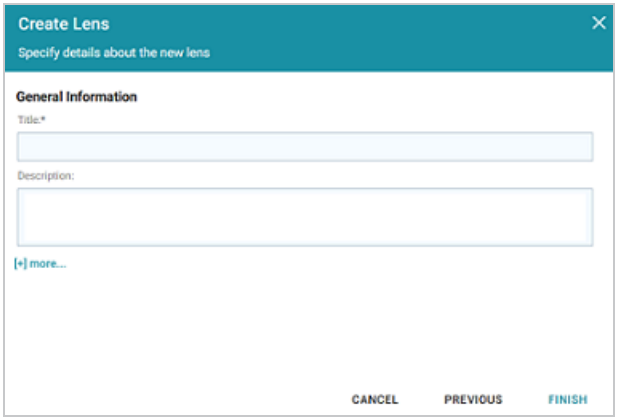
Complete the Minimum Configuration

Follow the instructions below to create a scatter chart. The instructions guide you through completing the minimum configuration needed to display your data in the chart. Additional, optional configuration settings are described in Optional Configuration Settings.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

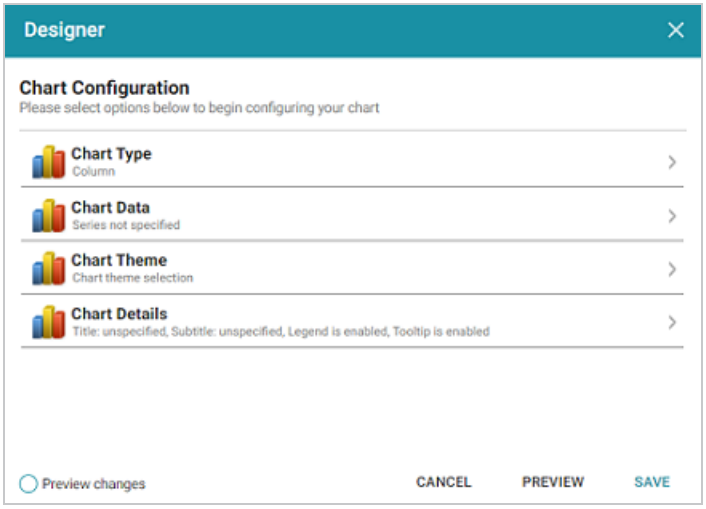


2. On the Create Lens dialog box, select **Chart**, and then click **Next**. Anzo displays the General Information dialog box.

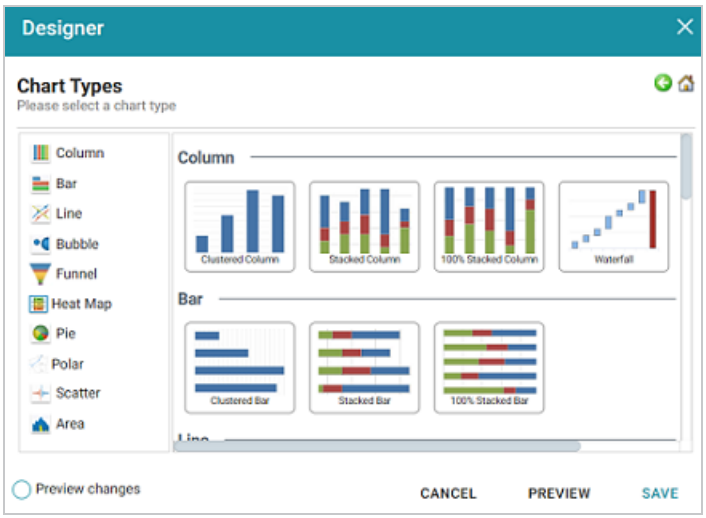


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed:

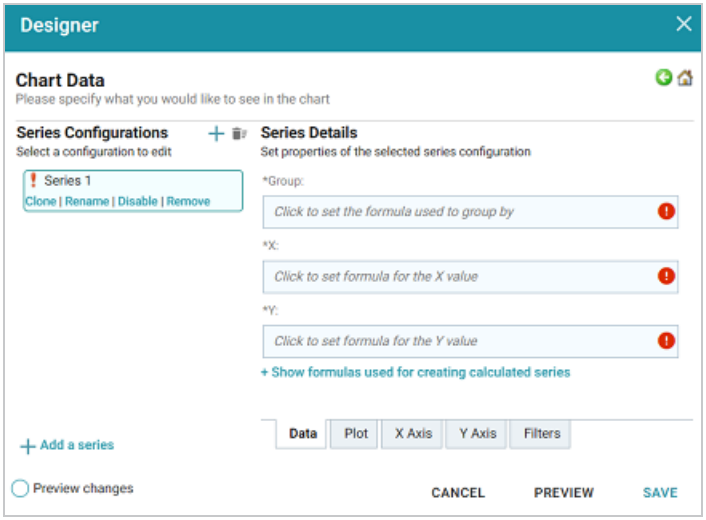


5. Click **Chart Type** to open the Chart Types dialog box:



6. On the left side of the screen, select **Scatter**, and then select the Scatter icon in the main part of the screen. Once the type is selected, you are returned to the Chart Configuration screen, and the Chart Type value is set to the type that you chose.

7. Next, select **Chart Data** to add the data that will populate the chart.



The **Series Configurations** section of the screen contains settings to manage each series. One series is created by default. A series is a set of data to display on the chart, for example a line on a line or area chart or one set of columns on a column or bar chart. The details for the selected series appear in **Series Details**, which contains settings to define the details of the selected series, such as the data to display as well as formatting and labels. From Series Configurations, you can clone, rename, disable (remove the series from the chart without deleting it), or remove a series. You can add a series by clicking the plus icon (+) or delete all series by clicking the trashcan icon ( ). Removing a series cannot be undone.

8. Under Series Configurations, click **Rename** under Series 1 and give the series a meaningful name that describes the data that will populate the chart.

9. Under Series Details, click the **Group** field and select the property to use for grouping the data in the series. You can use functions to derive the values for the groups. The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** ( ) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

10. Next, click the **X** field and select the property to use for the X axis values in the series. You can include functions to derive the values.
11. Then select the property to use for the Y axis values by clicking the **Y** field and choosing a property. You can also include functions.
12. If you want to add a group for the series, you can click **Show formulas used for creating calculated series** and add values for the following fields:
    - **Series Group**: Specifies a property to use for grouping data in addition to the `Group` value.
    - **Series Label**: Specifies the property whose values should serve as the Series Group

label. It is typically set to the same value as `Series Group`.

- **Series Sort Direction**: Specifies the sort direction for the series groups.

13. You can click **Save** any time to render on the dashboard the data that you have configured so far. You can add another series and repeat the steps above. You can also configure several more options for the chart by changing the optional parameters. The additional tabs and options are described below in Optional Configuration Settings.

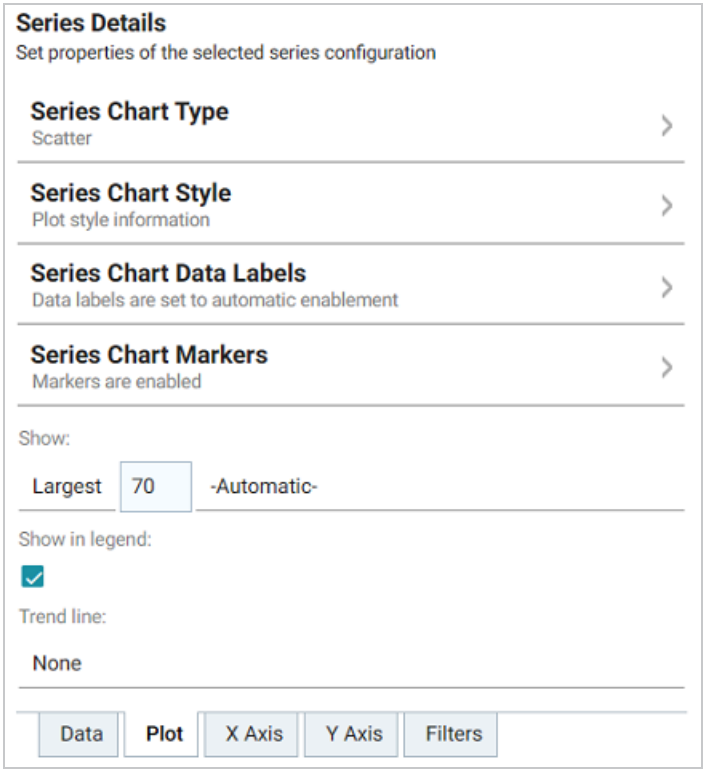
Optional Configuration Settings

This section describes each of the tabs and additional settings that are available for customizing a scatter chart.

- Chart Data: Plot Tab
- Chart Data: X Axis Tab
- Chart Data: Y Axis Tab
- Chart Data: Filters Tab
- Chart Theme
- Chart Details

Chart Data: Plot Tab

This tab contains settings that control series formatting, such as data labels, legends, and other display options.

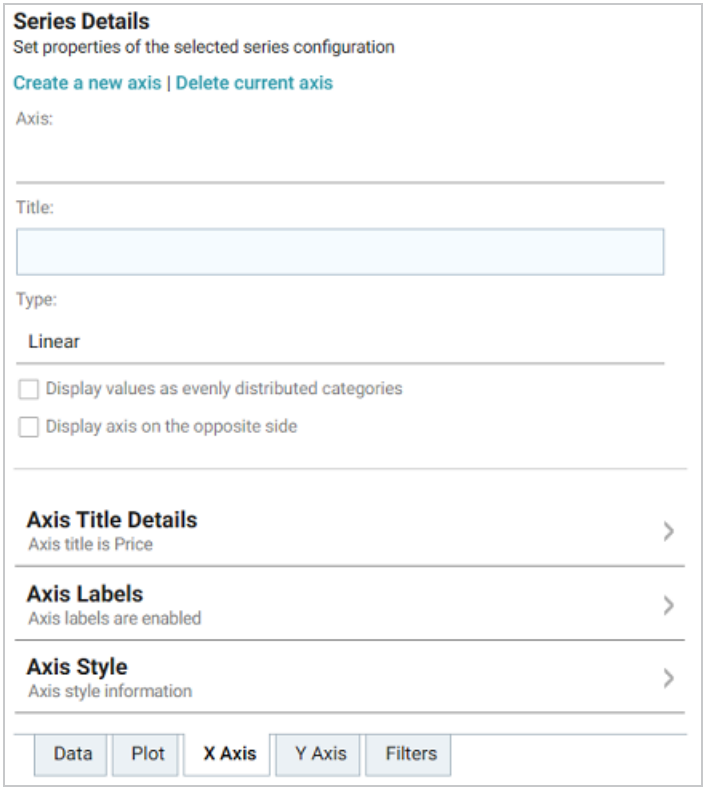


Field	Description
Series Chart Type	This option opens the Chart Types selection screen where you can select a different chart type for the selected series. The selection must be compatible with the types selected for any other series in the chart, however, or the incompatible series will fail validation and not be displayed on the chart until it is corrected.
Series Chart Style	This option contains the plot style settings that control the color of the dots for the series.
Series Chart Data Labels	This option contains the data label settings, which control the font, alignment, and font effects of the labels for the series.
Series Chart Markers	This option accesses the data marker settings. These settings control the placement and format of the data points for a series that appear when a user hovers the pointer over a dot in the chart. For example, the image below shows a data marker:

Field	Description
	
Show	This setting can be used to limit the data included in the series. The limit can be defined by the largest or smallest X or Y axis values.
Show in legend	This setting controls whether to show the name of the series in the legend.
Trend line	This setting controls whether to display a linear or exponential trend line for the series.

Chart Data: X Axis Tab

This tab defines the formats and labels for the X axis values in the series.

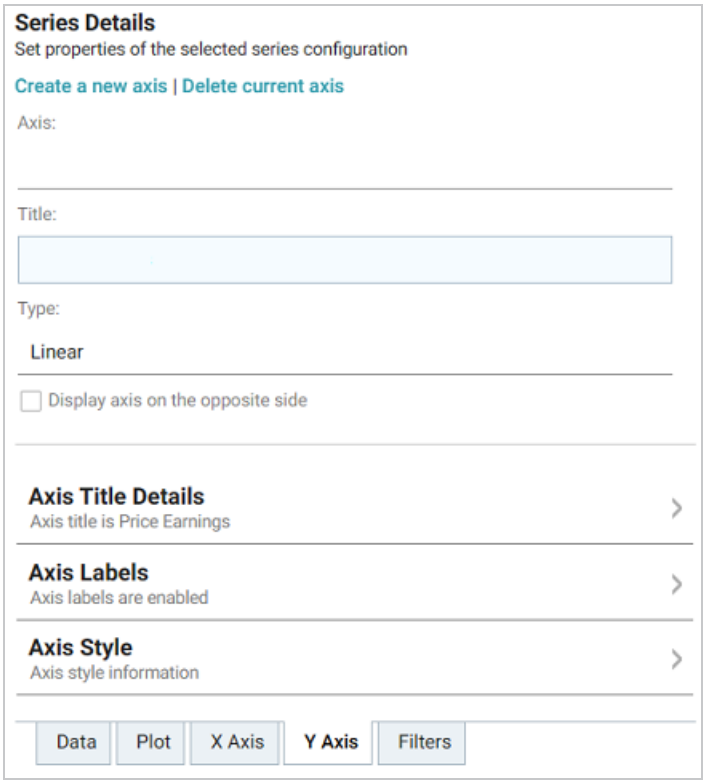


Field	Description
Axis	The Axis and Title settings are populated with the X axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the X axis.
Title	This setting defines the title for the X axis.
Type	This setting controls the scale for the X axis, linear or logarithmic.
Display values as evenly distributed categories	This setting controls whether to evenly distribute X axis values.
Display axis on the opposite side	This setting can be enabled to move the X axis to the opposite side of the chart.

Field	Description
Axis Title Details	This option contains the X axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the X axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the X axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Y Axis Tab

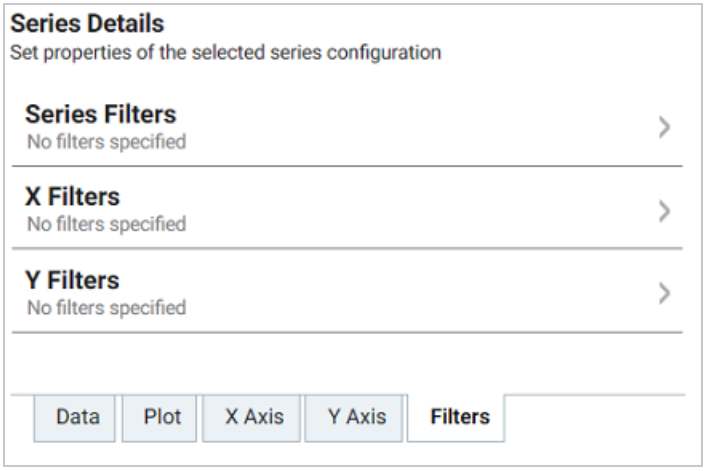
This tab defines the formats and labels for the Y axis values in the series.



Field	Description
Axis	The Axis and Title settings are populated with the `Y` axis value from the Data tab. If multiple axes exist, you can selected a different value to use for the Y axis.
Title	This setting defines the title for the Y axis.
Type	This setting controls the scale for the Y axis, linear or logarithmic.
Display axis on the opposite side	This setting can be enabled to move the Y axis to the opposite side of the chart.
Axis Title Details	This option contains the Y axis title settings, which control the font, alignment, and font effects of the axis title.
Axis Labels	This option contains the Y axis label settings, which control the font, alignment, and font effects of the labels for the axis.
Axis Style	This option accesses the Y axis style settings. These settings control the minimum and maximum values for the axis and the style and position of the grid lines and tick marks.

Chart Data: Filters Tab

This tab defines any filters to apply to the series.



Field	Description
Series Filters	This option can be used to define filters that apply to the entire series.
X Filters	This option can be used to define filters that apply only to the X axis values for the series.
Y Filters	This option can be used to define filters that apply only to the Y axis values for the series.

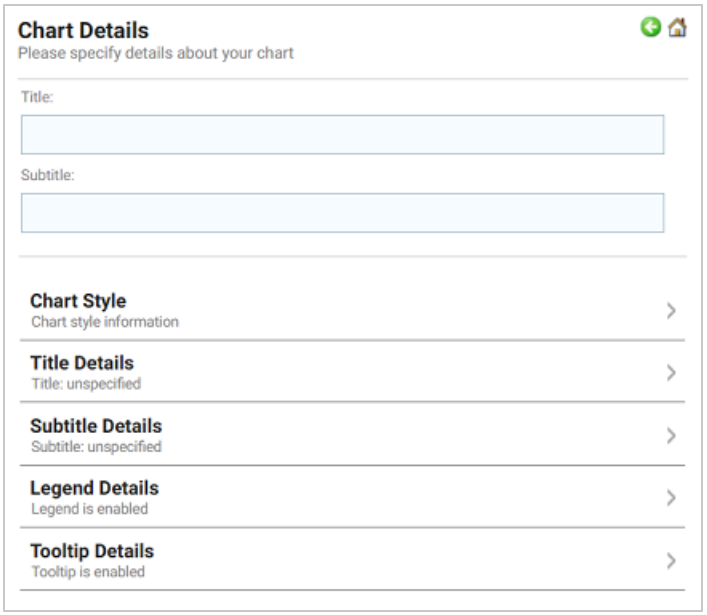
Chart Theme

This setting presents options for configuring the theme or color scheme for the chart.



Chart Details

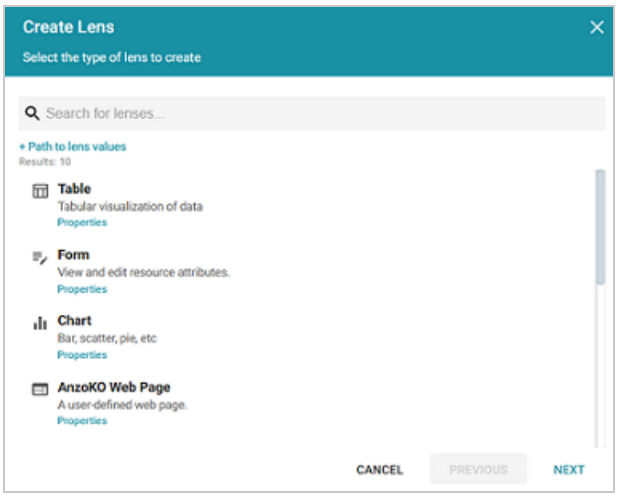
This option offers finer-grained customization settings than the Chart Theme. You can further customize the chart design by adding details such as a chart title and subtitle. You can also modify chart-level styles and fonts as well as legend and tooltip formats.



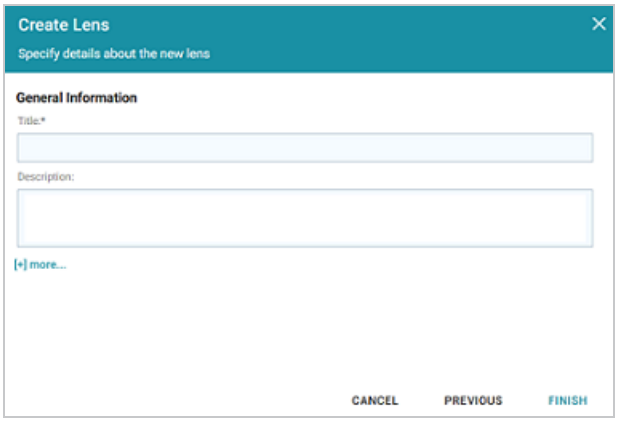
Creating a Drill Down Lens

Drill down lenses combine other lenses into a hierarchical interface. Clicking on an object in one lens opens the next lens in successive order. Follow the steps below to create a drill down lens.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

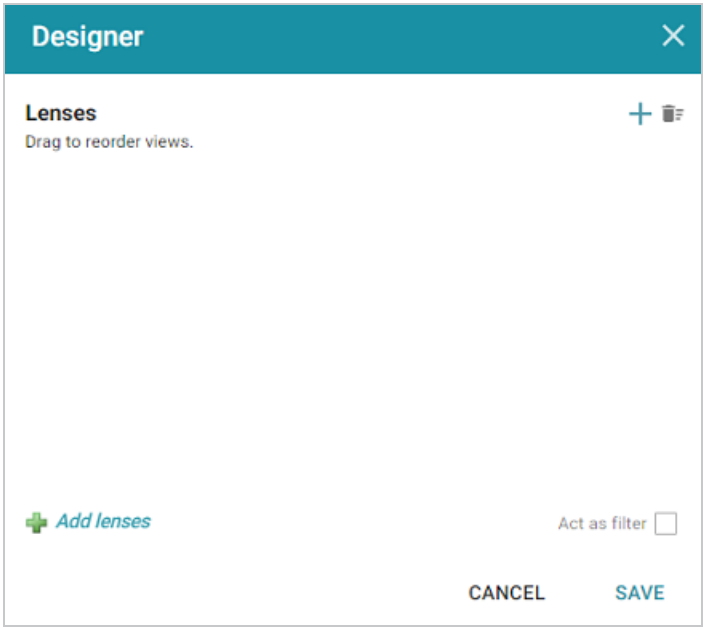


2. On the Create Lens dialog box, select **Drill Down**, and then click **Next**. Anzo displays the General Information dialog box.

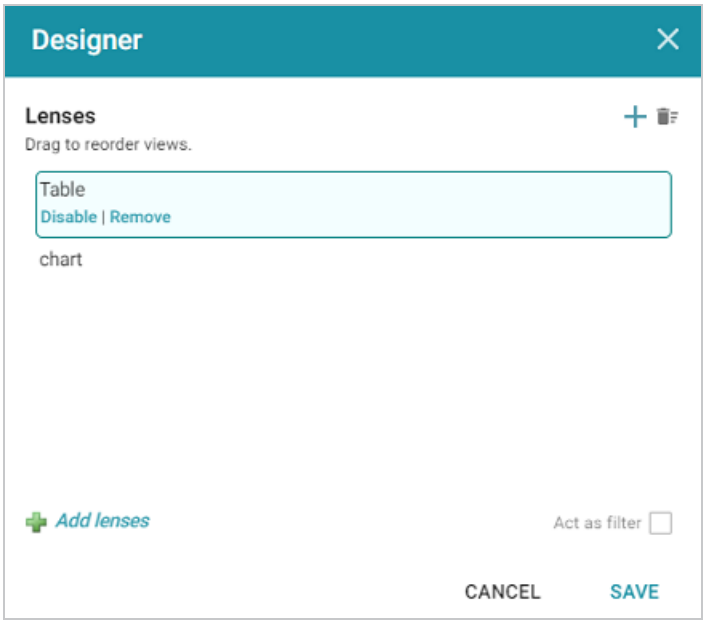


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



5. Drill down lenses do not require any property selections or format configurations. Click the plus icon (+) at the top or bottom of the Designer to add the lenses to include in the drill down functionality. The lens listed first becomes the first lens in the hierarchy. Clicking a drill down icon takes you to the next lens. You can drag the lenses in the Designer to change the display order.



6. When you finish adding lenses, click **Save**. Anzo adds the drill down functionality to the dashboard, and you can configure each lens in the hierarchy using the Designer for that lens.

The image below shows a dashboard with drill down functionality. Clicking the drill down icon (▽) in the left column displays a chart lens, which shows details about the venue for that event:



events

	Eventid ↑	Eventname
▽	1	Gotterdammerung
▽	2	Boris Godunov
▽	3	Salome
▽	4	La Cenerentola (Cinderella)
▽	5	Il Trovatore
▽	6	L Elisir d Amore
▽	7	Doctor Atomic
▽	8	The Magic Flute
▽	9	The Fly
▽	10	Rigoletto
▽	11	Doctor Atomic
▽	12	Ring Cycle
▽	13	Lucia di Lammermoor
▽	14	La Rondine
▽	15	Die Walkure
▽	16	La Gioconda
▽	17	La Gioconda
▽	18	Gotterdammerung

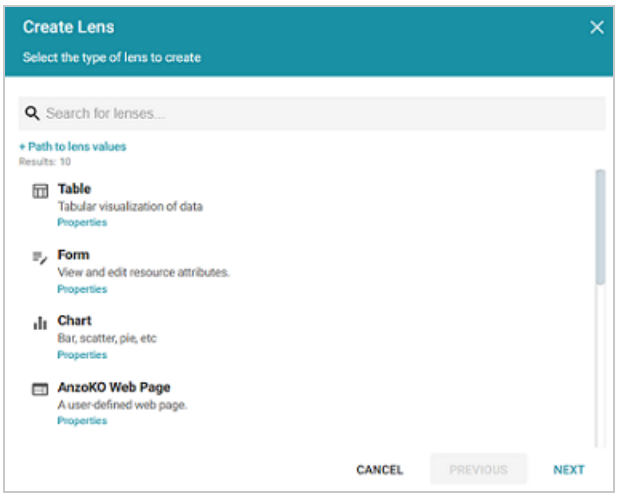
Creating a Form Lens

Form lenses enable you to create an editable or read-only form on the dashboard. Creating forms can be useful for displaying many details about each record instead of using a table where the large number of columns makes the data hard to read. Follow the steps below to create a Form lens.

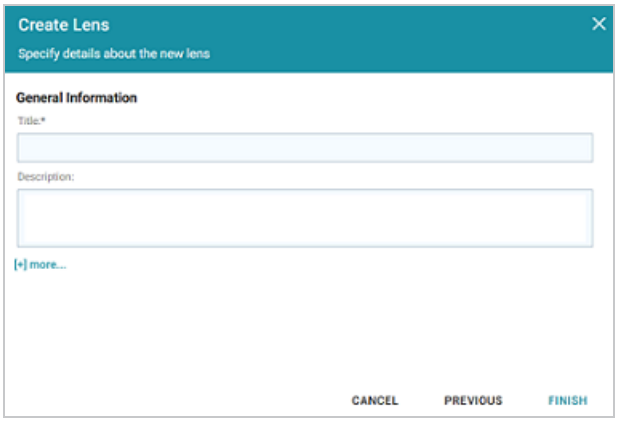
Important

By default, only the sysadmin user has access to create Form lenses. In addition, Form lenses are valid in only in Linked Dataset dashboards. They do not display data for graphmart dashboards. To create a Linked Dataset dashboard, select the **Show advanced dashboards** checkbox when creating a dashboard, and then select **Linked Dataset Dashboard**.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

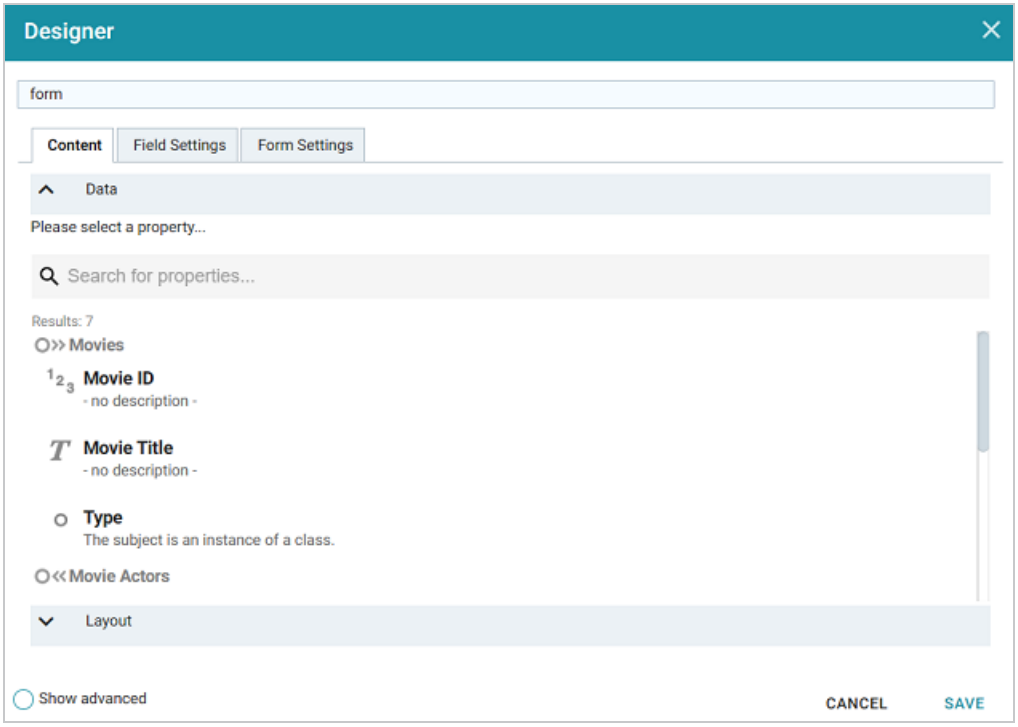


2. On the Create Lens dialog box, select **Form**, and then click **Next**. Anzo displays the General Information dialog box.

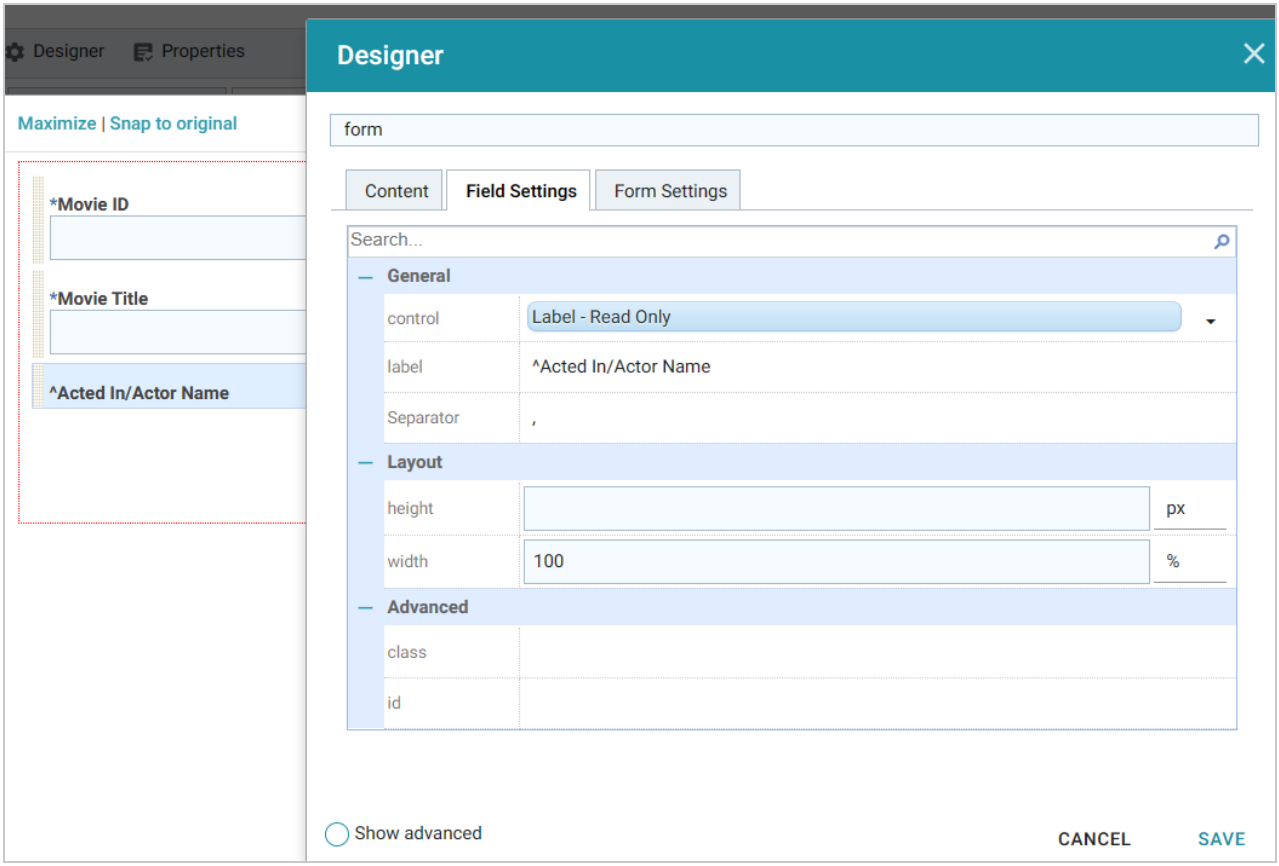


3. Type a **Title** and optional **Description** for the lens.

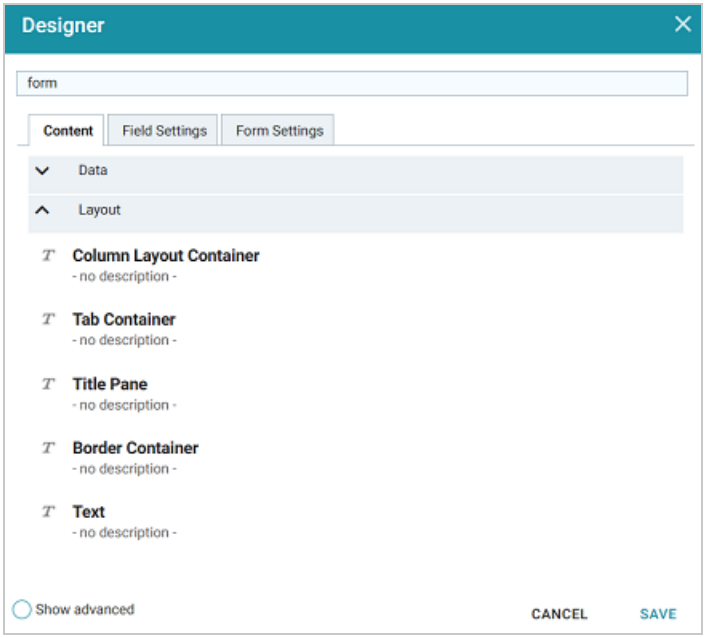
4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



5. On the Content tab, drag onto the dashboard each property or relative path that you want to appear as a field on the form. After adding objects, you can rearrange the form layout and use the Field Settings tab to further configure each field.



6. If you want to arrange the fields in a different layout, such as a two-column layout, click **Layout** below the list of properties on the Content tab. The Designer displays the available layout containers.

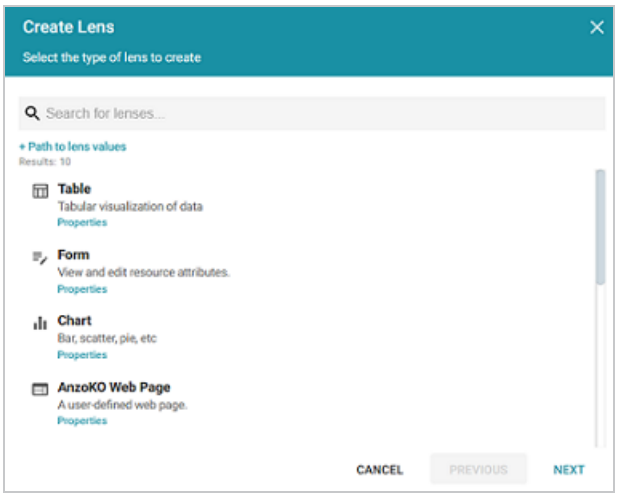


7. Drag a container onto the form to create the layout template. You can then drag properties into the template.
8. Click **Save** to save the configuration and add the lens to the dashboard.

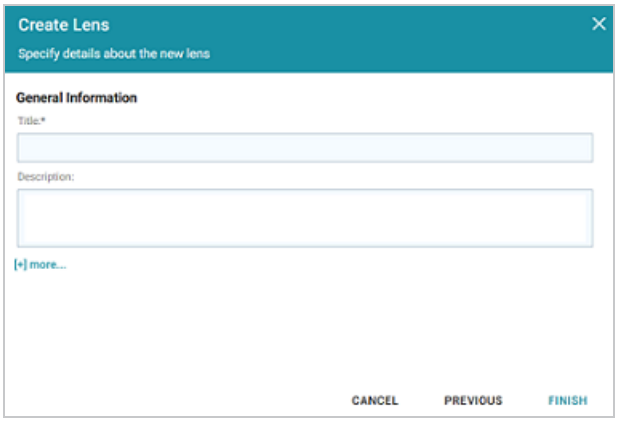
Creating a List Lens

List lenses display the values for the selected property in a list layout with icons, similar to a directory explorer view. Follow the steps below to create a List lens.

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

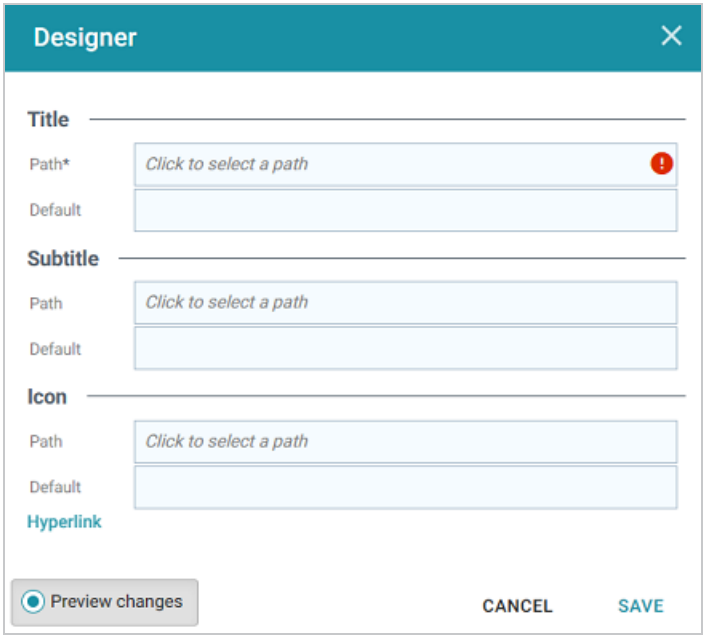


2. On the Create Lens dialog box, select **List**, and then click **Next**. Anzo displays the General Information dialog box.

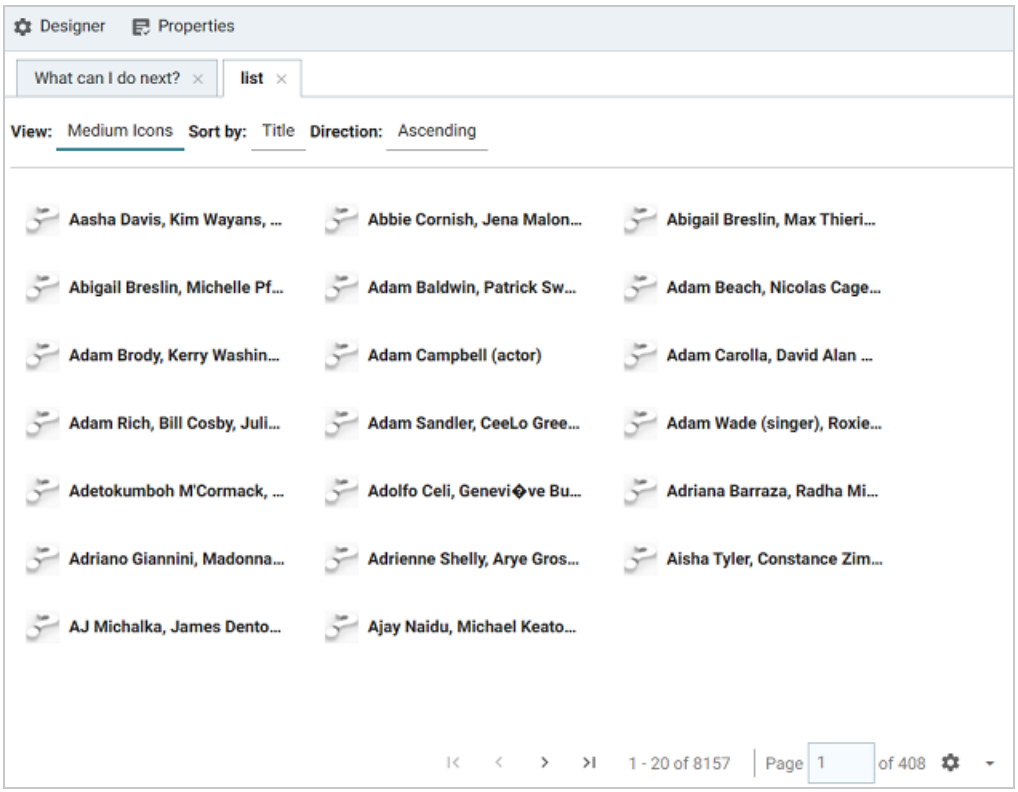


3. Type a **Title** and optional **Description** for the lens.

4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



5. In the **Path** field under Title, select the property that contains the values you want to display in the list.
6. If you want to include a subtitle for each value in the list, click the **Path** field under Subtitle and select the property to supply the subtitle values. For example, in the image below, Actor Name is chosen as the Title with Movie Name as the subtitle.



7. If you have a property that contains an icon to use in place of the default question mark icons, click the Path field under Icon and select the property.
8. If you want the Title to be formatted as a hyperlink to another lens, you can click **Hyperlink** and select the lens to link to.
9. When you have finished configuring the lens, click **Save** to save the configuration and add the lens to the dashboard.

Creating a Table Lens

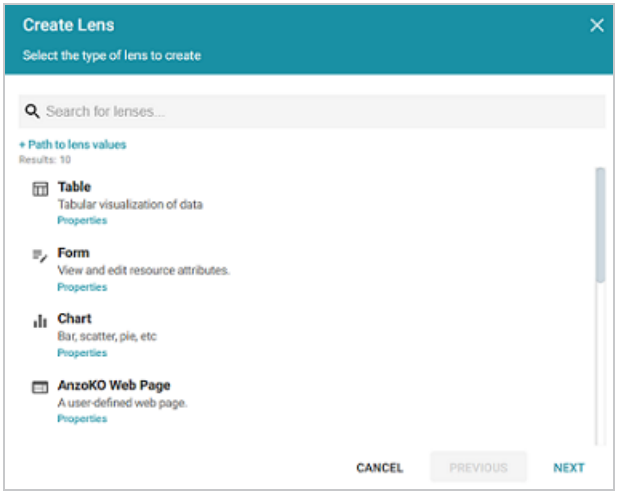
Table lenses display data in a standard column and row grid layout. Follow the steps below to create a table lens.

> **Note**
> The list below describes the default display formats for date and numeric values in tables.
>
> - **Date**: By default Anzo displays date values in "short" date format. The order of the month, day, and year depends on the location of your browser. For example, in the

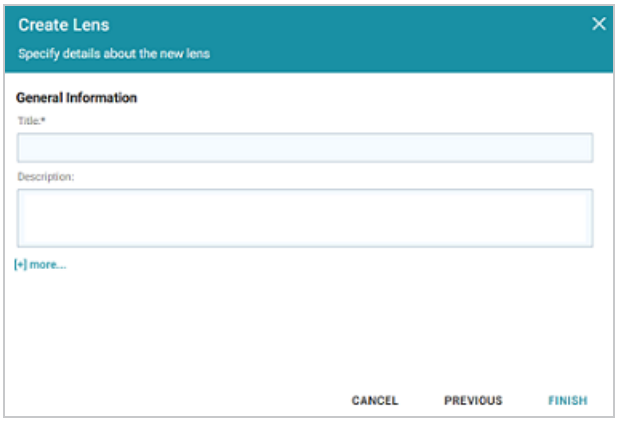
United States the default date format is `MM/DD/YYYY`. In Australia, the default date format is `DD/MM/YYYY`. Note that this is not dependent on the Anzo server location but on the location auto-detected by the browser.

- **Numeric**: Anzo displays the complete value without a limit on precision. Numeric formats are also dependent on the location of the browser. For example, in the United States the default format for a large number is `4,294,967,295.00` and in Canada the default format is `4 294 967 295,000`.

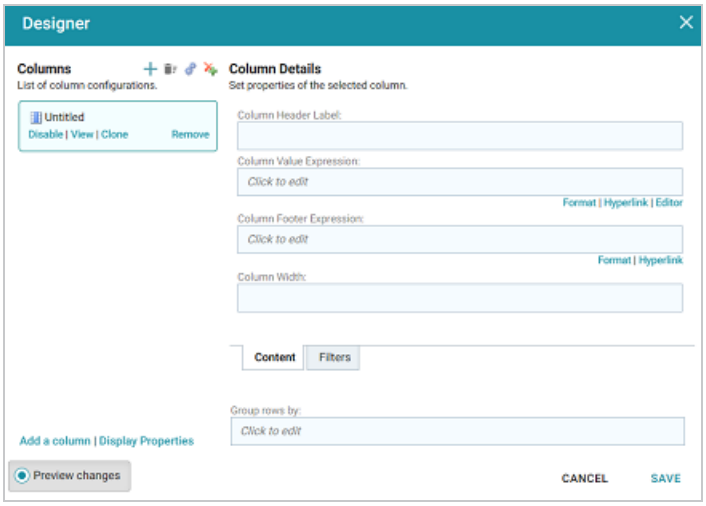
1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:



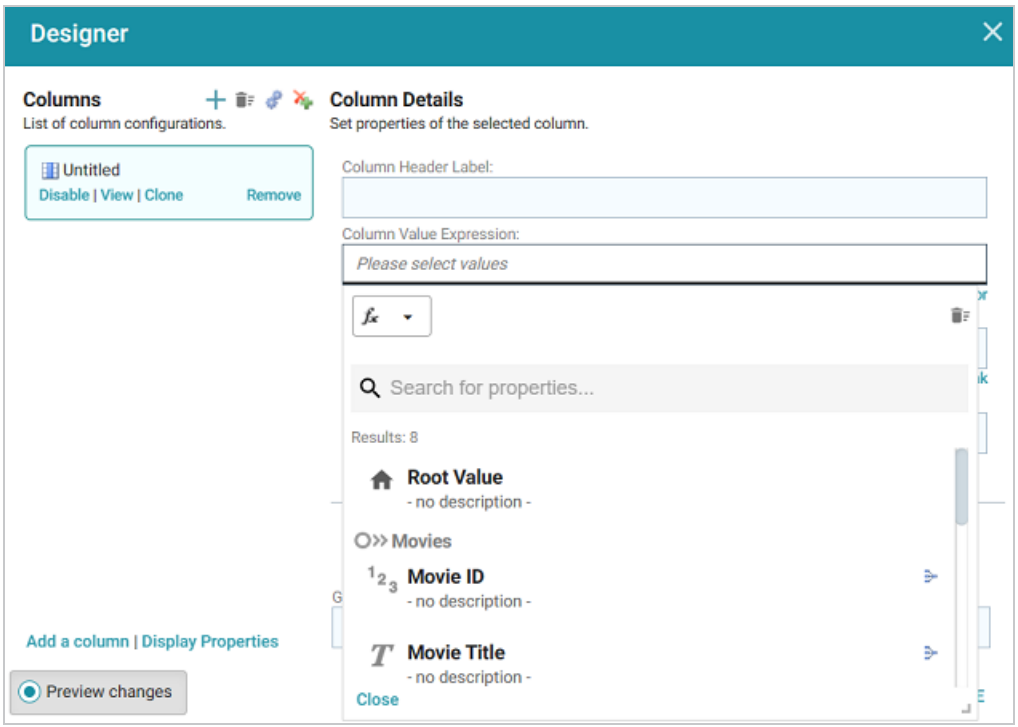
2. On the Create Lens dialog box, select **Table**, and then click **Next**. Anzo displays the General Information dialog box.



3. Type a **Title** and optional **Description** for the lens.
4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



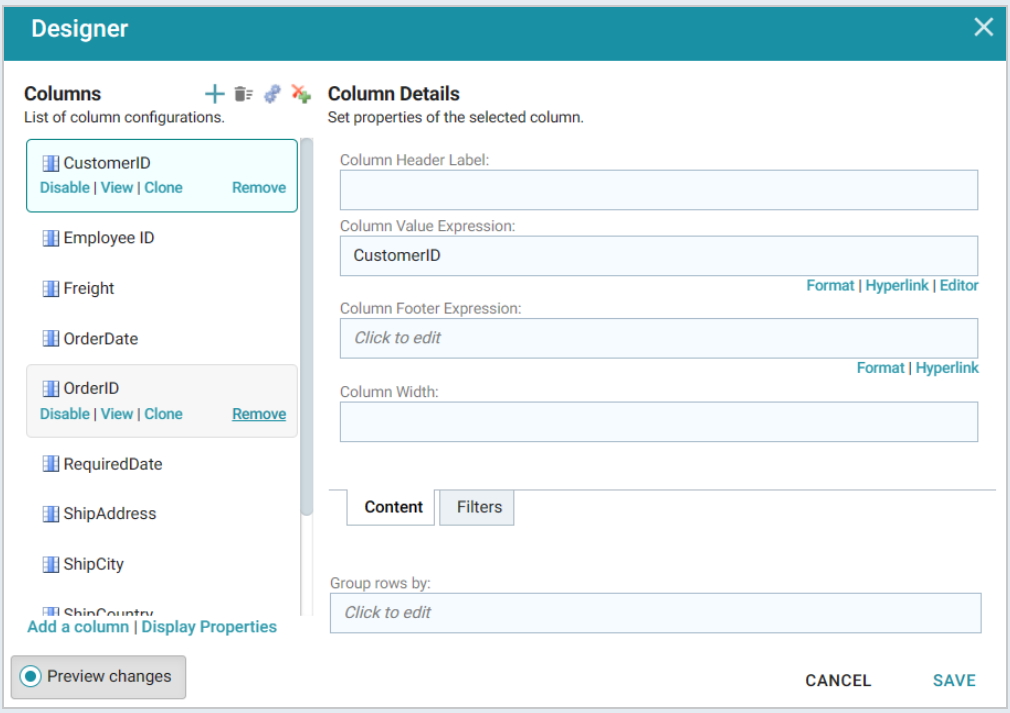
5. To add a column to the table, click the **Column Value Expression** field and select the property name or calculation to use to populate the values in the column. For information about calculating values, see Calculating Values in Lenses and Filters.



Tip

As a shortcut to browse the available columns, you can click the Auto-generate columns icon ( ) to add all properties (for the data type selected on the dashboard) as columns.

Then you can hover the pointer over columns to remove the ones you do not want to keep. For example:



6. You have the option to create a filter or configure any of the following optional fields:
   - **Column Header Label**: The column name to display. Overrides the Column Value Expression property name.
   - **Column Footer Expression**: The property to use to supply the table footer.
   - **Column Width**: The width of the column in pixels.
   - **Group rows by**: The property to group data by.
   - **Filters** Tab: You can click **Create filter** to add a filter on the column. For more information, see Working with Filters.
7. Click **Save** to save the configuration and add the lens to the dashboard.

Advanced Lenses

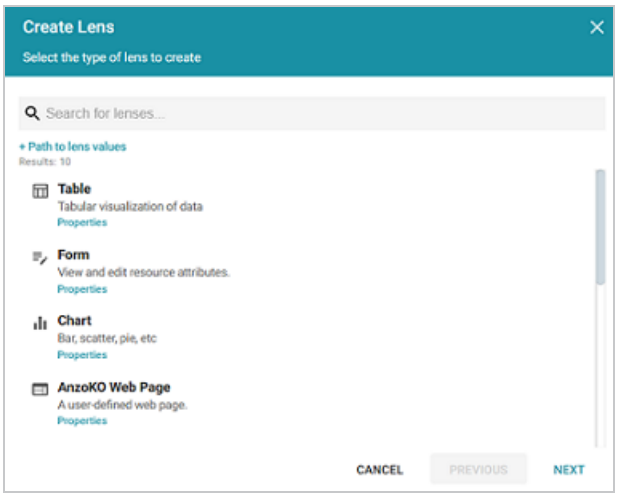
The topics in this section provide information about creating advanced dashboard lenses. Creating these types of lenses require users to advanced SPARQL query language skills or coding skills in the areas of HTML, CSS, and JavaScript. For information about Network Navigator lenses, see Creating a Network Navigator Dashboard.

- Creating an AnzoKO Web Page Lens
- Creating a Query Lens
- Creating a Resource Tree Navigator Lens
- Creating a Web Page Lens

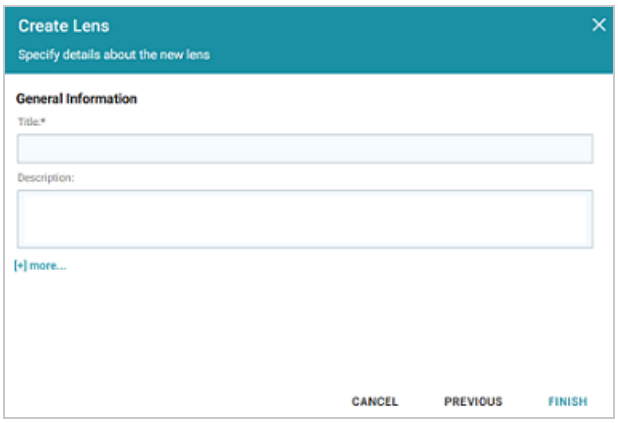
Creating an AnzoKO Web Page Lens

The custom AnzoKO Web Page lens includes the Knockout JavaScript framework and enables you to create visualizations of RDF resources and metadata using knockout.js-like syntax without needing to write additional JavaScript to declare which parts of the data to render in which sections of the HTML. Follow the steps below to create an AnzoKO Web Page Lens.

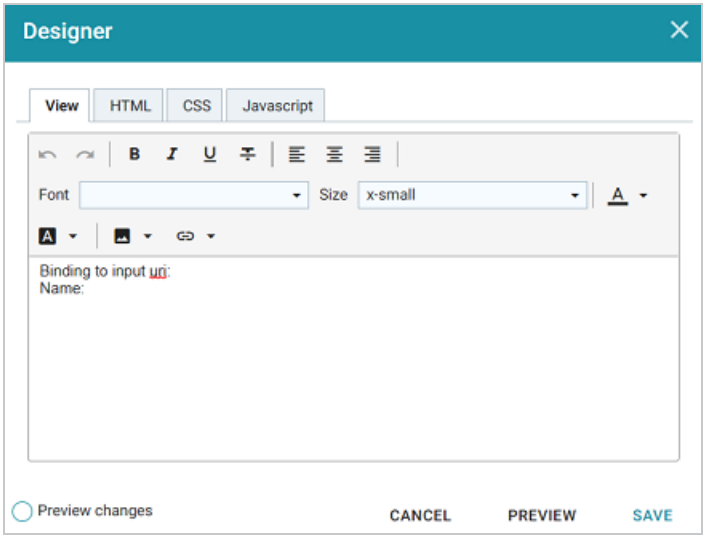
1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:



2. On the Create Lens dialog box, select **AnzoKO Web Page**, and then click **Next**. Anzo displays the General Information dialog box.



3. Type a **Title** and optional **Description** for the lens.
4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



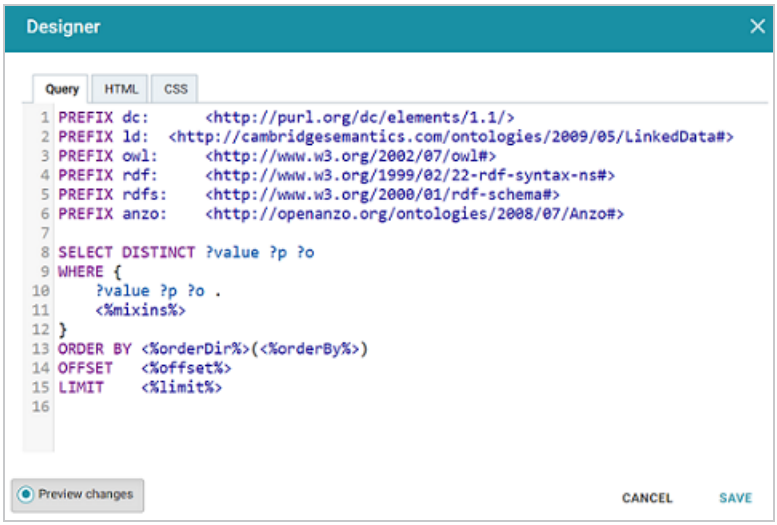
5. Configure the lens and then click **Save** to save the configuration and add the lens to the dashboard.

Creating a Query Lens

The query lens allows you to retrieve and display data using custom a SPARQL query. You format the query results using HTML and CSS. This lens can access external SPARQL-compatible data sources. See SPARQL Best Practices and Query Templates for guidance on writing SPARQL queries.

Query Lens Configuration

The Query lens Designer has three tabs:



- **Query**: This tab displays a SPARQL query template that you can use to write the query. Note the default code that reflects inherent functionality:
  - **<%mixins%>**: Incorporates a filter function.
  - **ORDER BY**: Incorporates a sort function.
- **HTML**: This tab includes default HTML and basic JavaScript code with sample values. You can edit the content to design the results that the query returns. The default HTML code automatically adds returned query data to a table and organizes it so that new rows are created for each record. Make sure that the <option> elements correspond to the elements in your query.
- **CSS**: This tab enables you to create a cascading style sheet to format the HTML and define the look and feel of the lens. Cambridge Semantics recommends that you define all CSS classes as namespaces to avoid global format changes.

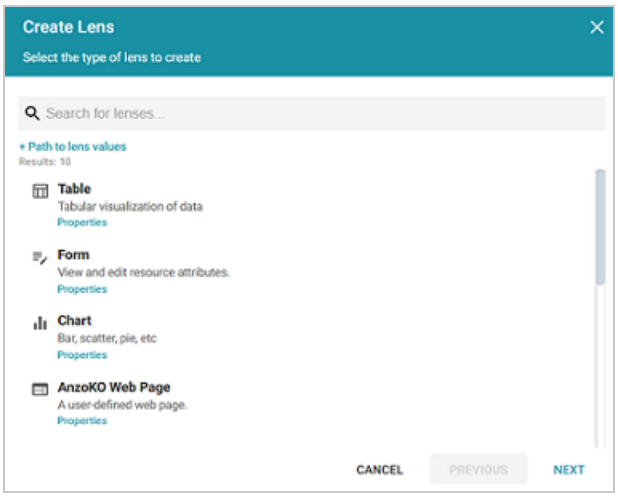
Creating a Resource Tree Navigator Lens

The Resource Tree Navigator lens displays data in a tree format with points that you can click to open successive child data points. Follow the steps below to create a lens.

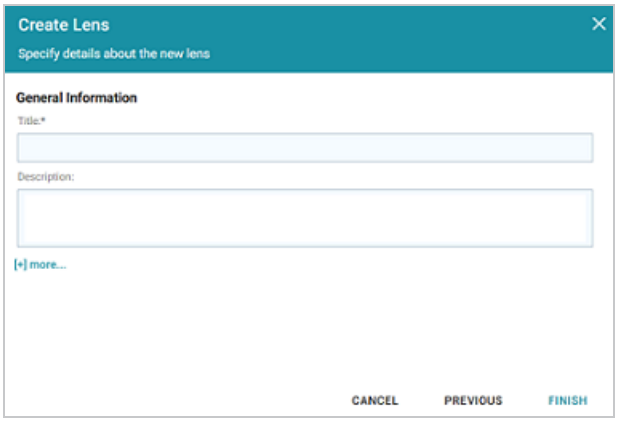
Note

By default, the only user who has permission to create a Resource Tree Navigator lens is the **sysadmin** user. However, the sysadmin user can share created lenses with other users and groups (see Sharing Access to Dashboards and Lenses).

1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:

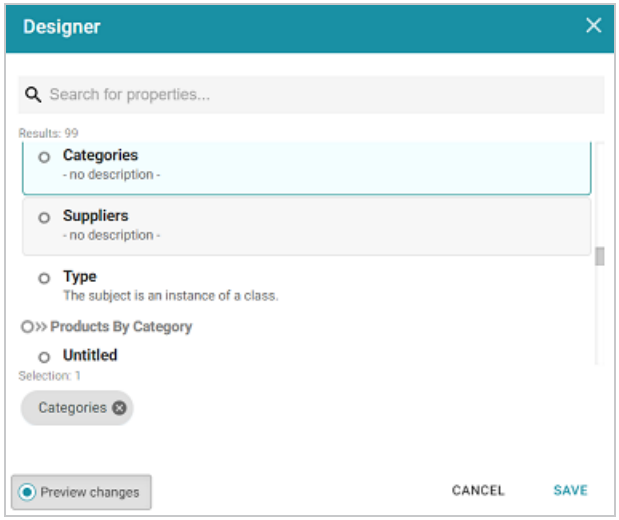


2. On the Create Lens dialog box, select **Resource Tree Navigator**, and then click **Next**. Anzo displays the General Information dialog box.

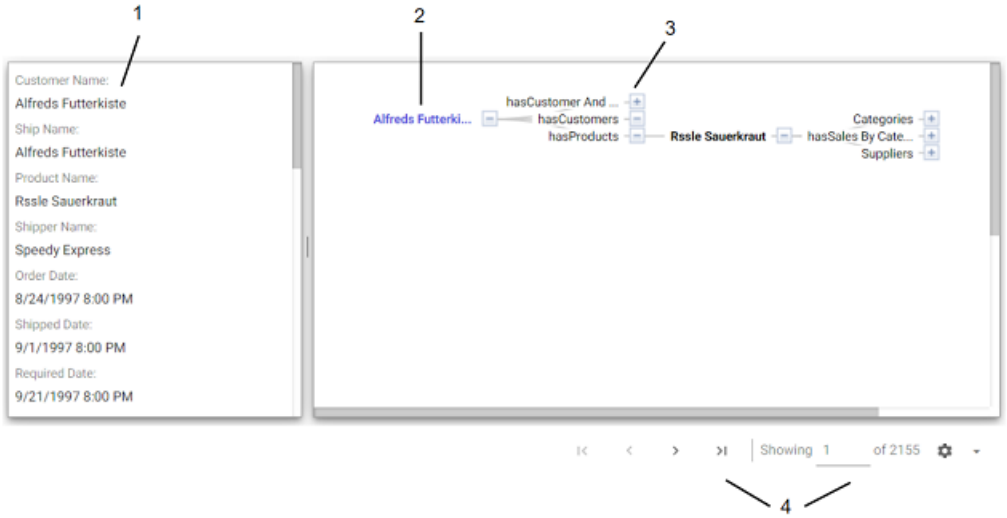


3. Type a **Title** and optional **Description** for the lens.
4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens. The Designer displays all of the properties for the linked classes. Select each property that

you want the resource tree to include.



5. Click **Save** to save the configuration and add the lens to the dashboard. The image below shows the information that is displayed.



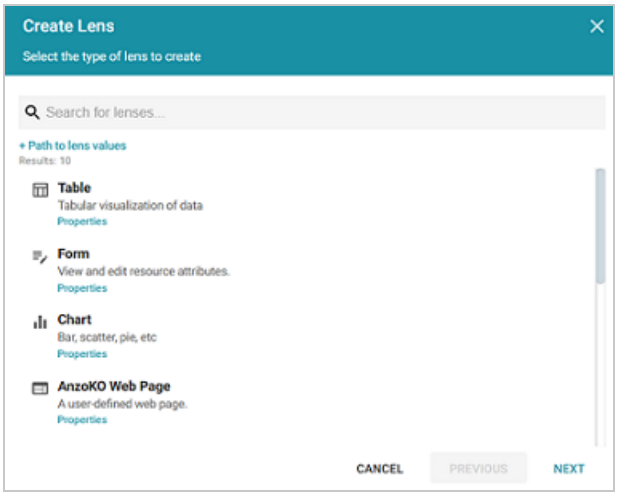
1. **Related data**: Displays the class data related to the selected data property. Data changes when another class is selected.
2. **Class property**: Displays the label property of the target class as the initial (start) point of the resource tree. Expand the tree to view child properties by clicking the plus icon for a data point.

3. **Selected linked property**: Displays the initial selected property that links to other classes.
4. **Navigation tools**: Use the arrows to navigate to other pages. The Showing text box displays the current page number and total number of pages.

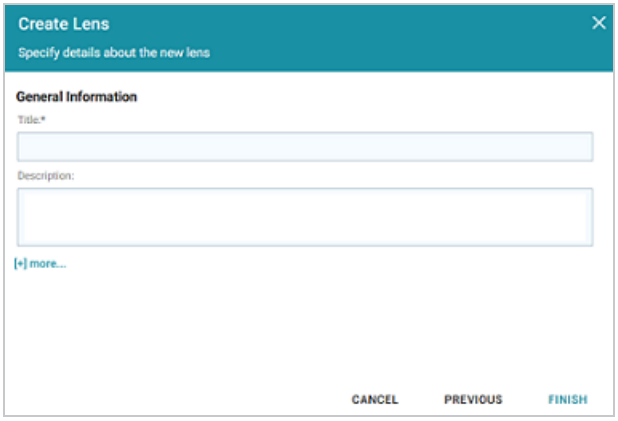
Creating a Web Page Lens

Web Page lenses enable you to display data by creating a web page using HTML, CSS, and JavaScript. This lens is for advanced users with coding skills in these areas. A powerful feature of the this lens is the ability to bind data to Anzo graphs so that updates are reflected in real time. Follow the steps below to create a Web Page lens.

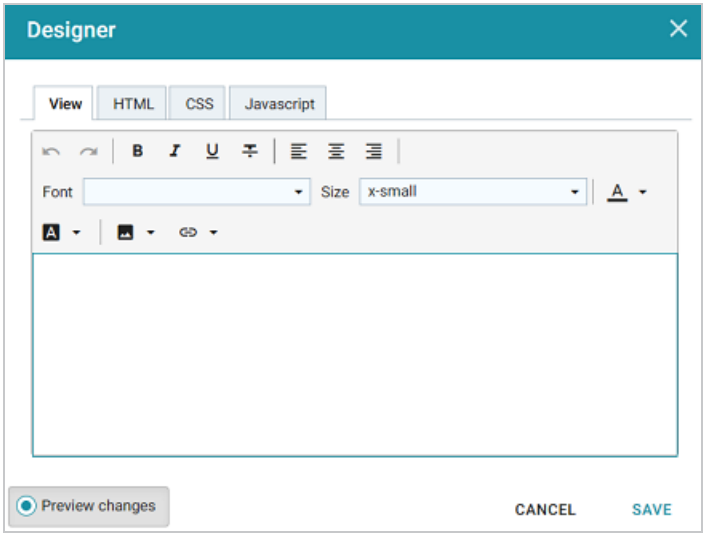
1. In the dashboard that you want to add a lens to, click **Lenses** in the main toolbar and select **New**. The Create Lens dialog box is displayed:



2. On the Create Lens dialog box, select **Web Page**, and then click **Next**. Anzo displays the General Information dialog box.



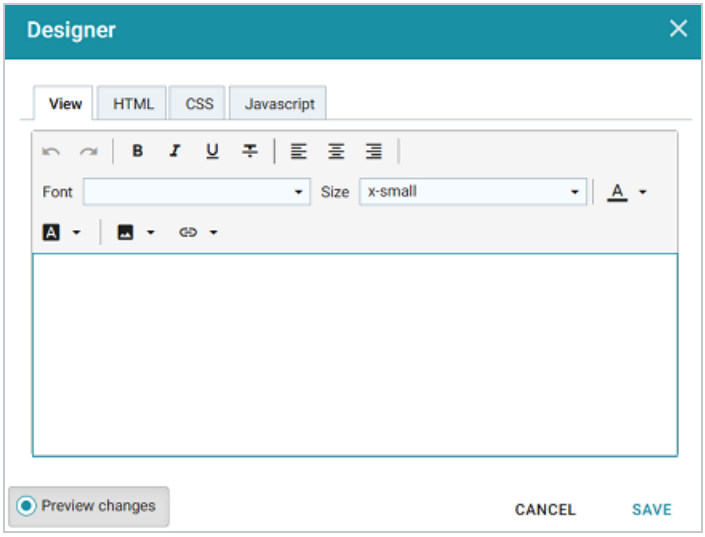
3. Type a **Title** and optional **Description** for the lens.
4. Click **Finish**. The lens Designer dialog box is displayed so that you can configure the lens.



5. Configure the lens and then click **Save** to save the configuration and add the lens to the dashboard. See Web Page Lens Configuration below for information about the configuration.

Web Page Lens Configuration

The Web Page Designer has four tabs:



- **View**: Provides a rich text interface for viewing the page (WYSIWYG). Changes made to this page are reflected in the HTML code.
- **HTML**: This tab enables HTML coding and data binding. The example HTML image below shows code that defines text format as well as data binding using the `anzowbind:innerhtml` command.



  For more information about data binding, see the Data Binding Example section below.

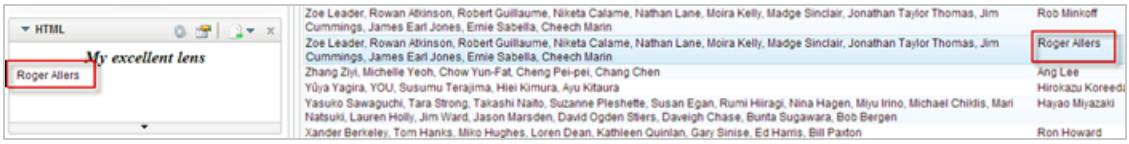
- **CSS**: This tab enables you to create a cascading style sheet to format the HTML and define the look and feel of the web page. Cambridge Semantics recommends that you define all CSS classes as namespaces to avoid global format changes.
- **Javascript**: This tab enables you to write JavaScript code to implement functions such as if statements, animations, or event notifications.

Data Binding Example

When data is bound to a web page lens using HTML code, the web page lens behaves as follows:

- The lens will reflects data changes in real time.
- If the lens is oriented to the left-hand column (using the Orientation drop-down), selecting data in an active lens prompts the web page lens to display the related data.

In the example below, the active table lens row is selected, prompting the web page lens on the left (“My excellent lens”) to display the corresponding data.

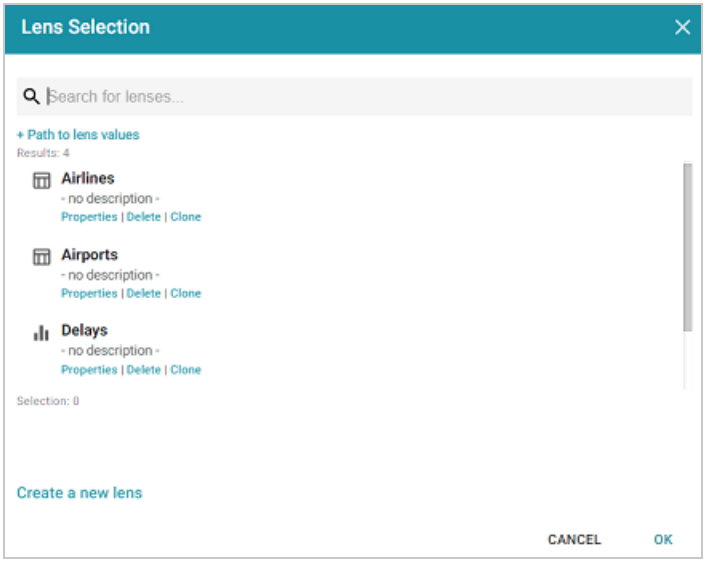


Cloning a Lens

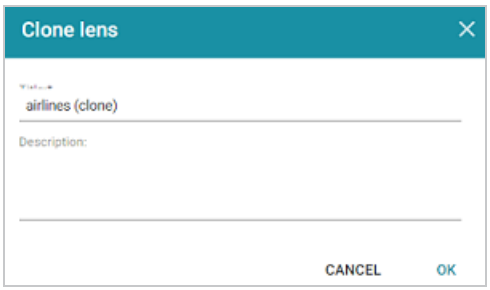
Cloning a lens makes a copy of the lens that can be changed without affecting the original lens or other dashboards. Follow the steps below to clone a lens.

> **Note**
> You can only clone lenses from dashboards that you have permission to modify. If you open a dashboard with read-only access, the clone options are not available. To clone a lens from a read-only dashboard, save a copy of the dashboard so that you become the owner of the copy. To save a copy, click the **Dashboard** button in the main toolbar and select **Save As**. Then follow the procedure below to clone a lens into the dashboard that you own.

1. Open a dashboard in the Hi-Res Analytics application, then click **Lenses** in the main toolbar and select **Open**. Anzo opens the Lens Selection dialog box, which lists the lenses that are available to open. For example:



2. Click the **Clone** link for the lens that you want to clone. Anzo displays the Clone lens dialog box and populates the Title field with the existing lens name and "(clone)." For example:

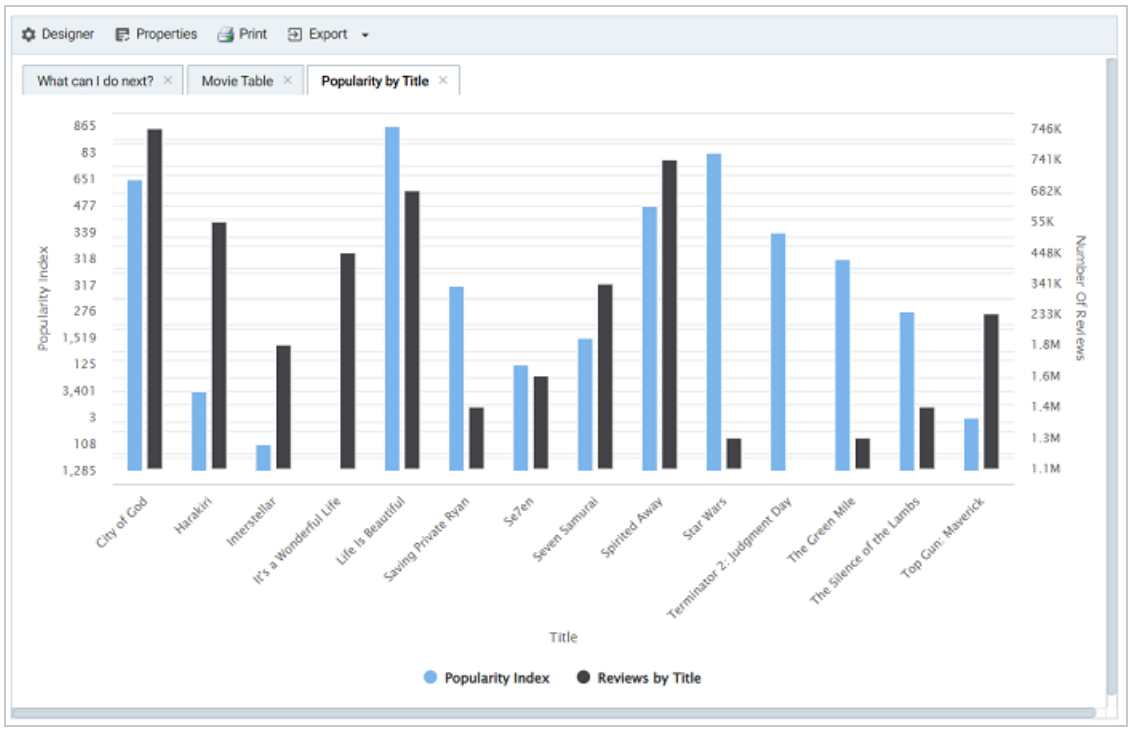


3. Modify the **Title** to name the new copy of the lens, and add or change the **Description** if necessary. Then click **OK**.
4. Anzo adds the new copy of the lens to the Lens Selection dialog box and selects it. Click **OK** to add the lens to the dashboard.

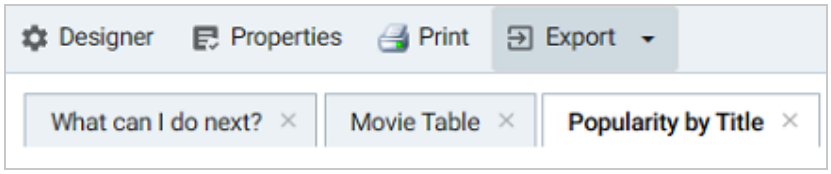
Exporting a Lens

If you have dashboards with Table, Chart, and/or Query lenses, you can export those lenses from the Hi-Res Analytics application. Charts can be exported as images in JPEG, PNG, SVG, or PDF format. Tables can be exported to CSV or JSON files. And Query lenses can be exported to CSV files. Follow the instructions below to export a lens.

1. Open the dashboard that contains the lens that you want to export.
2. If necessary click the tab for the lens to make it active. For example, the image below shows a chart lens.



3. In the object toolbar for the lens, click the **Export** button.

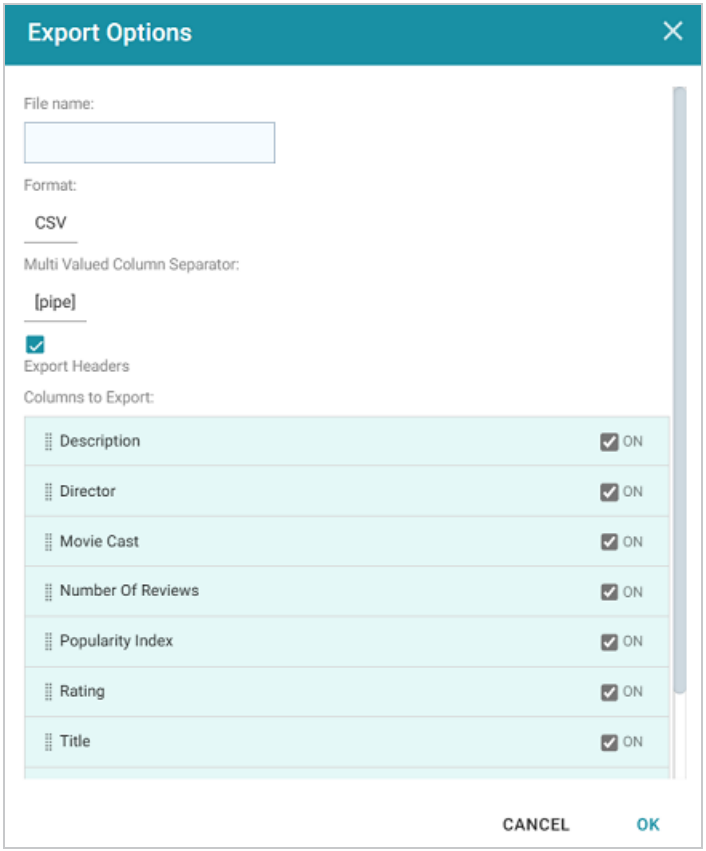


For Query lenses, adjust the Export options as needed, and then click **Export Results**. For example:



4. If the lens is a chart, select the one of the image types from the drop-down list. Anzo creates the image as that type and downloads the file to your computer.

5. If the lens is a table, the Export Options dialog box is displayed:



6. In the Export Options dialog box, specify the following file options:
   - **File name**: Specify a name for the file. Do not specify the file type extension.
   - **Format**: Click the Format field and select **CSV** to create a .csv file or **JSON** to create a .json file.
   - **Multi Valued Column Separator**: For CSV files, click this column to select the character to use as a separator in the file. This option does not apply to JSON files.
   - **Export Headers**: Indicates whether to include column headers in the file. Clear the checkbox to exclude headers from the file. This option does not apply to JSON files.
   - **Columns to Export**: By default, all columns are selected for export. You can clear the checkboxes next to columns that you do not want to include. In addition, you can reorder columns by dragging and dropping the rows.
7. Click **OK** to download the file to your computer.

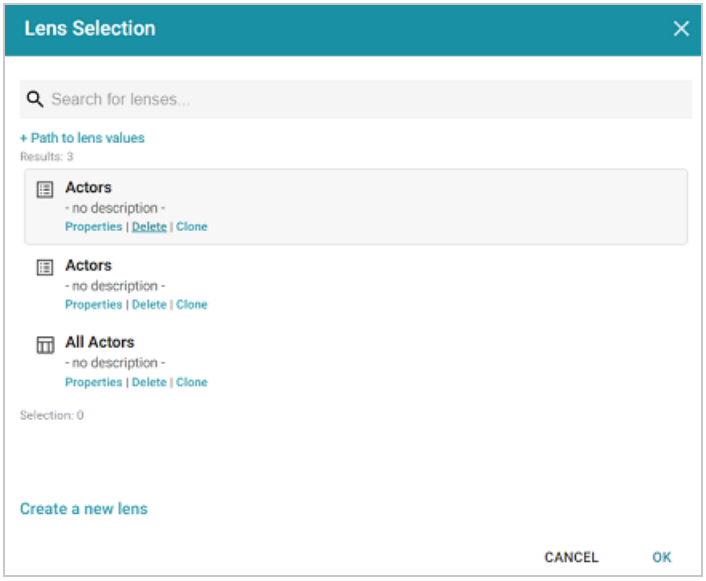
Deleting a Lens

This topic provides information about the permissions that are required for deleting lenses as well as instructions for deleting a lens.

> **Note**
> By default, only the **sysadmin** user and **lens creator** have permission to delete a lens. To delete a lens, a user must have the **Manage** permission assigned for that lens. The Manage permission is included in the **Admin** predefined lens permission set. If lens permissions have not been changed since the lens was created, the sysadmin user and the lens creator are the only users who have permission to delete that lens. The Manage permission is also required to change lens security settings and grant privileges to other users. Users who have read access to a lens (granted through the View, Modify, or Admin lens permission sets) can view the lens security settings to identify which non-sysadmin users have permission to delete the lens. For more information, see Sharing Access to Dashboards and Lenses.

Follow the steps below to delete a lens.

1. Open a dashboard that contains the lens that you want to delete.
2. In the Hi-Res Analytics application, click the **Lenses** menu in the main toolbar and select **Open**. The Lens Selection dialog box is displayed.
3. In the Lens Selection dialog box, find the lens that you want to delete and then click the **Delete** link for that lens. For example:



4. The application presents a confirmation message. Click **Yes** to delete the lens. The lens is removed from the Lens Selection dialog box, and you can repeat this process to delete additional lenses for which you have the required privileges.

Working with Filters

Filters narrow the data presented in a dashboard. You can define filter criteria using Microsoft Excel-like functions such as AVG, SUM, or UPPER, or groupings such as a date range or aggregation. Though you can filter data in some lens types, such as tables and charts, when you add a filter to a dashboard, all lenses on the dashboard update simultaneously based on your filter selection. Unlike lenses, filters cannot be shared by other users or dashboards and must be created for each dashboard.

The topics in this section provide instructions for creating and configuring each type of filter.

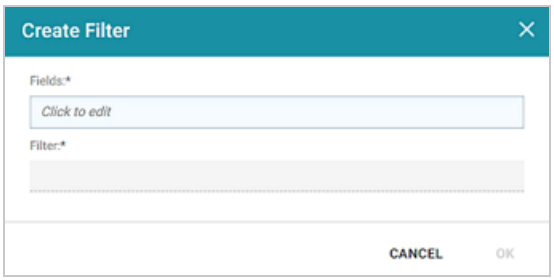
- Adding a Cloud Filter
- Adding a Date Range Filter
- Adding a Hierarchy Filter
- Adding a Limit Filter
- Adding a List Filter
- Adding a Numeric Range Filter
- Adding a Presence Filter
- Adding a Quartile Filter
- Adding a Range Slider Filter
- Adding a Relative Time Filter
- Adding a Search Filter
- Adding a Single Select List Filter
- Adding a Types Filter

Adding a Cloud Filter

Cloud filters display values in term clouds where each term is written in a font size that represents the number of results for that value. Unlike list filters, which enable you to select and filter on multiple values at once, cloud filters allow you to filter on one value at a time. The cloud filter is

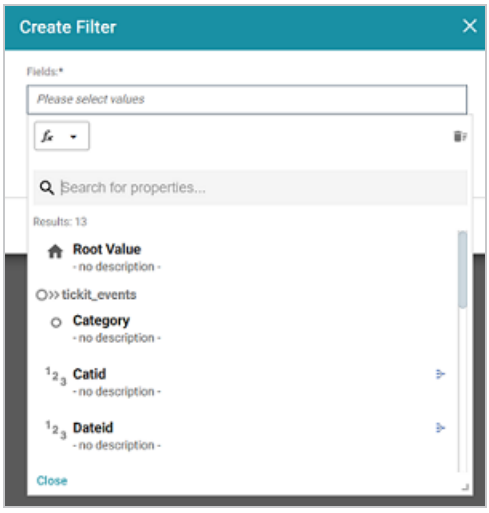
available for all data types. Follow the instructions below to create a cloud filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The values for this property will be the terms that are displayed in the cloud. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



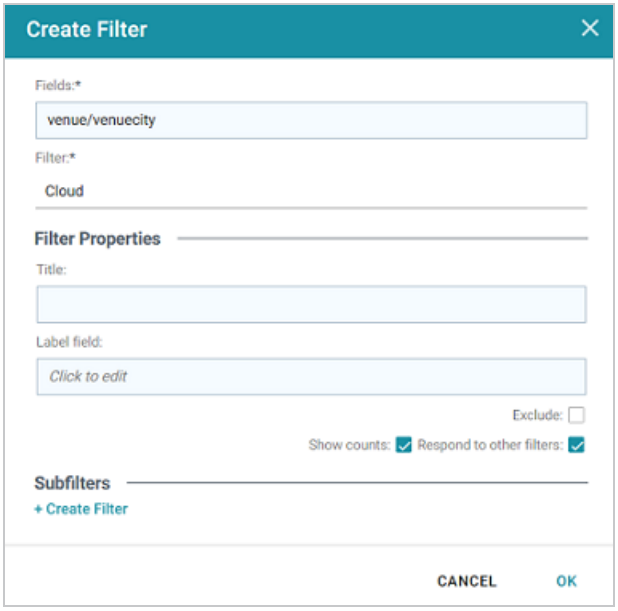
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Cloud** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:

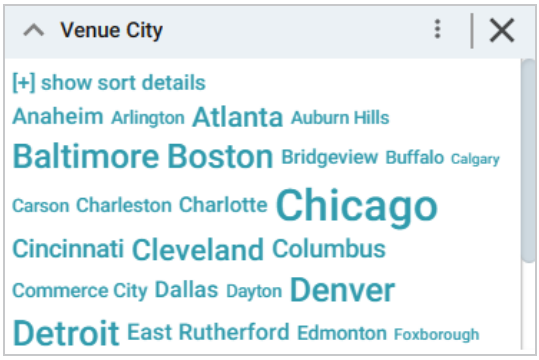


6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the cloud with values from a property other than the one specified in Fields, you can select an alternate property in this field.
   - **Exclude**: This setting controls whether selecting a term in the filter narrows the results to show only the records that *include* that term or whether selecting a term *excludes* the records that include that term. When Exclude is disabled, selecting a term narrows the dashboard results to show only the records that include that term. When Exclude is enabled, selecting a term filters out all of the records that include that term.
   - **Show Counts**: This setting controls whether the number of results for each term are displayed when you hover the pointer over a term.
   - **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not

displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.

8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the data.

   For example, the filter in the image below shows cities with event venues. The size of the terms represent the number of events that were held in that city.



   Depending on whether the Exclude option is enabled or disabled, clicking a term in the cloud refreshes the dashboard to show only the data that either contains or excludes the selected term.

When working with the filter on the dashboard, the following options are available for sorting and configuration:

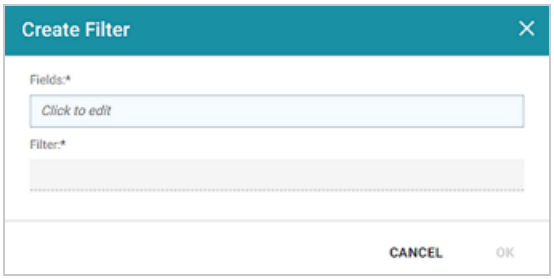
- **show/hide sort details**: Shows or hides the following options for sorting the results in the filter:
  - **Sort by**: Select **Count** to order results according to the total number of results for each value, or select **Value** to sort string values alphabetically.
  - **Direction**: Select **Ascending** to order results in alphabetical order. Or select **Descending** to order results in reverse order.
- **show/hide filters**: This option is displayed when a term is selected in the cloud. It shows or hides the selection.

- **Menu** (⋮): The menu contains the following options:
  - **Select All Visible**: This option does not work for cloud filters.
  - **Clear**: This option becomes available when a term is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Date Range Filter

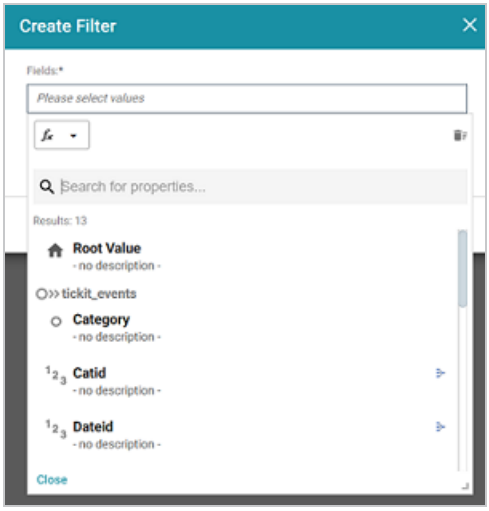
Date range filters are used to limit the results on a dashboard to data that falls in (or outside of) certain date and time groupings. Date range filters are available for properties with date, dateTime, and time data types. Follow the instructions below to create a date range filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the date, dateTime, or time type property to filter on. The values for this property will be used to determine the date ranges for the filter. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a property:

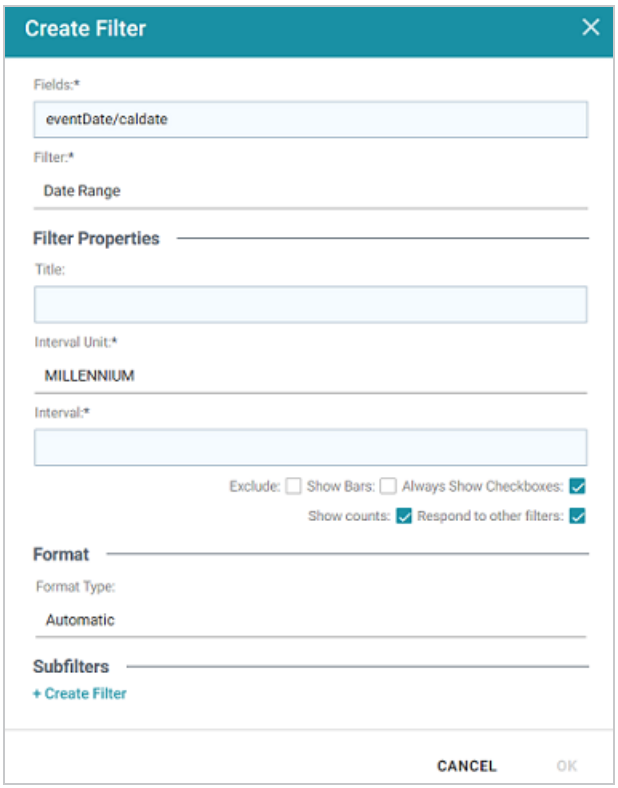
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties. For this example filter, selecting **EventDate**, navigates to the `tickit_dates` class where a date type property can be selected.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on

the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Date Range** from the drop-down list. The dialog box is refreshed to show the Filter Properties, Format, and other options that are available for the filter type:



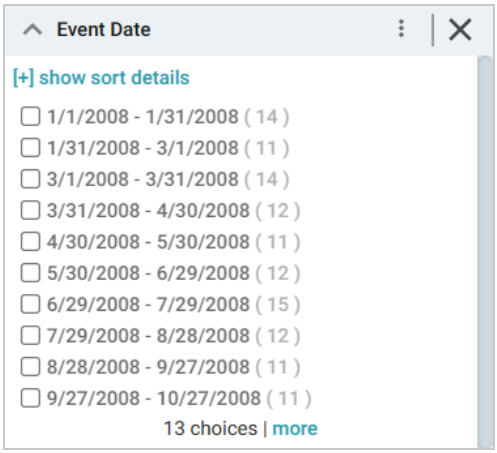
6. Configure any of the following properties. **Interval Unit** and **Interval** are required fields, and the rest of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the ranges with a label other than the one specified in Fields, you can select an alternate property in this field.

- **Interval Unit**: Click this field to choose the unit of time for the **Interval** value. Depending on the data type of the selected property, a subset of the following values are available to choose from: Millennium, Century, Decade, Year, Month, Week, Day, Hour, Minute, or Second.
- **Interval**: This setting specifies a number that defines the length of time in each grouping. For example, if the Interval Unit is "Decade," an Interval value of 2 creates groups of two-decade increments.
- **Exclude**: This setting controls whether selecting a range in the filter narrows the results to show only the records that are *included* in that range or whether selecting a range *excludes* the records that fall in that range. When Exclude is disabled, selecting a range narrows the dashboard results to show only the records that fall in that range. When Exclude is enabled, selecting a range filters out all of the records that fall in that range.
- **Show Bars**: This setting controls whether the total values for the selected property appear as a bar graphic in the background of the filter.
- **Always Show Checkboxes**: This setting controls whether checkboxes are shown next to the items in the filter.
- **Show Counts**: This setting controls whether the number of results for each range are displayed in parentheses next to the range.
- **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.

7. If you would like to specify the format to for date values that are displayed in the filter, click the **Format Time** field and select a format from the drop-down list.
8. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.

9. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane and displays the values that are available for filtering the data.

   For example, the filter in the image below shows date ranges for events. The Interval Unit is Month, and the Interval is 3 months.



   Depending on whether the Exclude option is enabled or disabled, clicking a range in the filter refreshes the dashboard to show only the data that is in the selected range or only the data that is outside of the range.

When working with the filter on the dashboard, the following options are available for sorting and configuration:

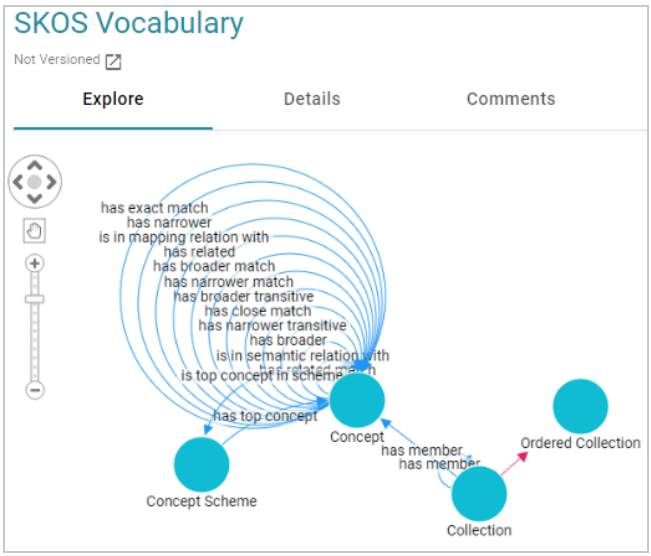
- **show/hide sort details**: Shows or hides the following option for sorting the results in the filter:
  - **Direction**: This option controls how you want to order the date ranges in the filter, depending on the Format specified for the values in the filter. For number values, **Count Ascending** orders results from the earliest to latest date and **Count Descending** orders results from the latest to earliest date. For character values, **Name Ascending** orders results in alphabetical order and **Name Descending** orders results in reverse alphabetical order.
- **show/hide filters**: This option is displayed when a range is selected. It shows or hides the selection.

- **Menu** (⋮): The menu contains the following options:
  - **Select All Visible**: This option selects all of the ranges that are listed in the filter.
  - **Clear**: This option is available when one or more ranges are selected. Clicking **Clear** removes the selection..
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Hierarchy Filter

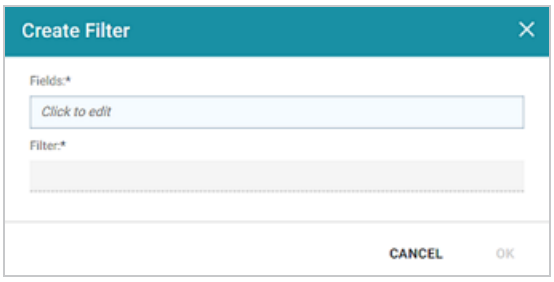
If hierarchies exist in your knowledge graph, you can create a hierarchy filter to explore the parent and child relationships and filter the dashboard based on the relationships. Unlike the majority of dashboard filters, where you select a property to filter on, hierarchy filters operate on relationships and are only available as a filter type when you select a path to filter on.

In order to produce hierarchies in the data, you typically need a self-referential data model, where properties have relationships to themselves. The SKOS ontology is a good example of a self-referential model. As shown in the image below, many of the properties in the Concept class refer to themselves. These paths define a hierarchy.



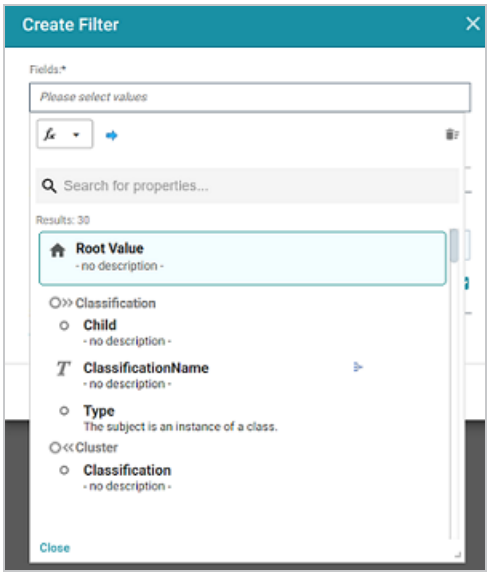
Follow the instructions below to create a hierarchy filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the path to filter on. The list of available classes, paths, and properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about protein classification. The type for the dashboard is `Classification`:



   The list below describes the icons and options that are available when choosing a path:

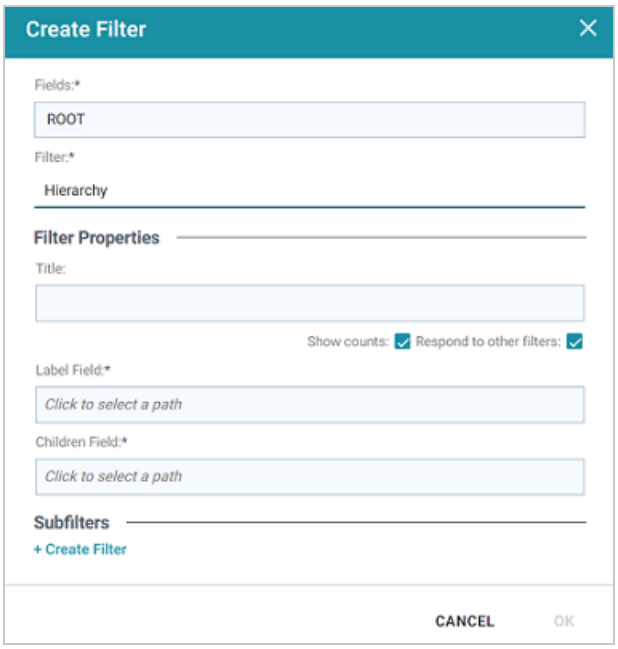
   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard. Typically **Root Value** is

the chosen Field for Hierarchy filters.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a path, you can apply a function or formula to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that become available depend on the data type of the selected path. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

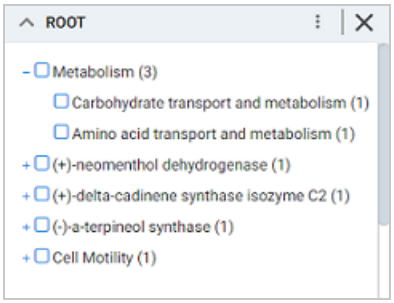
4. After you have selected the path to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Hierarchy** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:



6. Configure any of the following properties. **Label Field** and **Children Field** are required fields, and the rest of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Show Counts**: This setting controls whether the number of results for each filter value are displayed in parentheses next to the value.
   - **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.
   - **Label Field**: Species the property that supplies the label for the parent values in the hierarchy.
   - **Children Field**: Specifies the child value in the relationship.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.

8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a hierarchy of classifications.



   Selecting checkboxes in the filter refreshes the dashboard to show only the data that includes the selected values.

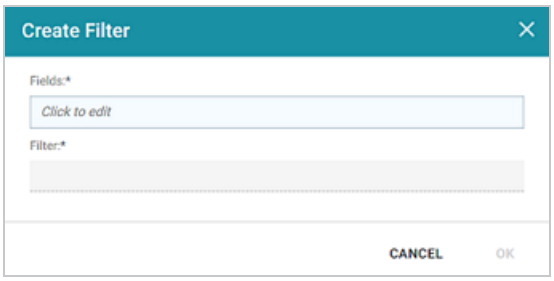
When working with the filter on the dashboard, the following options are available:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: This option becomes available when an item is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Limit Filter

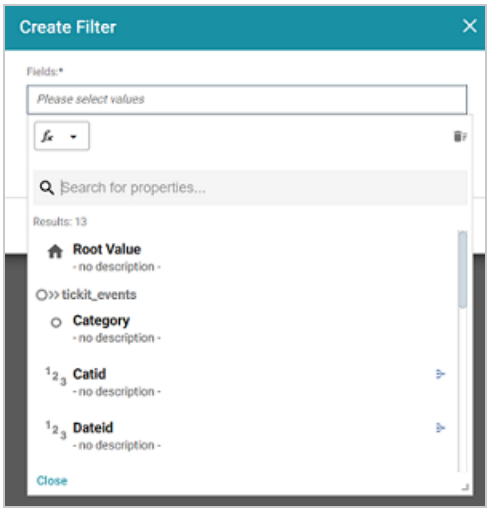
Limit filters are used to limit the results on the dashboard to a specified number of either the largest or smallest values. The limit filter is available for any data type. For strings, results are ordered alphabetically. "Largest" orders by the last letters in the alphabet and "Smallest" orders by the first letters in the alphabet. Follow the instructions below to create a limit filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



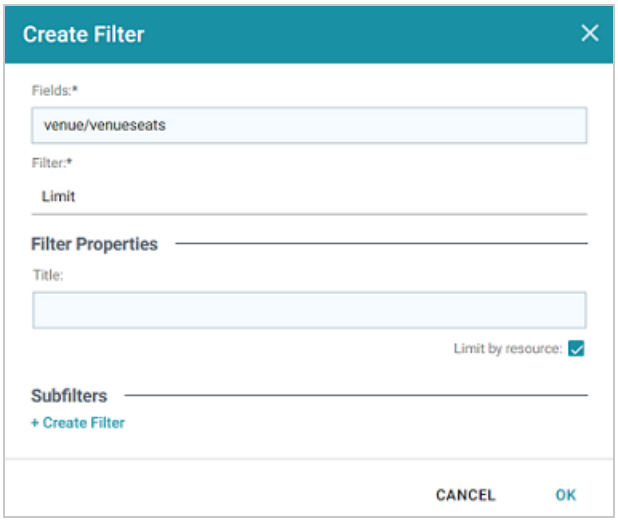
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

     > **Tip**
     > To view the root values, you can use the STR function to show a string representation of the URIs.

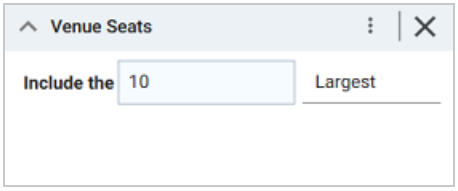
- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties. For this example filter, selecting the **Venue** path navigates to the tickit_venues class where an integer type property, Venueseats, is selected.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Limit** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:



6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Limit by Resource**: This setting controls whether the limit is also applied to the resource that is the data type of the dashboard. When **Limit by Resource** is enabled, the specified limit applies to the resource as well as the property. When **Limit by Resource** is disabled, the limit applies only to the specified property. Using the example above for the property of Venueseats and the resource (data type) tickit_events, when Limit by Resource is enabled, filtering for the 5 largest values of Venueseats returns the 5 events with the largest venues. When Limit by Resource is disabled, filtering for the 5 largest venues returns all of the events that were held in one of the 5 largest venues.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the options for configuring the limit.
9. To configure the limit for filtering data, specify a number in the **Include the** field. And click the drop-down list on the right to choose **Largest** or **Smallest**.

   For example, the filter in the image below excludes all but the 10 venues with the most number of seats.



   Changing the values in the filter refreshes the dashboard according to the new limit.

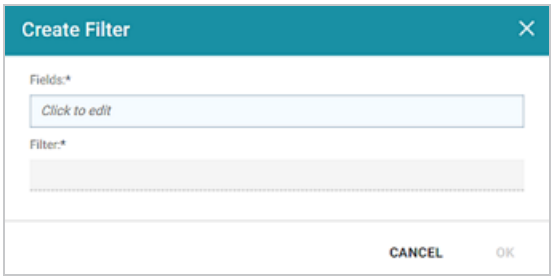
When working with the filter on the dashboard, the following options are available:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: Clicking **Clear** removes the value in the **Include the** field.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a List Filter

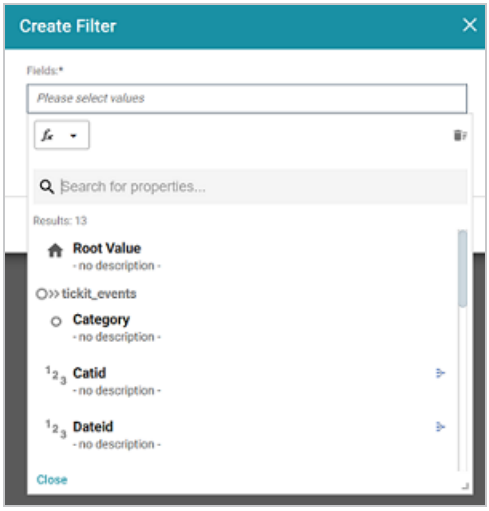
List filters display values in a list and allow you to select and filter on multiple values at the same time. List filters are available for all data types. Follow the instructions below to create a list filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a property:

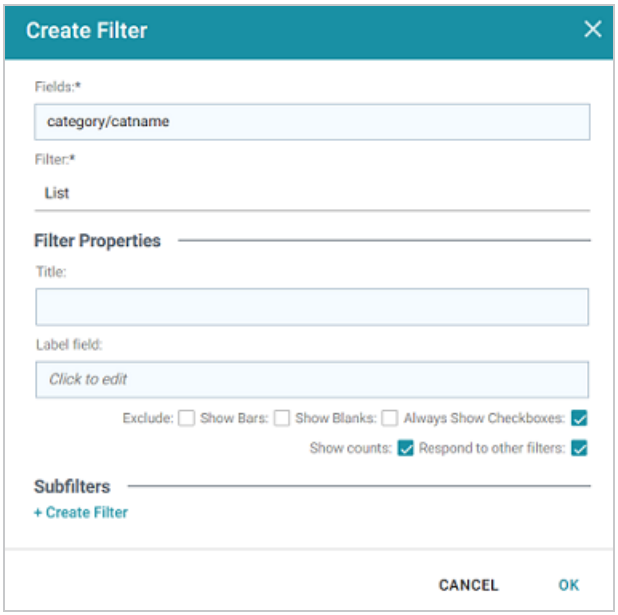
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to

choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **List** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:

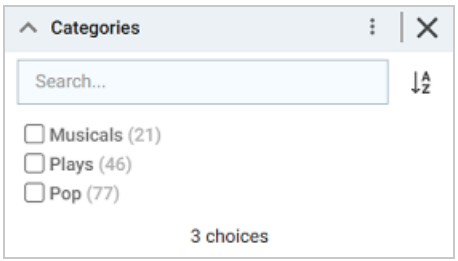


6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the list with values from a property other than the one specified in Fields, you can select an alternate property in this field.
   - **Exclude**: This setting controls whether selecting a value in the List Filter narrows the results to show only the records that *include* that value or whether selecting a value *excludes* the records that include that value. When Exclude is disabled, selecting a value in the list narrows the Dashboard results to show only the records that include that value. When Exclude is enabled, selecting a value filters out all of the records that include that value.

- **Show Bars**: This setting controls whether the total values for the selected property appear as a bar graphic in the background of the filter.
- **Show Blanks**: This setting controls whether to include null values for the selected property by listing them as **Blank** in the filter.
- **Always Show Checkboxes**: This setting controls whether checkboxes are shown next to the items in the filter.
- **Show Counts**: This setting controls whether the number of results for each item in the filter is displayed in parentheses next to the item.
- **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.

7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a filter that lists the categories of events. One or more of the categories can be selected to filter the data on the dashboard.



   Depending on whether the Exclude option is enabled or disabled, selecting an item in the filter refreshes the dashboard to show only the data includes the selected value or only the data that does not include the selected value.

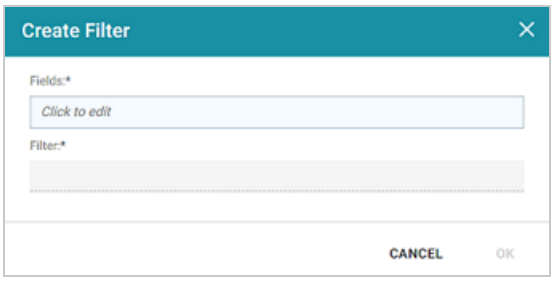
When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **Search**: Enables you to search for a value in the list. The search is case-insensitive.
- **Sort** (⇅): Shows the following options for sorting the results in the filter:
  - **Count Ascending**: Orders results from the smallest count to the largest.
  - **Count Descending**: Orders results from the largest count to the smallest.
  - **Name Ascending**: Orders results in alphabetical order.
  - **Name Descending**: Orders results in reverse alphabetical order.
- **show/hide filters**: This option is displayed when a value is selected. It shows or hides the selection.
- **Menu** (⋮): The menu contains the following options:
  - **Select All Visible**: This option selects all of the items that are listed in the filter.
  - **Clear**: This option becomes available when an item is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Numeric Range Filter

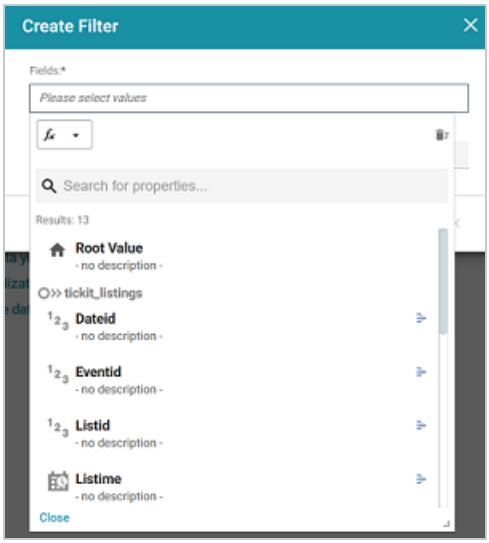
Numeric range filters are used to limit the results on a dashboard to data that falls in (or outside of) certain numeric groupings. Numeric range filters are available for properties with integer and double data types. Follow the instructions below to create a numeric range filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the integer or double type property to filter on. The values for this property will be used to determine the numeric ranges for the filter. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_listings`:



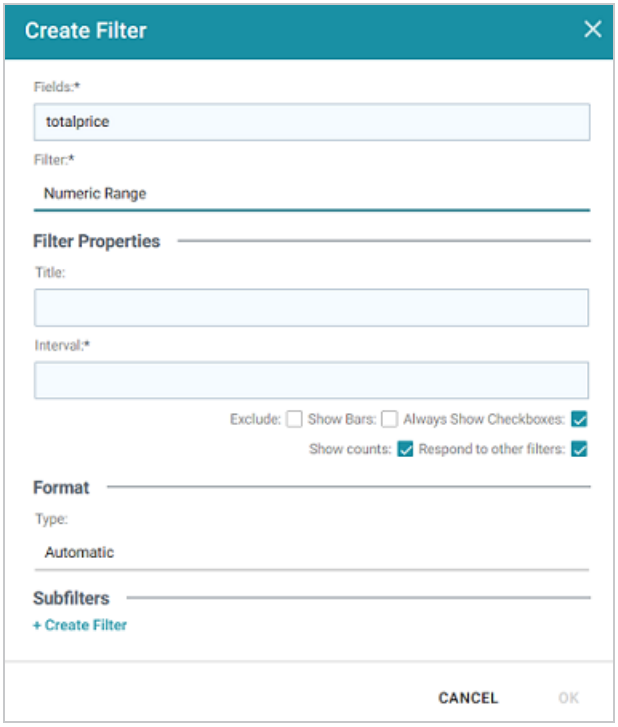
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Numeric Range** from the drop-down list. The dialog box is refreshed to show the Filter Properties, Format, and other options that are available for the filter type:

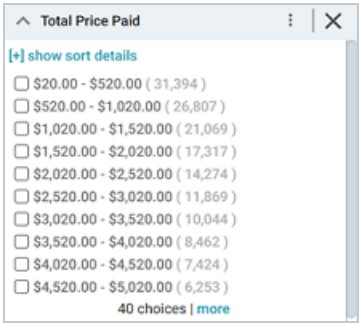


6. Configure any of the following properties. **Interval** is a required field, and the rest of the fields are optional:
    - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
    - **Interval**: This setting specifies a number that defines the size of the groupings.
    - **Exclude**: This setting controls whether selecting a range in the filter narrows the results to show only the records that are *included* in that range or whether selecting a range *excludes* the records that fall in that range. When Exclude is disabled, selecting a range narrows the dashboard results to show only the records that fall in that range. When Exclude is enabled, selecting a range filters out all of the records that fall in that range.
    - **Show Bars**: This setting controls whether the total values for the selected property appear as a bar graphic in the background of the filter.
    - **Always Show Checkboxes**: This setting controls whether checkboxes are shown next to the items in the filter.

- **Show Counts**: This setting controls whether the number of results for each range are displayed in parentheses next to the range.
- **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.

7. If you would like to specify the format for the numeric values that are displayed in the filter, click the **Type** field and select a format from the drop-down list.
8. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
9. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows numeric ranges for the total price paid for tickets to events. The Interval is 500, and format is Money, resulting in ranges for each group of $500.



   Depending on whether the Exclude option is enabled or disabled, clicking a range in the filter refreshes the dashboard to show only the data that is in the selected range or only the data that is outside of the range.

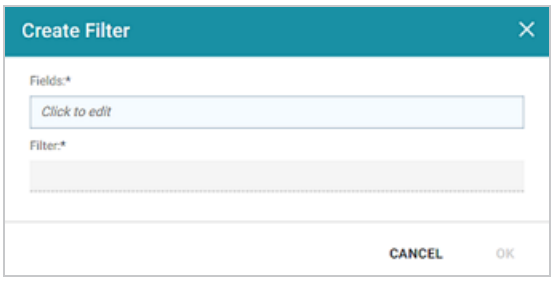
When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **show/hide sort details**: Shows or hides the following option for sorting the results in the filter:
  - **Direction**: This option controls how you want to order the ranges in the filter, depending on the Format specified for the values in the filter. **Count Ascending** and **Name Ascending** order results from the smallest range to the largest and **Count Descending** and **Name Descending** order results from the largest range to the smallest.
- **show/hide filters**: This option is displayed when a range is selected. It shows or hides the selection.
- **Menu** (⋮): The menu contains the following options:
  - **Select All Visible**: This option selects all of the ranges that are listed in the filter.
  - **Clear**: This option becomes available when a range is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Presence Filter

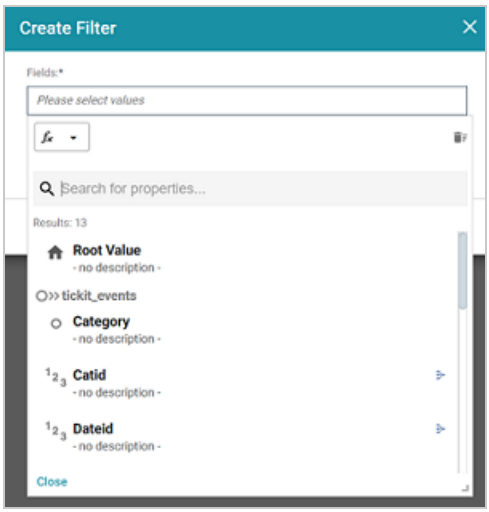
Presence filters group results based on whether the value exists or does not exist. This type of filter is useful for testing whether there are records that are missing a particular value. Presence filters are available for paths and properties of all data types. Follow the instructions below to create a presence filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the path or property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



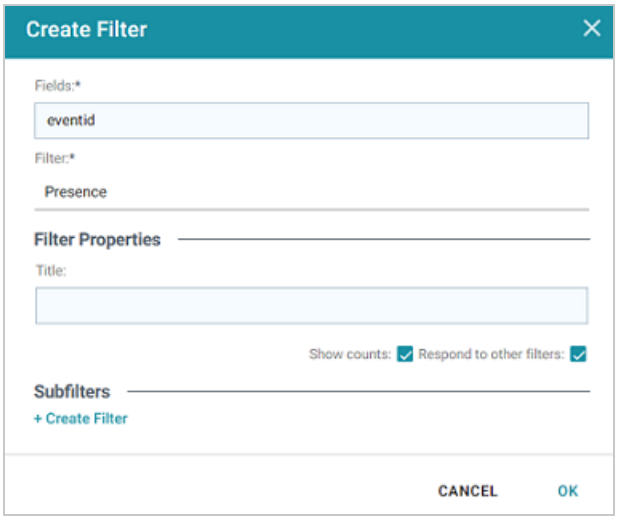
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

     > **Tip**
     > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Presence** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:



6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Show Counts**: This setting controls whether the number of results for each item in the filter is displayed in parentheses next to the item.
   - **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the Dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a Presence Filter that tests whether there are records that are missing the Category ID value. Selecting **Exists** or **Does not exist** filters the dashboard data to show only the records that fall into that category.



When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: This option becomes available when an item is selected. Clicking **Clear** removes the selection.

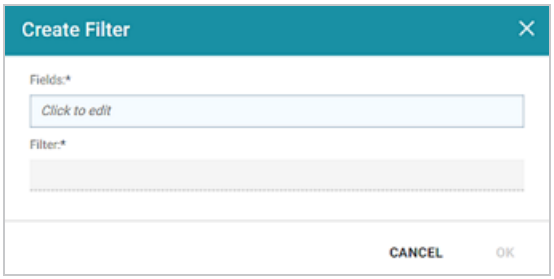
- **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.

- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Quartile Filter

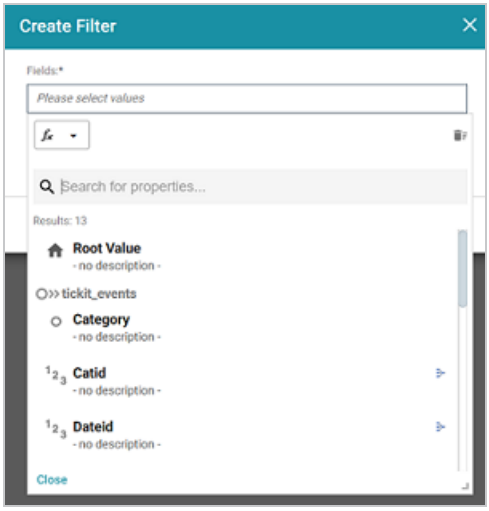
Quartile filters group and rank the values for a property into four equal ranges. This filter is available for properties with integer, double, date, time, and dateTime data types. It is not available for paths. Follow the instructions below to create a quartile filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a property:

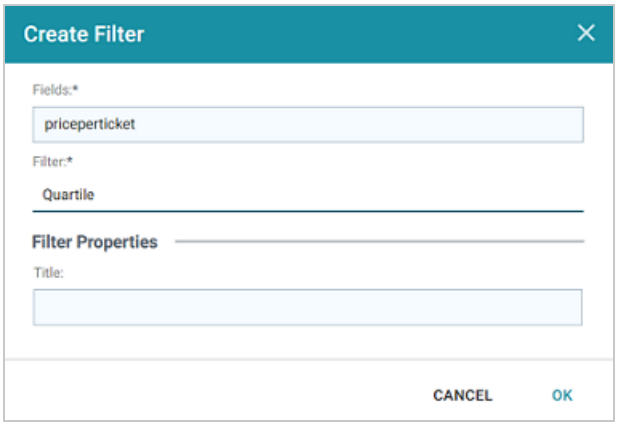
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to

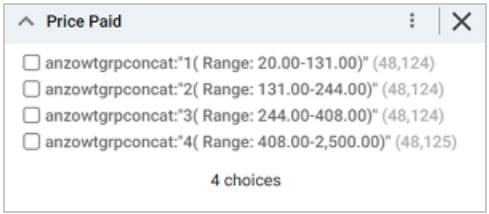
choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Quartile** from the drop-down list. The dialog box is refreshed to show the Filter Properties that are available for the filter type:



6. The only property that is configurable for Quartile filters is the **Title**. This field is optional and defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the value from the Fields field is used as the title.
7. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a filter that groups the price paid for tickets to events into four equal ranges.



   One or more ranges can be selected to filter the dashboard data to show only the records from the selected ranges.

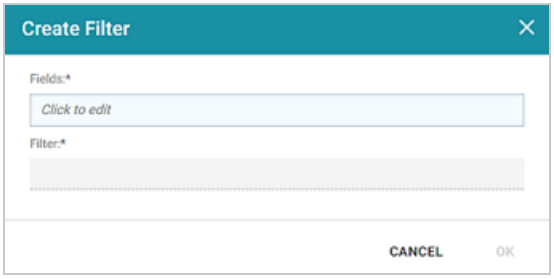
When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **show/hide filters**: This option is displayed when a value is selected. It shows or hides the selection.
- **Menu** (⋮): The menu contains the following options:
  - **Select All Visible**: This option selects all of the items that are listed in the filter.
  - **Clear**: This option becomes available when an item is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Range Slider Filter

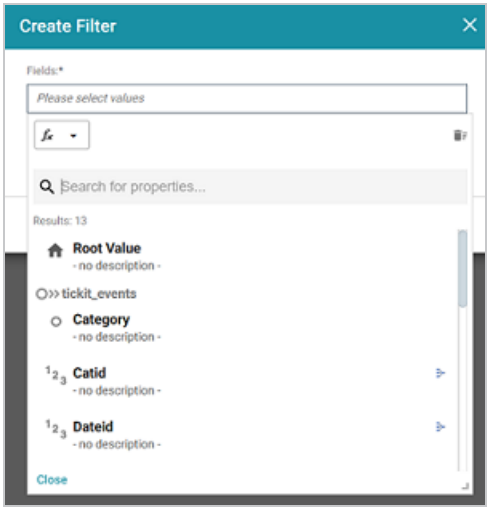
Range slider filters display a slider control that enables you to filter dashboard results by setting one range that you can adjust as needed. This type of filter is available for properties with integer, double, date, time, and dateTime data types. It is not available for paths. Follow the instructions below to create a range slider filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a property:

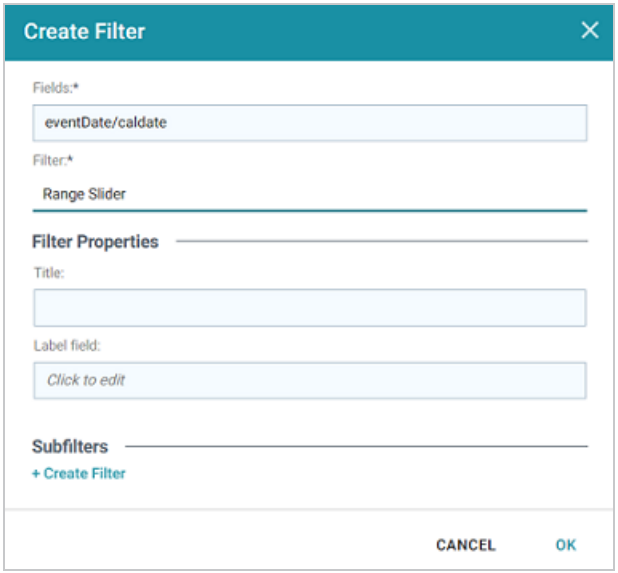
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function

button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

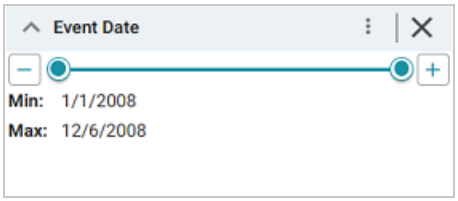
4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Range Slider** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:



6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the list with values from a property other than the one specified in Fields, you can select an alternate property in this field.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.

8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a Range Slider filter that uses a date property to define the range. The dashboard can be filtered by adjusting the Min and Max values to decrease or increase the range.



9. To refresh the results on the dashboard you can click and drag the Min or Max end of the slider or click the plus and minus buttons to adjust the range in small increments.

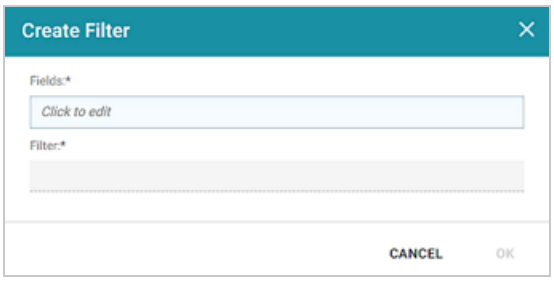
When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: This option is available once the slider has been adjusted. Clicking **Clear** resets the range to the default configuration.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Relative Time Filter

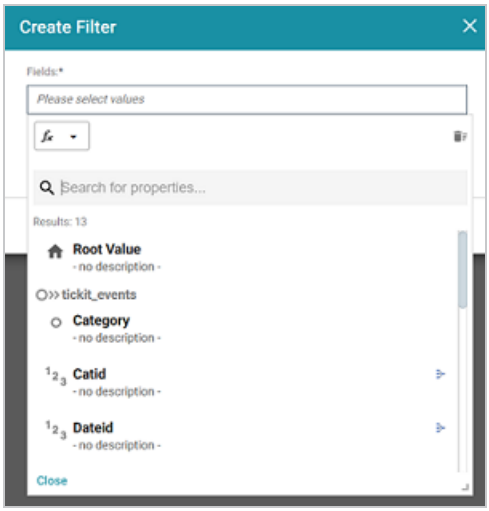
Relative time filters are used to filter for records that fall into a specified increment of time relative to the current time. This type of filter is available for date, time, and dateTime data types. Follow the instructions below to create a relative time filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the date, dateTime, or time type property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



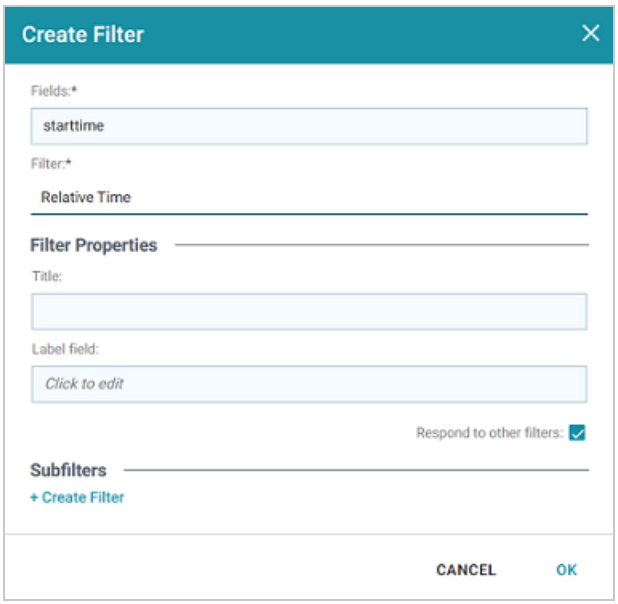
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

> **Tip**
> To view the root values, you can use the STR function to show a string representation of the URIs.

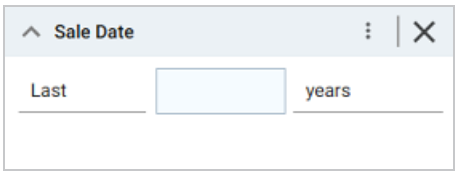
- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Relative Time** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:



6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the filter with a label other than the one specified in Fields, you can select an alternate property in this field.
   - **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below filters data by the Sale Date for tickets.



9. To configure the requirements for the data to display on the Dashboard, specify the following options:
    - On the left, select **Last** or **Next** to configure the relative time direction.
    - In the middle, specify a number to represent the amount of time.
    - On the right, select the time increment to use.

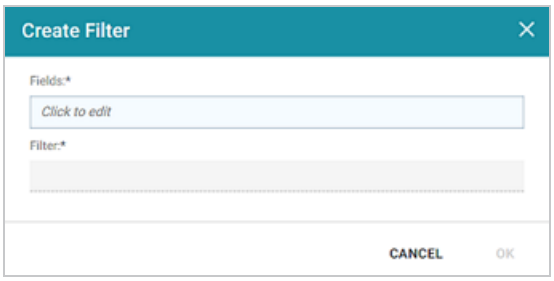
When working with the filter on the dashboard, the following options are also available:

- **Menu** (⋮): The menu contains the following options:
    - **Clear**: This option is available when once the filter options are selected. Clicking **Clear** resets the filter to the default values.
    - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Search Filter

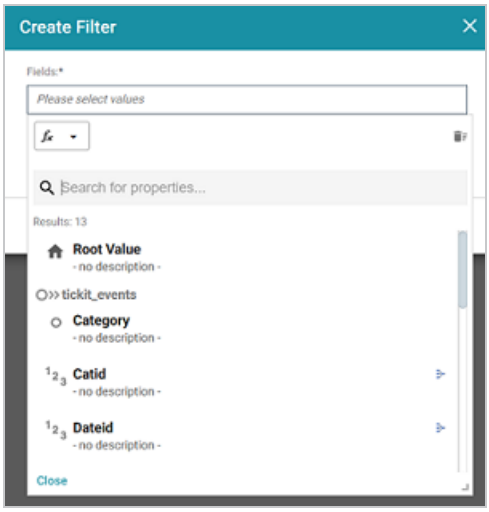
Search filters are used to search for values of a property that contain a partial match, exact match, or do not equal the text that you specify. The search is case-insensitive. This type of filter is available for all data types. It is not available for use with paths. Follow the instructions below to create a search filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



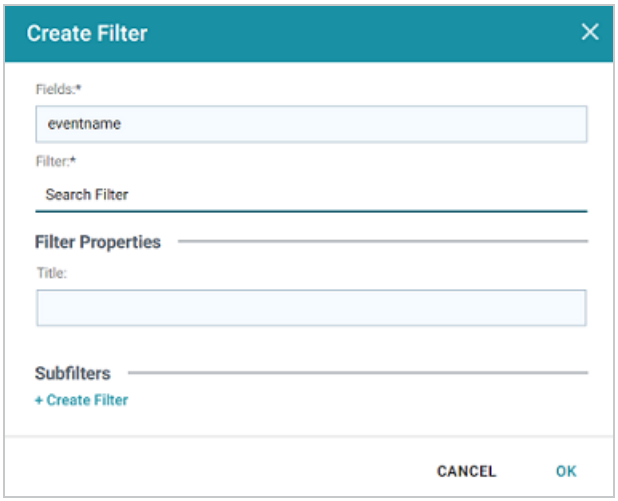
   The list below describes the icons and options that are available when choosing a property:

   - The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

     > **Tip**
     > To view the root values, you can use the STR function to show a string representation of the URIs.

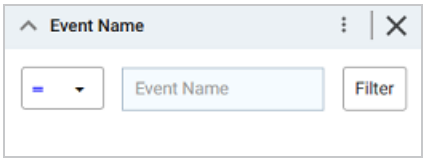
- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Search Filter** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:

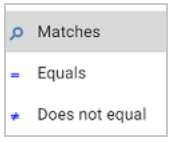


6. The only property that is configurable for Search Filters is the **Title**. This field is optional and defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the value from the Fields field is used as the title.
7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below defines Event Name as the property to search on.



9. To perform a search, configure the following options:
   - On the left, select the type of match to perform. **Matches** includes partial matches and **Equals** is an exact match. Note that matches are case-insensitive.



   - In the middle, specify the value to search for. The search is case-insensitive.
10. Press **Enter** or click **Filter** to perform the search and refresh the data on the dashboard.

When working with the filter on the dashboard, the following options are also available:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: This option becomes available when a search term is entered. Clicking **Clear** removes the term.

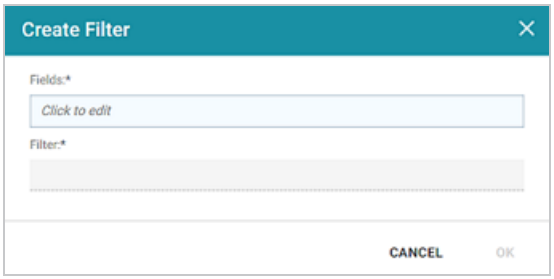
- **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.

- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Single Select List Filter

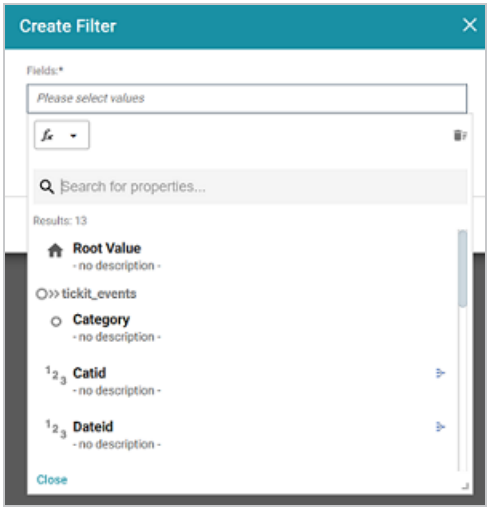
Single select list filters are similar to list filters but only allow you to filter one value from the list at a time. This type of filter is available for properties of all data types but is not available for paths. Follow the instructions below to create a single select list filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the property to filter on. The list of available properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a property:

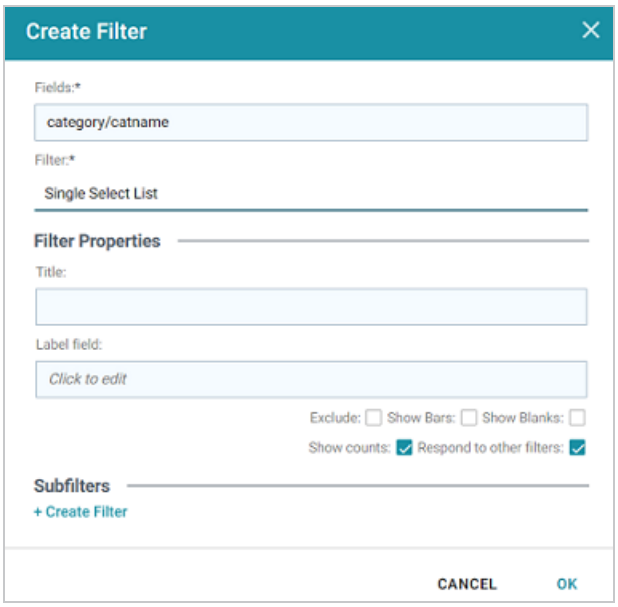
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a property, you can apply a function or formula to that property to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that are available depend on the data type of the selected property. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to

choose additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the property to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Single Select List** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:

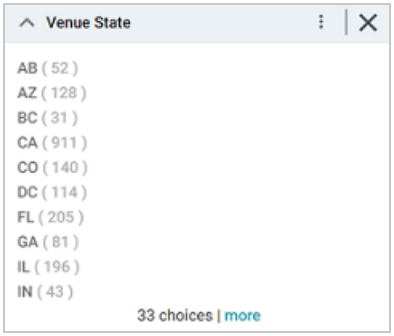


6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Label Field**: If you want to populate the list with values from a property other than the one specified in Fields, you can select an alternate property in this field.
   - **Exclude**: This setting controls whether selecting a value in the filter narrows the results to show only the records that *include* that value or whether selecting a value *excludes* the records that include that value. When Exclude is disabled, selecting a value in the list narrows the dashboard results to show only the records that include that value. When Exclude is enabled, selecting a value filters out all of the records that include that value.

- **Show Bars**: This setting controls whether the total values for the selected property appear as a bar graphic in the background of the filter.
- **Show Blanks**: This setting controls whether to include null values for the selected property by listing them as **Blank** in the filter.
- **Show Counts**: This setting controls whether the number of results for each item in the filter is displayed in parentheses next to the item.
- **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.

7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows a filter that lists each of the states that have event venues. One state at a time can be selected to filter the data on the dashboard.



   Depending on whether the Exclude option is enabled or disabled, selecting an item in the filter refreshes the Dashboard to show only the data includes the selected value or only the data that does not include the selected value.

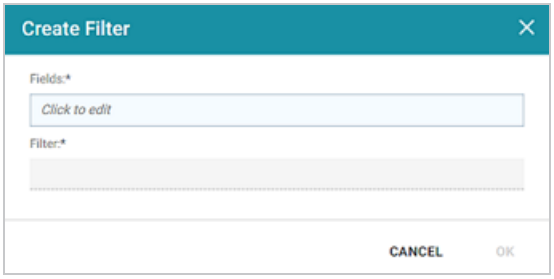
When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **Menu** (⋮): The menu contains the following options:
  - **Clear**: This option is available when an item is selected. Clicking **Clear** removes the selection.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Adding a Types Filter

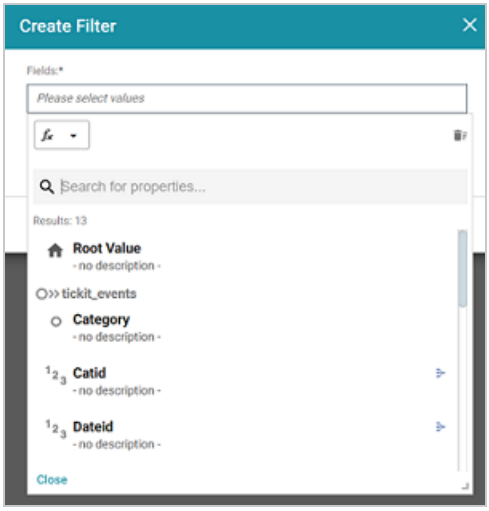
Types filters are used to filter data according to the types of data (classes) that are connected by a specified path. This type of filter is available only for paths and not properties. Follow the instructions below to create a types filter.

1. Open the dashboard that you want to add the filter to.
2. In the Hi-Res Analytics main toolbar, click **Filters** and select **Create a Filter**. The Create Filter dialog box is displayed.



3. Click in the **Fields** field to open the property drop-down list and determine the path to filter on. The list of available classes, paths, and properties depends on the selected data type for the dashboard.

   For example, the following image shows the list of properties that are available for a dashboard whose source is a graphmart that contains data about tickets sold for various types of events. The type for the dashboard is `tickit_events`:



The list below describes the icons and options that are available when choosing a path:

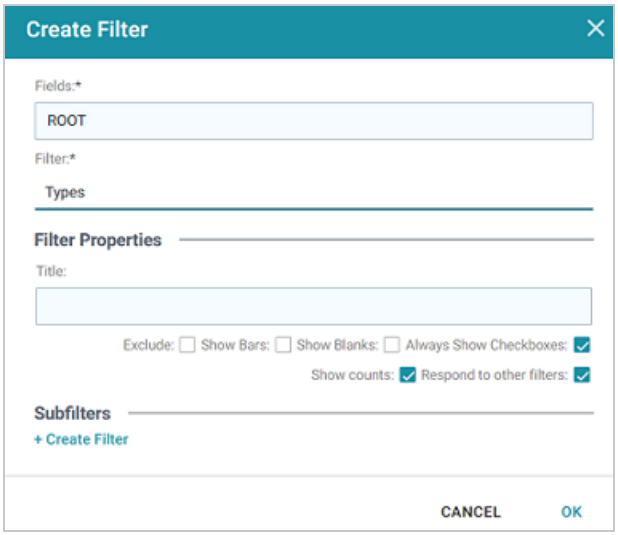
- The **Root Value** (🏠) is the instance URI for the root resource—the URI for the instances of the class that was chosen as the data type for the dashboard.

  > **Tip**
  > To view the root values, you can use the STR function to show a string representation of the URIs.

- Linked classes are represented by incoming (○«) and outgoing (○») connection icons. The properties in those classes with a path to another class are denoted with a circle icon (○). Selecting a linked property navigates to that class and displays its properties.
- When a property or path is selected, the breadcrumbs at the top of the dialog box show you the property path. You can click the clear icon (🗑) to clear the path and start again or you can click the forward (➡) or back (⬅) arrows to go forward or backward one or more levels at a time.
- After you have selected a path, you can apply a function or formula to calculate the values that are displayed in the filter. To add a function, click the function button (**fx**) at the top of the drop-down list. The functions that become available depend on the data type of the selected path. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose

additional properties and functions. For more information, see Calculating Values in Lenses and Filters.

4. After you have selected the path to filter on, click **Close** to close the Fields drop-down list.
5. Next, click the **Filter** field and select **Types** from the drop-down list. The dialog box is refreshed to show the Filter Properties and other options that are available for the filter type:

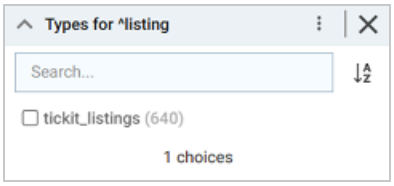


6. Configure any of the following properties. All of the fields are optional:
   - **Title**: Defines the title that appears at the top of the filter when it is added to the dashboard. If Title is blank, the Fields value is used as the title.
   - **Exclude**: This setting controls whether selecting a value in the filter narrows the results to show only the records that *include* that value or whether selecting a value *excludes* the records that include that value. When Exclude is disabled, selecting a value in the filter narrows the dashboard results to show only the records that include that value. When Exclude is enabled, selecting a value filters out all of the records that include that value.
   - **Show Bars**: This setting controls whether the total values for the selected property appear as a bar graphic in the background of the filter.
   - **Show Blanks**: This setting controls whether to include null values for the selected property by listing them as **Blank** in the filter.

- **Always Show Checkboxes**: This setting controls whether checkboxes are shown next to the items in the filter.
- **Show Counts**: This setting controls whether the number of results for each filter value are displayed in parentheses next to the value.
- **Respond to Other Filters**: This setting controls whether the results of this filter change based on the selections made in other filters on the dashboard.

7. If you would like to be able to further constrain the data that appears in the filter, you can add one or more subfilters. To add a subfilter, click **Create Filter** under Subfilters. The process of creating a subfilter is the same as the process for the parent filter. However, the subfilter is not displayed on the dashboard. It is visible only when editing the parent filter, and the subfilter's configuration affects only the parent filter and any sibling subfilters.
8. When you have finished configuring the filter, click **OK** to add it to the dashboard. The new filter appears in the left pane of the dashboard and displays the values that are available for filtering the displayed data.

   For example, the filter in the image below shows that there is one type of class of data that is connected by the chosen path.



   Depending on whether the Exclude option is enabled or disabled, selecting an item in the filter refreshes the dashboard to show only the data includes the selected value or only the data that does not include the selected value.

When working with the filter on the dashboard, the following options are available for sorting and configuration:

- **Search**: Enables you to search for a value in the filter. The search is case-insensitive.
- **Sort** (): Shows the following options for sorting the results in the filter:
  - **Count Ascending**: Orders results from the smallest count to the largest.
  - **Count Descending**: Orders results from the largest count to the smallest.
  - **Name Ascending**: Orders results in alphabetical order.
  - **Name Descending**: Orders results in reverse alphabetical order.
- **show/hide filters**: This option is displayed when a value is selected. It shows or hides the selection.
- **Menu** (): The menu contains the following options:
  - **Select All Visible**: This option selects all of the items that are listed in the filter.
  - **Clear**: This option is available when one or more items are selected. Clicking **Clear** removes the selections.
  - **Designer**: Selecting this option opens the designer so that you can view or change the filter configuration.
- **Close** (**X**): Clicking **X** deletes the filter from the dashboard. This action cannot be undone.

Calculating Values in Lenses and Filters

Anzo provides many standard and advanced functions that you can use to compute the values that are displayed on a dashboard. When selecting properties and paths for lenses and filters, you can add calculations by selecting functions from a list or by writing your own formula. The Hi-Res Analytics application enables you to save your formulas as computed properties that can be reused on other dashboards, lenses, and filters. This topic provides instructions for using functions and formulas to calculate displayed values, saving formulas as computed properties, and reusing computed properties.

- Computations in Filters and Lenses
- Applying Functions and Formulas to Properties
- Saving Formulas for Reuse
- Reusing Computed Properties

Computations in Filters and Lenses

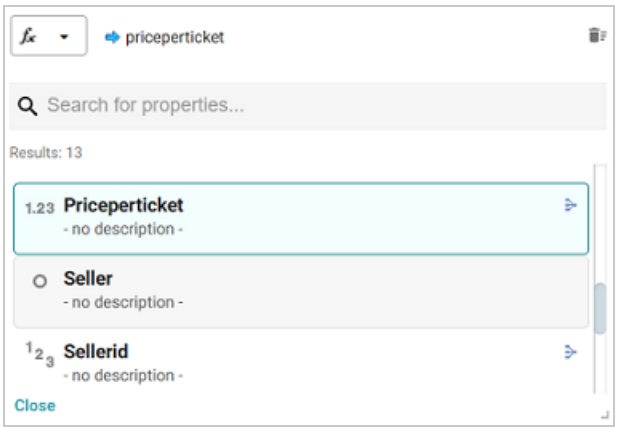
When you apply formulas to properties in filters, Anzo performs the calculation across all of the values that exist for the selected property and then groups the results into the list of values that the calculations return. For multiple value properties, all values for that property are included in the calculations.

When you apply formulas to properties in lenses, the calculation results depend on the data type of the dashboard or lens. If the property belongs to a class that allows multiple values, Anzo performs the calculation on each set of multiple values and returns the results as one record in the lens. If the class includes single value properties, the calculation is performed separately for each single value.

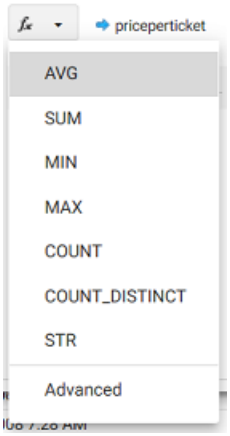
Applying Functions and Formulas to Properties

Follow the instructions below to use a function or formula to compute the values in a lens or filter.

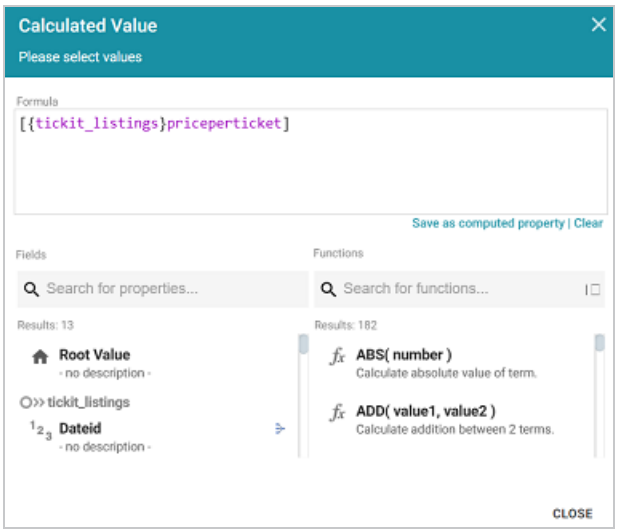
1. Create a new lens or filter or open the Designer for an existing lens or filter.
2. In the drop-down list for selecting properties or fields, select the property or path for which you want to compute the values. For example:



3. Click the function button (**fx**) to display the list of standard functions. The list varies depending on the data type of the selected property.



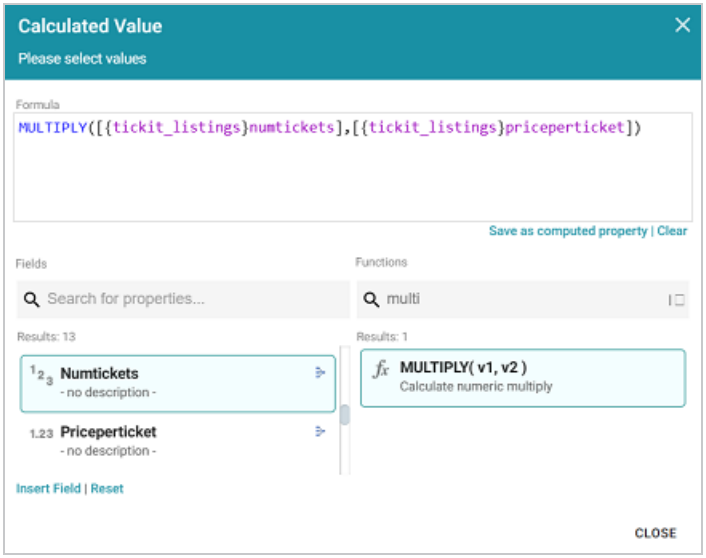
4. Click a function to apply it to the property that you chose. For information about each of the available functions, see Function and Formula Reference.
5. To choose a more advanced function or type a formula, click **Advanced**. The Calculated Value dialog box opens and enables you to choose additional properties and functions as well as type your own calculation.



> **Tip**
>
> To create an advanced formula, it might help to get started by viewing the functions listed in the Functions column on the right side of the screen. Each function includes the syntax to use for creating a formula that uses that function.

6. In the Functions column, double-click a function to add it to the Formula field at the top of the dialog box.
7. Place your cursor in the Formula where you want to insert the property to perform the calculation on (for example, inside the parentheses) and then double-click the property in the Fields column. If the syntax for the function includes characters such as commas, type the characters in the appropriate location in the formula. You can click the **Clear** link on the bottom right of the Formula field any time to clear that field and start over.

   For example, the formula below calculates the total price paid by multiplying the values in the price per ticket column with the value in the number of tickets column:



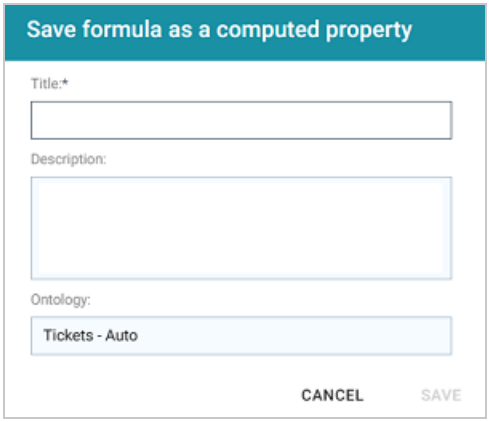
8. When you are finished writing a formula, you have two options:
   - If you want to use the formula now without saving it for later use, click **Close** to close the Calculated Value dialog box. Then complete the lens or filter configuration.
   - If you want save the formula for reuse, click the **Save as computed property** link and follow the instructions below in Saving Formulas for Reuse.

Saving Formulas for Reuse

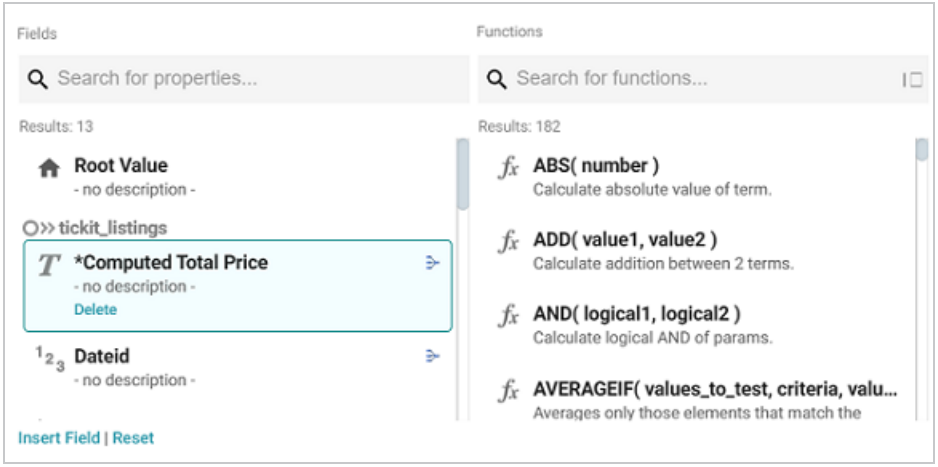
Follow these instructions to save a formula as a computed property that you can use in other lenses and filters that target the same class of data.

1. When you have finished writing a formula in the Calculated Value dialog box, click the **Save as computed property** link below the Formula field. The Save formula as computed property dialog box opens.

2. In the **Title** field, type a name for the new computed property.



3. Type a description of the new property in the **Description** field.
4. If necessary, click in the **Ontology** field to choose another ontology to save the property in. If you want to save this property in multiple ontologies, you can click the **Save as computed property** link again after saving the property in the current ontology.
5. Click **Save**. Anzo saves the new property and labels it with an asterisk (*). The property becomes available in the Fields column in the Calculated Value dialog box. For example, the image below shows the Computed Total Price property that saves the formula from the Applying Functions and Formulas to Properties section above.

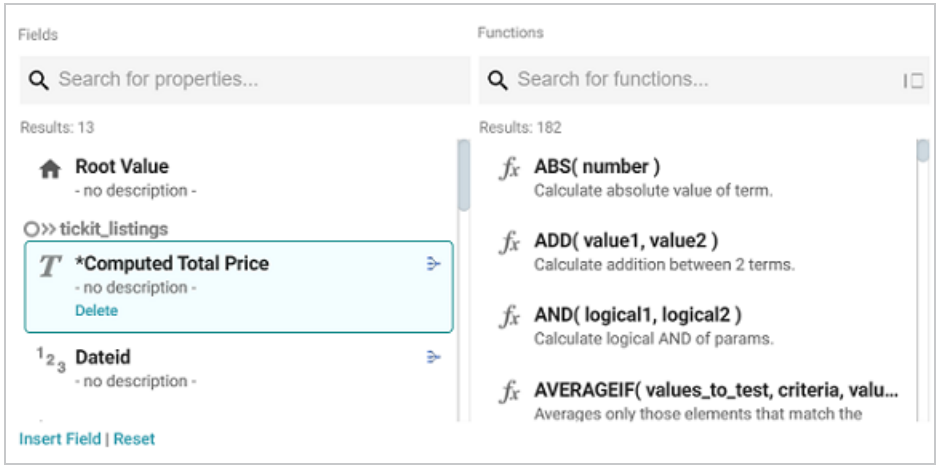


6. Click **Close** to close the dialog box.

Reusing Computed Properties

When an ontology contains computed properties, any other dashboards, lenses, and filters that use that ontology can also use the computed properties as long as they also use the same data type or class of data that the computed property is saved in. Follow the instructions below to use a computed property.

1. Open the Designer for the filter or lens where you want to apply the computed property.
2. Click in the **Fields** or **Column Value Expression** field to open the property selection drop-down list. The drop-down list includes any computed properties that are available for use with the selected data type. Computed properties are labeled with an asterisk (*).

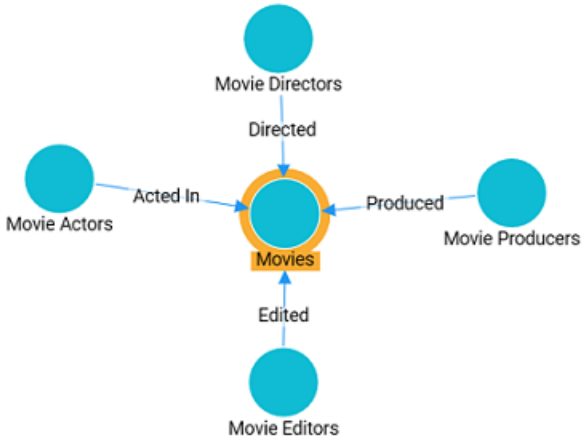


3. To use the property as-is, select the property and then close the drop-down list.
4. If you want to make changes to the formula and save it as a different computed property, select the property and then click the function button (**fx**) to open the Calculated Value dialog box. Follow the instructions in Saving Formulas for Reuse above to edit the formula and then save a new computed property.

Combining Data from Multiple Classes

Though you must choose one base data type (or class) for each dashboard, selecting a data type with connections to other classes enables you to configure lenses and filters that combine the data from those classes. This powerful capability can help surface the semantic relationships in your data and enable you to leverage those relationships to access and integrate all of the data in the knowledge graph. When choosing the base data type for a dashboard, it helps to consider all of the desired filters.

For example, consider the following data model for a movie data set:

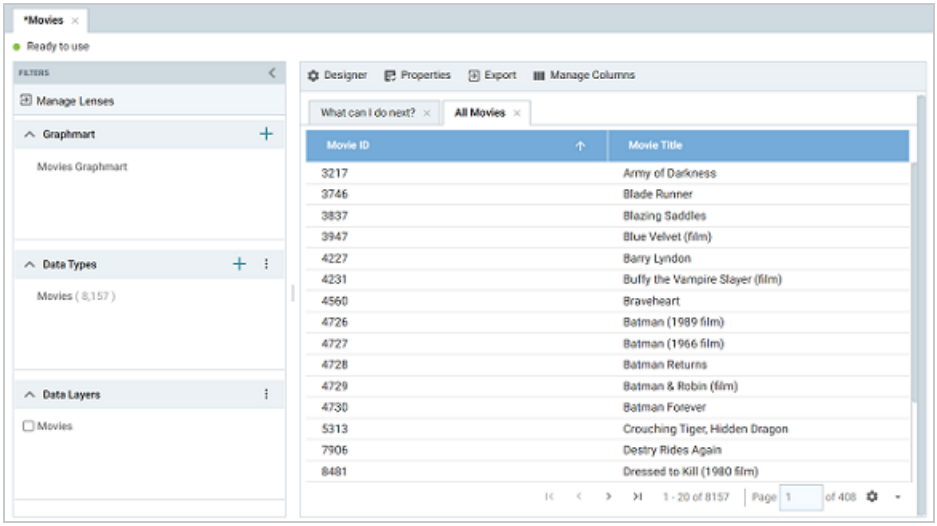


By creating a dashboard that specifies **Movies** as the base data type, the lenses and filters in the dashboard can navigate the paths to properties in the other classes, Movie Actors, Movie Directors, Movie Producers, and so on. This topic provides guidance on accessing data from multiple classes in filters and lenses.

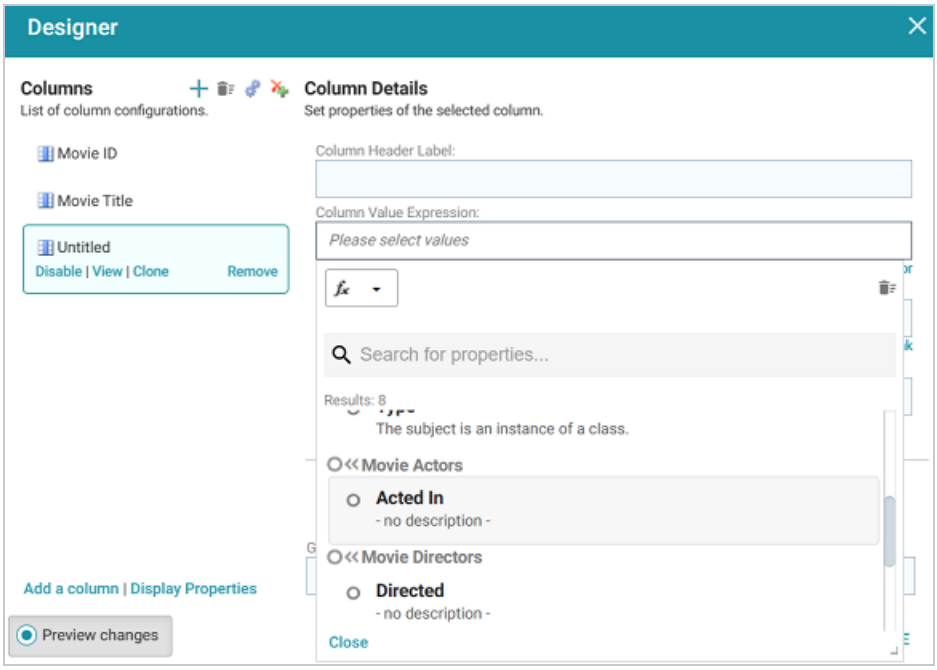
- Combining Classes in a Lens
- Filtering on Multiple Classes

Combining Classes in a Lens

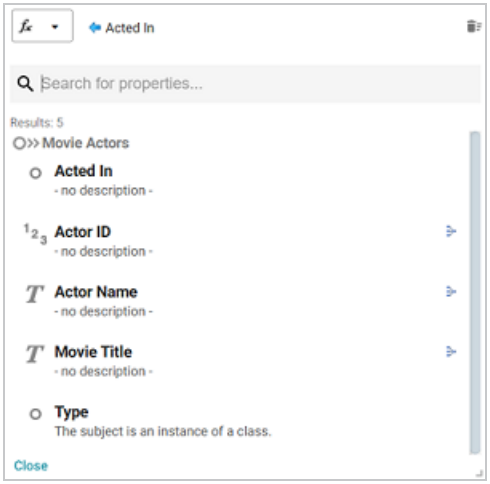
The image below shows a dashboard that accesses the graph for the above model. The specified data type is **Movies**, and a table lens displays all of the columns/properties in the Movies class:



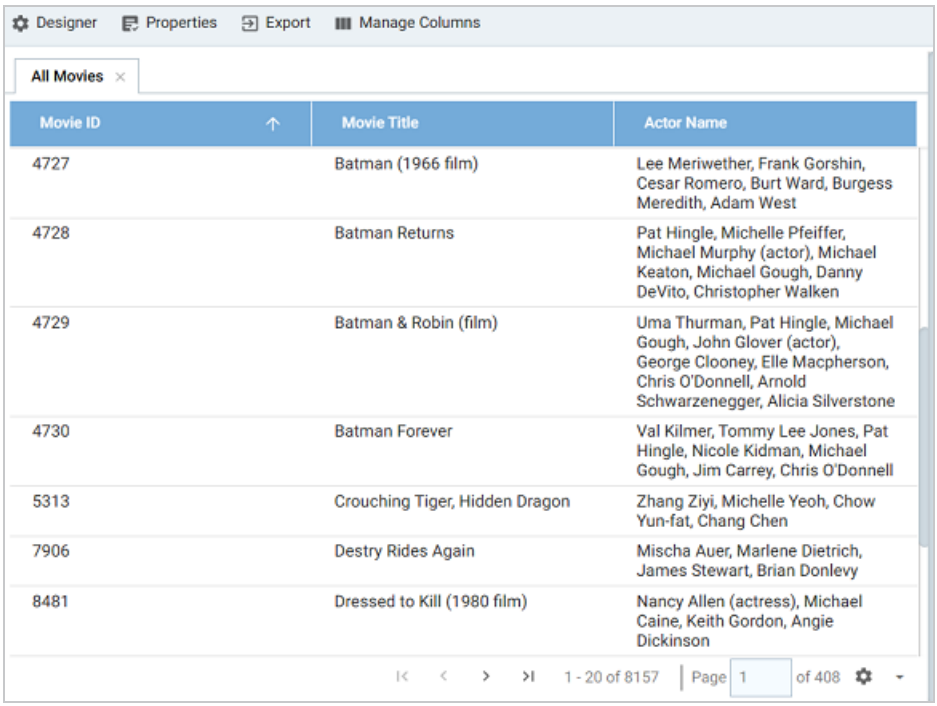
Lenses and filters can be configured to leverage the relationships from the base class to the connected classes. For example, adding a column that navigates the **ActedIn** path to access the **Movie Actors** class could be used to display values such as the names of the actors who starred in the movies. To navigate the relationship in the lens Designer, the **ActedIn** path from the **MovieActors** class is selected for the new column:



Once the path is chosen, all of the properties from the **Movie Actor** class are displayed:

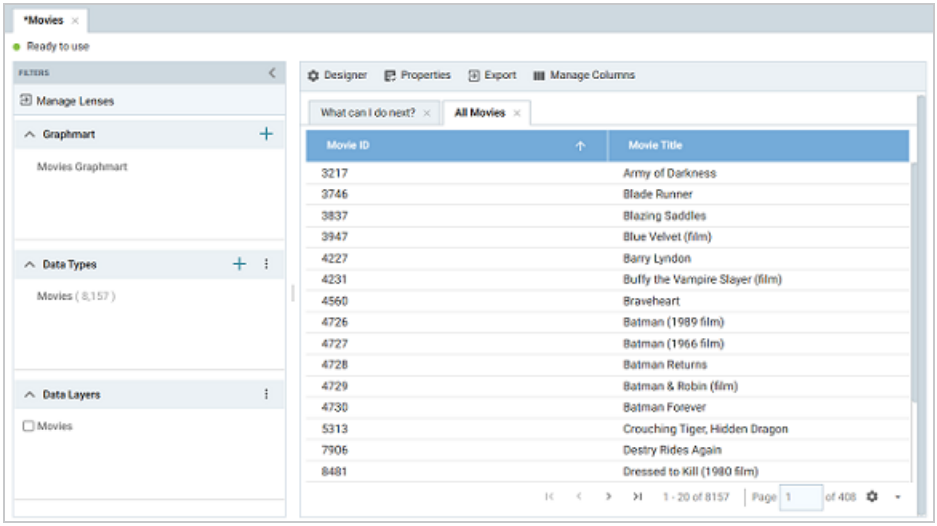


Selecting **ActorName** adds the column to the dashboard. The actors from each movie are now integrated into the lens even though the actor name values are not in the base class.

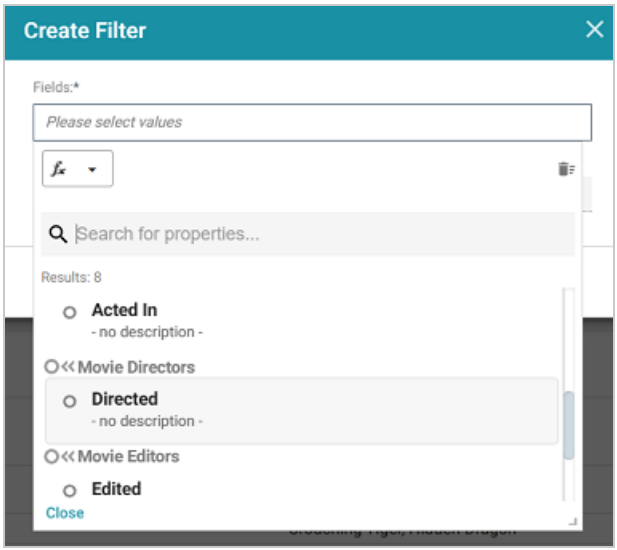


Filtering on Multiple Classes

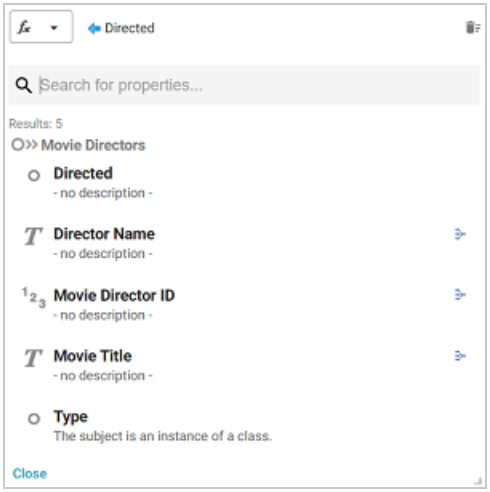
In addition to combining classes in lenses, you can also apply filters across classes. Like the example above, the image below shows a dashboard where the specified data type is **Movies**, and a Table lens displays all of the columns/properties in the Movies class:



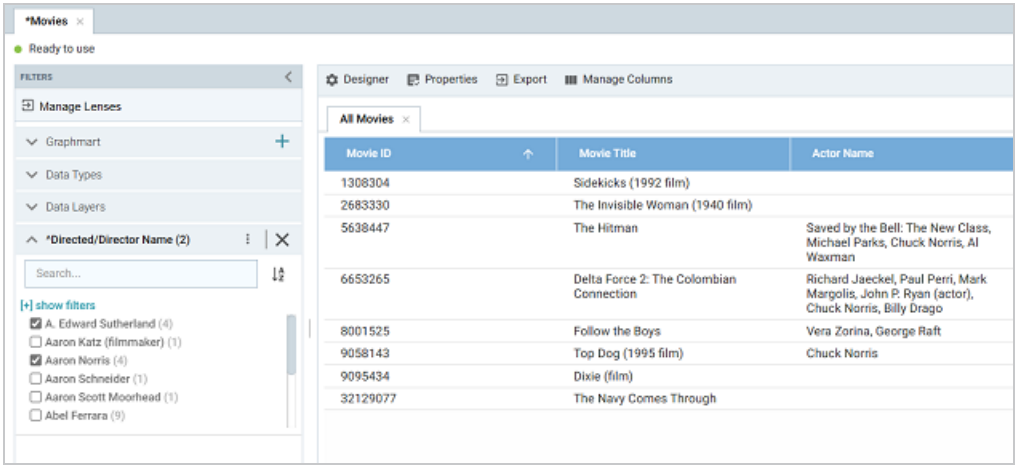
A filter can be configured to leverage the relationships from the base class to the connected classes. For example, adding a filter that navigates the **Directed** path to access the **Movie Directors** class could be used to display, and filter on, the name of the director for each movie. To navigate the relationship in the Create Filter dialog box, the **Directed** path from the **MovieDirectors** class is selected for the Field to filter on:



Once the path is chosen, the properties from the **Movie Directors** class are displayed:



Selecting the **Director Name** property and choosing **List** as the type of filter adds a filter to the dashboard that lists all of the directors in the graph. Users can select particular director names to filter the lens so that it only shows the movies that include one or more of the selected directors.

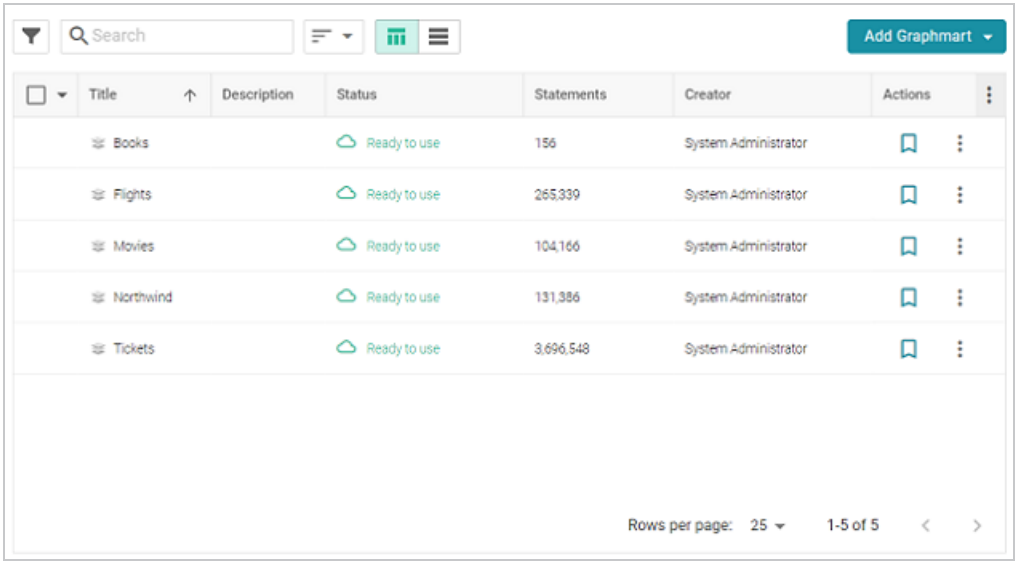


Using the same path traversal, filters could also be created to narrow the data to certain actors or producers by following the relationships to the Movie Actors or Movie Producers classes. For more information about creating or editing lenses and filters, see Creating a Lens and Working with Filters.

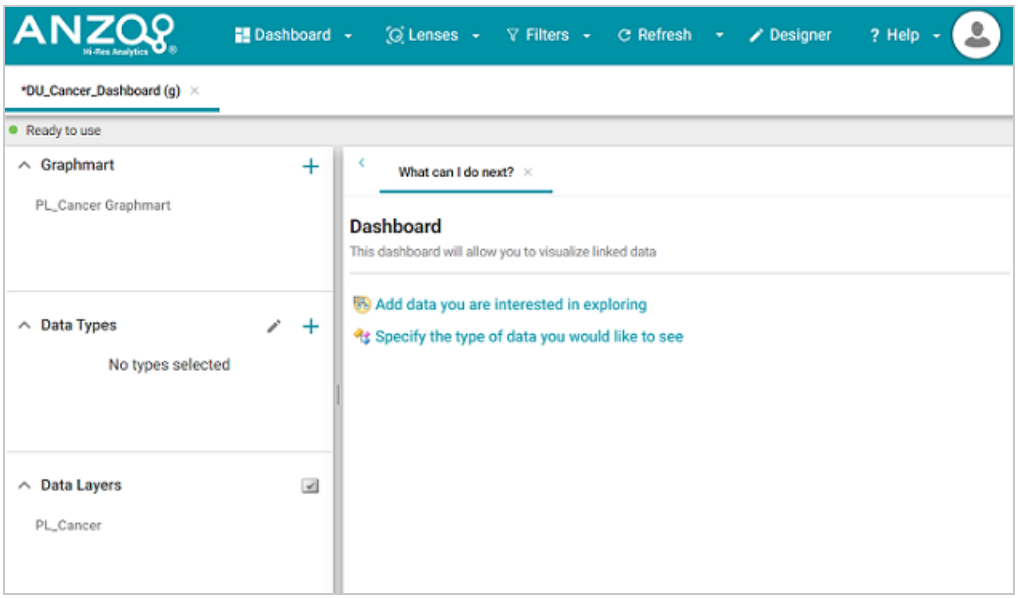
Searching for Text in Unstructured Documents

The Hi-Res Analytics application incorporates the Elasticsearch search engine to enable you to perform full text searches on unstructured documents. This topic provides instructions for creating a dashboard with text search capability and running a search across unstructured documents.

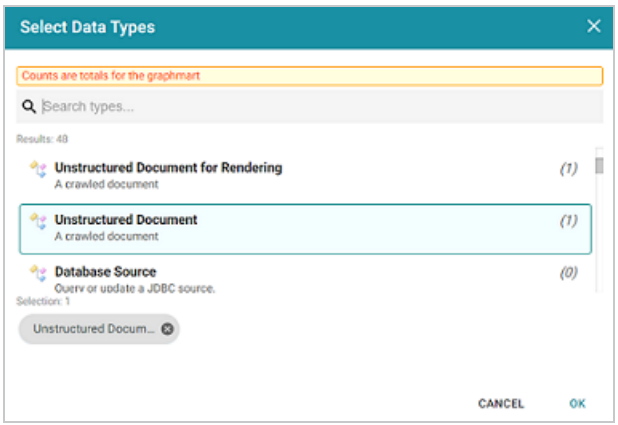
1. In the Anzo application, expand the **Blend** menu and click **Graphmarts**. Anzo displays a list of the existing graphmarts. For example:



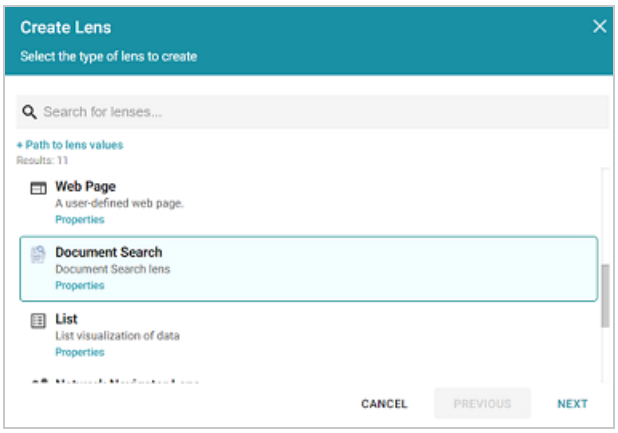
2. On the Graphmarts screen, click the name of the graphmart that contains the unstructured data. Anzo displays the graphmart overview screen.
3. Click the **Create Dashboard** button. The Hi-Res Analytics application opens and displays the Create Dashboard dialog box. Leave **Graphmart Dashboard** selected and click **Next**.
4. Type a name for the dashboard in the **Title** field and enter an optional **Description**. Then click **OK**. Anzo creates the dashboard and populates the Graphmart and Data Layers panels. For example:



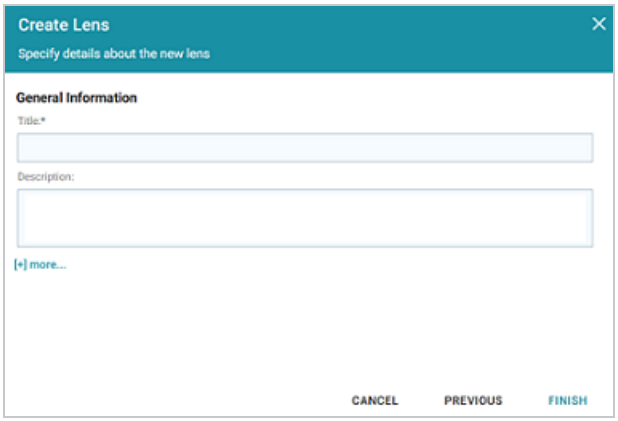
5. In the Data Types panel, click the plus icon (**+**) to open the Select Data Types dialog box. In the dialog box, select **Unstructured Document**.



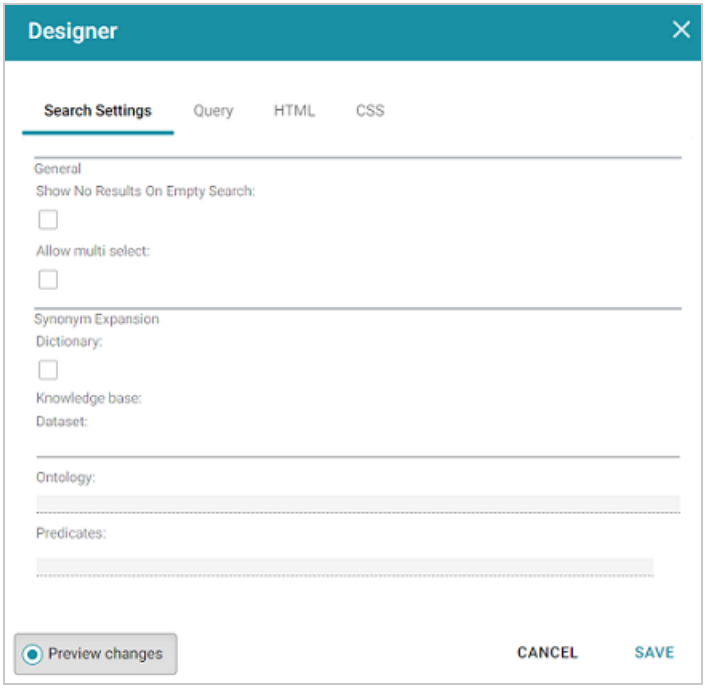
6. Click **OK**. Anzo adds the data type to the Data Types panel.
7. Next, click the **Lenses** button in the main toolbar and select **New** from the drop-down list. Anzo opens the Create Lens dialog box.



8. In the Create Lens dialog box, select **Document Search** and then click **Next**. Anzo displays the General Information dialog box.

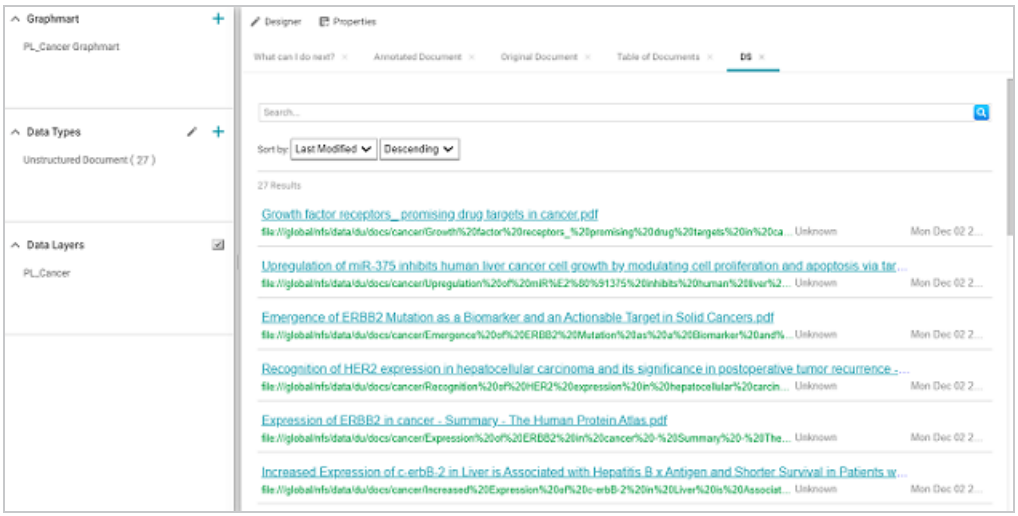


9. Type a name for the lens in the **Title** field and include an optional **Description**. Then click **Finish**. Anzo opens the Document Search Designer where you can configure the search settings or customize the style sheet, query, and HTML, if necessary. For example:

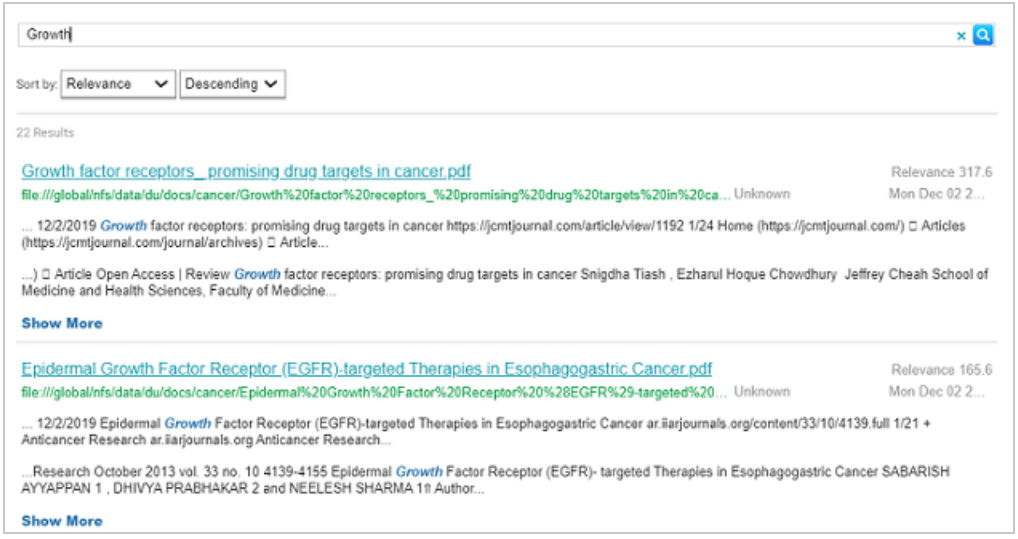


10. In the Designer, change the optional search settings as needed. The list below describes each option:
    - **Show No Results on Empty Search**: Determines whether documents are listed in the search results before a search is run. When enabled, the Document Search lens remains blank until a search is run.
    - **Allow Multi Select**: Determines whether a user can select multiple documents at a time in the results. When enabled, multiple documents can be selected by holding the Shift key and clicking documents in the results.
    - **Synonym Expansion Dictionary**: Determines whether to display an option for including synonyms in text searches. When enabled, the lens displays an **Include Synonyms** checkbox next to the Search field.
    - **Knowledge Base Dataset**: Enables you to include a knowledge base in the search if one exists. Click the field to select an available knowledge base.
    - **Ontology**: Enables you to select a data model to use for the search.
    - **Predicates**: Enables you to select specific predicates from the model.

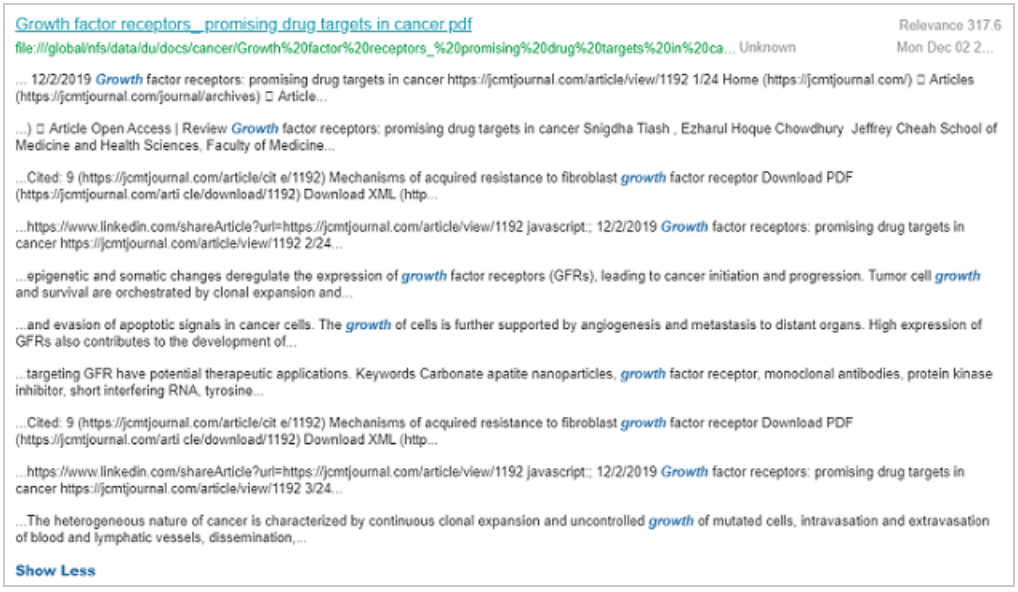
11. Click **Save**. Anzo add the lens to the dashboard. Depending on the search settings, the lens displays the list of documents. For example:



12. To run a search, type the text to find in the **Search** field and press **Enter**. See the Supported Search Syntax section below for information about supported search syntax. Anzo finds documents that include the search value and displays the documents, snippets of text to show the context of where the matches were found, and the Elasticsearch relevance score for the match. For information about how the relevance score is calculated, see What Is Relevance? in the Elasticsearch documentation. For example:



Clicking **Show More** expands the result to display additional matches. For example:



13. To refine the search, alter the text in the **Search** field and press **Enter** again. You can also click highlighted terms in the search results to open a dialog box that shows the full annotated document where the match was found. For example:



Supported Search Syntax

This section describes the keyword search syntax that Anzo supports.

Wildcard Characters: ? and *

- **?**: Use a question mark (?) to represent a single wildcard character. For example, in the search**co?l**, the resulting documents will include terms like "cool" or "coal."

- *: Use an asterisk (*) to represent multiple wildcard characters. For example, in the search**col***, the resulting documents will include terms like "collect" or "color."

Boolean Operators: +, -, OR, AND, NOT

- **+**: Use a plus (+) character to indicate mandatory matches. For example, in the search **flight +New York**, the resulting documents can include "flight" as an optional match and must include "New York."
- **-**: Use a minus (-) character to indicate a term that must not match. For example, in the search **flight +New York -Los Angeles**, the resulting documents can include "flight" as an optional match, must include "New York," and must not include "Los Angeles."
- **OR**: In the search **New York OR Los Angeles**, the resulting documents will include a match for either "New York" or "Los Angeles."
- **AND**: In the search **New York AND Los Angeles**, the resulting documents must include matches for both "New York" and "Los Angeles."
- **NOT**: In the search **New York NOT Los Angeles**, the resulting documents must include "New York" and cannot contain "Los Angeles."
- Grouping operators: In the search**(flight AND New York) OR Los Angeles**, the resulting documents will include "flight" and "New York" and optionally include "Los Angeles."

Fuzzy Matches: ~*n*

To search for a fuzzy match, use a tilde (~) character followed by a number to represent the number of fuzzy or incorrect characters. For example, in the search **Flgth~3**, the resulting documents could include the term "Flight."

Regular Expressions

For example, the following search expression matches email addresses: **/([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})/**.

For more information about the regular expression syntax that Elasticsearch supports, see Regular expression syntax in the Elasticsearch documentation.

Sharing Access to Dashboards and Lenses

This topic includes reference information about dashboard and lens permissions and provides instructions for configuring permissions.

- Dashboard Level Permissions
- Lens Level Permissions
- Configure Dashboard or Lens Permissions

Dashboard Level Permissions

Dashboard level permissions affect a user's ability to view, modify, delete, design, or configure dashboards and dashboard permissions. There are three predefined permission sets that can be assigned to an Anzo user or group. You also have the option to customize the set of permissions that are applied to a user or group.

The table below lists the predefined permission sets and describes the privileges that are granted for each permission that is part of the predefined set:

Set	Permission	Allows a user to:
View	Read	• Search for and open accessible dashboards. • Save As a new dashboard. • Share the dashboard. • View dashboard Properties. • View lens Properties. • Export lenses.
Modify	In addition to the **Read** permission described above, the **Modify** set includes the **Write** and **Delete** permissions described below.	
	Write	• Use the dashboard Designer to change the

Set	Permission	Allows a user to:
		dashboard. • Clone lenses.
	Delete	• Delete the dashboard.
Admin	In addition to the **Read**, **Write**, and **Delete** permissions described above, the **Admin** set includes the **Manage** permission described below.	
	Manage	• The Manage permission relates only to the Security tab. If a user has this permission, they can modify dashboard access by changing permissions for a user, group, or role.

Default Dashboard Permissions

The table below lists the predefined permission sets that are applied by default when a new dashboard is created. Besides the sysadmin user, the dashboard creator is granted **Admin** privileges by default. The Everyone role is granted **View** privileges by default. No other users, groups, or roles have dashboard permissions assigned by default.

User or Role	Applied Permission Set
Sysadmin	Admin
Dashboard Creator	Admin
Everyone Role	View

Lens Level Permissions

Lens level permissions affect a user's ability to view, modify, delete, design, or configure lenses and lens permissions. There are three predefined lens permission sets that can be assigned to an Anzo user or group. You also have the option to customize the set of permissions that are applied. While dashboard level permissions can affect a user's ability to clone a lens, the appropriate lens level permissions are required to be able to perform functions such as deleting or redesigning a lens.

The table below lists the predefined permission sets and describes the privileges that are granted for each permission that is part of the predefined set:

Set	Permission	Allows a user to:
View	Read	• Search for and open accessible lenses. • View lens Properties. • Export lenses.
Modify	In addition to the **Read** permission described above, the **Modify** set includes the **Write** and **Delete** permissions described below.	
	Write	• Use the lens Designer to change the lens. • Rename the lens. • Clone the lens.
	Delete	• Delete the lens.
Admin	In addition to the **Read**, **Write**, and **Delete** permissions described above, the **Admin** set includes the **Manage** permission described below.	
	Manage	• The Manage permission relates only to the Security tab. If a user has this permission, they can modify

Set	Permission	Allows a user to:
		lens access by changing permissions for a user, group, or role.

Default Lens Permissions

The table below lists the predefined permission sets that are applied by default when a new lens is created. Besides the sysadmin user, the lens creator is granted **Admin** privileges by default. The Everyone role is granted **View** privileges by default. No other users, groups, or roles have lens permissions assigned by default.

User or Role	Applied Permission Set
Sysadmin	Admin
Lens Creator	Admin
Everyone Role	View

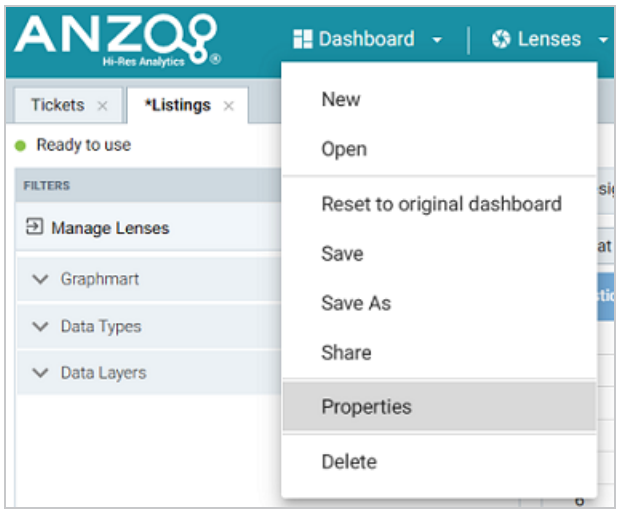
Configure Dashboard or Lens Permissions

This section provides instructions for modifying dashboard or lens properties to grant or restrict access to your dashboards and lenses.

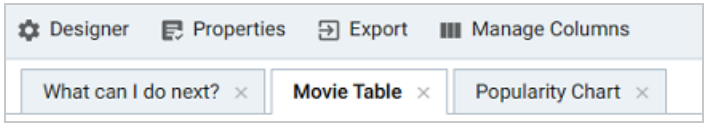
> **Note**
> Data can be restricted at a higher level than a dashboard. Though users might have access to view your dashboards and lenses, graphmart permissions determine whether they can view the data that the dashboard displays.

1. Open the dashboard for which you want to modify access.
2. Open the Properties dialog box for the either dashboard or for a specific lens:
   - To change access at the dashboard level, click **Dashboard** in the main toolbar and select **Properties**.

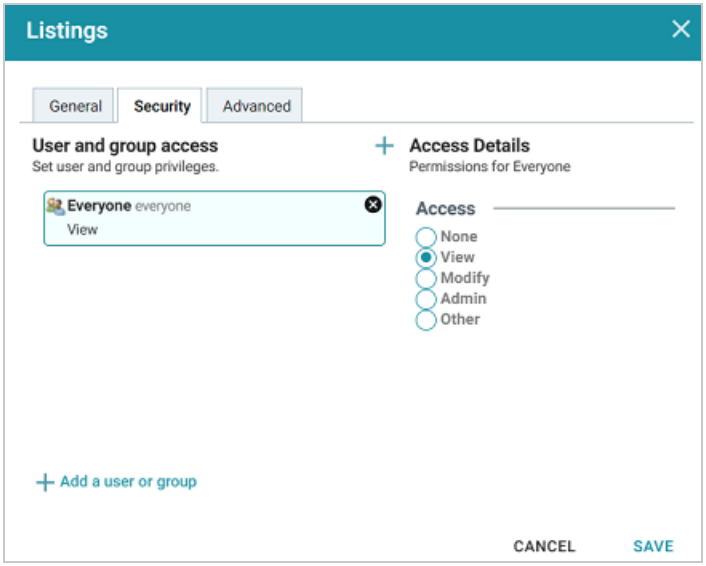
     > **Note**
     > Sharing a dashboard automatically shares the lenses in that dashboard.

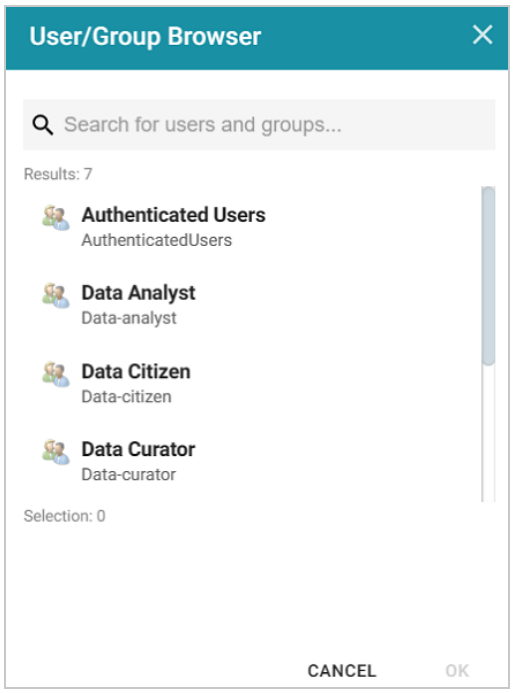
   - To change access for a lens in the dashboard, click the lens to display it and then click the **Properties** button in the object toolbar.

3. In the Properties dialog box, click the **Security** tab. This tab lists the available groups and users who can view this dashboard or lens. The image below shows the Security tab at the dashboard level:



4. Select a user or user group to manage, and then modify any of the following options:
   - **Remove a user or group**: Click the delete icon (X) next to the user or group.
   - **Add a user or group**: Click **Add a user or group**. On the User/Group Browser dialog box, select the users or groups that you want to add. Then click **OK**.



   - **Access Details**: Select the access level for the selected user or group. Refer to Dashboard Level Permissions or Lens Level Permissions above for details about each of

the access options.

- **None**: No permissions set for the selected dashboard or lens.
- **View**: Grants the **View** predefined permission set for the selected dashboard or lens.
- **Modify**: Grants the **Modify** predefined permission set for the selected dashboard or lens.
- **Admin**: Grants the **Admin** predefined permission set for the selected dashboard or lens.
- **Other**: Enables you to set custom access levels for the selected dashboard or lens. Select the checkboxes to enable any combination of the following permissions: Read, Write, Delete, or Manage (administrator permissions).

5. Click **Save** to save the changes.

To get a URL to your dashboard that you can send to users, click **Dashboard** in the main toolbar and select **Share**. The Share Dashboard dialog box opens and displays a URL for the dashboard. You can copy the link and send it to users.

Access Data with the Query Builder

The Query Builder in the user interface provides options for accessing data in various data sources. The Query Builder includes a **Find** option that enables users to search for quads by specifying a single subject, object, predicate, or graph name. It also includes a **Query** option that enables users to write, run, and save SPARQL queries. The topics in this section provide information about accessing data using the Query Builder.

In this section:

Running SPARQL Queries in the Query Builder 907

Searching for Quads in the Query Builder 915

Running SPARQL Queries in the Query Builder

The Query Builder includes a Query tab for writing and running SPARQL queries. The query editor provides syntax assistance, type-ahead suggestions for model entity names, and automated prefix creation and query formatting for readability. It also includes the option to save queries for later use.

The Query tab supports running queries against the following data sources:

- Graphmarts and specific data layers within graphmarts
- Linked Datasets
- Data sources: Anzo System Data Source, AnzoGraph, Anzo System Tables, Data Profiling Metrics, LDAP Primary Data Source

Note

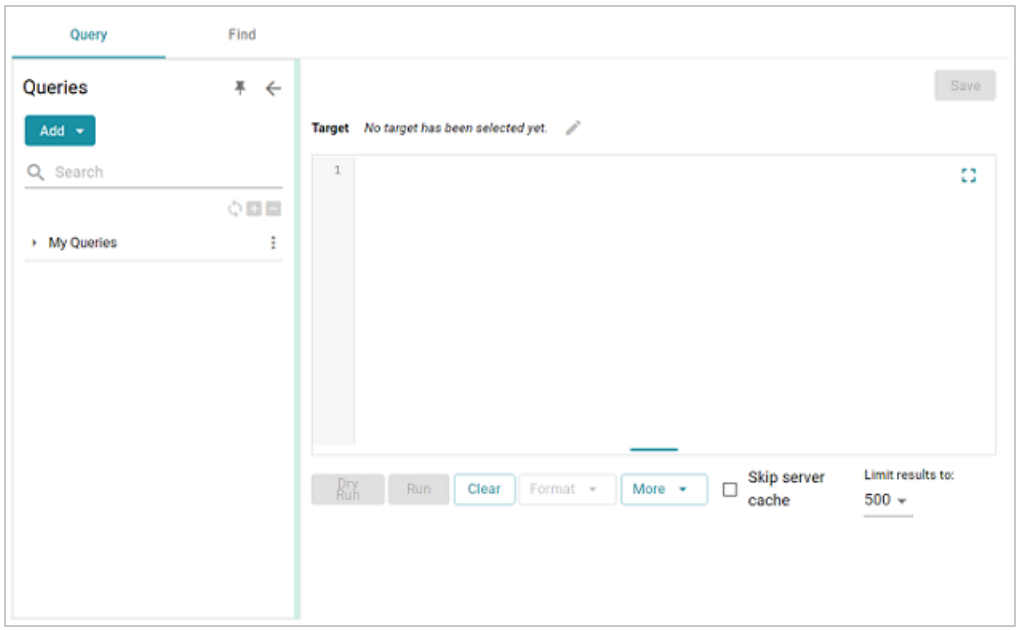
To ensure that queries perform well and do not consume too many resources on the system, keep the following guidelines in mind when developing and testing queries:

- Set a limit on the number of results to return.
- Avoid cross-product joins
- Consider using VALUES clauses instead of FILTER clauses.
- When retrieving a large number of values, use subqueries instead of OPTIONAL clauses.

For query templates and additional details about best practices, see SPARQL Best Practices and Query Templates.

Follow the instructions below to write and run SPARQL queries against any of the supported data sources.

1. In the Anzo application, expand the **Access** menu and click **Query Builder**. Anzo displays the query editor.



2. At the top of the screen, click the edit (✎) icon next to the Target data source:

   - To query data that is in a graphmart, select the **Graphmart** radio button.

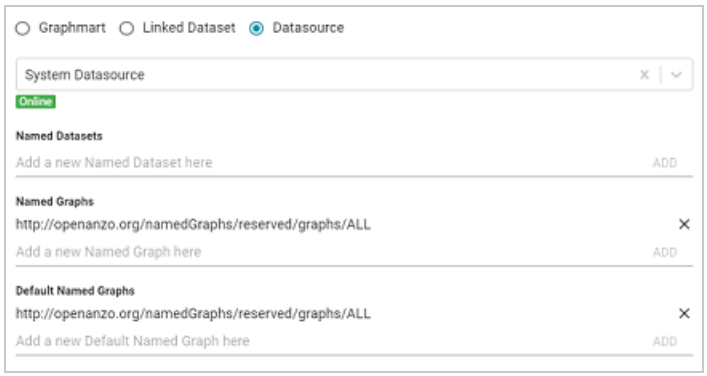
     Click the **Select Graphmart** drop-down list and select the graphmart to query. If you want to narrow the scope of the query by selecting one or more data layers in the graphmart, click the **Select Layers** drop-down list and select the layer or layers to target.

   - To query data that is in the Datasets catalog, select the **Linked Dataset** radio button.

     Click the **Select linked dataset** drop-down list and select the linked dataset to query.

   - To run queries against the system data source, data profiling metrics, Anzo system tables, LDAP server, or AnzoGraph, select the **Datasource** radio button.



Click the **Datasource** drop-down list and select the target source:

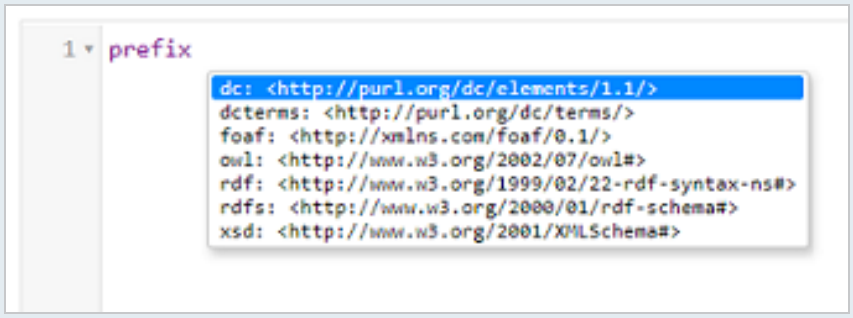
- Select **System Datasource** to search the local Anzo system volume.
- Select the name of an AnzoGraph instance to search for data in graphmarts that are loaded to that instance.
- Select **Data Profiling Metrics** to search the data metrics volume.
- Select **LDAP Primary Datasource** to search the directory server.
- Select **System Tables** to search Anzo system table data.

By default, the Named Graphs and Default Named Graphs values are set to all named graphs (`http://openanzo.org/namedGraphs/reserved/graphs/ALL`). If you want to narrow the scope of the query, you can replace the values with specific graph URIs. To list multiple graphs, separate URIs with a space.

3. When you have finished configuring the target, click outside of the dialog box to return to the Query screen.
4. In the text box below the target, compose the SPARQL query. See SPARQL Best Practices and Query Templates for tips on writing queries. For information about the supported SPARQL functions, see Function and Formula Reference.

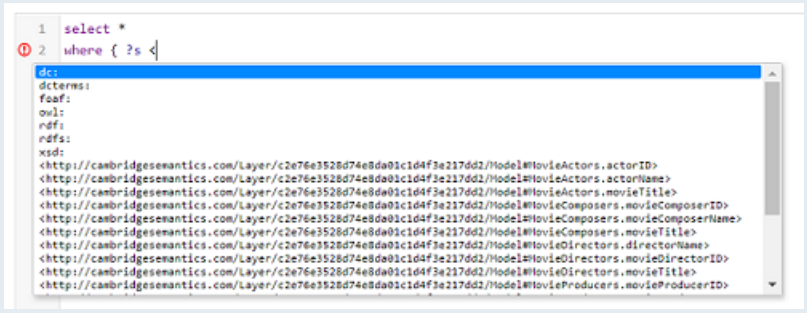
Tip

When adding PREFIX statements, once you type **prefix** followed by a space Anzo displays a tooltip that lists all of the global prefixes that are defined for your system. Clicking a prefix in the list inserts a PREFIX statement into the query. For example:

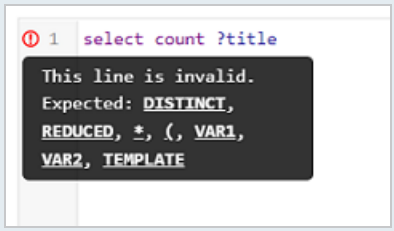


In addition, typing the abbreviation for a global prefix followed by a colon (:) automatically inserts the PREFIX statement into the query without opening the tooltip.

When typing entity URIs in the WHERE clause, the query builder also offers suggestions by listing the properties in the data source. You can click an item in the list to insert that entity. For example:

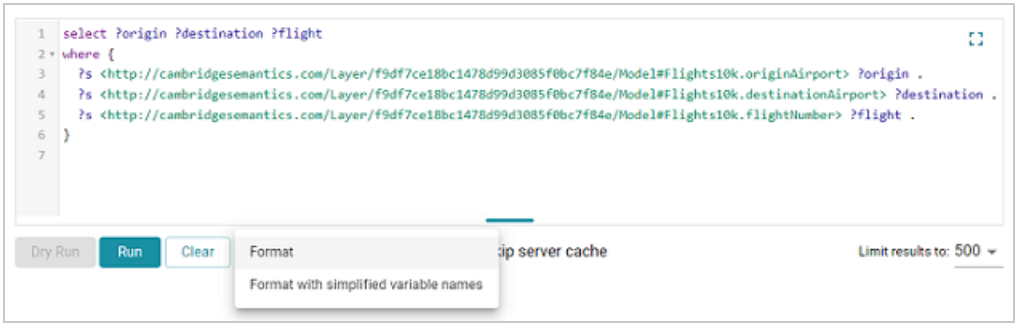


When a red exclamation mark icon () is displayed next to a line number, you can hover the pointer over the icon to view guidance on how to resolve the issue. For example:



5. If you want to format the query for readability, click the **Format** button and select from the following options:
   - **Format**: Auto-creates prefixes, inserts URI abbreviations, and restructures the query for readability.
   - **Format with simplified variable names**: Auto-creates prefixes, inserts URI abbreviations, simplifies variable names by changing them to ?_var1, ?_var2, ?_var*N*, and restructures the query for readability.

   For example, the image below shows a query before it is formatted.



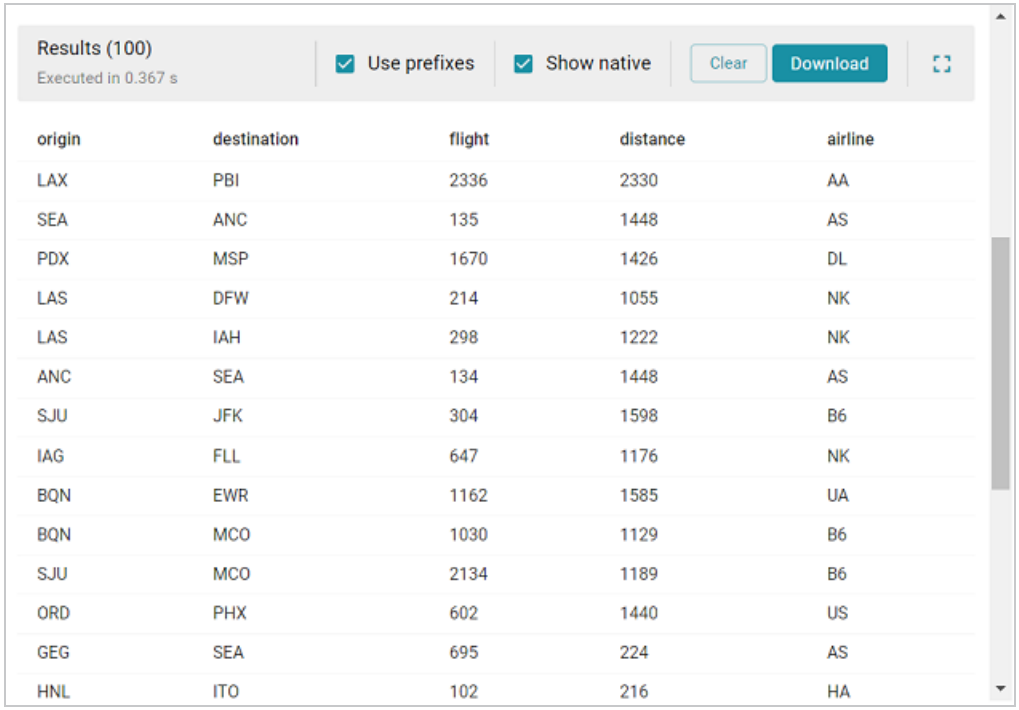
   After the query is formatted, prefixes and URI abbreviations are added. For example:



6. If the query is an INSERT or DELETE query, the Dry Run button becomes active. You can click **Dry Run** to do a test run of the update. In a test run, Anzo runs a version of the query where INSERT or DELETE is replaced with CONSTRUCT, and the results report the number of statements that the query affects, i.e., the number of additions or removals per graph. If the results are unexpected, you can adjust the query before clicking running the query and committing the updates.
7. If necessary, change the query limit. By default, query results are limited to 500. To adjust the limit, click the **Limit results to** drop-down list below the query editor and select a value. For example:



8. To run the query, click **Run**. The results appear at the bottom of the screen. For example:



origin	destination	flight	distance	airline
LAX	PBI	2336	2330	AA
SEA	ANC	135	1448	AS
PDX	MSP	1670	1426	DL
LAS	DFW	214	1055	NK
LAS	IAH	298	1222	NK
ANC	SEA	134	1448	AS
SJU	JFK	304	1598	B6
IAG	FLL	647	1176	NK
BQN	EWR	1162	1585	UA
BQN	MCO	1030	1129	B6
SJU	MCO	2134	1189	B6
ORD	PHX	602	1440	US
GEG	SEA	695	224	AS
HNL	ITO	102	216	HA

> **Tip**
> You can click any value in the result list to copy that value to the clipboard.

9. To save the query for later use, click the **Save** button at the top of the screen. Anzo displays the New Query dialog box.



10. In the dialog box, specify a name for the query in the **Title** field and an optional description in the **Description** field.

11. You can click the **Folder** drop-down list to select a different folder or create a new one to save the query in. Then click **Save**. The query is saved and added to the **Queries** panel on the left side of the screen.

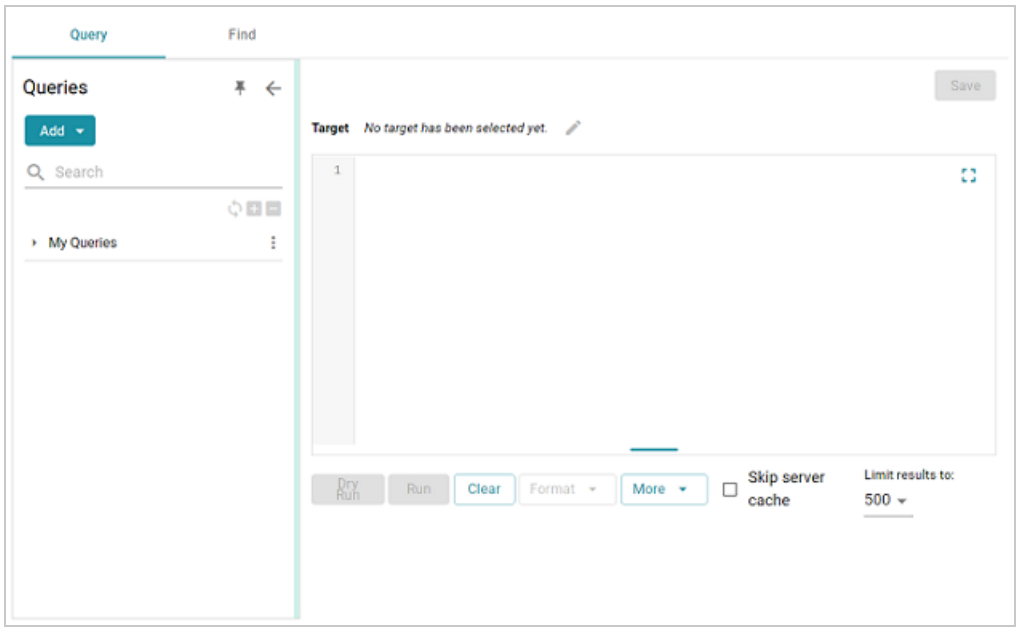
Searching for Quads in the Query Builder

The Query Builder includes a Find tab for searching for data by specifying a single subject, object, predicate, graph name or any combination of those elements. Statements that match the search criteria are returned in quads, and the screen includes quick filters that enable users to toggle filters on and off to show or hide any of the quad elements. The Find tab supports searches against the following data sources:

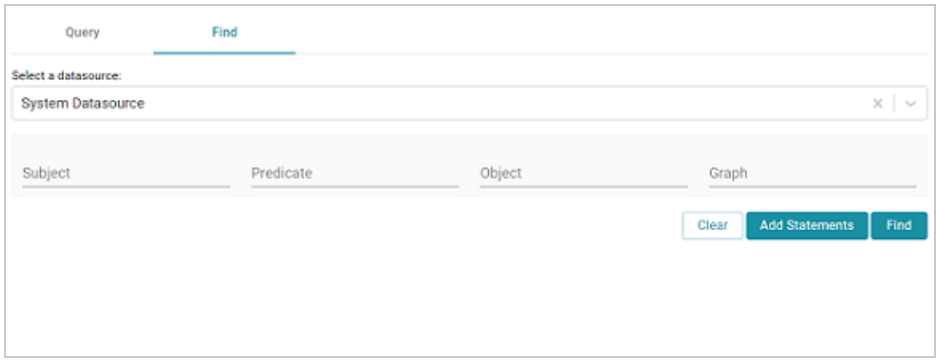
- Anzo System Data Source
- AnzoGraph
- Anzo System Tables
- Data Profiling Metrics
- LDAP Primary Data Source

When finding data in the system data source, users have the option to modify or delete statements directly in the user interface. Follow the instructions below to find data in any of the supported data sources.

1. In the Anzo application, expand the **Access** menu and click **Query Builder**. Anzo displays the query editor.

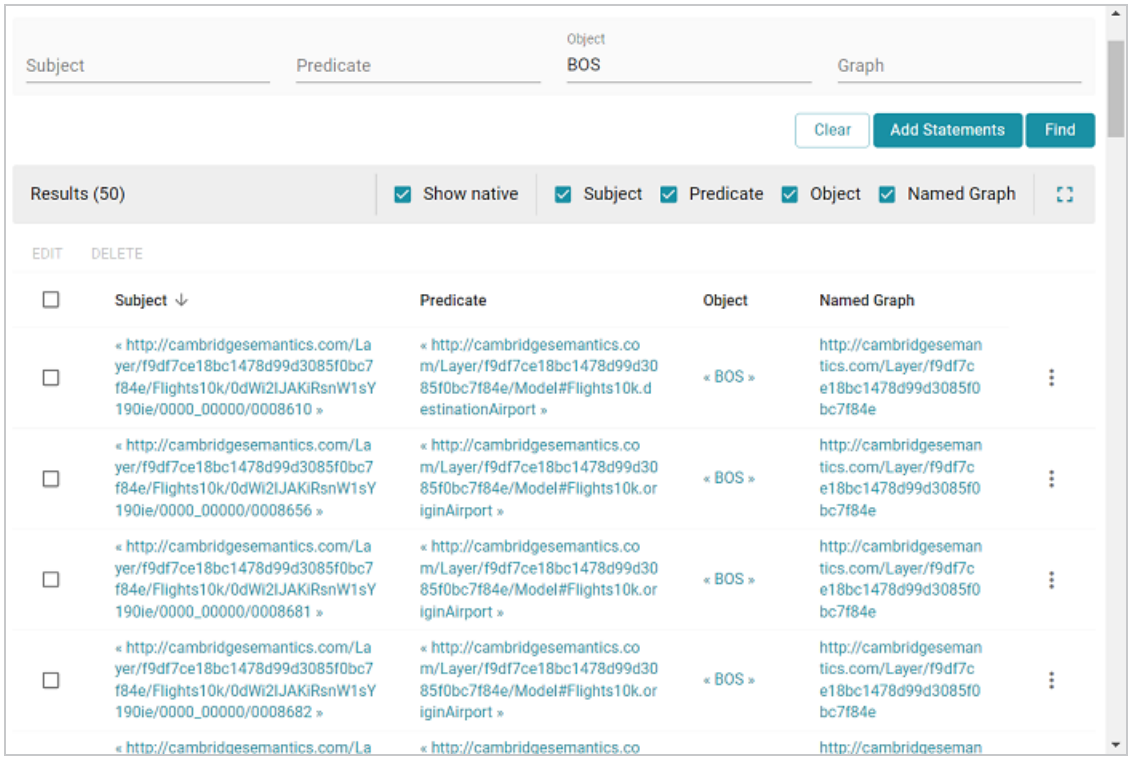


2. Click the **Find** tab.

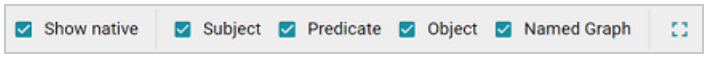


3. Click the **Select a Datasource** drop-down list and select the data source that you want to search.
   - Select **System Datasource** to search the local Anzo system volume.
   - Select the name of an AnzoGraph instance to search for data in graphmarts that are loaded to that instance.
   - Select **Data Profiling Metrics** to search the data profiling metrics volume.
   - Select **LDAP Primary Datasource** to search the directory server.
   - Select **System Tables** to search Anzo system table data.

4. Follow the guidelines below to specify the data to find in the data source:
   - Specify any subject, predicate, object, or graph name in the appropriate field. You can specify a value for one field in the quad or any combination of fields.
   - Any URIs and/or literal values that you specify must match the value in the data. Partial values, wildcard characters, and regular expressions are not supported.
   - If you want to get a list of all of the statements in the data source, you can leave all of the fields blank.
5. Click **Find** to search for the statements that match the search criteria. Anzo displays the matching statements. For example:



6. The following options are available for working with the results:
   - To filter results by showing or hiding parts of the quads in the statements, you can select or clear the following checkboxes above the results.



Clearing a checkbox hides that part of the quad in the result list. You can display the item again by selecting the checkbox.

- To modify the search parameters, you can click any of the graph, subject, predicate, or object values in the results. The search is automatically run again using only the value that you clicked.
- If the source that you searched is the **System Datasource**, you can edit, delete, or add statements directly. See System Datasource Options below for details.

System Datasource Options

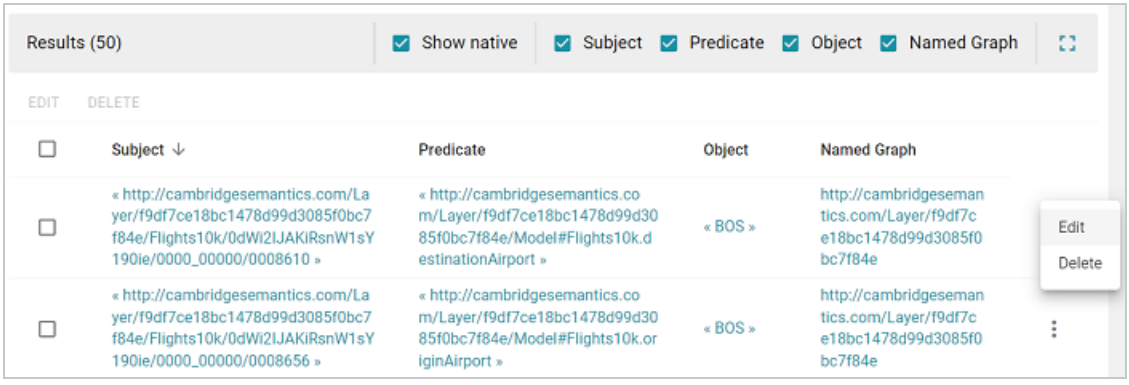
This section provides information about editing, deleting, and adding statements on the Find screen.

> **Note**
> Though the options described below are available for all data sources, adding, deleting, or editing statements is only successful when the data source is **System Datasource**.

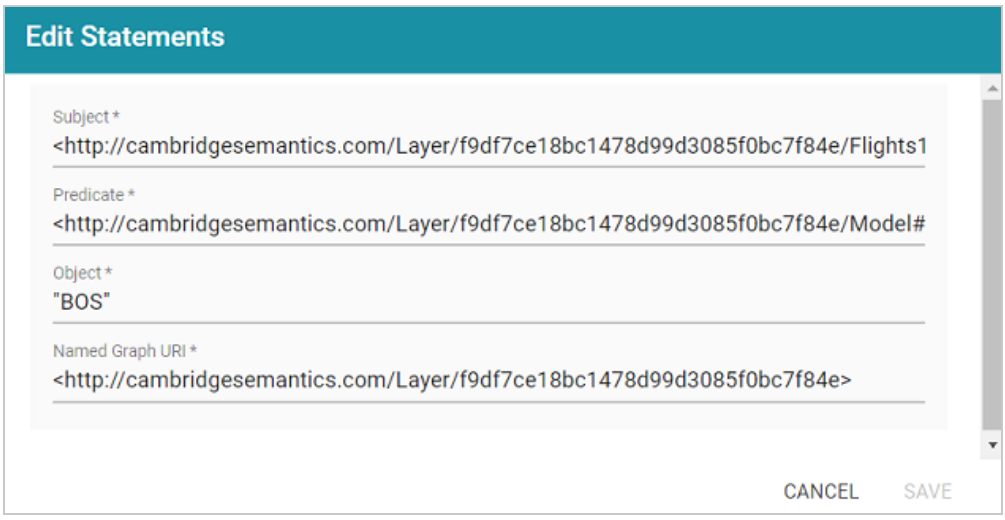
- Editing a Statement
- Deleting a Statement
- Adding a Statement

Editing a Statement

To edit a statement, click the menu icon (⋮) to the right of the statement and select **Edit**.



Anzo displays the Edit Statement dialog box. For example:



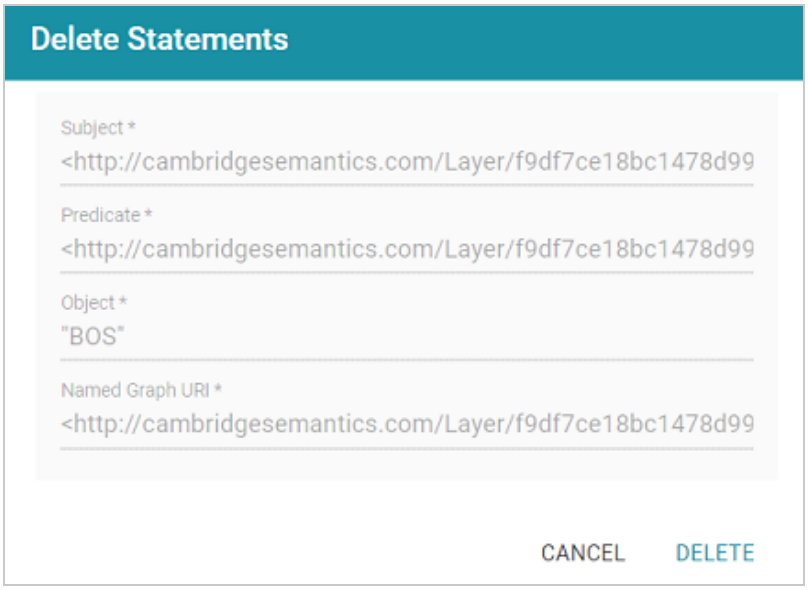
Change any of the quad values, and then click **Save**.

Important

If you edit URI values, make sure that the modified value is a valid URI.

Deleting a Statement

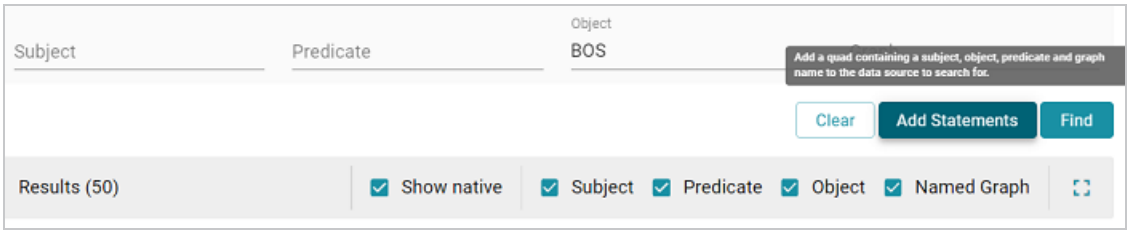
To delete a statement, click the menu icon (⋮) to the right of the statement and select **Delete**. Anzo displays the statement in a confirmation dialog box. For example:



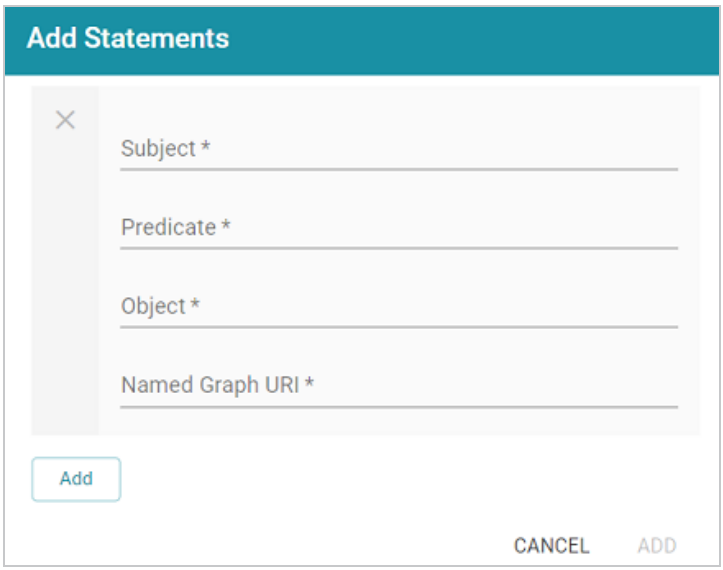
Click **Delete** to remove the statement from the system data source.

Adding a Statement

To add a quad to the data source, click the **Add Statements** button at the top of the result list.



Anzo displays the Add Statements dialog box.



Specify the new quad by adding the subject, predicate, object, and named graph URI in the appropriate fields. Each field is required. URIs must be valid, and the Named Graph URI that you specify must be present in the data source. You cannot add a new named graph. Click **Save** to add the new quad to the data source.

Access Data on Demand Endpoints

The topics in this section provide information about accessing Data on Demand endpoints and using the OData API as well as the Anzo ODBC and JDBC drivers. For information about creating endpoints, see Creating Data on Demand Endpoints.

In this section:

Accessing an Endpoint Programmatically 922

Accessing an Endpoint from an Application 926

OData Reference 944

Accessing an Endpoint Programmatically

This topic provides guidance on accessing Data on Demand endpoints programmatically by showing some example implementations using R and Python.

- Authentication and Data Access
- Access an Endpoint with R (RStudio)
- Access an Endpoint with Python (Linux Terminal)

Authentication and Data Access

Connections to Data on Demand endpoints must be authenticated. Users can submit their Anzo username and password when accessing data. Ultimately the data that is available to users from OData endpoints is subject to the security and composition of the graphmart as configured in Anzo.

Access an Endpoint with R (RStudio)

The following example shows how to connect to an OData endpoint from RStudio. The example uses the R programming language to access a Data on Demand endpoint and pull in data via a standard dataframe. New or existing R scripts can then be used with the data.

The first step in accessing data from RStudio is to prepare the R script that will construct the target URL and retrieve the resulting information via HTTP. The example script below accesses a pre-configured "Sample Data" endpoint. The script has sections for filtering the results as well as expanding the selection to include information from multiple classes:

```
require("httr")
require("jsonlite")
require("rstudioapi")

user   <- rstudioapi::showPrompt("Username", "Enter Anzo username", "sysadmin")
pw     <- rstudioapi::askForPassword(paste("Enter password for",user,sep=" "))

## Data on Demand endpoint
odata  <- "https://10.100.0.10/dataondemand/Sample-Graphmart/Sample-Data"

## Start from Probe class
```

```
startClass  <- "Probe?"

## Filter results for Homo sapiens species
filterKw   <- "$filter="
filterVal  <- "Species eq 'Hs'"
urlify     <- URLencode(filterVal)
filterStr  <- paste(filterKw,urlify,sep="")

## Select properties of interest (FeatureID) from base class
selectKw   <- "&$select="
selectVal  <- "FeatureID"
selectStr  <- paste(selectKw,selectVal,sep="")

## Select properties of interest (symbol) from Gene class
## via corresponds_to property on base Probe class
expandKw   <- "&$expand="
expandClass  <- "corresponds_to"
expandProps  <- "symbol"
expSelStr   <- "$select="
expandStr   <- paste(expandKw,expandClass,"(",expSelStr,expandProps,")",sep="")

## Specify format
format  <- "&$format=json"

## Generate OData URL using fragments above
url <- paste(odata,startClass,filterStr,selectStr,expandStr,format,sep="")

## Access OData endpoint
resultRaw  <- GET(url, (authenticate(user,pw, type = "basic")))
resultTxt  <- content(resultRaw, "text")
resultJson  <- fromJSON(resultTxt, flatten = TRUE)

print(url)

## Read results into dataframe
resultDataFrame  <- as.data.frame(resultJson)
View(resultDataFrame)
```

Executing the above R script from RStudio results in a dataframe that represents columns from the **Probe** and **Gene** classes.

Access an Endpoint with Python (Linux Terminal)

Many users have existing Python scripts to use with data in Anzo or a familiarity with Python that would make exploring, retrieving, and leveraging the data easier. The following example shows how to connect to an OData endpoint by executing a Python script from a Linux terminal.

The first step in accessing data using Python is to prepare the Python script that will construct the target URL and retrieve the resulting information via HTTP. The example script below accesses a pre-configured "Sample Data" endpoint. The script has sections for filtering the results as well as expanding the selection to include information from multiple classes (the same filter and class properties that were used in the R example above).

```
import requests
import getpass
from urllib.parse import urlparse

un = getpass.getpass(prompt='Username: ')
pw = getpass.getpass(prompt='Password: ')

## OData endpoint
# Data on Demand URL
odata = 'https://10.100.0.10/dataondemand/Sample-Graphmart/Sample-Data/'

## Start from Lease class
startClass = "Probe?"

## Filter results
filterKw  = "$filter="
filterVal = "Species eq 'Hs'"
urlify    = urlparse(filterVal)
filterStr = filterKw + urlify.geturl()

## Select properties of interest (start date, missed payments, lease status) from base
class
selectKw  = "&$select="
selectVal = "FeatureID"
selectStr = selectKw + selectVal

## Select properties of interest (name, social security number, credit score) from
Individual class
expandKw  = "&$expand="
```

```
expandClass = "corresponds_to"
expandProps = "symbol"
expSelStr   = "$select="
expandStr = expandKw + expandClass + "(" + expSelStr + expandProps + ")"

## Specify format
format = "&$format=text/csv"

## Generate OData URL using fragments above
url = odata + startClass + filterStr + selectStr + expandStr + format

## Access OData endpoint
r = requests.get(url, auth=(un, pw), verify=False)

print("URL")
print(url)
print("CONTENT")
print(r.content.decode('unicode_escape'))
print(type(r))
print(type(r.content))
```

In this example, the output is returned in CSV format (rather than JSON like the R example).

Accessing an Endpoint from an Application

Since Anzo's Data on Demand service conforms to the OData standard, any tool that supports the OData V4 REST API can access a Data on Demand endpoint to leverage data in Anzo. In addition, applications that support ODBC or JDBC APIs can use the Anzo CData ODBC or JDBC drivers to interact with Data on Demand endpoints. This capability enables users to leverage the benefits of Anzo's semantic layer, data model, and data blending capabilities in their favorite analytics tools.

This topic provides information about accessing Data on Demand endpoints from third-party applications.

- JDBC Driver Considerations
- Authentication and Data Access
- Accessing Data via the OData API
- Downloading the Anzo ODBC and JDBC Drivers
- JDBC Driver Documentation

JDBC Driver Considerations

This section describes important items to consider when using JDBC clients for accessing Data on Demand endpoints:

- Join Performance
- Querying Multi-Valued Properties
- Working with Long Column Names

Join Performance

To join results from multiple classes, Cambridge Semantics strongly recommends using OData or SPARQL. Hi-Res Analytics and SPARQL are designed to quickly return large results from multiple classes and should be strongly considered for these use cases. Joins on large data sets are also well-supported with OData when best practices around paging are applied.

You can also join tables upstream in AnzoGraph by creating data layers. For example, you can create a view that joins the data using a CONSTRUCT query. The view becomes available as an OData table. For information about view steps, see Construct a View of the Data (View Step).

In addition, Custom Data on Demand endpoints (sometimes called Table endpoints) enable you to join classes, add filters, and apply functions to properties during endpoint creation. The tables that you create are automatically translated to SPARQL queries that create views in AnzoGraph, allowing you to perform complex analytics on the graph yet generate results in the tabular format that BI tools expect.

Because the JDBC driver generates multiple OData queries and joins the results in memory, SQL queries that include JOINs on large data sets may take a very long time to complete. When using the JDBC driver, Cambridge Semantics recommends that you query one class at a time and then use the BI tool to do analytics on the returned data. For more information, see JDBC Performance Details below.

Querying Multi-Valued Properties

Some applications do not directly support Anzo's RDF graph data structures. For example, sometimes the JDBC driver presents multi-valued properties as arrays, which are not supported by some BI tools. When creating a Data on Demand endpoint for a graphmart that includes multi-valued properties, consider denormalizing the results to expand the properties into new rows so that they can be viewed in BI tools. For more information, see Creating Data on Demand Endpoints.

Working with Long Column Names

By default, the JDBC driver creates column names based on the property labels in the data model. The property labels can be too long for some clients. For example, Informatica is limited to 128 characters. When ingesting data from a tabular source, the label is a concatenation of the table and column name. Users may need to shorten the property labels to work with JDBC clients. If the label is missing, Anzo uses the localName of the IRI. For information about configuring the column names to be used for a Data on Demand endpoint, see Creating Data on Demand Endpoints.

Authentication and Data Access

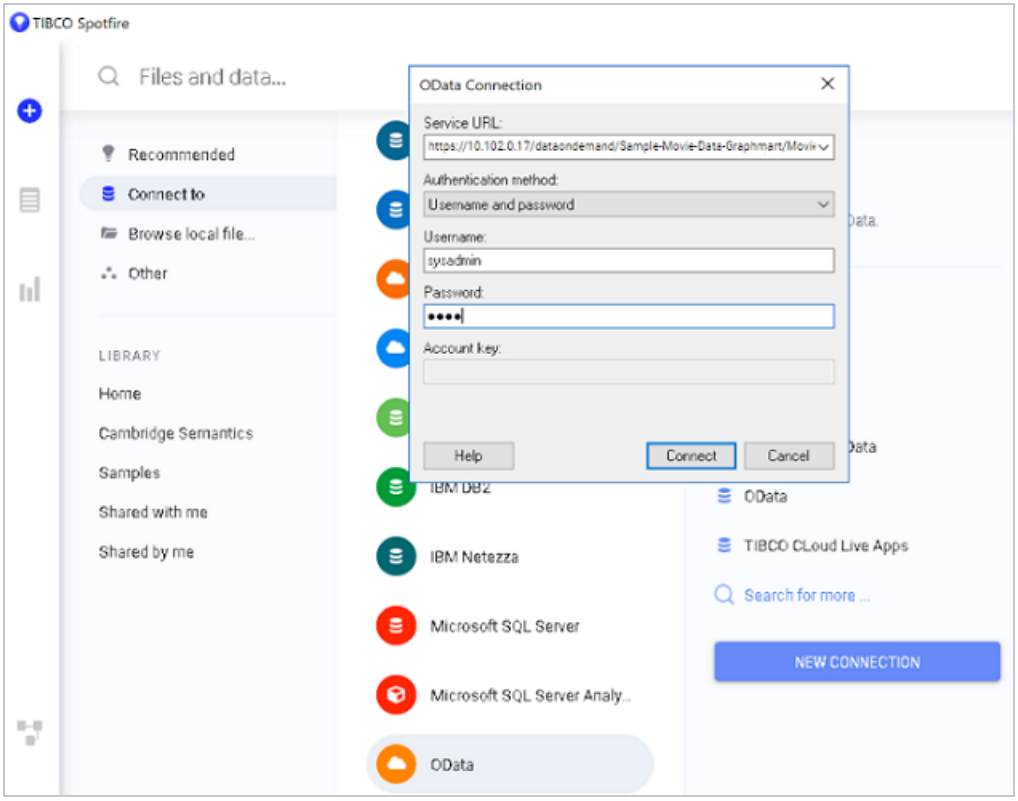
Connections to Data on Demand endpoints must be authenticated. Users can submit their Anzo username and password when accessing data. If your applications use single sign-on (SSO) authentication, you can also use SSO with Anzo. When using SSO, the client authenticates the user against the SSO provider and then passes the credentials to Anzo. All data is secured according to the user's SSO profile. For information about the supported SSO providers and instructions on configuring SSO access, see Connecting to an SSO Provider in the Administration Guide.

> **Note**
> Ultimately the data that is available to users from Data on Demand endpoints is subject to the access control configuration of the graphmart in Anzo.

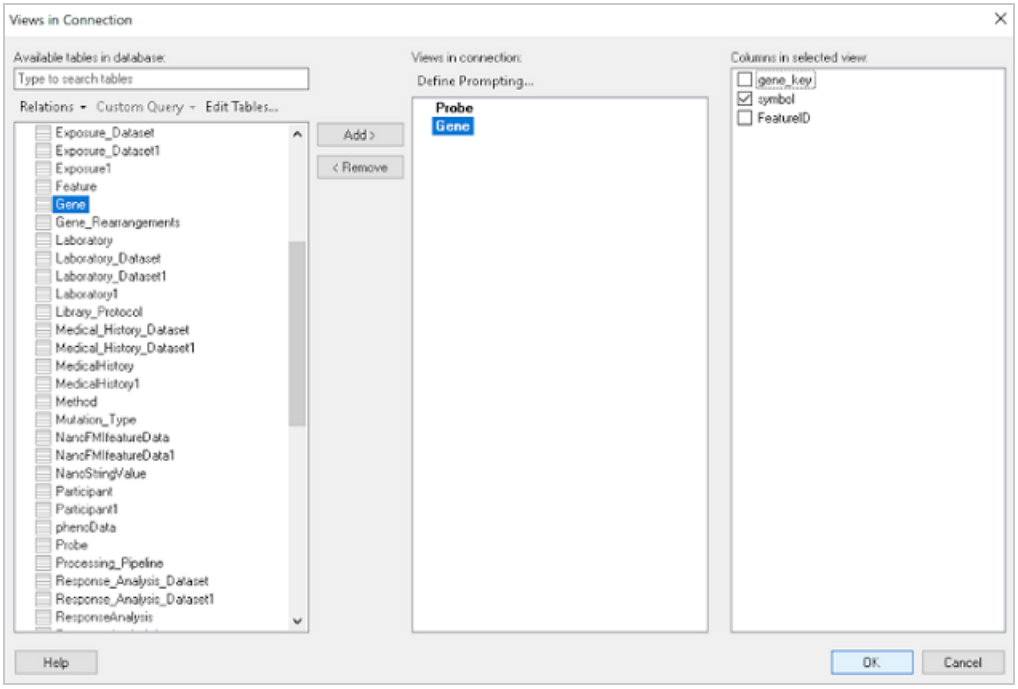
Accessing Data via the OData API

This section provides guidance on accessing a Data on Demand endpoint from an application that supports the OData REST API. It includes an example that configures an OData connection in TIBCO Spotfire. The example steps can also be applied to OData connections in other similar business intelligence tools.

The first step is to connect to the OData endpoint using the Spotfire Data sources user interface. When setting up the OData connection, the Service URL is the OData/ODBC URL from the Data on Demand endpoint configuration details in Anzo. The OData connection uses the user’s Anzo credentials for authentication.



Once the connection is established, Sportfire prompts the user to select the classes and properties to work with. In this example, the **FeatureID** property from the **Probe** class and the **symbol** property from the **Gene** class are selected:



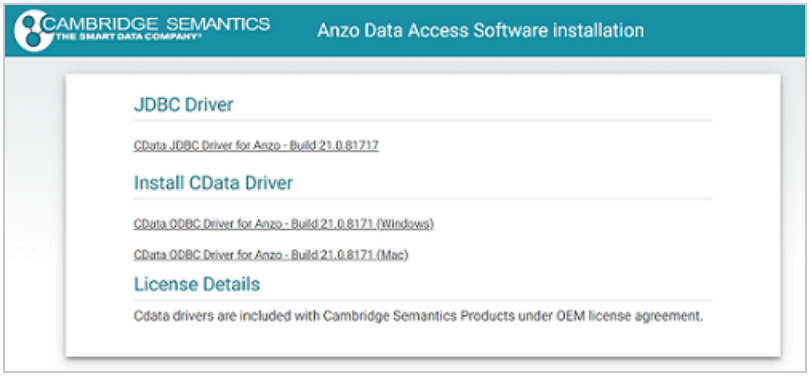
Once the properties are chosen, the data is loaded in Spotfire and can be used to inform existing analytics and data visualizations or create new ones.

Downloading the Anzo ODBC and JDBC Drivers

This section provides guidance on accessing Data on Demand endpoints from applications that support ODBC or JDBC APIs. Your Anzo deployment includes CData ODBC and JDBC drivers to use with applications. The first step is to retrieve the appropriate driver for your client. To download a driver, open a web browser and go to the following URL:

```
https://<Anzo_server>/installs/anzodataaccess
```

Where <Anzo_server> is the Anzo server DNS name or IP address. The Anzo Data Access Software Installation page provides links to download each driver. For example:



Download the appropriate driver to the client server:

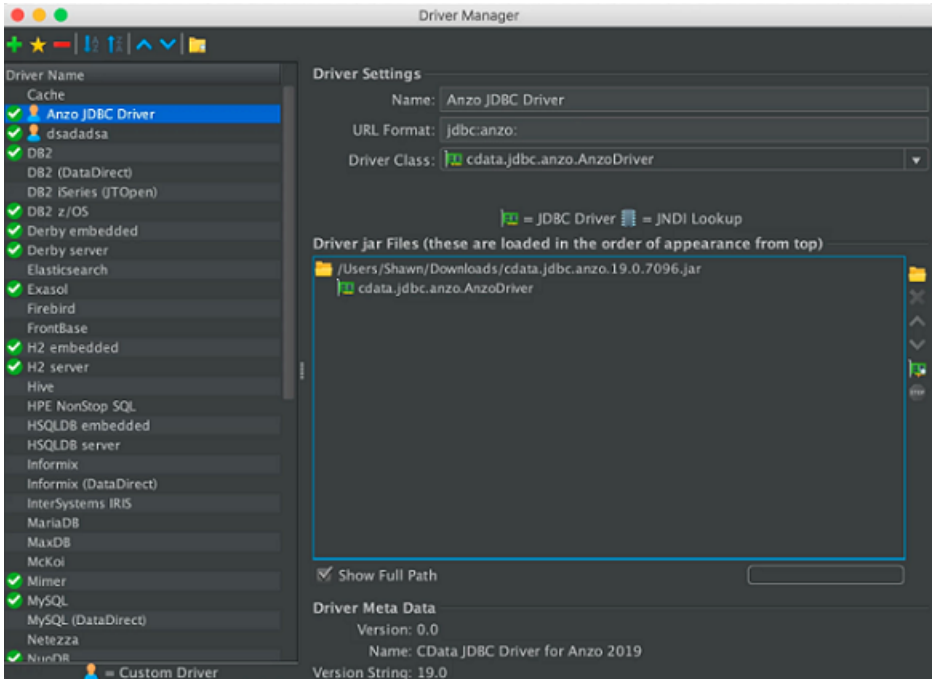
- The **CData JDBC Driver for Anzo** is used to connect to Anzo from most Java applications and database management tools.
- The **CData ODBC Driver for Anzo** for Windows or Mac is for use with applications and database management tools that support open database connectivity, such as Microsoft Excel or Tableau.

Configuring the Driver and Connecting to the Endpoint

This section provides guidance configuring an ODBC or JDBC driver by showing examples of configuring DbVisualizer and Tableau to access a Data on Demand endpoint using Anzo's JDBC driver and configuring Power BI to access an endpoint using the ODBC driver.

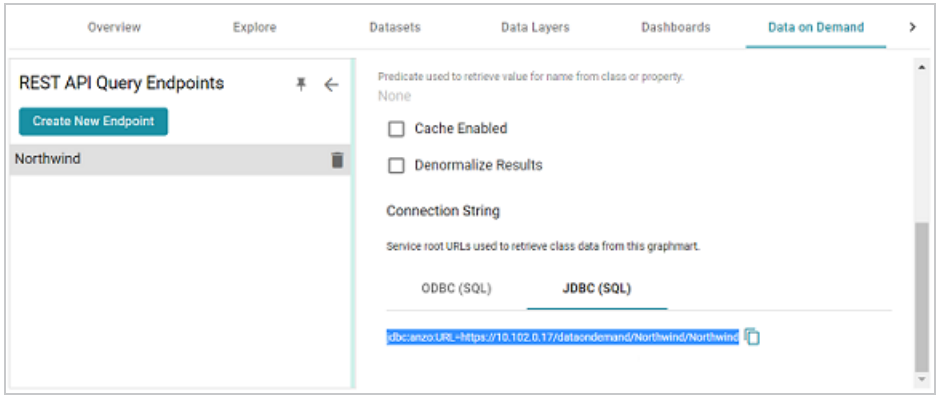
Example JDBC Setup with DbVisualizer

1. In DbVisualizer, go to **Tools** → **Driver Manager**.
2. In the Driver Manager, click the green plus icon to create a new driver.
3. Specify a name for the driver. For example, **Anzo JDBC Driver**.
4. In the URL Format field, specify the format **jdbc:anzo**.
5. In the **Driver File Paths** or **Driver jar Files** section of the screen, click the folder icon and then browse to and select the directory where you saved the CData JDBC Driver for Anzo cdata.jdbc.anzo.jar file that you downloaded to the server. DbVisualizer reads the jar and sets the Driver Class to **cdata.jdbc.anzo.AnzoDriver**. For example:

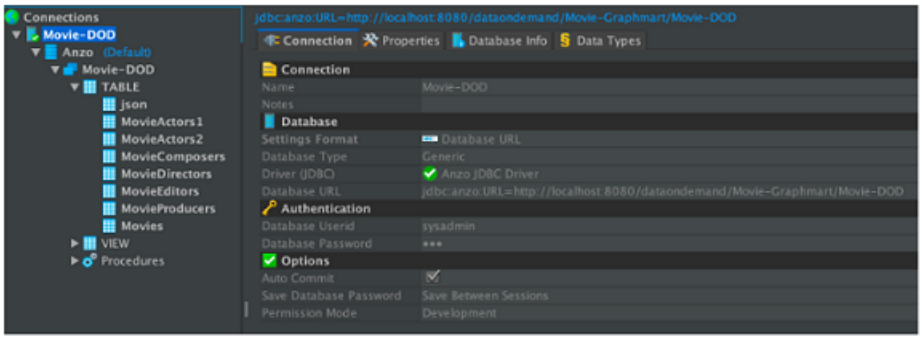


6. To connect to the endpoint in DbVisualizer, go to **Database** → **Create Database Connection**. Click **No Wizard** when prompted.
7. Specify a name for the connection in the **Name** field.
8. In the **Driver (JDBC)** field, select the Anzo JDBC driver connection.
9. In the **Database URL** field, specify the JDBC URL from the Anzo Data on Demand endpoint configuration. For example:

```
jdbc:anzo:URL=https://10.10.0.11/dataondemand/Northwind/Northwind
```



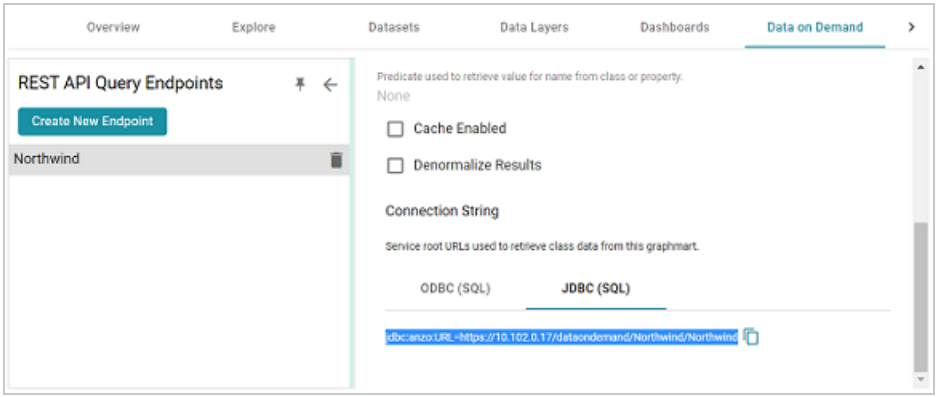
10. Under **Authentication**, enter your Anzo user ID and password. You should now be able to connect to the endpoint and view the available schemas. For example:



Example JDBC Setup with Tableau

1. After downloading the CData JDBC Driver for Anzo cdata.jdbc.anzo.jar file, place the .jar in the appropriate directory depending on your operating system:
   - **Windows**: C:\Program Files\Tableau\Drivers
   - **MacOS**: ~/Library/Tableau/Drivers
   - **Linux**: /var/opt/tableau/tableau_server/data/tabsvc/vizqlserver/Datasources/
2. Restart Tableau and then go to **Add a Connection → To a Server**.
3. Click **Other Databases (JDBC)**.
4. In the **Database URL** field, specify the JDBC URL from the Anzo Data on Demand endpoint configuration. For example:

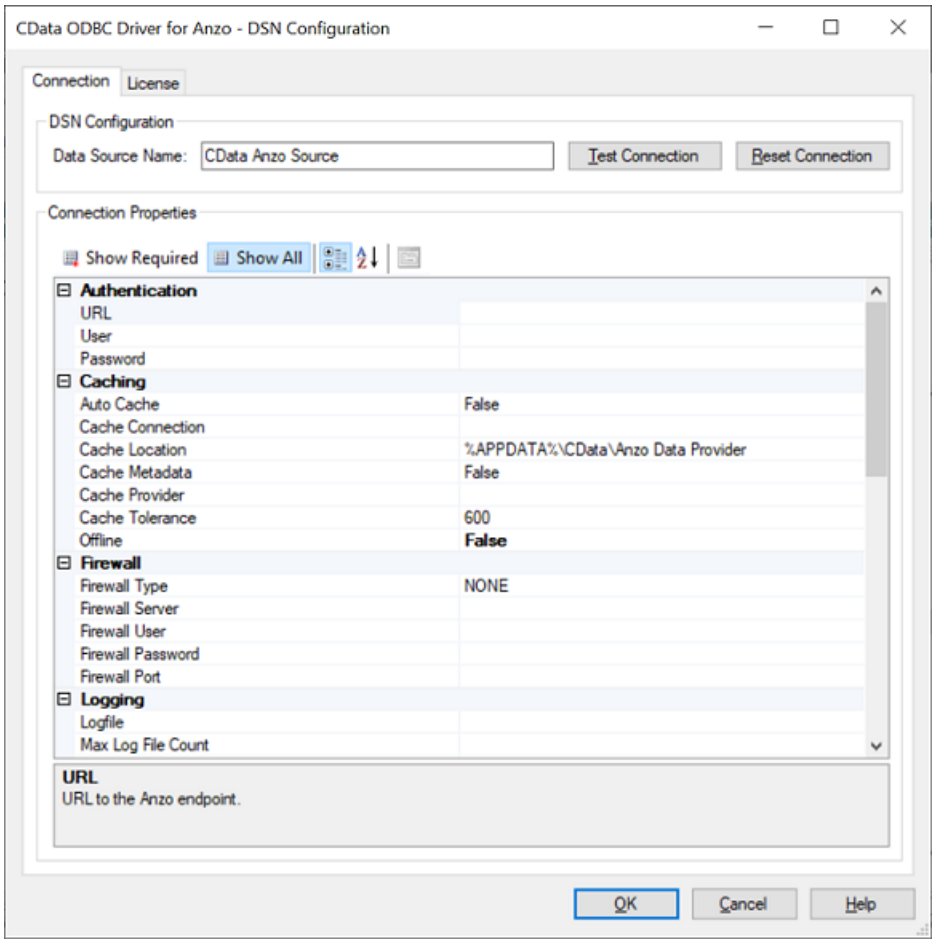
```
jdbc:anzo:URL=https://10.10.0.11/dataondemand/Northwind/Northwind
```



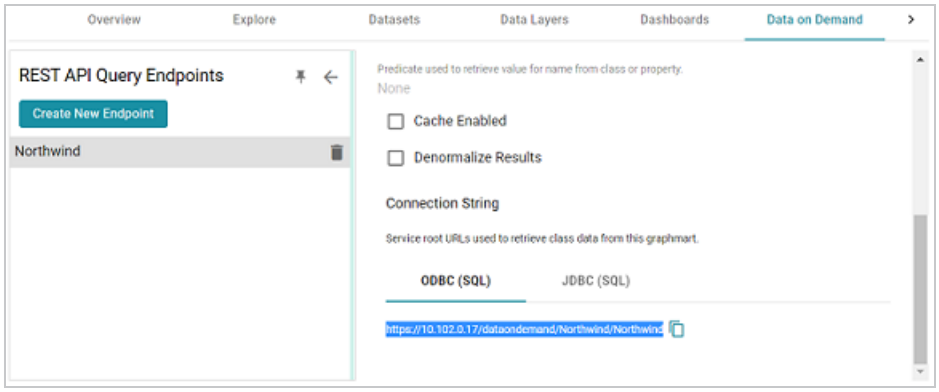
5. Enter your Anzo username and password and click **Sign In**. You should now be able to connect to the endpoint and view the available schemas.

Example ODBC Setup with Microsoft Power BI

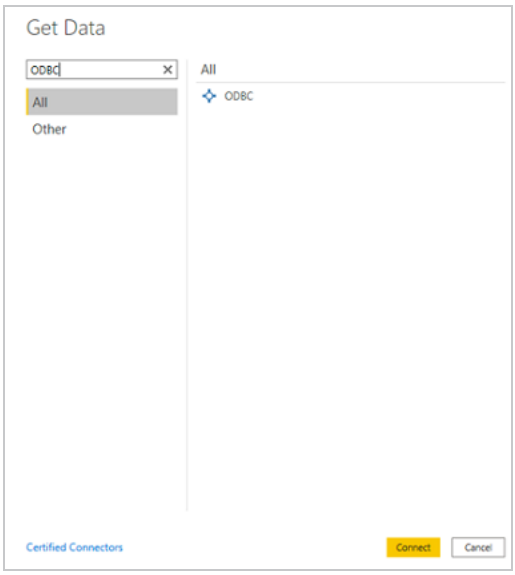
1. After downloading the Windows CData ODBC Driver for Anzo executable file, run the executable to start the installation wizard. The wizard guides you through installing the driver.
2. At the end of the installation, make sure the **Configure ODBC Data Source** checkbox is selected and click **Finish**. The wizard opens the driver's DNS Configuration screen. For example:



3. Under **Authentication** in Connection Properties, specify the **URL**, **User**, and **Password** to use for connecting to the Data on Demand endpoint. The User and Password are the Anzo username and password to use for authentication, and URL is the OData/ODBC service root URL for the endpoint. You can retrieve the URL from the Data on Demand screen for the endpoint. For example:



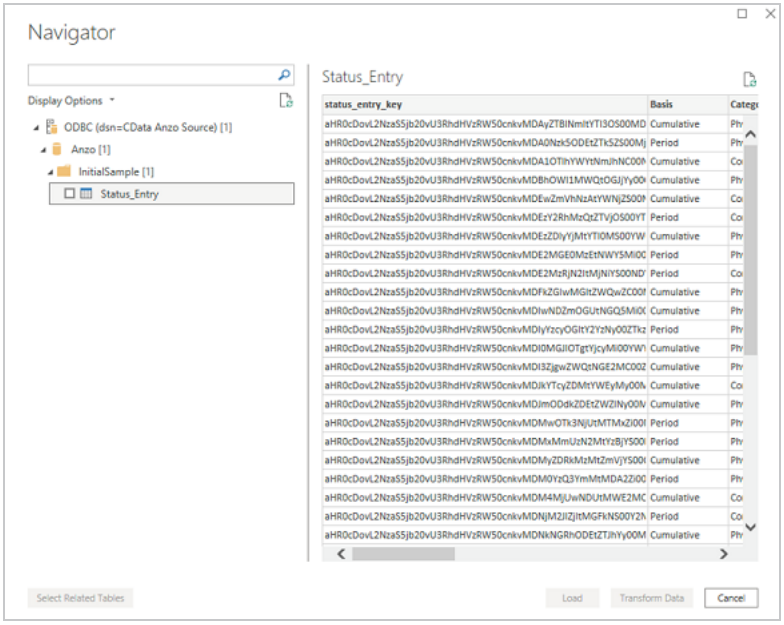
4. Click **OK** to save the configuration changes and close the dialog box.
5. Next, connect to the ODBC data source from Power BI. Open Power BI and click the **Get Data** button in the tool bar. In the Get Data dialog box, search for "ODBC." For example:



6. The search opens the wizard for creating an ODBC connection to a specified data source. Select **CData Anzo Source** from the drop-down list. You do not need to configure the advanced options.



7. Click **OK** to create the connection. Power BI opens the Navigator screen. For example:



Under Display Options, the top level container in the view represents the ODBC driver, the **Anzo** item represents the database, and the **InitialSample** item represents the schema. Each table is represented as a table entry under the schema. In the example above there is one table. If you select a table, sample data from that table is displayed on the right side of the screen. To load table(s), select the checkbox for each table and click the **Load** button. You can also use the advanced features of Power BI to transform the data as you load it into the tool.

JDBC Driver Documentation

This section provides a quick reference for JDBC driver support.

- For the complete JDBC driver documentation, see CData JDBC Driver for Anzo.
- For the complete ODBC driver documentation, see CData ODBC Driver for Anzo.

SQL Compliance

The JDBC driver supports most of the standard operations for querying data. The exceptions are listed below.

- The driver does not currently support transactions.
- The driver does not support batching of SQL statements.
- The driver has support for inserting, updating, and deleting records. However, performing updates via the driver can have unexpected consequences.

For more information about SQL compliance, see the SQL Compliance section in the CData JDBC Driver documentation.

JDBC Performance Details

By default, the JDBC driver offloads to Anzo as much of the SELECT statement processing as possible and then processes the rest of the query locally in memory.

- For joins, the driver generates multiple OData queries and joins the results in memory. As a result, SQL queries that include JOINs can take up to several minutes to complete.
- For aggregates, the driver retrieves all rows necessary to process in memory.
- For predicates, the driver determines which clauses Anzo supports and sends them to Anzo to retrieve the smallest possible superset of rows that would satisfy the query. It then filters the rest of the rows client-side.
- The driver's **SupportEnhancedSQL** setting can be disabled to limit SQL execution to only what the Anzo API supports. For more information, see the Support Enhanced SQL section in the CData JDBC Driver documentation.

> **Tip**
> To determine which query capabilities the driver can offload to the Anzo API, you can query the **sys_sqlinfo** system table. The table contains information about the functionality that is supported by the connected source. For example:
>
> ```
> SELECT * FROM sys_sqlinfo WHERE name='AGGREGATE_FUNCTIONS'
> or name = 'COUNT' or name = 'SUPPORTED_OPERATORS' or name = 'GROUP_BY'
> or name = 'OUTER_JOINS' or name = 'OJ_CAPABILITIES' or name = 'SUBQUERIES'
> or name = 'STRING_FUNCTIONS' or name = 'NUMERIC_FUNCTIONS'
> or name = 'TIMEDATE_FUNCTIONS';
> ```

For more information, see the sys_sqlinfo section in the CData JDBC Driver documentation.

Data Caching

Due to the client-side in-memory processing of aggregates and joins, the performance of queries against extremely large data sets may suffer. If this is a common use case, consider leveraging caching in the JDBC driver. If the driver maintains a local copy of the data, it reduces the number of API calls and can increase performance for long-running queries. For more information, see the Caching Data section in the CData JDBC Driver documentation.

Supported SELECT Statement Clauses

The following list shows the supported SELECT statement clauses. For more information, see the SELECT Statement section in the CData JDBC Driver documentation.

- SELECT
- INTO
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY
- LIMIT

Supported Aggregate Functions

The following list shows the supported aggregate functions. For more information, see the Aggregate Functions section in the CData JDBC Driver documentation.

- COUNT
- COUNT_DISTINCT
- AVG
- MIN
- MAX
- SUM

Supported Joins

The following list shows the supported JOIN types. For more information, see the JOIN Queries section in the CData JDBC Driver documentation.

- **Inner Join**: Selects only the rows from both tables that match the join condition.
- **Left Join**: Selects all of the rows in the FROM table and only matching rows in the JOIN table.

SQL Function Reference

The JDBC driver provides implementations of the following common SQL functions. For more information, see the SQL Functions section in the CData JDBC Driver documentation.

> **Note**
> The driver interprets all function input as either column names or strings. Therefore, all string literals must be escaped with single quotes. For example, `SELECT DATENAME ('yy',GETDATE()))`.

String Functions

- ASCII(character_expression)
- CHAR(integer_expression)
- CHARINDEX(expressionToFind ,expressionToSearch [,start_location])
- CONCAT(string_value1, string_value2 [, string_valueN])
- CONTAINS(expressionToSearch, expressionToFind)

- ENDSWITH(character_expression, character_suffix)
- FORMAT(value, format)
- FROM_UNIXTIME(time, format, issecond)
- INDEXOF(expressionToSearch, expressionToFind [,start_location])
- ISNULL(check_expression , replacement_value)
- JSON_AVG(json, jsonpath)
- JSON_COUNT(json, jsonpath)
- JSON_EXTRACT(json, jsonpath)
- JSON_MAX(json, jsonpath)
- JSON_MIN(json, jsonpath)
- JSON_SUM(json, jsonpath)
- LEFT(character_expression , integer_expression)
- LEN(string_expression)
- LOWER(character_expression)
- LTRIM(character_expression)
- NCHAR(integer_expression)
- PATINDEX(pattern, expression)
- QUOTENAME(character_string [, quote_character])
- REPLACE(string_expression, string_pattern, string_replacement)
- REPLICATE(string_expression ,integer_expression)
- REVERSE(string_expression)
- RIGHT(character_expression , integer_expression)
- RTRIM(character_expression)

- SOUNDEX(character_expression)
- SPACE(repeatcount)
- STARTSWITH(character_expression, character_prefix)
- STR(float_expression [, integer_length [, integer_decimal]])
- STUFF(character_expression , integer_start , integer_length , replaceWith_expression)
- SUBSTRING(expression,integer_start,integer_length)
- TOSTRING(string_value1)
- TRIM(character_expression)
- UNICODE(ncharacter_expression)
- UPPER(character_expression)
- XML_EXTRACT(xml, xpath [, separator])

Date Functions

- CURRENT_DATE()
- CURRENT_TIMESTAMP()
- DATEADD(datepart , integer_number , date [, dateformat])
- DATEDIFF(datepart , startdate , enddate)
- DATEFROMPARTS(integer_year, integer_month, integer_day)
- DATENAME(datepart , date)
- DATEPART(datepart, date [,integer_datefirst])
- DATETIME2FROMPARTS(integer_year, integer_month, integer_day, integer_hour, integer_minute, integer_seconds, integer_fractions, integer_precision)
- DATETIMEFROMPARTS(integer_year, integer_month, integer_day, integer_hour, integer_minute, integer_seconds, integer_milliseconds)
- EOMONTH(start_date [, integer_month_to_add])

- GETDATE()
- GETUTCDATE()
- ISDATE(date, [date_format])
- SMALLDATETIMEFROMPARTS(integer_year, integer_month, integer_day, integer_hour, integer_minute)
- SYSDATETIME()
- SYSUTCDATETIME()
- TIMEFROMPARTS(integer_hour, integer_minute, integer_seconds, integer_fractions, integer_precision)
- YEAR(date)

Math Functions

- ABS(numeric_expression)
- ACOS(float_expression)
- ASIN(float_expression)
- ATAN(float_expression)
- ATN2(float_expression1 , float_expression2)
- CEILING(numeric_expression)
- COS(float_expression)
- COT(float_expression)
- DEGREES(numeric_expression)
- EXP(float_expression)
- EXPR(expression)
- FLOOR(numeric_expression)
- LOG(float_expression [, base])

- LOG10(float_expression)
- PI( )
- POWER(float_expression , y)
- RADIANS(float_expression)
- RAND([integer_seed])
- ROUND(numeric_expression , integer_length [,function])
- SIGN(numeric_expression)
- SIN(float_expression)
- SQRT(float_expression)
- SQUARE(float_expression)
- TAN(float_expression)

OData Reference

The Anzo Data on Demand service follows the OData Version 4.0 specification, which defines the standard URL conventions and query options. This topic provides a quick reference for learning OData basics and viewing the supported string operators and output formats. It also provides some example queries.

- OData URL Conventions
- Supported Query Operators
- Example OData Requests

OData URL Conventions

An OData service URL has three main parts:

1. The **Service Root URL** that Anzo provides. The service root URL is the metadata that describes all of the available feeds (tables).
2. The optional **Resource Path** that narrows the scope of the available data to the individual table (class) level, property level, or the schema.
3. The **Query Options** for analyzing the data.

For example, the following OData URL shows the service root from the Data on Demand screen in Anzo, a resource path that narrows the scope of the data to the Employees table (class), and query options that filter the result set to show data for the NA region only:

```
https://10.100.0.10/dataondemand/Northwind-Graphmart/Northwind/Employees?$filter=contains(Region, 'NA')
\_____________________________________________________________/ \_______/  \_____________________________/
                       Service Root URL                        Resource Path       Query Options
```

OData requests need to be URL-encoded. Typically you can configure programs to encode requests automatically. And browsers encode URLs that are pasted into the address bar.

Supported Query Operators

OData query options are used to dynamically query data via the endpoint and control the amount and order of the data returned. The Data on Demand service supports the following OData query operators. See Example OData Requests below for example queries that employ the operators.

Operator	Description
$count	Used to count the number of matching resources in the result set.
$expand	Used to retrieve related data and include it in the results. When you query data via OData, the default response does not include related entities. The $expand option provides flexibility for exploring data across the data model. It allows the related information to be embedded in the response.
$filter	Used to filter a result set. The expression specified with $filter is evaluated for each resource identified by the resource path, and only items where the expression evaluates to true are included in the response.
$format	Used to specify the output format for the results. The supported formats are text/CSV, JSON, and XML. For example: $format=json
$metadata	Used to return the schema, entity set, and property metadata.
$orderby	Used to return results in ascending (asc) or descending (desc) order. If asc or desc is not specified, solutions are returned in ascending order.
$select	Used to specify the subset of properties to include in the result set.
$skip	Used to specify the number of solutions to exclude in the results. The $top and $skip OData query options are similar to the LIMIT and OFFSET clauses in SPARQL queries.
$top	Used to limit the number of solutions that are returned.

Example OData Requests

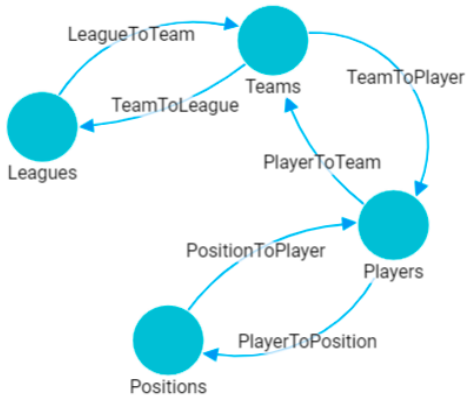
This section demonstrates the use of OData query operators by providing examples of common types of OData requests.

The examples below are run against a sample graphmart, called **LeagueGM**, that contains data about the teams and players in a small local baseball league. The Data on Demand endpoint is named **LeagueData**. The following service root URL was created by Anzo:

```
https://10.100.0.10/dataondemand/LeagueGM/LeagueData
```

For readability, the examples below abbreviate "https://10.100.0.10/dataondemand" to **dataondemand**. In addition, the examples are not URL-encoded.

The data has Leagues, Teams, Players, and Positions classes (or entities in OData). And the image below shows a graph view of the data model. To view the TriG version of the model, click here.



To view the instance data for each class, you can click a link below to view the data for that class. The data is in JSON format.

Leagues Teams Players Positions

Retrieving Metadata

The request below retrieves the schema, entity set, and property metadata for the endpoint.

```
dataondemand/LeagueGM/LeagueData/$metadata
```

The results are in XML format. A snippet of the results is shown below. To view the complete response, click here.

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Feeds">
      <EntityContainer Name="Default">
        <EntitySet Name="Leagues"
EntityType="com.cambridgesemantics.ont.autogen.LeagueDict.LeagueData.Leagues">
          <NavigationPropertyBinding Path="LeagueToTeam" Target="Teams"/>
        </EntitySet>
        <EntitySet Name="Teams"
EntityType="com.cambridgesemantics.ont.autogen.LeagueDict.LeagueData.Teams">
          <NavigationPropertyBinding Path="TeamToLeague" Target="Leagues"/>
          <NavigationPropertyBinding Path="TeamToPlayer" Target="Players"/>
        </EntitySet>
        <EntitySet Name="Positions"
EntityType="com.cambridgesemantics.ont.autogen.LeagueDict.LeagueData.Positions">
          <NavigationPropertyBinding Path="PositionToPlayer" Target="Players"/>
        </EntitySet>
        <EntitySet Name="Players"
EntityType="com.cambridgesemantics.ont.autogen.LeagueDict.LeagueData.Players">
          <NavigationPropertyBinding Path="PlayerToPosition" Target="Positions"/>
          <NavigationPropertyBinding Path="PlayerToTeam" Target="Teams"/>
        </EntitySet>
      </EntityContainer>
    </Schema>
...
```

Counting an Entity

The request below returns the number of teams in the graphmart. Adding the resource path **Teams** to the request narrows the scope to the Teams entity (or class in Anzo).

```
dataondemand/LeagueGM/LeagueData/Teams/$count
```

Result

```
4
```

This request returns the number of players:

```
dataondemand/LeagueGM/LeagueData/Players/$count
```

Result

```
12
```

Counting a Property of an Entity

The request below counts the number of players on the Al Thomas team. The request uses the team_key to identify the team and the TeamToPlayer to identify each player.

```
dataondemand/LeagueGM/LeagueData/Teams
('aHR0cDovL2NzaS5jb20vVGVhbXMvMQ')/TeamToPlayer/$count
```

Result

```
3
```

This request counts the number of positions played by James Smith:

```
dataondemand/LeagueGM/LeagueData/Players
('aHR0cDovL2NzaS5jb20vUGxheWVycy8y')/PlayerToPosition/$count
```

Result

```
2
```

Filtering Data via Text Search

The request below filters the results to show data for the TeamName that equals "Black Sox." The request also returns results in JSON format:

```
dataondemand/LeagueGM/LeagueData/Teams?$filter=TeamName eq 'Black Sox'&$format=json
```

Result

```
{
  "@odata.context":
"https://10.100.0.10/dataondemand/LeagueGM/LeagueData/$metadata#Teams",
  "value": [
    {
      "teams_key": "aHR0cDovL2NzaS5jb20vVGVhbXMvMg",
      "TeamId": 2,
      "teamtoleague_key": [
          "aHR0cDovL2NzaS5jb20vTGVhZ3Vlcy8x"
      ],
```

```
      "TeamName": "Black Sox",
      "teamtoplayer_key": [
          "aHR0cDovL2NzaS5jb20vUGxheWVycy80",
          "aHR0cDovL2NzaS5jb20vUGxheWVycy81",
          "aHR0cDovL2NzaS5jb20vUGxheWVycy82"
      ]
    }
  ]
}
```

This request filters the data to find the players whose name contains "Ted."

```
dataondemand/LeagueGM/LeagueData/Players?$filter=contains(PlayerName,'Ted')
```

The request can also use "startswith" in place of contains to filter specifically for player names that start with "Ted."

```
dataondemand/LeagueGM/LeagueData/Players?$filter=startswith(PlayerName,'Ted')
```

Result

```
{
  "@odata.context":
"https://10.100.0.10/dataondemand/LeagueGM/LeagueData/$metadata#Players",
  "value": [
    {
      "players_key": "aHR0cDovL2NzaS5jb20vUGxheWVycy8xMA",
      "playertoposition_key": [
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzM",
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzI"
      ],
      "PlayerId": 10,
      "playertoteam_key": [
          "aHR0cDovL2NzaS5jb20vVGVhbXMvNA"
      ],
      "PlayerName": "Ted James",
      "DefensiveRating": 92.55
    },
    {
      "players_key": "aHR0cDovL2NzaS5jb20vUGxheWVycy84",
      "playertoposition_key": [
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzI",
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzEw"
```

```
      ],
      "PlayerId": 8,
      "playertoteam_key": [
          "aHR0cDovL2NzaS5jb20vVGVhbXMvMw"
      ],
      "PlayerName": "Ted Sale",
      "DefensiveRating": 77.33
    }
  ]
}
```

Selecting Properties and Ordering Results

The request below selects player names and their defensive ratings. The results are ordered by defensive rating in descending order so that the player with the highest defensive rating is listed first. The request also formats the results in text/csv.

```
dataondemand/LeagueGM/LeagueData/Players?$select=PlayerName,DefensiveRating&$orderby=De
fensiveRating desc&$format=text/csv
```

Result

```
PlayerName,DefensiveRating
James Smith,98.33
Alex Granderson,98.22
Matt Butler,95.66
Tim Hooper,93.43
Steve Jones,93.28
Ted James,92.55
Fred Wynn,88.68
Jared Bonds,86.34
Billy Roper,83.44
Mike Magazine,78.33
Ted Sale,77.33
Chris Underwood,66.22
```

Expanding the Results to Include Related Entities

The request below uses the $expand operator to retrieve data from the Players entity and include the related Positions data for each player. For this example, the request limits the number of results returned to 2 players by adding $top=2:

```
dataondemand/LeagueGM/LeagueData/Players?$expand=PlayerToPosition&$top=2
```

Result


```
{
  "@odata.context":
"https://10.100.0.10/dataondemand/LeagueGM/LeagueData/$metadata#Players",
  "value": [
    {
      "players_key": "aHR0cDovL2NzaS5jb20vUGxheWVycy8x",
      "playertoposition_key": [
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzg"
      ],
      "PlayerId": 1,
      "playertoteam_key": [
          "aHR0cDovL2NzaS5jb20vVGVhbXMvMQ"
      ],
      "PlayerName": "Steve Jones",
      "DefensiveRating": 93.28,
      "PlayerToPosition": [
          {
            "positions_key": "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzg",
            "PositionId": 8,
            "ShortName": "CF",
            "positiontoplayer_key": [
                "aHR0cDovL2NzaS5jb20vUGxheWVycy8xMg",
                "aHR0cDovL2NzaS5jb20vUGxheWVycy8x"
            ],
            "Description": "Centerfield"
          }
      ]
    },
    {
      "players_key": "aHR0cDovL2NzaS5jb20vUGxheWVycy8xMA",
      "playertoposition_key": [
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzI",
          "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzM"
      ],
      "PlayerId": 10,
      "playertoteam_key": [
          "aHR0cDovL2NzaS5jb20vVGVhbXMvNA"
      ],
      "PlayerName": "Ted James",
      "DefensiveRating": 92.55,
      "PlayerToPosition": [
```

```
          {
            "positions_key": "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzI",
            "PositionId": 2,
            "ShortName": "C",
            "positiontoplayer_key": [
                "aHR0cDovL2NzaS5jb20vUGxheWVycy84",
                "aHR0cDovL2NzaS5jb20vUGxheWVycy8xMA"
            ],
            "Description": "Catcher"
          },
          {
            "positions_key": "aHR0cDovL2NzaS5jb20vUG9zaXRpb25zLzM",
            "PositionId": 3,
            "ShortName": "1B",
            "positiontoplayer_key": [
                "aHR0cDovL2NzaS5jb20vUGxheWVycy83",
                "aHR0cDovL2NzaS5jb20vUGxheWVycy8xMA"
            ],
            "Description": "First Base"
          }
        ]
      }
   ]
}
```

Access the SPARQL Endpoint

Anzo offers a standard HTTP(S) SPARQL endpoint for sending SPARQL requests between client applications and Anzo. The endpoint is enabled by default. This topic provides the base endpoint URL and describes the supported HTTP methods and parameters.

> **Tip**
> You can generate a cURL request against the SPARQL endpoint from a query in the Query Builder. Once you have a valid query, click the **More** button under the query and select **Copy CURL Command**. For more information about writing queries in the Query Builder, see Running SPARQL Queries in the Query Builder.

Authentication

The Anzo SPARQL endpoint supports Basic Authentication. The endpoint can be configured to enable other Anzo-supported authentication methods. However, implementing alternate authentication mechanisms can have unexpected results. For more information, contact Cambridge Semantics Support.

> **Note**
> Ultimately the data that is available to users from SPARQL endpoints depends on the access control configuration of the graphmart or linked data set as configured in Anzo.

HTTP Methods and Options

The Anzo SPARQL endpoint accepts HTTP GET and POST methods. GET is used to retrieve data from the endpoint, and POST is used to send data to the endpoint. Update queries must use the POST method, and read queries can be submitted using GET or POST.

Endpoint Base URL

Use the following base URL to access data in Anzo via the SPARQL endpoint. The table below describes each base URL component:

```
<protocol>://<hostname>:<port>/sparql/<store_type>/<url-encoded_dataset_uri>
```

Option	Description
protocol	The protocol to use for the connection: **http** for HTTP protocol or **https** for SSL protocol.
hostname	The DNS name or IP address of the Anzo server.
port	The port for the endpoint. The port that you specify depends on the protocol that you choose. By default, the HTTP port is **80** and the HTTPS port is **443**. To view the ports that are configured for your Anzo instance, see **Server Settings** in the **Administration** menu.
sparql	Required keyword for the SPARQL endpoint.
store_type	The type of RDF store for the data. Typically users specify **graphmart** to query data that is in a graphmart. It is also possible to query the metadata for a linked data set (LDS) in the Dataset catalog. To query an LDS that is stored in a local volume, specify **lds** as the store type.
url-encoded_dataset_uri	The URI for the graphmart or the catalog entry for the LDS. The URI must be URL-encoded using upper case hexadecimal digits. Lower case hexadecimal digits are not supported at this time. How do I find the URI for a graphmart? How do I find the catalog entry URI for a dataset?

For example, the following base endpoint URL targets the data in a graphmart:

```
https://10.100.10.20:8443/sparql/graphmart/http%3A%2F%2Fcambridgesemantics.com%2FGraphmart%2F1ad0ee911b834097ad7f71ee0ae1c0ff
```

The example below shows a base endpoint URL that targets a dataset catalog entry:

```
https://10.100.10.20:8443/sparql/lds/http%3A%2F%2Fopenanzo.org%2FcatEntry
(%255Bhttp%253A%252F%252Fcsi.com%252FFileBasedLinkedDataSet%252F001e517db4f0eaea9f27942
7e4e2a828%255D%2540%255Bhttp%253A%252F%252Fopenanzo.org%252Fdatasource%252FsystemDataso
urce%255D)
```

HTTP Header Options

The HTTP header provides information related to the transfer of data between the requesting client and the SPARQL endpoint. The table below describes the supported HTTP header options. Both of the fields are optional.

Option	Description
Content-Type	The Content-Type specifies the type of request that is being sent by the client. Anzo supports the following Content-Type values: • **application/x-www-form-urlencoded**: Including this value specifies that the query string will be passed as the value of a "query" or "update" HTTP parameter. This is the default value. When Content-Type is not specified, the endpoint behaves as if `Content-Type: application/x-www-form-urlencoded` is specified. • **application/sparql-query**: Including this value specifies that the HTTP request body includes a SPARQL read (non-update) query. • **application/sparql-update**: Including this value specifies that the HTTP request body includes a SPARQL update query.
Accept	The Accept field specifies the response formats that are acceptable for the server to send back to the client. You can use this field to specify the output serialization format for query results in place of the format HTTP parameter. For details about the supported formats, see Format Options below.

HTTP Body Parameters

The HTTP parameters in the body of the request provide the rest of the information about the request. Certain parameters are appropriate for read-only queries, SELECT and CONSTRUCT, and others are appropriate for updates, INSERT and DELETE. The tables below describe the supported parameters for query and update requests.

Query Parameters

Parameter	Description
query	Specifies the full read-only query string to run. If you do not specify a url-encoded_dataset_uri, default-graph-uri or named-graph-uri in the request, the query string should contain the appropriate FROM clauses. To run an update query (INSERT or DELETE), use the update parameter.
default-graph-uri	Specifies a default graph URI to query. You can include this parameter multiple times in a request. When the base URL specifies a graphmart URI, you can specify a data layer URI to narrow the scope of the query to a specific data layer in the graphmart.
named-graph-uri	Specifies a named graph URI to query. You can include this parameter multiple times in a request. When the base URL specifies a graphmart URI, you can specify a data layer URI to narrow the scope of the query to a specific data layer in the graphmart.
format	Specifies the serialization format to use for the results of the query. For details about the supported formats, see Format Options below.

Parameter	Description
includeMetadataGraphs	A boolean value that specifies whether to query the metadata graphs. Only valid for queries that target a linked data set (LDS) that is stored in a local volume. The default value is **includeMetadataGraphs=false**.
delim	Specifies a custom delimiter character to use in CSV output results. Valid only for SELECT queries where the output format is **text/csv**. This field accepts any character. When delim is not specified the default value is a **,** (comma).
dedup	A boolean value that specifies whether to deduplicate CONSTRUCT results on the client side. When dedup is not specified, the default value is **dedup=true**.
serverDedup	A boolean value that specifies whether to deduplicate CONSTRUCT results on the server side. When serverDedup is not specified, the default value is **serverDedup=true**.
skipCache	A boolean value that specifies whether to skip the reuse of any query cache that exists from a previous run of the query. When skipCache is not specified, the default value is **skipCache=false**. Anzo server's query cache should be forcibly skipped/ignored
hasHeader	A boolean value that specifies whether to include headers in CSV results. Valid only for SELECT queries where the output format is **text/csv**. When hasHeader is not specified, the default value is **hasHeader=false**.
attachResult	A boolean value that specifies whether to provide the query response as a file "attachment," i.e. the HTTP response will include the Content-Disposition of **attachment**. When attachResult is not specified, the default value is **attachResult=false**. When returning

Parameter	Description
	results as an attachment, you can specify a file name in filename the parameter.
filename	If attachResult is true, this parameter specifies the file name to use for the attachment, excluding the file extension. If attachResult is true and filename is not specified, the default file name is **QueryResult**.

Format Options

The table below describes the options for specifying the serialization format of the results that the server sends back to the client. These format options, i.e., MIME types or file extensions, can be specified in the format parameter in the body of the request or in the Accept header.

> **Note**
> When the request does not include the format parameter or Accept header, the default result format for SELECT queries is SPARQL XML (**application/sparql-results+xml**). For CONSTRUCT queries, the default format depends on whether the query includes GRAPH clauses. If no GRAPH clause is present, the default format for CONSTRUCT results is RDF Turtle. If GRAPH clauses are present, the default format is RDF TriG.

Format	Accepted Values	Query Type	Description
XML	**application/sparql-results+xml** **application/xml** **xml** **xml2** **srx**	SELECT only	Returns results in SPARQL Query Results XML Format.
	application/rdf+xml **rdf**	CONSTRUCT only	Returns results in RDF 1.1 XML format.

Format	Accepted Values	Query Type	Description
	owl **rdfs**		
JSON	**application/json** **json**	SELECT and CONSTRUCT	For SELECT queries, results are returned in SPARQL Query Results JSON Format. For CONSTRUCT queries, results are returned in Anzo's native JSON RDF serialization format. See Anzo JSON RDF Serialization for details.
	application/sparql-results+json	SELECT only	Returns results in SPARQL Query Results JSON Format.
CSV	**text/csv** **csv**	SELECT only	Returns results in SPARQL Query Results CSV Format.
TriG and Gzipped TriG	**application/x-trig** **trig** **application/x-trigz** **trigz** **gz** **trig.gz**	CONSTRUCT only	CONSTRUCT queries with a GRAPH clause return RDF 1.1 TriG by default if no format is specified.
Turtle and Gzipped Turtle	**application/x-turtle** **ttl** **application/x-turtlez** **ttlz** **ttl.gz**	CONSTRUCT only	Returns RDF 1.1 Turtle. CONSTRUCT queries without a GRAPH clause return Turtle by default if no format is specified.

Format	Accepted Values	Query Type	Description
N-Triples	**text/plain** **nt**	CONSTRUCT only	Returns results in RDF 1.1 N-Triples format.
Notation3 and Gzipped Notation3	**text/rdf+n3** **n3** **text/rdf+n3z** **n3z** **n3z.gz**	CONSTRUCT only	Returns results in RDF Notation3 format.
N-Quads	**text/x-nquads** **nq** **nquad** **nquads**	CONSTRUCT only	Returns results in RDF 1.1 N-Quads format.
TriX	**application/trix** **trix**	CONSTRUCT only	Returns results in RDF Triples in XML format.

Update Parameters

Parameter	Description
update	Specifies the full update string to run. If you do not specify a url-encoded_dataset_uri, using-graph-uri or using-named-graph-uri in the request, the update query should contain the appropriate USING clauses. To run a non-update query (SELECT or CONSTRUCT), use the query parameter.
using-graph-uri	Specifies a default graph URI to update. You can include this parameter multiple times in a request. When the base URL specifies a graphmart URI, you can specify a data layer URI to narrow the scope of the update to a specific data layer in the

Parameter	Description
	graphmart.
using-named-graph-uri	Specifies a named graph URI to update. You can include this parameter multiple times in a request. When the base URL specifies a graphmart URI, you can specify a data layer URI to narrow the scope of the update to a specific data layer in the graphmart.
includeMetadataGraphs	A boolean value that specifies whether to query the metadata graphs. Only valid for queries that target a linked data set (LDS) that is stored in a local volume. The default value is **false**.

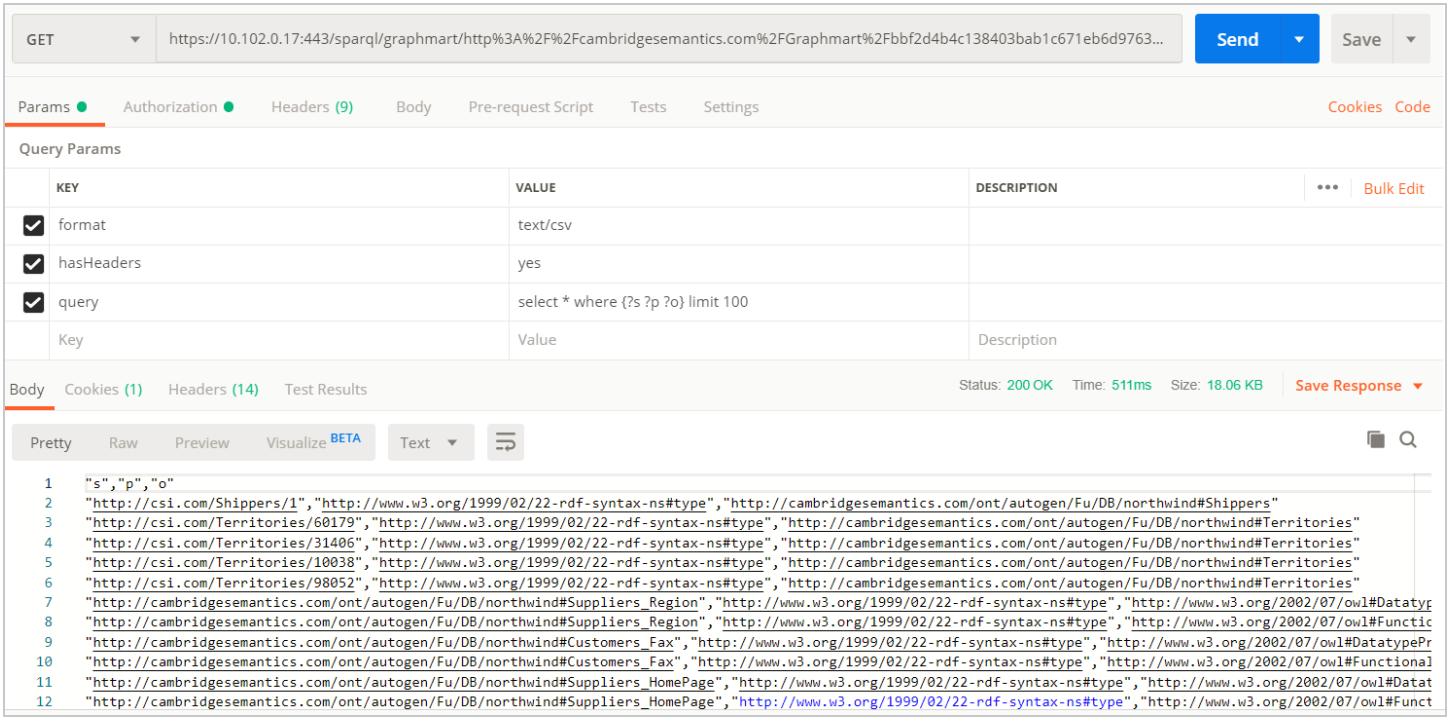
Examples

The following example uses cURL to send a request that runs a SELECT query against a graphmart. Since the request does not include an Accept header or format parameter, results will be returned in SPARQL XML format.

```
curl --user sysadmin:@nz0 -c cookiejar.txt -L -v -k

http://10.100.10.20/sparql/graphmart/http%3A%2F%2Fcambridgesemantics.com%2FGraphmart%2F
2dc579b101654ae29eb91b0c7d046ca1
--data-urlencode "query=SELECT * WHERE{ ?s ?p ?o . } LIMIT 100"
```

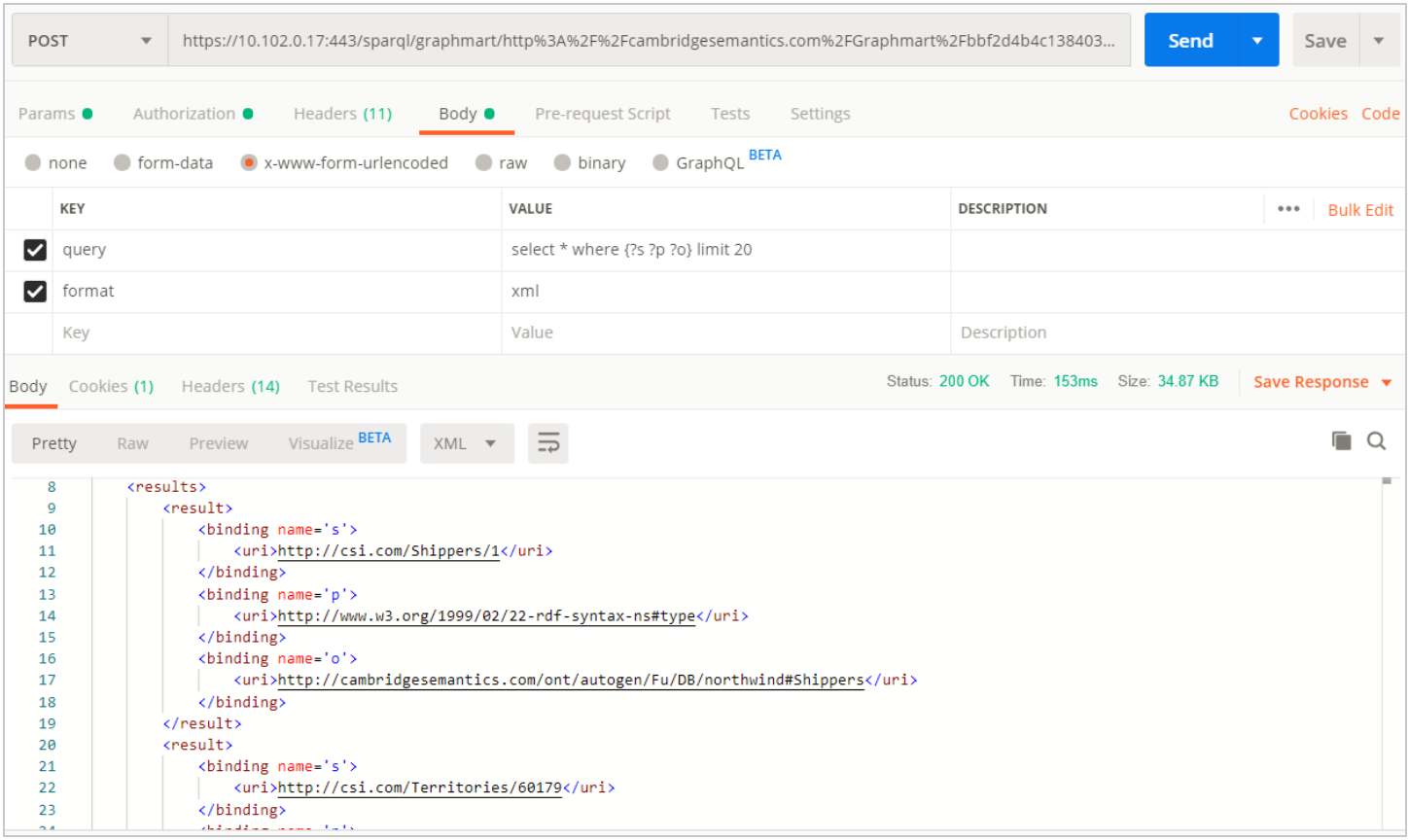
The following example sends a GET request that runs a SELECT query against a graphmart. The format parameter is included to format the results in text/csv serialization.



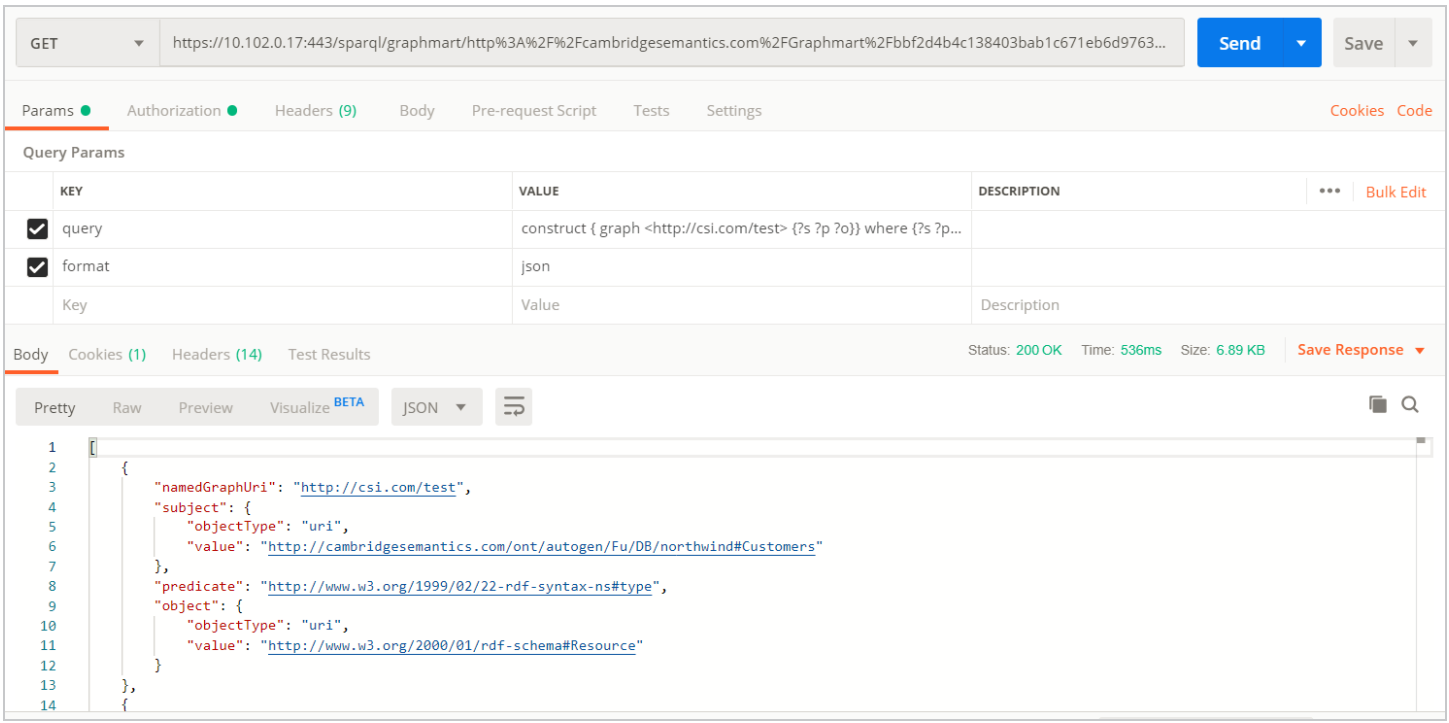
For reference, below is the URL-encoded version of the request string shown in the image above. When sending a request from a client that does not automatically encode requests, you must convert the string. Line breaks are added for readability:

```
http://10.100.10.20/sparql/graphmart/http%3A%2F%2Fcambridgesemantics.com%2F
Graphmart%2F646861d1bab54d67bc79dea94e02f3e6
?query=select%20*%20where%20%7B%3Fs%20%3Fp%20%3Fo%7D%20limit%20100
```

The example below sends a POST request that runs a SELECT query. In this example, the query is included in the body of the request and the response format is XML.



The example below sends a GET request that runs a CONSTRUCT query. The response format is set to JSON, and the results are formatted in Anzo JSON RDF Serialization.



The example below uses a Python script to send a request that runs a SPARQL query.

```
import requests
import urllib

server = 'https://company.anzo.com:'
port = 443
graphmart = 'http://cambridgesemantics.com/Graphmart/be4bd080c5654628b6fff90ca1b647d6'
url = server + str(port) + '/sparql/graphmart/' + urllib.quote_plus(graphmart)
#urllib.parse.quote_plus(graphmart) in Python 3

queryText = 'SELECT * WHERE {?instance a ?type .} LIMIT 10'
payload = {'query':queryText, 'format':'text/csv'}

r = requests.post(url, data = payload, auth = ('sysadmin','<pw>'))
print r.text
```

Access the HTTP Client Interface

In addition to the SPARQL HTTP(S) endpoint that enables users to send SPARQL queries to Anzo over HTTP, Anzo provides an HTTP(S) servlet that can be used to invoke Anzo client operations over HTTP. The client servlet enables external systems to interact with Anzo semantic services as well as custom services. It also enables remote servers to interact with Anzo without needing the Anzo command line interface.

HTTP Methods and Options

The Anzo client servlet accepts HTTP GET and POST methods. GET is used for operations that retrieve data, and POST is used for update operations that add or remove data. Update operations must use the POST method, and read operations can be submitted using GET or POST.

Client Servlet Base URL

Use the following URL to access Anzo services via the HTTP client servlet. The table below describes each URL component:

```
<protocol>://<hostname>:<port>/anzoclient/<client_operation>
```

For example:

```
https://10.100.10.20:8443/anzoclient/call
```

Option	Description
protocol	The protocol to use for the connection: **http** for HTTP protocol or **https** for SSL protocol.
hostname	The DNS name or IP address of the Anzo server.
port	The port for the endpoint. The port that you specify depends on the protocol that you choose. By default, the HTTP port is **80** and the HTTPS port is **443**. To view the ports that are configured for your Anzo instance, see **Server Settings** in the

Option	Description
	Administration menu.
anzoclient	Required keyword for the client servlet.
client_ operation	The type of Anzo client operation to invoke. The list below provides an overview of the supported operation types. For more information about the operations, see Client Operations below. • **call**: Invokes the semantic service operation identified by the URI provided in the request (analogous to the `anzo call` CLI command) • **add**: Imports the specified statements to Anzo (analogous to the `anzo import` CLI command) • **remove**: Removes the specified statements from Anzo (analogous to the `anzo update -r` CLI command) • **get**: Gets the specified named graph from Anzo (analogous to the `anzo get` CLI command) • **find**: Finds the statements in Anzo that match the specified pattern (analogous to the `anzo find` CLI command)

Client Operations

This section provides usage information and examples for each of the Anzo client operations.

- Call
- Add
- Remove
- Get
- Find

Call

The call operation invokes a semantic service. Identify the service to call by providing the URI for the service in the request header. The call operation is supported with HTTP GET and POST methods. When including RDF data as input to the service, the request must use the POST method.

Call Header Options

Call operations support the following header parameters. Only the **uri** parameter is required:

- **uri**: **Required** parameter that specifies the URI of the semantic service to invoke.
- **mimeType**, **Content-Type**, or **format**: Include one of these optional parameters to specify the MIME type for the RDF serialization used in the request body as well as the response from the service. The default type is **application/json** if the header does not specify the format. For more information about the supported RDF serialization types, see Format Options.

Call Body Options

If the call operation supplies data as input to the service, include the data in the request body. The data must be serialized as specified in the request header, or **application/json** if the header does not specify a serialization type.

Call Examples

The following cURL example uses a GET call to invoke a health check service.

```
curl https://10.100.10.20:8443/anzoclient/call \
  --user sysadmin:123 \
  --header 'uri: http://www.csi.com/service/genericIngestManager#healthCheck'
```

The example below uses a POST call to invoke a service operation. The call passes in a request data set that is serialized as RDF JSON.

```
curl https://10.100.10.20:8443/anzoclient/call \
  --header 'Content-Type: application/json' \
  --user sysadmin:123 \
  --header 'uri: http://someServiceURI#someOperation' \
  --data '{"subject" : {"objectType": "uri" ,"value" : "urn://test"},
           "predicate" : "urn://predicate",
```

```
          "object" : {"objectType": "uri" ,"value" : "urn://object"},
          "namedGraphUri" : "urn://ng"}'
```

The example below uses a POST call to invoke a service operation. The call passes in a request data set that is serialized as TriG.

```
curl https://10.100.10.20:8443/anzoclient/call \
  --header 'Content-Type: application/x-trig' \
  --user sysadmin:123 \
  --header 'uri: http://www.csi.com/service/genericIngestManager#healthCheck'
  --data '<urn://ng> { <urn://test> <urn://predicate> <urn://object> .}'
```

Add

The add operation adds statements to the Anzo graphstore. Add is supported with the HTTP POST method. The header can include **mimeType**, **Content-Type**, or **format** to specify the MIME type for the RDF serialization used in the request body as well as the response from the service. The default type is **application/json** if the header does not specify the format. For more information about the supported RDF serialization types, see Format Options. The request body includes the statements to add.

Add Examples

The following example add operation uses cURL to issue a POST call to add a statement to the graphstore. The statement is specified in Anzo JSON RDF serialization format.

```
curl https://10.100.10.20:8443/anzoclient/add \
  --user sysadmin:123 \
  --data '{"subject" : {"objectType": "uri" ,"value" : "urn://test"},
           "predicate" : "urn://predicate",
           "object" : {"objectType": "uri" ,"value" : "urn://object"},
           "namedGraphUri" : "urn://ng"}'
```

Remove

The remove operation deletes statements from the Anzo graphstore. Remove is supported with the HTTP POST method. The header can include **mimeType**, **Content-Type**, or **format** to specify the MIME type for the RDF serialization used in the request body as well as the response from the

service. The default type is **application/json** if the header does not specify the format. For more information about the supported RDF serialization types, see Format Options. The request body specifies the statements to remove.

Remove Examples

The following example remove operation uses cURL to issue a POST call to remove a statement from the graphstore. The statement is specified in Anzo JSON RDF serialization format.

```
curl https://10.100.10.20:8443/anzoclient/remove \
  --user sysadmin:123 \
  --data '{"subject" : {"objectType": "uri" ,"value" : "urn://test"},
           "predicate" : "urn://predicate",
           "object" : {"objectType": "uri" ,"value" : "urn://object"},
           "namedGraphUri" : "urn://ng"}'
```

Get

The get operation retrieves a named graph from the Anzo graphstore. The get operation is supported with HTTP GET and POST methods. The named graph URI that contains the contents to retrieve can be included as a query parameter or as a **uri** parameter in the request body. The get operation also returns the metadata graph, which is equivalent to running `anzo get -m <named_graph_uri>` with the Anzo admin CLI. The header can include **mimeType**, **Content-Type**, or **format** to specify the MIME type for the RDF serialization used to format the response from the service. The default type is **application/json** if the header does not specify the format.

Get Examples

The following example get operation uses cURL to retrieve the contents of a named graph.

```
curl -k -XPOST https://10.100.10.20:8443/anzoclient/get --user sysadmin:123
     --data-urlencode
     "uri=http://cambridgesemantics.com/Graphmart/9da211618a15476daa10cead2292d8e7"
```

This example uses Python with requests:

```
import requests

url = "https://10.100.10.20:8443/anzoclient/get"
data = {"uri":
```

```
"http://cambridgesemantics.com/Graphmart/9da211618a15476daa10cead2292d8e7"}
username = "sysadmin"
password = "123"
r = requests.post(url, data=data, auth=(username, password), verify=False)
print (r.text
```

Find

The find operation finds the statements in the graphstore that match the pattern that is specified in the request. The find operation is supported with HTTP GET and POST methods. Header options are not applicable. The list below describes each of the supported parameters. These parameters can be included as query parameters in the URL or as parameters in the request body:

- **graph**: The named graph URI for the find pattern.
- **sub**: The subject of the find pattern.
- **pred**: The predicate of the find pattern.
- **lit**: The object of the find pattern if that object is a literal value.
- **uri**: The object URI of the find pattern if that object is a URI.
- **type**: If the object is a literal, this parameter can be used to specify the data type of the literal value.
- **lang**: If the object is a literal, this parameter can be used to specify the language of the literal value.

Results returned by the find operation are in Anzo JSON RDF serialization format. See Anzo JSON RDF Serialization below for details.

Find Examples

The following example find operation (using the GET HTTP method) finds all of the statements in the graphstore with predicate `http://w3.org/1999/02/22-rdf-syntax-ns#type` and an object URI of `http://cambridgesemantics.com/ontologies/2009/05/LinkedData#LinkedDataSet`. The parameters are specified as query parameters in the URL.

```
curl https://10.100.10.20:8443/anzoclient/find?pred=http://www.w3.org/1999/02/22-rdf-
syntax-
ns%23type&amp;uri=http://cambridgesemantics.com/ontologies/2009/05/LinkedData%23LinkedD
ataSet' \
  --user sysadmin:123
```

The example below finds the same statements but issues a POST call. The URL-encoded parameters are specified in the request body.

```
curl https://10.100.10.20:8443/anzoclient/find \
  --user sysadmin:123
  --data 'pred=http%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-
ns%23type&uri=http%3A%2F%2Fcambridgesemantics.com%2Fontologies%2F2009%2F05%2FLinkedData
%23LinkedDataSet'
```

Anzo JSON RDF Serialization

Anzo's JSON RDF serialization standard is straightforward but differs from the common public JSON RDF serialization standards. In Anzo JSON serialization format, a set of statements (quads) are represented as an array of JSON objects. Each JSON object (statement) is defined as a key/value pair, where the key specifies the component of the statement, i.e., the subject, predicate, object, or namedGraphUri. Depending on the component, properties such as the component's value and data type are specified in nested objects.

The following example array shows Anzo's JSON serialization. The list below the example describes the structure.

```
[
  {
    "subject" : {
        "objectType": "uri" ,
        "value" : "urn://test"
    },
    "predicate" : "urn://predicate",
    "object" : {
        "objectType": "uri" ,
        "value" : "urn://object"
    },
    "namedGraphUri" : "urn://ng"
    },
```

```
  {
    "subject" : {
        "objectType": "uri" ,
        "value" : "urn://test"
    },
    "predicate" : "urn://predicate2",
    "object" : {
        "objectType": "literal" ,
        "value" : "test literal",
        "dataType" : "http://www.w3.org/2001/XMLSchema#string"
    },
    "namedGraphUri" : "urn://ng"
  }
]
```

- **subject** is a JSON object with two properties:
  - **objectType**: The resource type of the subject value. This is either a "uri" or "bnode" (blank node).
  - **value**: The blank node value or a string literal that specifies the URI.
- **predicate** is a string literal that specifies the predicate URI.
- **object** is a JSON object with two required properties and two optional properties:
  - **objectType**: Required property that specifies whether the object is a "uri," "literal," or "bnode."
  - **value**: Required property that specifies the string representation of the object value.
  - **dataType**: Optional property for use if the objectType is "literal." This property describes the data type of the literal value. It is a string literal of the XSD data type URI. For example: "http://www.w3.org/2001/XMLSchema#string"
  - **language**:Optional property for use if the objectType is "literal." This property describes the language of the literal value.
- **namedGraphUri** is a string literal that specifies the named graph URI.

Share Access to Artifacts

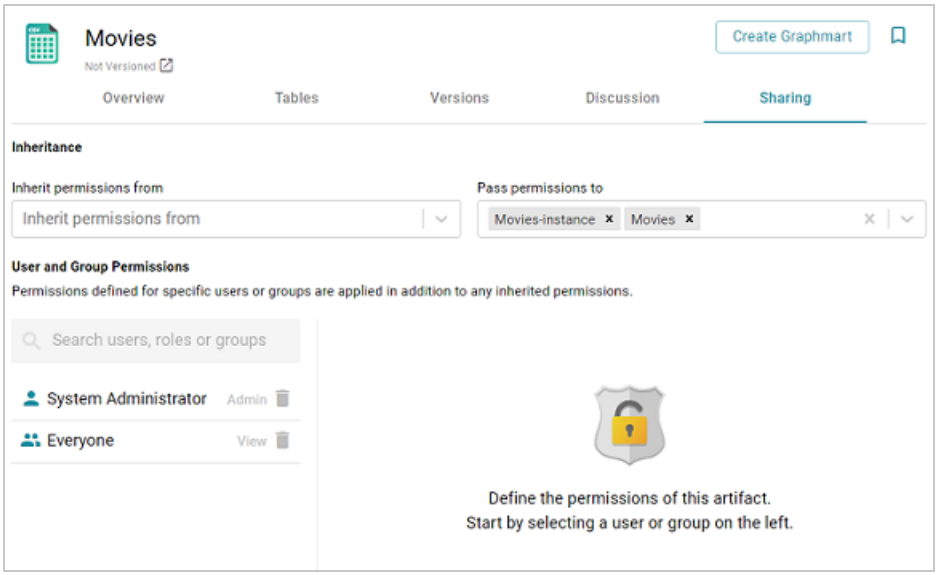
All Anzo artifacts—data sources, schemas, models, graphmarts, etc.—that you create can be shared with other groups (or users) from the **Sharing** tab in the Anzo application. This topic provides an overview of the Sharing tab and basic instructions for configuring artifact permissions.

> **Note**
> For specifics about sharing multifaceted artifacts like graphmarts that include layers and dashboards that include multiple lenses, see Sharing Access to Graphmarts and Sharing Access to Dashboards and Lenses.

- Sharing Tab Overview
- Permission Settings
- Sharing an Artifact

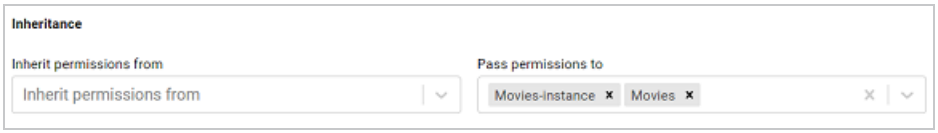
Sharing Tab Overview

Access the Sharing tab by navigating to an artifact and clicking **Sharing**. For example, the image below shows the Sharing tab for a data source.



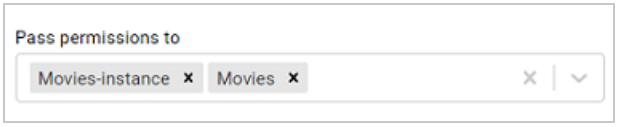
Inheritance

The top of the screen displays the permission inheritance settings:



To facilitate common workflows, the Anzo application applies logic so that artifacts in the same workflow inherit the same permissions. You can alter the inheritance configuration by choosing the artifact or artifacts that this artifact should inherit from by choosing the artifacts to **Inherit permissions from**. Since the example above is a data source and no artifacts precede the source in Anzo, the Inherit Permissions From setting is empty.

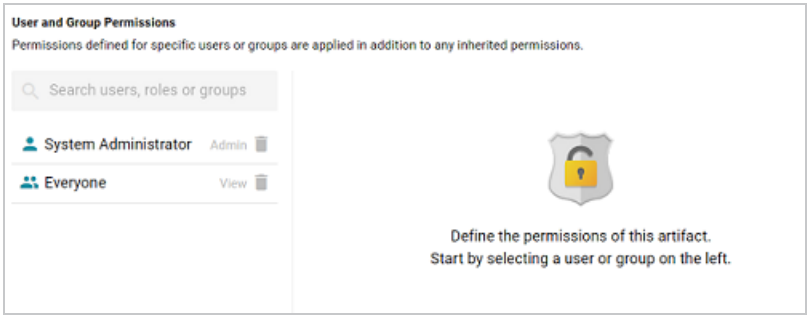
You can also configure an artifact to **Pass permissions to** other artifacts. In the example above, the Movies data source passes permissions to the Movies schema instance and the Movies graphmart.



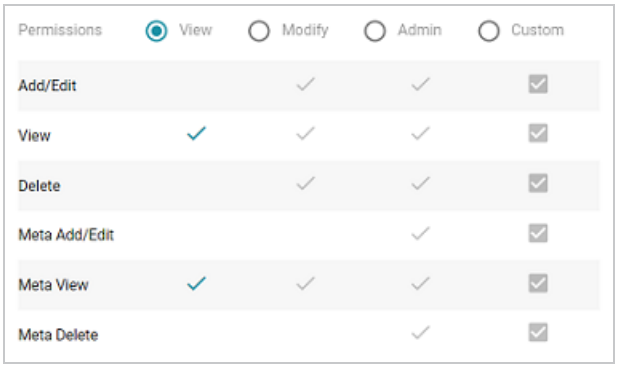
> **Note**
> Permissions are additive. Copying permissions from multiple artifacts with differing permission levels results in the super set being acquired by the artifact that is inheriting the permissions. In addition, any permissions that are configured in the table at the bottom of the screen are also added to the set. For more conceptual information about permission inheritance, see Permission Inheritance in the Administration Guide.

User and Group Permissions

In addition to the inheritance settings, the bottom of the screen lists the users and groups that this artifact has been shared with. For example:



The current level of access is listed next to each name: **View**, **Modify**, or **Admin**. View, Modify, and Admin are predefined permission sets. Each predefined set selects a certain combination of six permissions. You also have the option to create a **Custom** set of permissions. Selecting a user or group from the list displays the following permissions table on the right side of the screen:



Typing a value in the **Search users, roles or groups** field finds and displays the users or groups that you can add to the list. The Permission Settings section below defines each of the permission sets.

Permission Settings

The tables below list the predefined permission sets and describe the privileges that are granted for each permission that is part of the set:

View

The following table describes the permissions in the **View** set.

Permission	Allows a user to:
View	• See the artifact in the Anzo application. • Create versions of the artifact.
Meta View	Relates only to an artifact's permissions. A user with Meta View can see the permissions on the Sharing tab but they cannot modify, add, or remove permissions.

Modify

In addition to the **View** and **Meta View** permissions described above, the **Modify** set includes the **Add/Edit** and **Delete** permissions described below.

Permission	Allows a user to:
Add/Edit	• Change an artifact, such as to rename it or edit its description. • Add an entity to an artifact. For example, add a schema to a data source or a layer to a graphmart.
Delete	• Remove an entity from the artifact. For example, delete a layer from a graphmart or a schema from a data source. • **Does not give permission to remove the parent artifact**. For example, a user can remove a schema from a source but cannot delete the data source.

Admin

In addition to the **View**, **Meta View**, **Add/Edit**, and **Delete** permissions described above, the **Admin** set includes the **Meta Add/Edit** and **Meta Delete** permissions described below.

Permission	Allows a user to:
Meta Add/Edit	Relates only to an artifact's permissions. A user with Meta Add/Edit can add permissions to a user or group. They cannot remove permissions from any user or group.
Meta Delete	• Remove permissions from a user or group. • Delete the parent artifact and its related entities.

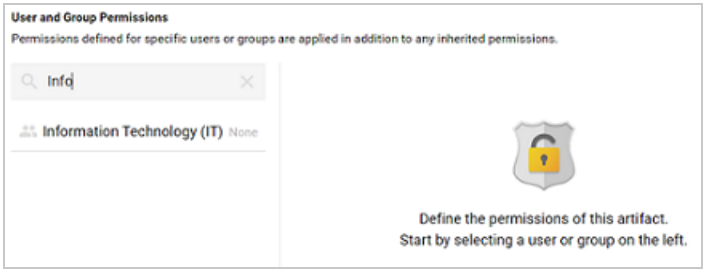
Sharing an Artifact

Follow the instructions below to share access to an artifact.

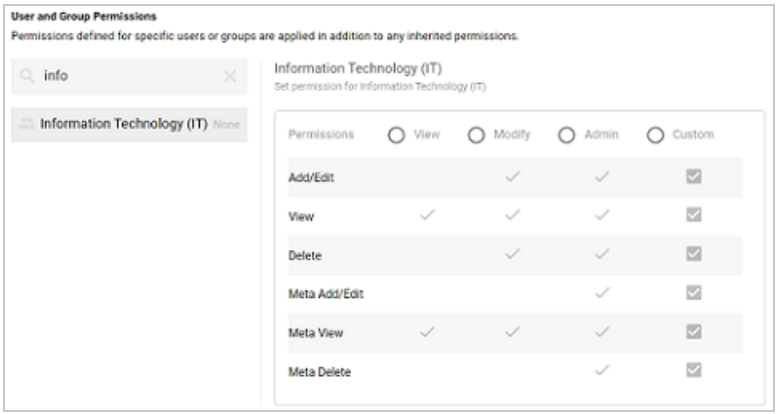
1. In the Anzo application, navigate to the artifact that you want to share access to. Then click the **Sharing** tab.
2. If you want to change the inheritance for the artifact, use the fields at the top of the screen:
   - To apply all of the permissions from another artifact to this one, select the artifact to inherit from in the **Inherit permissions from** field.
   - To pass this artifact's permissions to other artifacts, select the artifacts to pass permissions to in the **Pass permissions to** field.

   Note
   Permissions are additive. Copying permissions from multiple artifacts with differing permission levels results in the super set being acquired by the artifact that is inheriting the permissions. In addition, any permissions that are configured in the table at the bottom of the screen are also added to the set. For more conceptual information about permission inheritance, see Permission Inheritance in the Administration Guide.
3. To share access to this artifact with a particular user or group, type a value in the **Search users, roles or groups** field to find and display the user or group. The resulting list shows the current permission level that is set for each user or group in the search results. For example,

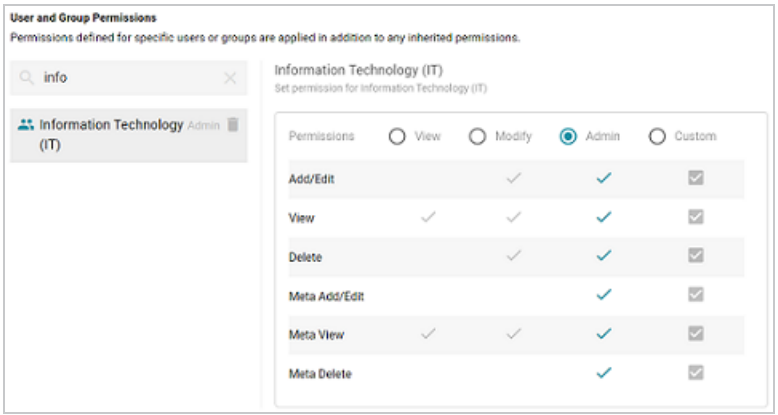
the image below shows the current permissions for the IT group (None):



4. Select the user or group for which you want to configure permissions. The permissions settings are displayed on the right side of the screen. For example:



5. To assign a predefined set of permissions, click the **View**, **Modify**, or **Admin** radio button to assign that level of access to the selected user or group. Refer to Permission Settings above for details about the permissions sets. For example, the image below gives **Admin** permissions to users in the IT group:



If you want to customize the permissions, click the **Custom** radio button and then select or deselect the permissions checkboxes. To clear permissions for a user or group, click the trashcan icon (🗑) next to the name.

Repeat the steps above to share the artifact with additional groups. Changes to permissions take effect immediately. Users do not need to log out of the application and log back in.

Version and Migrate Artifacts

The topics in this section provide guidance on creating backup versions of artifacts and migrating artifacts by exporting them from one server and importing them to another server.

Tip

For information about migrating artifacts and their related entities in bulk, see Migration Packages in the Administration Guide.

In this section:

Creating and Restoring Versions of Artifacts 981

Exporting an Artifact 986

Making Properties Replaceable on Export 991

Importing Exported Versions of Artifacts 992

Creating and Restoring Versions of Artifacts

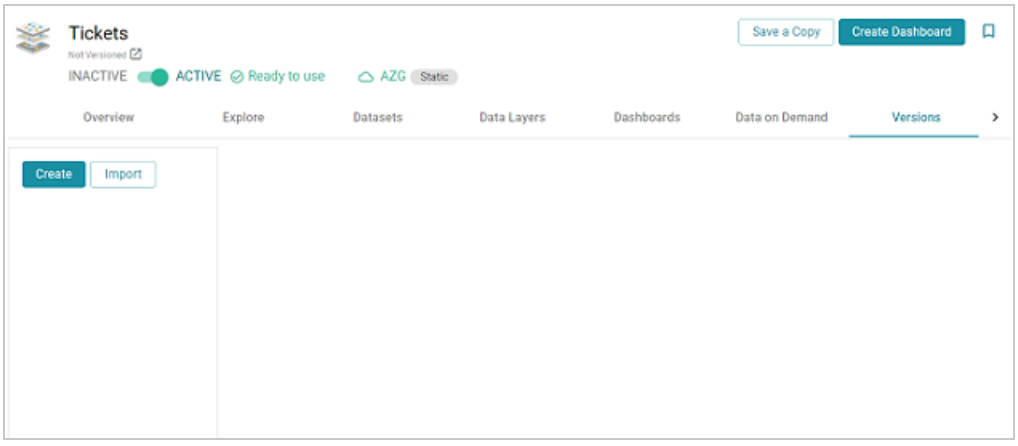
Before making changes to data sources, schemas, unstructured pipelines, models, graphmarts, etc., users can take a snapshot of the current version of that artifact. When a backup is created, Anzo automatically creates a backup version of each entity that is related to that artifact. For example, backing up a graphmart backs up the same version of any related models, datasets, and dashboards. In addition, Anzo backs up the metadata graphs for all of the entities. Metadata graphs store information about the artifacts such as the creator and creation date and the permissions configuration. Changed artifacts can be reverted at any time to any of the saved versions. If an artifact is restored to a previous version, Anzo automatically saves a version of the current state of the artifact and its related entities and metadata. Follow the appropriate instructions to create or restore a backup version of an artifact.

- Create a Backup Version
- Restore a Backup Version

Create a Backup Version

Follow the instructions below to save a snapshot of an artifact.

1. In the Anzo application, navigate to the artifact that you want to back up, and then click the **Versions** tab. For example, the image below shows the Versions tab for a graphmart. In this example, the graphmart does not have any backup versions:

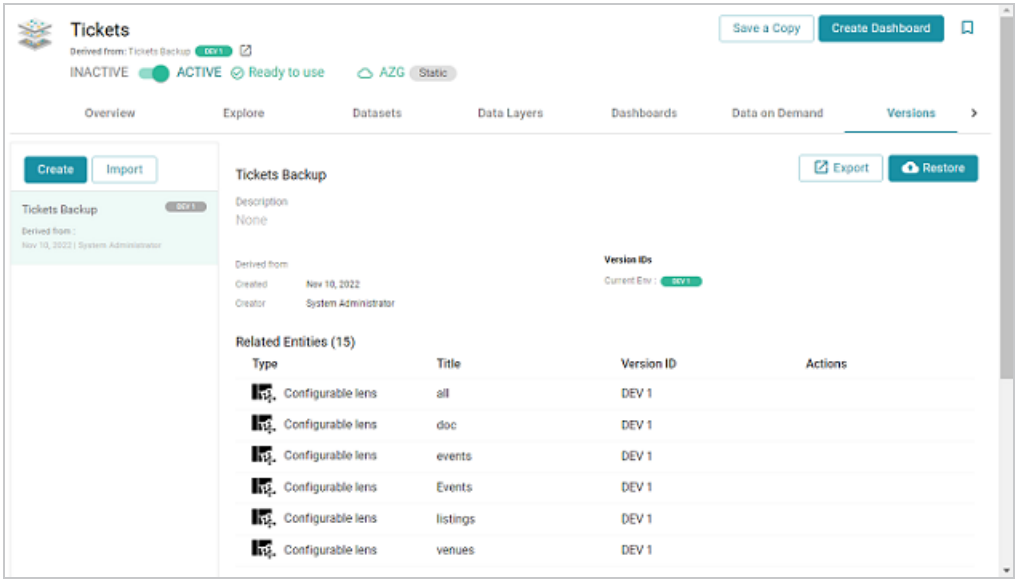


> **Note**
> For data models, add the model that you want to back up to the Working Set and then open it in the Model editor. Then click the **Versions** tab.

2. Click the **Create** button. Anzo displays the Create New Version screen.



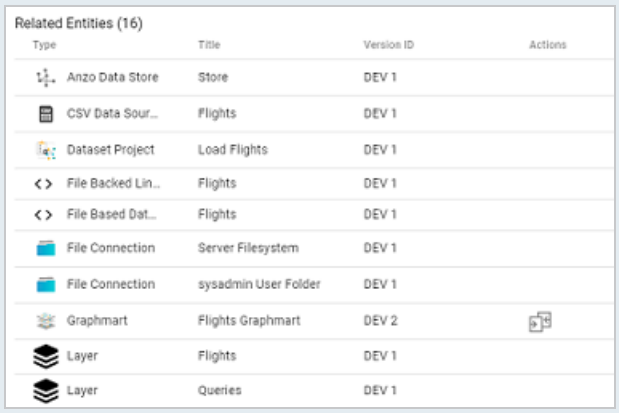
3. In the **Name for New Version** field, type a name for the backup version. Then type details about the version in the optional **Comment for New Version** field.
4. Click **Save**. Anzo takes a snapshot of the artifact as well as its related entities and adds the version to the list on the left side of the screen. Depending on the size and number of related entities, the backup operation can take a few minutes to complete. For example:



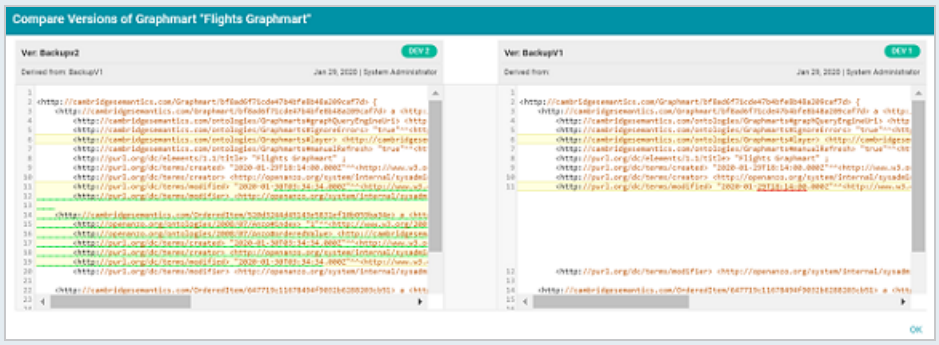
5. If necessary, select the new version in the list to view details on the right side of the screen. The screen displays details such as the version creator and created date and lists each of the related entities that were also backed up. In the list of related entities, the **Actions** column displays a compare icon next to each entity that has changed since the previous version.

Example

In the image below, the compare icon in the Graphmart row indicates that this version of the graphmart includes changes that were not in the previous version:



Clicking the icon in the Actions column opens the Compare Versions dialog box, which shows a side-by-side comparison of the TriG files for the two versions:



Users can now make changes to the current version of the backed up artifacts, and the new changes can be reverted to a backup version at any time.

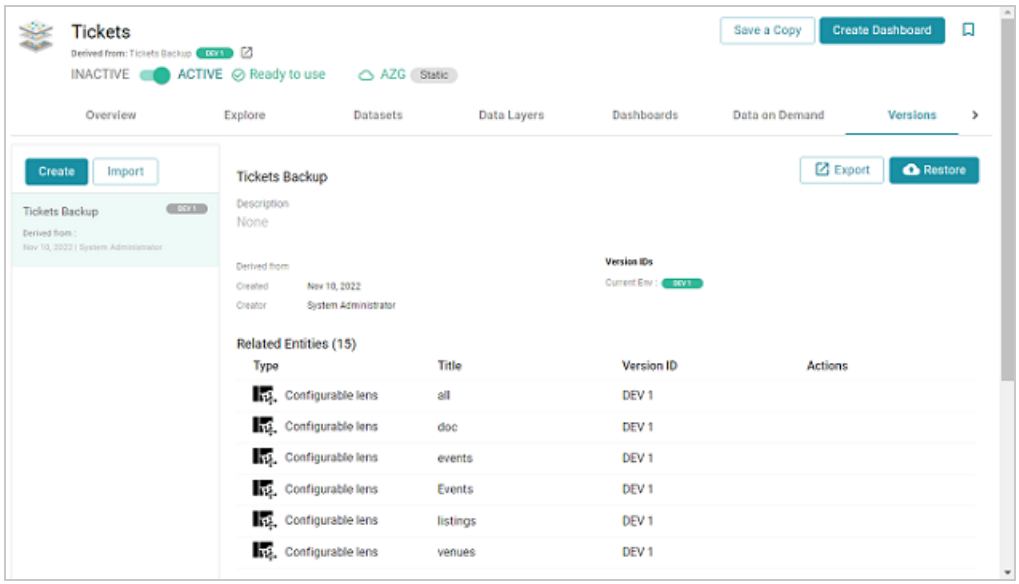
Restore a Backup Version

Follow the instructions below to restore an artifact and its related entities to a previous backup version.

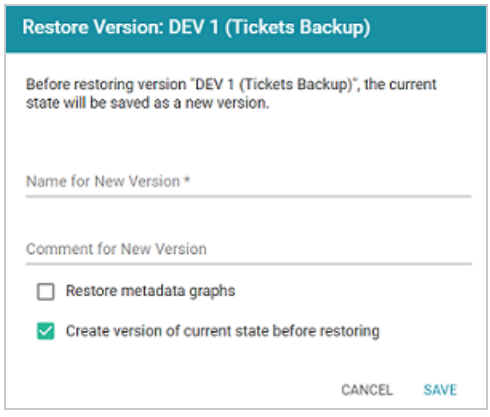
1. In the Anzo application, go to **Versions** tab for the artifact that you want to restore.

> **Note**
> For data models, add the model that you want to restore to the Working Set and then open it in the Model editor. Then click the **Versions** tab.

2. On the Versions screen, select the backup version that you want to restore. For example:



3. Click the **Restore** button to restore the artifact to the version that you selected. Since Anzo automatically creates a snapshot of the current version before you restore an artifact, Anzo displays the Restore Version dialog box so that you can specify a label for the new version.



4. In the Restore Version dialog box, type a name for the version in the **Name for New Version** field. You can also add an optional comment in the **Comment for New Version** field.
5. Specify whether you want to revert to the backed up version's metadata graphs for this artifact and its related entities:
   - If you want the restored version to use the metadata, such as permission configuration and last created date, that was saved at the time of the backup, select the **Restore metadata graphs** checkbox. Anzo will revert the metadata to the saved version.
   - If changes were made to the metadata for the current version of the artifact and you want to preserve those changes, such as if the permissions were modified to further restrict or allow access, leave the **Restore metadata graphs** checkbox blank. Anzo will preserve the current metadata graphs instead of reverting the metadata to the saved version.
6. Specify whether you would like Anzo to create a backup version of the current version of the artifact before restoring it:
   - If you want Anzo to make a backup copy of the current version before restoring it, leave the **Create version of current state before restoring** checkbox selected.
   - If you want do not want Anzo to create a backup copy of the current version before restoring it, clear the **Create version of current state before restoring** checkbox.
7. Click **Save**. Anzo saves the current version and restores the current files to the backup version. The new version is added to the list of available backups.

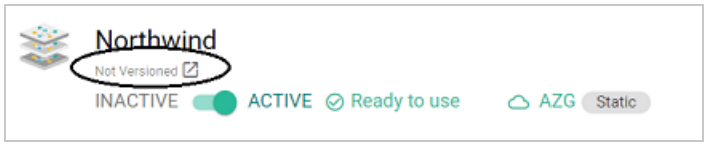
Exporting an Artifact

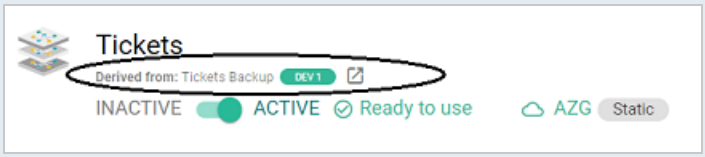
You can export the current version of an artifact and its related entities or any backup version. Follow the instructions below to export an artifact. For instructions on creating a backup version, see Creating and Restoring Versions of Artifacts.

1. In the Anzo application, navigate to the artifact that you want to export.

   > **Note**
   > For data models, add the model that you want to export to the Working Set and then open it in the Model editor.

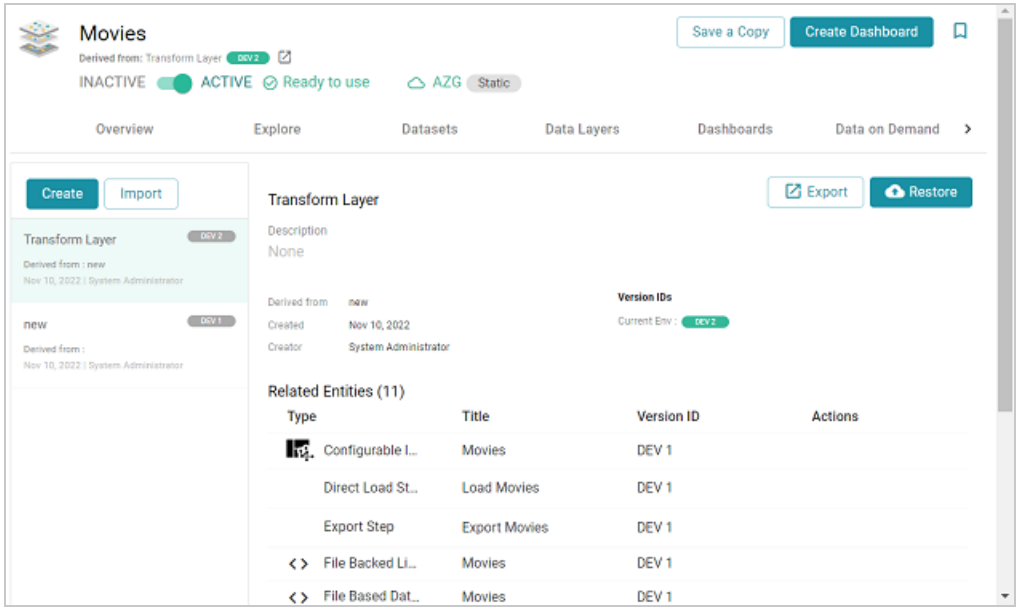
2. Follow one of the options below, depending on whether you want to export the current version of the artifact or a backup version:
   - If you want to export the current, working version of the artifact, click the Export icon (⧉) under the artifact name.

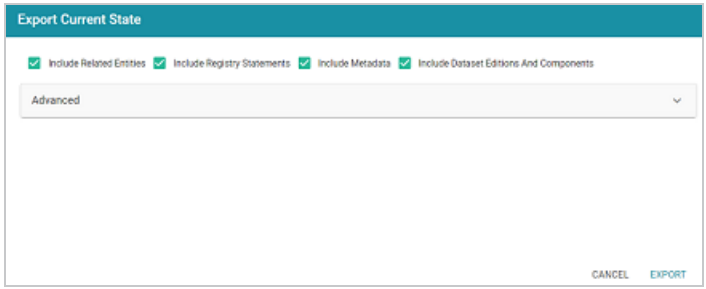
     > **Tip**
     > The image below shows an example of the Export icon for an artifact that has a backup version. Clicking the Export icon (⧉) exports the current version of the artifact, not the backup version that is listed.
     >
     > 
     >

   - If you want to export a backup version of the artifact, click the **Versions** tab, which lists the backups that exist for the artifact. Select the version that you want to export and then click the **Export** button. For example, the image below shows the Versions tab for a

graphmart:



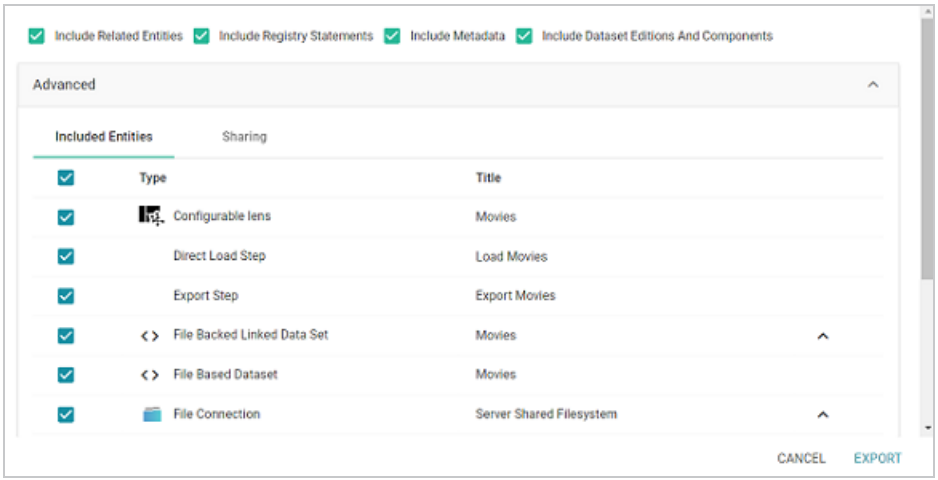
The Export dialog box is displayed. For example:



3. On the Export dialog box, configure the following export options as needed. The list below describes each option:
   - **Include Related Entities**: Indicates whether to export the artifact's related entities. Since most artifacts have dependencies with other artifacts, Cambridge Semantics recommends that you enable **Include Related Entities** (selected by default) and export all related entities. The number and type of related entities that are included varies by the type of artifact that is being exported.
   - **Include Registry Statements**: Indicates whether to export the registry statements for the artifact and each of its related entities.

- **Include Metadata**: Indicates whether to export the metadata graph for the artifact and its related entities, such as the permissions configuration and last modified date. If you exclude the metadata, the artifacts in this export will follow the metadata configuration on the destination server when they are imported. Select **Include Metadata** if you want to migrate the existing metadata to the destination server. Enabling this setting also gives you the option to change the permissions configuration for the exported entities.
- **Include Dataset Editions and Components**: If a dataset is included in the export package, this setting controls whether to include all of the editions and components with that dataset.

4. If you want to change permissions or replace the values for certain properties in the exported version of an entity, such as the user name and password for a data source, the base folder location for a File Store connection, or the file path for an Anzo Data Store, expand the **Advanced** option to view the Included Entities list. For example:

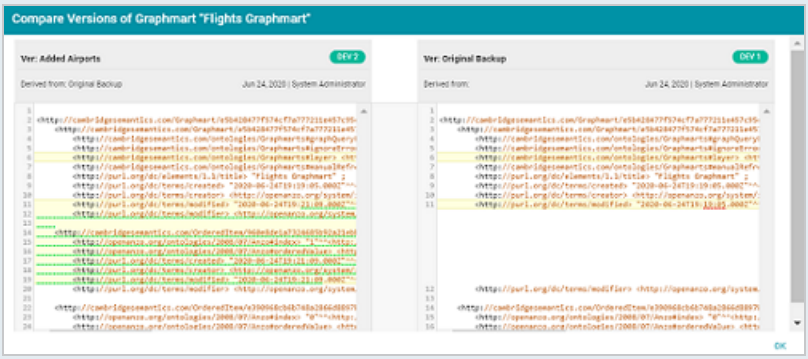


The entities with replaceable values are expandable. Click the ^ character to the right of an entity name to expand the options and view the editable properties. Replace any of the existing values with the new values that you want to define for the exported version of the artifact.

> **Tip**
> For information about configuring additional properties so that their values are replaceable on export, see Making Properties Replaceable on Export.

If you specified **Include Metadata** and want modify permissions for the exported entities, click the **Sharing** tab. For information about changing permissions on the Sharing tab, see Share Access to Artifacts.

> **Tip**
> When a row in the Related Entities list includes the compare versions icon ( ) in the Actions column, you can click the icon to open a side-by-side comparison of the TriG files for the two versions. For example:



5. Click **Export** to export the artifacts.

Anzo packages the files into a .zip file and downloads it to your computer. You do not need to extract the files in order to import the artifacts to another Anzo server. See Exported ZIP File Contents below for a description of the files that are included in the .zip file.

Exported ZIP File Contents

Depending on the options configured for the export, the resulting .zip file contains one or more of the following files:

- **<artifact_name>_export.trig** contains statements about the type of artifact that was exported and the export settings that were configured.
- **<artifact_name>_graph.trig** contains the artifact definitions.
- **<artifact_name>_metadata.trig** contains metadata statements such as the access control configuration and last modified date for the exported entities.
- **<artifact_name>_registry.trig** contains registry statements such as the named graph information for the artifacts.

Making Properties Replaceable on Export

When exporting artifacts, users can replace the existing values for certain properties like the user name and password for database data sources, the base folder location for file connections, and the file path for Anzo Data Stores. This topic provides instructions for configuring additional properties so that their values can be modified in the exported version of an artifact.

To configure a property so that its value is replaceable on export, add the following statement to the **http://cambridgesemantics.com/annotations/replaceStatements** graph:

```
<class_URI> http://cambridgesemantics.com/ontologies/2018/06/Export#replaceStatement
<property_URI>
```

Where **<class_URI>** is the URI for the class that defines the property whose value should be replaceable. And **<property_URI>** is the URI of the property.

> **Important**
> The specified property must be a datatype property that contains a literal value.

You can use the following TriG contents as a template for defining properties with replaceable values. The contents show the default replaceable properties. You can add your statements to the `ann:replaceStatements` list and then import the file.

```
@prefix ds:  <http://cambridgesemantics.com/ontologies/DataSources#> .
@prefix exp: <http://cambridgesemantics.com/ontologies/2018/06/Export#> .
@prefix ann: <http://cambridgesemantics.com/annotations/> .

#Mode:ADD

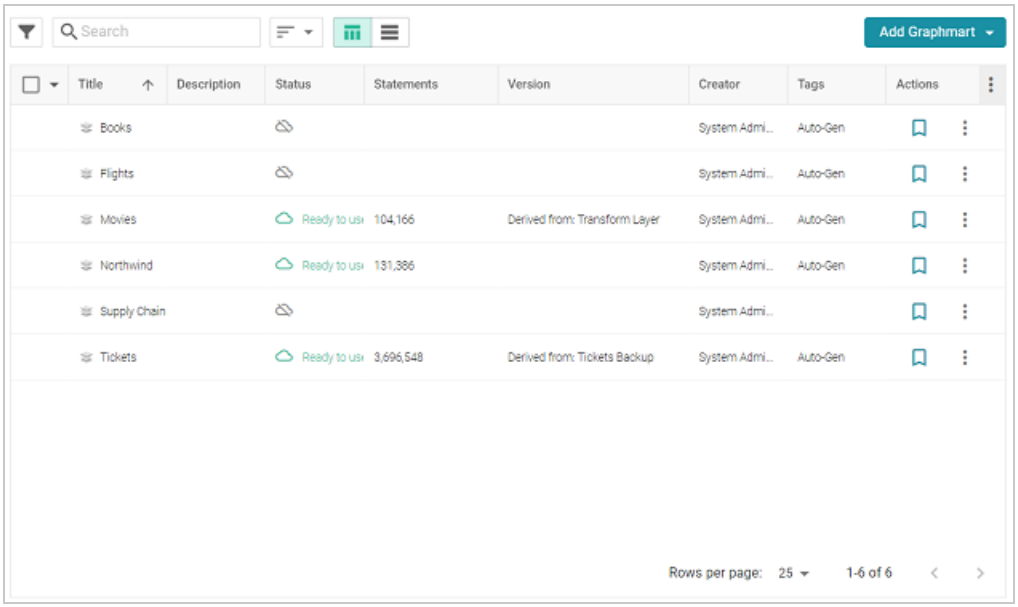
ann:replaceStatements {
  ds:PathConnection exp:replaceStatement ds:filePath .
  ds:FileConnection exp:replaceStatement ds:fileConnectionBaseFolder .
  ds:DbDataSource exp:replaceStatement ds:dbUser , ds:dbDatabase, ds:dbPassword .
}
```

Importing Exported Versions of Artifacts

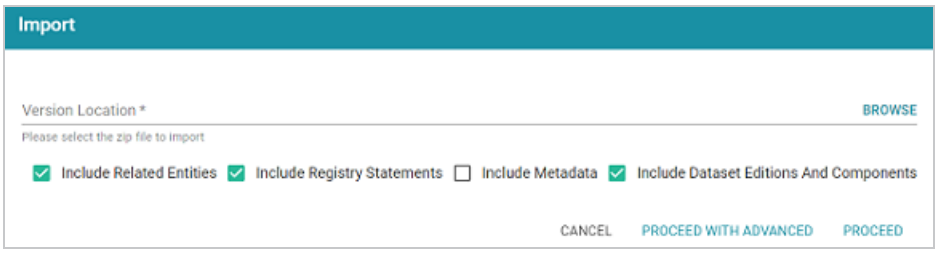
Follow the instructions below to import an exported version of an artifact (as described in Exporting an Artifact) and its related entities.

> **Note**
> To import a model that was created outside of Anzo or was downloaded from Anzo (as described in Downloading a Model), see Uploading a Model.

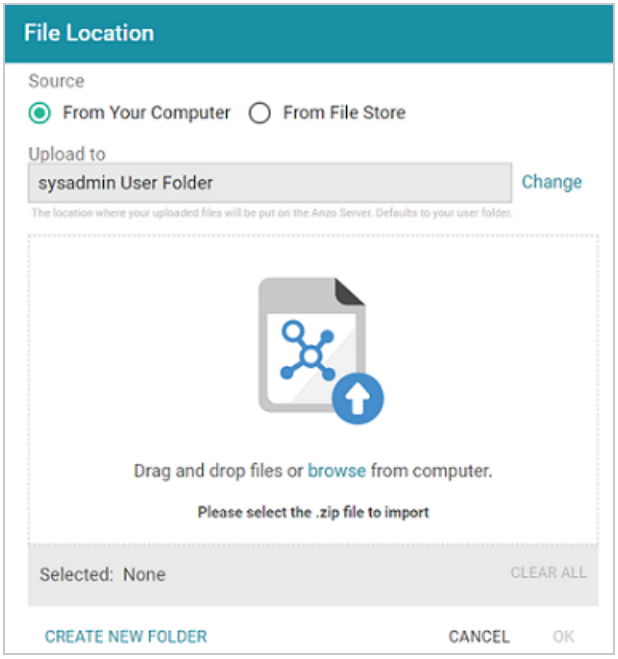
1. In the Anzo application, go to the resource selection screen for the type of artifact that you want to import. For example, the image below shows the Graphmarts screen:



2. Click the **Add ...** button on the top of the screen and select **Import ...**. Anzo opens the Import dialog box.



3. On the Import screen, click the **Version Location** field to open the File Location dialog box.

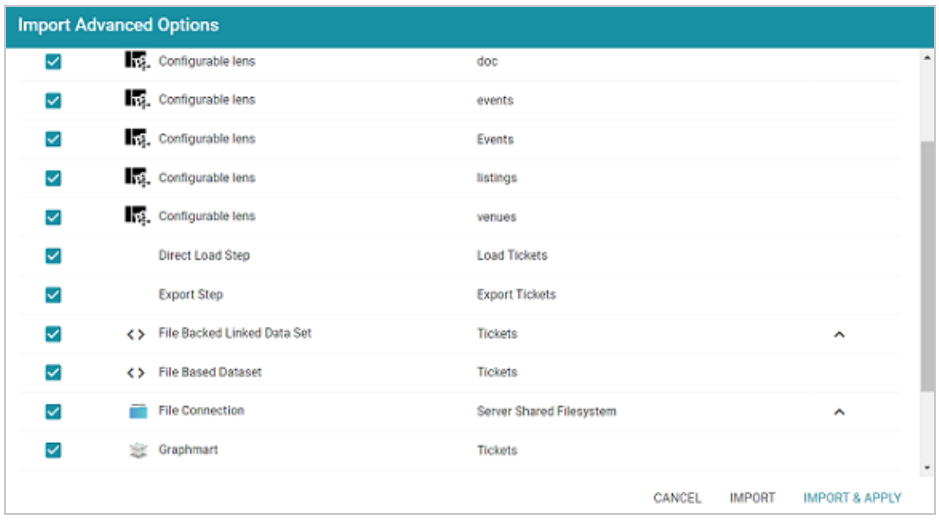


4. If the exported .zip file is on your computer, drag and drop the file onto the screen or click **browse** to navigate to the file and select it. If the .zip file is on a File Store, select the **From File Store** radio button and select the file on the File Store.
5. Click **OK** to save the file location value and close the File Location dialog box.
6. Enable or disable the following options as needed, depending on the data that the import file contains:
   - **Include Related Entities**: Indicates whether to import the artifact's related entities. Since most artifacts have dependencies with other artifacts, the **Include Related Entities** option is selected by default when an artifact is exported. Capturing all related entities on export ensures that all of an artifact's dependencies are included when that artifact is migrated. For example, an exported pipeline has the data source, schema, and model artifacts that it relies on when the pipeline is run. If the exported package includes related entities, Cambridge Semantics recommends that you enable **Include Related Entities** on import.
   - **Include Registry Statements**: Indicates whether to import the registry statements for the artifact and its related entities. This option is selected by default. Registry statements

should be included in imports. When registry statements are not included, the imported artifacts are not displayed in Anzo. For example, if a data source artifact is imported without registry statements, it would not be added to the Data Sources Registry and therefore not be displayed in the list of data sources in the Anzo application.

- **Include Metadata**: Indicates whether to import the metadata graph for the artifact and its related entities, such as the access control configuration and last modified date. If you select **Include Metadata**, you have the option to edit the permission configuration before importing the artifact.
- **Include Dataset Editions And Components**: This option specifies whether to import all of the editions and components for each dataset included in the import package.

7. Choose one of the following options to proceed with the import:
   - If you want to import the files as alternate versions of artifacts and not as the current, working version, and you do not want to replace any values or change permissions, click **Proceed**. Anzo imports the data and the imported files become available as versions on the relevant Version screens for the imported artifacts.
   - If you want to import these files as the current working version, and/or you want to change values or modify the permissions, click **Proceed With Advanced**. Anzo opens the Import Advanced Options dialog box. For example:



Click the ^ character to the right of an entity name to expand the options and view the editable properties. Replace any of the existing values with the new values that you want to define for the imported version of the entity. If you specified **Include Metadata** and want modify permission settings for the import, click the **Sharing** tab and edit or add permissions for users and groups. For details about the Sharing tab, see Share Access to Artifacts.

When you are ready to import the entities, choose one of the following options:

- If you want to import the files as alternate versions and not as the current, working version, click **Import**. Anzo imports the files and the entities become available as versions on the relevant Version screens.
- If you want to import the files so that they become the current, working versions of the artifacts, click **Import & Apply**. Anzo creates a backup version of the existing working versions and then imports the artifacts as the new working versions.

SPARQL Best Practices and Query Templates

To provide guidance on developing performant SPARQL queries and avoiding unexpected results, this section offers SPARQL best practices and query templates that you can use as a starting point for writing SPARQL queries in Anzo, such as in data layers, dashboard query lenses, and the Query Builder.

In this section:

SPARQL Best Practices 997

SPARQL Query Templates 1004

SPARQL Best Practices

This topic provides some background information on query execution and gives general guidelines to follow to help ensure that your SPARQL queries are optimized and do not exhaust Anzo or AnzoGraph resources.

- Query Execution Overview
- Clause Execution Order and Optimization
- General Guidelines

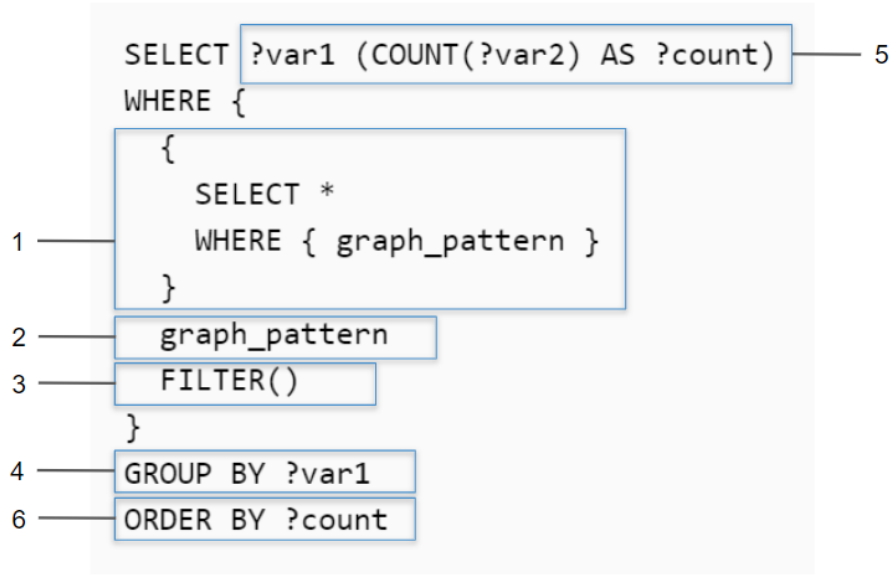
Query Execution Overview

When you run a new query against a graphmart, AnzoGraph parses the text and then plans the query execution strategy. The planner determines the steps that are required to compute the results in an optimal way. The plan is based on the statistics about your data that is gathered during graphmart activation. After the plan is determined, code for the plan is generated, compiled, and then distributed throughout the cluster to be executed. The plan XML file and compiled code are both saved to disk so that they can be reused when the query is run again. That is why the first run of a query takes longer than subsequent runs.

In addition to the caching that AnzoGraph performs, Anzo caches the query results. If you enable the **Skip server cache** option when running a query in the Query Builder, it means that Anzo will not access the cache that is available from a previous run of the query.

Clause Execution Order and Optimization

The following image shows the execution order of clauses in a read (SELECT or CONSTRUCT) query. The list below describes the order.



1. First, any subselects or subqueries are performed. Note that the ordering presented in the image also applies to the clauses included in subqueries.
2. Then the graph patterns that are not included in a subquery are scanned.
3. Next, filtering is applied if a FILTER clause is included.
4. Once the WHERE clause has been processed, grouping is performed if a GROUP BY clause is included.
5. Next, the result clause is processed, including keywords like DISTINCT and any aggregation or other functions.
6. Lastly, the results are ordered when ORDER BY is specified. The ORDER BY clause is more complex when GROUP BY and LIMIT are included: `Nlog(M)` where N is the number of grouped solutions and M is the LIMIT.

The GROUP BY and result clauses (steps 4 and 5) add linear complexity on top of the complexity in the WHERE clause. Query execution time increases proportionally to the total number of solutions found by the WHERE clause. To reduce complexity, trim the WHERE clause to have as few solutions as possible so that the expensive clauses are executed against as few solutions as possible.

General Guidelines

This section offers guidance on developing performant queries and avoiding unexpected results.

- Limit the Results
- Replace FILTER with VALUES or Triple Patterns
- Avoid Cross-Product Joins
- Use Subqueries for Large Amounts of Data

Limit the Results

The easiest way to reduce query execution time in some cases is to apply a LIMIT statement to limit the result set to a specific number of solutions. Limiting the number of results improves performance for cases where query results are calculated and returned in a streaming fashion. Limiting results is particularly useful when results need to be ordered so that the first group of results are the only ones of interest.

Example: Find the sample ID and binding density for the top 10 most dense samples

```
PREFIX biom: <http://identifiers.csi.com/pharmakg/def/biomarker#>

SELECT ?sampleId ?bindingDensity
WHERE {
  ?sample a biom:Sample ;
      biom:sampleId ?sampleId ;
      biom:bindingDensity ?bindingDensity .
}
ORDER BY DESC(?bindingDensity)
LIMIT 10
```

Replace FILTER with VALUES or Triple Patterns

While a FILTER clause is useful for narrowing down selected data per a set of requirements, only use FILTER when the logic does not lend to other operations. In many cases, replacing FILTER with a VALUES clause or a well-organized set of triple patterns increases query performance. When

processing a FILTER statement, all non-filtered data must be retrieved before the FILTER can be applied. Using a VALUES clause or triple pattern, however, reduces the amount of data that is retrieved and processed after the retrieval.

Example 1: Inappropriate use of FILTER for a value-driven SELECT query

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?var1 ?var2
WHERE {
  ?instanceOfClass a uriRoot:Class ;
      uriRoot:varName1 ?var1 ;
      uriRoot:varName2 ?var2 ;
      uriRoot:filteredVar ?filteredVar .
  FILTER(?filteredVar = 'COMPAREDDATA1' || ?filteredVar = 'COMPAREDDATA2' ||
?filteredVar = 'COMPAREDDATA3')
  # filteredVar is first retrieved and then run through several comparisons
}
```

Solution 1: Use VALUES to select certain values

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?var1 ?var2
WHERE {
  ?instanceOfClass a uriRoot:Class ;
      uriRoot:varName1 ?var1 ;
      uriRoot:varName2 ?var2 ;
      uriRoot:filteredVar ?valueVar .

  VALUES (?valueVar) {
      ('COMPAREDDATA1')
      ('COMPAREDDATA2')
      ('COMPAREDDATA3')
  }
  # selection is performed once for each entry in the VALUES clause,
  # retrieving no more data than necessary
}
```

Example 2: Inappropriate use of FILTER for a value-driven SELECT query

```
PREFIX uriRoot: <http://example.com/rootOfUris#>
```

```
SELECT ?var1 ?filteredVar
WHERE {
  ?instanceOfClass a uriRoot:ClassName ;
      uriRoot:varName1 ?var1 ;
      uriRoot:varName2 ?var2 ;
      uriRoot:filteredVar ?filteredVar .
  FILTER(?filteredVar = 'COMPAREDDATA1')
  # filteredVar is first retrieved and then compared
}
```

Solution 2: Use a triple literal to select certain values

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?var1 ?filteredVar
WHERE {
  ?instanceOfClass a uriRoot:Class ;
      uriRoot:varName1 ?var1 ;
      uriRoot:filteredVar 'COMPAREDDATA' .
      # data is only retrieved if filteredVar matches desired compared data upon
initial retrieval
}
```

Avoid Cross-Product Joins

When trying to gather data from multiple classes at once, it is possible to accidentally create a cross-product join, a selection that combines the selected data in a hyper-linear way rather than simply assembling the data and returning an unprocessed set.

Example: Accidental cross-product query

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?var1 ?var2
WHERE {
  ?instanceOfClass1 a uriRoot:Class1 ;
      uriRoot:varName1 ?var1 .
  ?instanceOfClass2 a uriRoot:Class2 ;
      uriRoot:varName2 ?var2 .
}
```

In the above example, the goal may have been to retrieve IDs from all instances of Class1 and all instances of Class2, for example, all of the participants and all of the subjects. However, the result of the query would be every combination of participant and subject. If there are 10 participants and 5 subjects, there would be 50 results rather than 15. In large data sets, this severely affects performance and puts the system under unnecessary strain.

There are two straightforward ways to separate or parameterize data to write a more performant query:

Solution 1: Use UNION to replace the cross-product

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?commonVar
WHERE {
  {
    ?instanceOfClass1 a uriRoot:Class1 ;
        uriRoot:varName1 ?var1 .
    BIND(?var1 as ?commonVar)
  }
  UNION

{
    ?instanceOfClass2 a uriRoot:Class2 ;
        uriRoot:varName2 ?var2 .
    BIND(?var2 as ?commonVar)
  }
  # this creates a temporary graph that is a union of two graphs
  # in each of the two graphs, the desired data is saved under the same name
}
```

Solution 2: Use VALUES to replace the cross-product

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?commonVar
WHERE {
  ?instanceOfClass a ?classURI ;
      ?propertyURI ?commonVar .

  VALUES (?classURI ?propertyURI) {
      (uriRoot:Class1 uriRoot:varName1)
```

```
      (uriRoot:Class2 uriRoot:varName2)
  }
}
```

Use Subqueries for Large Amounts of Data

When analyzing data, there may be a need to aggregate data and then perform a selection or derivation on the resulting aggregate. In this case, it is advisable to use one or more subselects or subqueries, where a SELECT query is included inside the WHERE clause and the remainder of the WHERE clause operates on the results of that SELECT as though the data were immediately available in the graph.

Example: Aggregate a variable and then process the aggregation

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

SELECT ?var1 ?var2Aggregation
WHERE {
  {
    SELECT ?var1 (GROUP_CONCAT(?var2) as ?var2Aggregation)
    WHERE {
        ?instanceOfClass1 a uriRoot:Class1 ;
            uriRoot:varName1 ?var1 .
        ?instanceOfClass2 a uriRoot:Class2 ;
            uriRoot:varName2 ?var2 .
    }
    GROUP BY ?var1
  }
  # var1 and var2Aggregation are now available for the usual processing
  # while var2 is no longer available as it only existed within the subselect

  FILTER(regex(?var2Aggregation, 'DESIREDVAR2VAL'))
  # FILTER is used for illustrative purposes, but any processing would work
}
```

SPARQL Query Templates

This topic provides templates that you can use as a starting point for writing SPARQL queries. The templates are based on the best practices described in SPARQL Best Practices.

- Basic Data Selection
- Graph Traversal Data Selection
- Text Cleanup with REGEX
- Data Aggregation
- Apply a Filter to Selected Data
- Create or Derive New Variables

Basic Data Selection

The most fundamental use case for writing SPARQL queries is to select data from properties from a collection of instances. The following template and example query illustrate how to access a class in a model and return the properties on that class using their URIs.

Abstracted Query Template

Replace the bold text to modify the query.

```
PREFIX uriRoot: <http://example.com/rootOfUris#>

# select the variables that are populated in the WHERE clause
SELECT ?var1 ?var2
WHERE {
  ?instanceOfClass a uriRoot:ClassName ;
      uriRoot:varName1 ?var1 ;
      # use a prefix to abbreviate a property URI as shown above
      # or use the full URI as shown below
      <http://example.com/rootOfUris#varName2> ?var2 .
}
```

Example: Get the sample ID and anatomical location for each sample

```
PREFIX bm: <http://identifiers.csi.com/pharmakg/def/biomarker#>
```

```
SELECT ?sampleId ?anatomicalLocation
WHERE {
  ?sample a bm:Sample ;
      bm:sampleId ?sampleId ;
      <http://identifiers.csi.com/pharmakg/def/biomarker#fmi_anatomicalLocation>
?anatomicalLocation .
}
```

Graph Traversal Data Selection

The graph model enables the flexibility to combine data from different classes. The following template illustrates how to traverse between classes in the data model and access data from properties on multiple classes.

Abstracted Query Template

Replace the bold text to modify the query.

```
PREFIX uriRoot: <http://example.com/rootOfUris#>
# select the variables that are populated in the WHERE clause
SELECT ?var1 ?var2 ?varFromOtherClass
WHERE {
  ?instanceOfClass a uriRoot:ClassName ;
      uriRoot:varName1 ?var1 ;
      # use a prefix to abbreviate a property URI as shown above
      # or use the full URI as shown below
      <http://example.com/rootOfUris#varName2> ?var2 ;
      # getting data from other classes requires traversing per the model
      uriRoot:pointerToOtherClass ?instanceOfOtherClass .

  ?instanceOfOtherClass a uriRoot:OtherClassName ;
      uriRoot:varName3 ?varFromOtherClass .
}
```

Text Cleanup with REGEX

Once data is onboarded to Anzo, it is common to encounter strings that include issues such as unintended characters, missing spaces, and inconsistent formatting. You can use regular expressions in a data layer query to manipulate those values so that they are consistent and readable in analytics against the graphmart.

The BIND clause in the query below trims any white space from before and after the string, converts the characters to upper case, and removes all non-alphanumeric characters and non-spaces. Replace the bold text as needed:

```
PREFIX : <http://csi.com/>
DELETE {
  GRAPH ${targetGraph}{
    ?s ?pred ?old_val
  }
}
INSERT {
  GRAPH ${targetGraph}{
   ?s ?pred ?new_val
  }
}
${usingSources}
WHERE {
  ?s a :Class ;
    ?pred ?old_val .

  VALUES (?pred) {
    (:property)
  }
BIND(TRIM(UPPER(REPLACE(?val, "[^a-zA-Z0-9[[:space:]]]", ""))) as ?new_val)
}
```

Data Aggregation

Grouping data selections around a central property yields a more complete representation or summary of the data available. The following template illustrates how to use one property to act as a pivot point for collecting all the data from another property.

Abstracted Query Template

Replace the bold text to modify the query

```
PREFIX pref: <http://example.com/rootOfUris#>

SELECT
# data can be aggregated to yield counts, concatenations of data, etc.
  ?instanceId GROUP_CONCAT(DISTINCT(?instanceDetail) as ?instanceDetails)
WHERE {
```

```
  # apply selection/filtering logic to narrow the aggregation
  # or get summaries of total data by applying only simple restrictions
  ?instance a pref:Class ;
      pref:instanceId ?instanceId ;
      pref:instanceDetail ?instanceDetail .
}
GROUP BY ?instanceId
# all non-aggregated variables must be grouped in GROUP BY
```

Apply a Filter to Selected Data

Filtering the results for a query gives the ability to focus on specific aspects of the data. The following template illustrates how to restrict the total selected result set by including a filter on a variable.

Abstracted Query Template

Replace the bold text to modify the query.

```
PREFIX pref1: <http://example.com/rootOfUris1#>
PREFIX pref2: <http://example.com/rootOfUris2#>

SELECT ?varFromClass1 ?varFromClass2 ?varFromClass3 ?filteredVar
WHERE {
  ?instance1 a pref1:Class1 ;
      pref1:varName1 ?varFromClass1 ;
      # the path on the model points from Class1 to Class2
      pref1:pointerToClass2 ?instance2 .

  ?instance2 a pref1:Class2 ;
      pref1:varName2 ?varFromClass2 .

  # models with different prefixes can still be joined
  ?instance3 a pref2:Class3 ;
      # the path on the model points from Class3 to Class2
      pref2:pointerToClass2 ?instance2 ;
      pref2:filteredVarName ?filteredVar .

  # filters use comparisons to scope the selected data
  # they can use existence checks or other boolean expressions as well
  FILTER(?filteredVar = 'COMPAREDDATA')
}
```

Tip

For optimal query performance, replace FILTER clauses. See Replace FILTER with VALUES or Triple Patterns for more information.

Create or Derive New Variables

Storing intermediate or derived data within a query enables a single query to answer more complex questions. The following template illustrates how to bind a derived value to a variable. That variable is then available for selection or further manipulation.

Abstracted Query Template

Replace the bold text to modify the query.

```
PREFIX pref1: <http://example.com/rootOfUris1#>
PREFIX pref2: <http://example.com/rootOfUris2#>
PREFIX pref3: <http://example.com/rootOfUris3#>

SELECT ?var1 ?filterVar ?var2AndVar3
WHERE {
  ?instance1 a pref1:Class1 ;
      pref1:varName1 ?var1 .

  ?filterInstance a pref2:MedicalHistory ;
      pref2:filterVarName ?filterVar ;
      # multiple traversals between classes may be necessary to link appropriate data
      pref2:pointerToIntermediateClass ?intermediateInstance .

  ?intermediateInstance a pref2:IntermediateClass ;
      pref2:pointerToClass1 ?instance1 .

  ?instance2 a pref3:Class2 ;
      # forwards traversals tend to be more performant
      # it is still possible to identify a latter class and do a backwards traversal
      pref3:pointerToClass1 ?instance1 ;
      pref3:varName2 ?var2 .

  ?instance3 a pref3:Class3 ;
      pref3:pointerToClass2 ?instance2 ;
      pref3:varName3 ?var3 .
```

```
  # filters can be executed on various data types
  FILTER(?filterVar < "filterData"^^xsd:filterDataType)

  # binding allows population of new/derived variables
  BIND(CONCAT(?var2, "--", ?var3) as ?var2AndVar3)
}
```

Function and Formula Reference

This section describes the standard and advanced built-in functions that are available when working with Hi-Res Analytics dashboards or writing SPARQL queries in data layers or the Query Builder.

In this section:

String Functions 1011

Math Functions 1040

Aggregate Functions 1071

Date and Time Functions 1092

Casting Functions 1113

Logical Functions 1131

Informational or Testing Functions 1142

Hash Functions 1154

Window Aggregate and Ranking Functions 1158

String Functions

This topic describes the Anzo functions that operate on string data types.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- BUSINESS_ENTITY_EXCLUDER: Removes suffixes that represent business entities.
- CONCATENATE: Concatenates two or more strings and returns the result as a string.
- CONCATURL: Concatenates two or more strings and returns the result as a URI.
- CONTAINS: Evaluates whether the specified string contains the given pattern.
- ENCODE_FOR_URI: Encodes the specified string as a URI.
- ESCAPEHTML: Escapes the specified string for use in HTML.
- FIND: Returns the position—from left to right—of a string within another string.
- FINDREVERSE: Returns the position—from right to left—of a string within another string.
- GROUP_CONCAT: Concatenates a group of strings into a single string.
- GROUPCONCAT: Concatenates a group of strings into a single string. This function is a customizable version of GROUP_CONCAT.
- LANG: Returns any language tags that are included with strings.
- LANGMATCHES: Evaluates whether a string includes a language tag that matches the specified language range.
- LCASE: Converts the letters in a string literal to lower case.

- LEFT: Returns the specified number of characters starting from the beginning (left side) of the string.
- LEN: Calculates the length (number of characters) in a string.
- LEVENSHTEIN_DIST: Calculates the Levenshtein distance or measure of similarity between two strings.
- LOWER: Converts all letters in a string to lower case.
- MD5: Returns the MD5 checksum of a string as a hexadecimal string.
- MID: Returns the specified number of characters from a string, starting from a given position in the string.
- REGEX: Evaluates whether a string matches the specified regular expression pattern.
- REGEXP_SUBSTR: Searches a string for the specified regular expression pattern and returns the substring that matches the pattern.
- REPLACE: Extends the REGEX function to provide the ability to find a pattern in a string and replace it with another pattern.
- RIGHT: Returns the specified number of characters starting from the end (right side) of the string.
- SEARCH: Uses text search semantics to evaluate whether the specified string matches the given pattern.
- SHA1: Calculates the SHA-1 digest of a string value.
- SHA224: Calculates the SHA-224 digest of a string value.
- SHA256: Calculates the SHA-256 digest of a string value.
- SHA384: Calculates the SHA-384 digest of a string value.
- SHA512: Calculates the SHA-512 digest of a string value.
- STRAFTER: Returns the portion of a string that comes after the specified substring.
- STRBEFORE: Returns the portion of a string that comes before the specified substring.

- STRDT: Constructs a literal value with the specified data type.
- STRENDS: Evaluates whether the specified string ends with the specified substring.
- STRLANG: Constructs a literal value with the specified language tag.
- STRLEN: Calculates the length of a string.
- STRSTARTS: Evaluates whether the specified string starts with the specified substring.
- STRUUID: Returns a string that is the result of generating a Universally Unique Identifier (UUID).
- SUBSTITUTE: Substitutes the existing text for the specified new text.
- SUBSTR: Returns a substring from a string value.
- TOURI: Casts a string to a URI.
- TRIM: Removes all spaces from a string except for any single spaces between words.
- UCASE: Converts all letters in a string to upper case.
- UPPER: Converts the letters in a string literal to upper case.

BUSINESS_ENTITY_EXCLUDER

This function removes from strings the suffixes that represent business entities.

Syntax

```
BUSINESS_ENTITY_EXCLUDER(text)
```

Argument	Type	Description
text	string	The string from which you want to remove business entities.

Returns

Data Type	Description
string	The string without the business entity suffix.

CONCATENATE

This function concatenates two or more strings and returns the result as a string.

Syntax

```
CONCATENATE(text1, text2 [, textN ])
```

Argument	Type	Description
text1–N	string	The strings that you want to concatenate to form a single string.

Returns

Type	Description
string	The concatenated string.

CONCATURL

This function concatenates two or more strings and returns the result as a URI.

Syntax

```
CONCATURL(text1, text2 [, textN ])
```

Argument	Type	Description
text1–N	string	The strings that you want to concatenate to form a URI.

Returns

Type	Description
URI	The concatenated string as a URI.

CONTAINS

This function evaluates whether the specified strings contain the given pattern. Results are grouped under "true" or "false."

Syntax

```
CONTAINS(text, pattern)
```

Argument	Type	Description
text	string	The string value that you want to check against the specified pattern.
pattern	string	The string pattern that you want to look for in the supplied text.

Returns

Type	Description
boolean	`True` if the strings contain the pattern and `false` if they do not.

ENCODE_FOR_URI

This function encodes the specified string as a URI and returns a string in URI format.

Syntax

```
ENCODE_FOR_URI(text)
```

Argument	Type	Description
text	string	The string value to encode as a URI.

Returns

Type	Description
string	The string as a URI.

ESCAPEHTML

This function escapes the specified string for use in HTML.

Syntax

```
ESCAPEHTML(text)
```

Argument	Type	Description
text	string	The string value to escape for HTML.

Returns

Type	Description
string	The string escaped for HTML.

FIND

This function returns the position—from left to right—of a string within another string.

> **Tip**
> You can use FINDREVERSE to find the character or substring position from right to left.

Syntax

```
FIND(find_text, within_text, start_num)
```

Argument	Type	Description
find_text	string	The string to look for in the `within_text`.
within_text	string	The string to search within.
start_num	int	An integer that indicates the position to start from when looking for the `find_text`. The starting position is at the beginning of the `within_`

Argument	Type	Description
		`text` value and characters are counted from left to right.

Returns

Type	Description
int	The character position (from left to right) where the substring starts.

FINDREVERSE

Similar to FIND, this function returns the position—from right to left—of a string within another string.

Syntax

```
FINDREVERSE(find_text, within_text, start_num)
```

Argument	Type	Description
find_text	string	The string to look for in the `within_text` value.
within_text	string	The string to search within.
start_num	int	An integer that indicates the position to start from when looking for the `find_text`. The starting position is the end of the `within_text` value and characters are counted from right to left.

Returns

Type	Description
int	The character position (from right to left) where the substring starts.

GROUP_CONCAT

This function concatenates a group of strings into a single string. It is a simplified version of GROUPCONCAT as it takes only one argument.

Syntax

```
GROUP_CONCAT(text)
```

Argument	Type	Description
text	string	The string property whose values to concatenate into a single string.

Returns

Type	Description
string	The concatenated string.

GROUPCONCAT

This function concatenates a group of strings into a single string. Unlike GROUP_CONCAT, this function allows for customization of the separator to use as well as the configuration of limits and options like prefixes and suffixes.

Syntax

```
GROUPCONCAT(group1, [ group2, ..., groupN, ] group_value_separator, separator,
serialize,
            row_limit, value_limit, delimit_blanks [, prefix ] [, suffix ] [, max_
length ])
```

Argument	Type	Description
group1–N	string	The group(s) of strings to concatenate.
group_value_separator	string	The separator string to use between the groups of strings if you specified more than one `group`.

Argument	Type	Description
separator	string	The separator string to use between the values in a concatenated group of strings.
serialize	boolean	A boolean value that indicates whether returned values should be serialized with the value's data type.
row_limit	int	An integer that puts a maximum limit on the number of rows to retrieve for a group.
value_limit	int	An integer that puts a maximum limit on the number of values to retrieve from a group of rows.
delimit_blanks	boolean	A boolean value that indicates whether to delimit blanks with the `separator` value.
prefix	string	Optional string to add as a prefix to the resulting string.
suffix	string	Optional string to add as a suffix to the resulting string.
max_length	int	Optional integer that puts a maximum limit on the number of characters the resulting string can have.

Returns

Type	Description
string	The concatenated string.

LANG

This function returns any language tags that are included in the string. The results are grouped by each language tag or by "blank" if a value does not have a language tag.

Syntax

```
LANG(text)
```

Argument	Type	Description
text	string	The string to search for language tags.

Returns

Type	Description
string	The found language tags.

LANGMATCHES

This function tests whether a string includes a language tag that matches the specified language range.

Syntax

```
LANGMATCHES(text, language_range)
```

Argument	Type	Description
text	string	The string to evaluate.
language_range	string	The language tag to match in the `text`.

Example

```
LANGMATCHES(LANG(?prop),"en")
```

Returns

Type	Description
boolean	`True` if strings include a language tag that matches the range and `false` if they do

Type	Description
	not.

LCASE

This function converts the letters in a string literal to lower case.

Syntax

```
LCASE(text)
```

Argument	Type	Description
text	string	The string literal to convert to lower case.

Returns

Type	Description
string	The string with lower case letters.

LEFT

This function returns the specified number of characters starting from the beginning (left side) of the string.

Syntax

```
LEFT(text, num_chars)
```

Argument	Type	Description
text	string	The string from which to return the specified number of characters.
num_chars	int	An integer that specifies the number of characters to return, starting from the left side of the `text`.

Returns

Type	Description
string	The specified number of characters from the string.

LEN

This function calculates the length (number of characters) in a string.

Syntax

```
LEN(text)
```

Argument	Type	Description
text	string	The string for which to calculate the length.

Returns

Type	Description
int	The number of characters in the string.

LEVENSHTEIN_DIST

This function calculates the Levenshtein distance or measure of similarity between two strings. The distance is the number of edits required to transform the first string into the second string.

Syntax

```
LEVENSHTEIN_DIST(text1, text2)
```

Argument	Type	Description
text1	string	The string that would be transformed into `text2`.
text2	string	The string to measure `text1` against.

Returns

Type	Description
int	The Levenshtein distance between the strings.

LOWER

This function converts all letters in a string to lower case.

Syntax

```
LOWER(text)
```

Argument	Type	Description
text	string	The string to convert to lower case.

Returns

Type	Description
string	The string with lower case letters.

MD5

This function returns the MD5 checksum of a string as a hexadecimal string.

Syntax

```
MD5(text)
```

Argument	Type	Description
text	string	The string for which to return the MD5 checksum.

Returns

Type	Description
string	The hexadecimal string.

MID

This function returns the specified number of characters from a string, starting from a given position in the string.

Syntax

```
MID(text, start_num, num_chars)
```

Argument	Type	Description
text	string	The string from which to return the specified characters.
start_num	int	An integer that indicates the starting position in the string.
num_chars	int	An integer that specifies the number of characters to return, starting with the character indicated by `start_num`.

Returns

Type	Description
string	The specified number of characters from the string.

REGEX

This function tests whether a string matches the specified regular expression pattern.

Syntax

```
REGEX(text, pattern [, flags ])
```

Argument	Type	Description
text	string	The string to test against the `pattern`.
pattern	string	The regular expression pattern to look for in the `text`. For information about the supported regular expression syntax, see the Regular Expression Syntax section of the W3C XQuery 1.0 and XPath 2.0 Functions and Operators specification.
flags	string	You can include one or more optional modifier flags that further define the pattern. For information about flags, see the Flags section of the W3C Functions and Operators specification.

Returns

Type	Description
boolean	`True` if the string matches the regular expression pattern and `false` if it does not.

REGEXP_SUBSTR

This function searches a string for the specified regular expression pattern and returns the substring that matches the pattern.

Syntax

```
REGEXP_SUBSTR(text, pattern [, start_position ] [, nth_appearance ])
```

Argument	Type	Description
text	string	The string to test against the `pattern`.
pattern	string	The regular expression pattern to look for in the `text`. For information about the supported regular expression syntax, see the Regular Expression Syntax section of the W3C XQuery 1.0 and XPath 2.0 Functions and Operators specification.

Argument	Type	Description
start_position	int	An optional integer that specifies the number of characters from the beginning of the string to start searching for matches (the default value is **1**).
nth_ appearance	int	An optional integer that specifies which occurrence of the pattern to match (the default value is **1**).

Returns

Type	Description
string	The substring that matches the regular expression pattern.

REPLACE

This function extends the REGEX function to provide the ability to find a pattern in a string and replace it with another pattern. The function returns the replaced string.

Syntax

```
REPLACE(text, pattern, replacement_pattern [, flags ])
```

Argument	Type	Description
text	string	The string to test against the `pattern`.
pattern	string	The regular expression pattern to look for in the `text`. For information about the supported regular expression syntax, see the Regular Expression Syntax section of the W3C XQuery 1.0 and XPath 2.0 Functions and Operators specification.
replacement_ pattern	string	The pattern to replace the `pattern` with.

Argument	Type	Description
flags	string	You can include one or more optional modifier flags that further define the pattern. For information about flags, see the Flags section of the W3C Functions and Operators specification.

Returns

Type	Description
string	The string that contains the replacement pattern.

RIGHT

This function returns the specified number of characters starting from the end (right side) of the string.

Syntax

```
RIGHT(text, num_chars)
```

Argument	Type	Description
text	string	The string from which to return the specified number of characters.
num_chars	int	An integer that specifies the number of characters to return, starting from the right side of the `text`.

Returns

Type	Description
string	The specified characters from the string.

SEARCH

This function uses text search semantics to evaluate whether the specified string matches the given pattern.

Syntax

```
SEARCH(text, pattern [, required ] [, wildcard ] [, escape ])
```

Argument	Type	Description
text	string	The string to search.
pattern	string	The search string to look for in the `text`. Anzo automatically converts the value to a regular expression pattern that uses text search semantics.
required	boolean	An optional boolean value that indicates whether the `text` must include all elements of the search pattern to qualify as a match or whether matching just part of the pattern qualifies as a match.
wildcard	boolean	An optional boolean value that indicates whether or not to add the wildcard character `*` to the end of the search pattern.
escape	boolean	An optional boolean value that indicates whether or not escape all of the special characters (such as `+`, `-`, or `|`) in the `text`.

Returns

Type	Description
boolean	`True` if strings match the pattern and `false` if they do not.

SHA1

This function calculates the SHA-1 digest of a string.

Syntax

```
SHA1(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-1 digest.

Returns

Type	Description
string	The SHA-1 digest.

SHA224

This function calculates the SHA-224 digest of a string.

Syntax

```
SHA224(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-224 digest.

Returns

Type	Description
string	The SHA-224 digest.

SHA256

This function calculates the SHA-256 digest of a string.

Syntax

```
SHA256(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-256 digest.

Returns

Type	Description
string	The SHA-256 digest.

SHA384

This function calculates the SHA-384 digest of a string.

Syntax

```
SHA384(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-384 digest.

Returns

Type	Description
string	The SHA-384 digest.

SHA512

This function calculates the SHA-512 digest of a string.

Syntax

```
SHA512(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-512 digest.

Returns

Type	Description
string	The SHA-512 digest.

STRAFTER

This function returns the portion of a string that comes after the specified substring.

Syntax

```
STRAFTER(text, substring)
```

Argument	Type	Description
text	string	The string from which to return the characters that follow the `substring`.
substring	string	The string to match in the `text`. The function will return the part of the text that comes after this substring.

Returns

Type	Description
string	The part of the string that comes after the substring.

STRBEFORE

This function returns the portion of a string that comes before the specified substring.

Syntax

```
STRAFTER(text, substring)
```

Argument	Type	Description
text	string	The string from which to return the characters that precede the `substring`.
substring	string	The string to match in the `text`. The function will return the part of the text that comes before this substring.

Returns

Type	Description
string	The part of the string that comes before the substring.

STRDT

This function constructs a literal value with the specified data type.

Syntax

```
STRDT(text, datatype)
```

Argument	Type	Description
text	string	The string to add a data type specification to.
datatype	URI	The data type URI to add to the `text`. For example, `xsd:integer` or `<http://www.w3.org/2001/XMLSchema#integer>`.

Returns

Type	Description
string	The typed literal value.

STRENDS

This function evaluates whether the specified string ends with the specified substring.

Syntax

```
STRENDS(text, substring)
```

Argument	Type	Description
text	string	The string to search for the `substring`.
substring	string	The string to match at the end of `text`. The function returns true if the text ends in the specified substring and false if it does not.

Returns

Type	Description
boolean	`True` if strings end with the specified substring and `false` if they do not.

STRLANG

This function constructs a literal value with the specified language tag.

Syntax

```
STRLANG(text, language_tag)
```

Argument	Type	Description
text	string	The string to add the language tag to.
language_tag	string	The language tag to add to the `text`.

Returns

Type	Description
string	The literal value with the language tag.

STRLEN

This function calculates the length (in characters) of a string value.

Syntax

```
STRLEN(text)
```

Argument	Type	Description
text	string	The string for which to return the length.

Returns

Type	Description
long	The number of characters in the string.

STRSTARTS

This function evaluates whether the specified string starts with the specified substring.

Syntax

```
STRENDS(text, substring)
```

Argument	Type	Description
text	string	The string to search for the `substring`.
substring	string	The string to match at the beginning of `text`. The function returns true if the text starts with the specified substring and false if it does not.

Returns

Type	Description
boolean	`True` if strings begin with the specified substring and `false` if they do not.

STRUUID

This function returns a string that is the result of generating a Universally Unique Identifier (UUID).

Syntax

```
STRUUID()
```

Returns

Type	Description
string	The UUID.

SUBSTITUTE

This function substitutes the existing text for the specified new text.

Syntax

```
SUBSTITUTE(text, old_text, new_text [, instance_num ])
```

Argument	Type	Description
text	string	The string to substitute text in.

Argument	Type	Description
old_text	string	The string within the `text` to replace.
new_text	string	The string to replace the `old_text` with.
instance_num	int	An optional integer that specifies the number of `old_text` instances to replace.

Returns

Type	Description
string	The string with the new text.

SUBSTR

This function returns a substring from a string value.

Syntax

```
SUBSTR(text, start [, length ])
```

Argument	Type	Description
text	string	The string to find the substring in.
start	int	An integer that specifies the number of the character in the `text` that should be the start of the substring.
length	int	An optional integer that specifies the total number of characters to include in the substring. If not specified, the substring will end at the end of the `text` value.

Returns

Type	Description
string	The substring.

TOURI

This function casts a string literal value to a URI.

Syntax

```
TOURI(text)
```

Argument	Type	Description
text	string	The string literal to cast to a URI.

Returns

Type	Description
URI	The literal value as a URI.

TRIM

This function removes all spaces from a string except for any single spaces between words.

Syntax

```
TRIM(text)
```

Argument	Type	Description
text	string	The string to trim.

Returns

Type	Description
string	The string with spaces removed.

UCASE

This function converts all letters in a string to upper case.

Syntax

```
UPPER(text)
```

Argument	Type	Description
text	string	The string value to convert to upper case.

Returns

Type	Description
string	The string with upper case characters.

UPPER

This function converts all letters in a string literal to upper case.

Syntax

```
UPPER(text)
```

Argument	Type	Description
text	string	The string literal to convert to upper case.

Returns

Type	Description
string	The string with upper case characters.

Math Functions

This topic describes the mathematical functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- ABS: Calculates the absolute value of the specified number.
- ADD: Adds two numeric values.
- AVERAGEIF: Calculates the average of the range of values that meet the specified criterion.
- AVERAGEIFS: Calculates the averages of the ranges of values that meet the specified criteria.
- AVG: Calculates the average (arithmetic mean) value for a group of numbers.
- CEILING: Rounds up a numeric value to the nearest integer.
- COS: Calculates the cosine of an angle.
- DIVIDE: Divides a number by another number.
- EQUAL: Evaluates whether two values are equal.
- EXP: Raises e to the specified power.
- FACT: Calculates the factorial of the specified number.
- FLOOR: Rounds down a numeric value to the nearest integer.
- GE: Evaluates whether one value is greater than or equal to (>=) another value.
- GT: Evaluates whether one value is greater than (>) another value.

- HAMMING_DIST: Calculates the hamming distance between two values.
- HAVERSINE_DIST: Computes the haversine distance between two latitude and longitude values.
- LE: Evaluates whether one value is less than or equal to (<=) another value.
- LN: Calculates the natural logarithm of a double value.
- LOG: Calculates the specified base logarithm of a double value.
- LOG2: Calculates the base two logarithm of a double value.
- LT: Evaluates whether one value is less than (<) another value.
- MAXVAL: Determines the maximum value from the given literal values.
- MINVAL: Determines the minimum value from the given literal values.
- MOD: Calculates the modulo of the division between two numbers.
- MULTIPLY: Multiplies two number values.
- NOT_EQUAL: Evaluates whether two values are not equal.
- NPV: Calculates the net present value of an investment.
- NUMERIC-ADD: Adds two numeric values.
- NUMERIC-SUBTRACT: Subtracts one numeric value from another numeric value.
- PI: Returns the value for PI.
- POWER: Raises the specified number to the specified power.
- PRODUCT: Calculates the product of a group of numbers.
- QUOTIENT: Calculates the quotient between two numbers.
- RAD: Converts to radians an angle value that is in degrees.
- RAND: Returns a random double value between 0 and 1.
- RANDBETWEEN: Returns a random integer that falls between two specified integers.
- ROUND: Rounds a numeric value to the nearest integer.

- ROUNDDOWN: Rounds a numeric value down to the specified number of digits.
- ROUNDUP: Rounds a numeric value up to the specified number of digits.
- SIN: Calculates the sine of an angle.
- SQRT: Calculates the square root of a number.
- SUBTRACT: Subtracts one RDF term from another RDF term type value.
- SUM: Calculates the sum of the numbers within a group.
- SUMIF: Calculates the sum of the range of values that meet the specified criterion.
- SUMIFS: Calculates the sums of the ranges of values that meet the specified criteria.
- SUMPRODUCT: Multiplies the numbers in a group and adds the results.
- SUMSQ: Calculates the square root of each number in a group and adds the results.
- TAN: Calculates the tangent of an angle.

ABS

This function calculates the absolute value of the specified number.

Syntax

```
ABS(number)
```

Argument	Type	Description
number	numeric	The numeric value for which to calculate the absolute value.

Returns

Type	Description
number	The absolute value.

ADD

This function adds two numeric values.

Syntax

```
ADD(value1, value2)
```

Argument	Type	Description
value1	numeric	The first numeric value to add.
value2	numeric	The second numeric value to add.

Returns

Type	Description
number	The result of the addition operation.

AVERAGEIF

This function calculates the average of the range of values that meet the specified criterion.

> **Tip**
> You can use AVERAGEIFS to specify multiple value ranges and conditions.

Syntax

```
AVERAGEIF(values_to_test, criterion [, range_of_values ])
```

Argument	Type	Description
values_to_test	RDF term	The literal, URI, or blank node value that defines the values to test against the `criteria`.
criterion	RDF term	The literal, URI, or blank node value that defines the condition to

Argument	Type	Description
		test values against.
range_of_values	numeric	An optional number that defines the range of values to average. When omitted, `values_to_test` is used.

Returns

Type	Description
number	The average value from the range of values that meet the criterion.

AVERAGEIFS

This function calculates the averages of the ranges of values that meet the specified criteria. Unlike AVERAGEIF, this function enables you to specify multiple ranges and multiple conditions.

Syntax

```
AVERAGEIFS(values_to_average, value_range1, criteria1,
           value_range2, criteria2
           [, value_rangeN, criteriaN ])
```

Argument	Type	Description
values_to_average	numeric	The numeric value that defines the overall range of values to evaluate.
value_range1–N	RDF term	The literal, URI, or blank node value that defines the range of values to test against the corresponding `criteria`.
criteria1–N	RDF term	The literal, URI, or blank node value that defines the condition to test the corresponding `value_range` against.

Returns

Type	Description
number	The average values from the ranges of values that meet the criteria.

AVG

This function calculates the average (arithmetic mean) value for a group of numbers.

Syntax

```
AVG(number)
```

Argument	Type	Description
number	numeric	The numeric value for which to calculate the average.

Returns

Type	Description
number	The arithmetic mean of the input values.

CEILING

This function rounds up a numeric value to the nearest integer if the value has a fractional part. CEILING returns the value itself if it is a whole number.

Syntax

```
CEILING(number)
```

Argument	Type	Description
number	numeric	The numeric value to round up.

Returns

Type	Description
number	The rounded up value.

COS

This function calculates the cosine of the specified angle.

Syntax

```
COS(angle)
```

Argument	Type	Description
angle	double	The angle in radians (double data type) to calculate the cosine for. If you have angle values in degrees, you can use RAD to convert the degrees to radians.

Returns

Type	Description
double	The cosine of the angle.

DIVIDE

This function divides one number by another number.

Syntax

```
DIVIDE(value1, value2)
```

Argument	Type	Description
value1	numeric	The number that is the dividend in the equation.

Argument	Type	Description
value2	numeric	The number to divide `value1` by.

Returns

Type	Description
number	The result of the division operation.

EQUAL

This function evaluates whether value1 is equal to value2.

Syntax

```
EQUAL(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if the values are equal and `false` if they are not.

EXP

This function raises the base of the natural logarithms, e, to the specified power.

Syntax

```
EXP(power)
```

Argument	Type	Description
power	numeric	The number to raise e to.

Returns

Type	Description
number	E raised to the specified power.

FACT

This function calculates the factorial of the specified number.

Syntax

```
FACT(number)
```

Argument	Type	Description
number	int	The number for which to calculate the factorial.

Returns

Type	Description
int	The factorial of the input values.

FLOOR

This function rounds down a numeric value to the nearest integer if the value has a fractional part. FLOOR returns the value itself if it is a whole number.

Syntax

```
FLOOR(number)
```

Argument	Type	Description
number	numeric	The numeric value to round down.

Returns

Type	Description
number	The rounded down value.

GE

This function evaluates whether value1 is greater than or equal to (>=) value2.

Syntax

```
GE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be checked to see if it is greater than or equal to value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 >= value2. `False` if not.

GT

This function evaluates whether value1 is greater than (>) value2.

Syntax

```
GE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be checked to see if it is greater than value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 > value2. `False` if not.

HAMMING_DIST

This function calculates the hamming distance between two values.

Syntax

```
HAMMING_DIST(number1, number2)
```

Argument	Type	Description
number1	long	The first number.
number2	long	The second number.

Returns

Type	Description
int	The hamming distance.

HAVERSINE_DIST

This function computes the haversine distance between two latitude and longitude values and returns the distance in kilometers. The function uses the Haversine formula, which is accurate for most purposes but assumes a spherical Earth. Since the Earth is elliptical, distances involving points near the poles will be more inaccurate than other points.

Syntax

```
HAVERSINE_DIST(latitude1, longitude1, latitude2, longitude2)
```

Argument	Type	Description
latitude1	double	The first latitude value.
longitude1	double	The first longitude value.
latitude2	double	The second latitude value.
longitude2	double	The second longitude value.

Returns

Type	Description
double	The distance in kilometers.

LE

This function evaluates whether value1 is less than or equal to (<=) value2.

Syntax

```
LE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be evaluated to see if it is less than or equal to value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 <= value2. `False` if not.

LN

This function calculates the natural logarithm of a double value.

Syntax

```
LN(number)
```

Argument	Type	Description
number	double	The double value for which to calculate the natural logarithm.

Returns

Type	Description
double	The natural logarithm of the input value.

LOG

This function calculates the specified base logarithm of a double value.

Syntax

```
LOG(number [, base ])
```

Argument	Type	Description
number	double	The double value for which to calculate the `base` logarithm.
base	double	An optional double value that specifies the base for the logarithm. If omitted, base e is used.

Returns

Type	Description
double	The base logarithm of the input value.

LOG2

This function calculates the base two logarithm of a double value.

Syntax

```
LOG2(number)
```

Argument	Type	Description
number	double	The double value for which to calculate the base 2 logarithm.

Returns

Type	Description
double	The base two logarithm of the input value.

LT

This function evaluates whether value1 is less than (<) value2.

Syntax

```
LT(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, or RDF term type value to compare to `value2`. This is the value that will be evaluated to see if it is less than value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, or RDF term type value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 < value2. `False` if not.

MAXVAL

This function determines the maximum value from the given literal values.

Syntax

```
MAXVAL(value1 [, value2 ] [, valueN ])
```

Argument	Type	Description
value1–N	literal	A literal value from which you want to find the maximum value.

Returns

Type	Description
literal	The maximum value.

MINVAL

This function determines the minimum value from the given literal values.

Syntax

```
MINVAL(value1 [, value2 ] [, valueN ])
```

Argument	Type	Description
value1–N	literal	A literal value from which you want to find the minimum value.

Returns

Type	Description
literal	The minimum value.

MOD

This function calculates the modulo or remainder of the division between two numbers.

Syntax

```
MOD(number, divisor)
```

Argument	Type	Description
number	numeric	The number that is the dividend in the equation.
divisor	numeric	The number to divide the dividend by.

Returns

Type	Description
number	The modulo between the input numbers.

MULTIPLY

This function multiplies two numbers.

Syntax

```
MULTIPLY(value1, value2)
```

Argument	Type	Description
value1	numeric	The first number in the multiplication equation.

Argument	Type	Description
value2	numeric	The number to multiply `value1` by.

Returns

Type	Description
number	The result of the multiplication operation.

NOT_EQUAL

This function evaluates whether value1 is not equal to value2.

Syntax

```
NOT_EQUAL(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 is not equal to value2. `False` if they are equal.

NPV

This function calculates the net present value of an investment by using a discount rate and a series of future payments (negative values) and income (positive values).

Syntax

```
NPV(rate, year, value)
```

Argument	Type	Description
rate	double	A double value that defines the discount rate to use in the calculation.
year	double	A double value that indicates which year the `value` is in.
value	double	The double values that represent payments and income.

Returns

Type	Description
double	The net present value.

NUMERIC-ADD

This function adds two numeric values.

Syntax

```
NUMERIC-ADD(value1, value2)
```

Argument	Type	Description
value1	numeric	The first number in the addition equation.
value2	numeric	The number to add to `value1`.

Returns

Type	Description
number	The result of the addition operation.

NUMERIC-SUBTRACT

This function subtracts one numeric value from another numeric value.

Syntax

```
NUMERIC-SUBTRACT(value1, value2)
```

Argument	Type	Description
value1	numeric	The first number in the subtraction equation.
value2	numeric	The number to subtract from `value1`.

Returns

Type	Description
number	The result of the subtraction operation.

PI

This function returns the value for PI.

Syntax

```
PI()
```

Returns

Type	Description
double	The PI value.

POWER

This function raises the specified number to the specified power.

Syntax

```
POWER(value, power)
```

Argument	Type	Description
value	numeric	The number to raise by the `power`.
power	numeric	The number to raise `value` by.

Returns

Type	Description
number	The result of `value` raised to the specified `power`.

PRODUCT

This function calculates the product of a group of numbers.

Syntax

```
PRODUCT(number)
```

Argument	Type	Description
number	numeric	The group of numbers to multiply.

Returns

Type	Description
number	The product of the group.

QUOTIENT

This function calculates the quotient between two numbers.

Syntax

```
QUOTIENT(numerator, denominator)
```

Argument	Type	Description
numerator	numeric	The number to divide by the `denominator`.
denominator	numeric	The number to divide the `numerator` by.

Returns

Type	Description
long	The quotient between the input values.

RAD

This function converts to radians an angle value that is in degrees.

Syntax

```
RAD(angle)
```

Argument	Type	Description
angle	double	The angle value to convert to radians.

Returns

Type	Description
double	The angle in radians.

RAND

This function returns a random double value between 0 and 1.

Syntax

```
RAND()
```

Returns

Type	Description
double	The random value between 0 and 1.

RANDBETWEEN

This function returns a random integer that falls between the two specified integers. The two integers are included as options to be returned.

Syntax

```
RANDBETWEEN(low_number, high_number)
```

Argument	Type	Description
low_number	int	The lowest integer in the range of values.
high_number	int	The highest integer in the range of values.

Note

If the arguments are decimal values, Anzo returns a random integer between `CEIL(low_number)` and `FLOOR(high_number)`.

Returns

Type	Description
int	The random value between the given low and high numbers.

ROUND

This function rounds a numeric value to the nearest integer.

Syntax

```
ROUND(number)
```

Argument	Type	Description
number	numeric	The number to round to the nearest integer.

Returns

Type	Description
long	The rounded value.

ROUNDDOWN

This function rounds a numeric value down to the specified number of digits.

Syntax

```
ROUNDDOWN(number, num_digits)
```

Argument	Type	Description
number	numeric	The number to round down.
num_digits	int	An integer that specifies the number of digits to round down to.

Returns

Type	Description
number	The rounded down value.

ROUNDUP

This function rounds a numeric value up to the specified number of digits.

Syntax

```
ROUNDUP(number, num_digits)
```

Argument	Type	Description
number	numeric	The number to round up.
num_digits	int	An integer that specifies the number of digits to round up to.

Returns

Type	Description
number	The rounded up value.

SIN

This function calculates the sine of the specified angle.

Syntax

```
SIN(angle)
```

Argument	Type	Description
angle	double	The angle in radians to calculate the sine for. If you have angle values in degrees, you can use RAD to convert the degrees to radians.

Returns

Type	Description
double	The sine of the angle.

SQRT

This function calculates the square root of the specified number.

Syntax

```
SQRT(number)
```

Argument	Type	Description
number	numeric	The number for which to calculate the square root.

Returns

Type	Description
double	The square root of the input value.

SUBTRACT

This function subtracts one RDF term type (a literal value, URI, or blank node) value from another RDF term type value.

Syntax

```
SUBTRACT(term1, term2)
```

Argument	Type	Description
term1	RDF term	The literal, URI, or blank node value that `term2` will be subtracted from.
term2	RDF term	The literal, URI, or blank node value to subtract from `term1`.

Returns

Type	Description
RDF term	The result of the subtraction operation.

SUM

This function calculates the sum of the numbers within a group.

Syntax

```
SUM(number)
```

Argument	Type	Description
number	numeric	The group of numbers to sum.

Returns

Type	Description
number	The sum of the values in the group.

SUMIF

This function calculates the sum of the range of values that meet the specified criterion.

> **Tip**
> You can use SUMIFS to specify multiple value ranges and conditions.

Syntax

```
SUMIF(values_to_test, criterion [, range_of_values ])
```

Argument	Type	Description
values_to_test	RDF term	The literal, URI, or blank node value that defines the values to test against the `criterion`.
criterion	RDF term	The literal, URI, or blank node value that defines the condition to test values against.
range_of_values	numeric	An optional number that defines the range of values to sum. When omitted, `values_to_test` is used.

Returns

Type	Description
number	The sum of the range of values.

SUMIFS

This function calculates the sums of the ranges of values that meet the specified criteria. Unlike SUMIF, this function enables you to specify multiple ranges and multiple conditions.

Syntax

```
SUMIFS(values_to_sum, value_range1, criteria1,
       value_range2, criteria2
       [, value_rangeN, criteriaN ])
```

Argument	Type	Description
values_to_sum	numeric	The numeric value that defines the overall range of values to evaluate.
value_range1–N	RDF term	The literal, URI, or blank node value that defines the range of values to test against the corresponding `criteria`.
criteria1–N	RDF term	The literal, URI, or blank node value that defines the condition to test the corresponding `value_range` against.

Returns

Type	Description
number	The sums of the ranges of values.

SUMPRODUCT

This function multiplies the numbers in a group and adds the results.

Syntax

```
SUMPRODUCT(number)
```

Argument	Type	Description
number	numeric	The group of numbers to multiply and then sum the results.

Returns

Type	Description
number	The sum of the product of the numbers in the group.

SUMSQ

This function calculates the square root of each number in a group and adds the results.

Syntax

```
SUMSQ(number)
```

Argument	Type	Description
number	numeric	The group of numbers for which to calculate the square root and then sum the results.

Returns

Type	Description
number	The sum of the square root of the numbers in the group.

TAN

This function calculates the tangent of the specified angle.

Syntax

```
TAN(angle)
```

Argument	Type	Description
angle	double	The angle in radians to calculate the tangent for. If you have angle values in degrees, you can use RAD to convert the degrees to radians.

Returns

Type	Description
double	The tangent of the angle.

Aggregate Functions

This topic describes the aggregate functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- AVERAGEIF: Calculates the average of the range of values that meet the specified criterion.
- AVERAGEIFS: Calculates the averages of the ranges of values that meet the specified criteria.
- AVG: Calculates the average (arithmetic mean) value for a group of numbers.
- CHOOSE_BY_MAX: Returns the value from a group that corresponds to the maximum value from another group.
- CHOOSE_BY_MIN: Returns the value from a group that corresponds to the minimum value from another group.
- COUNT: Counts the number of values that exist for a group.
- COUNT_DISTINCT: Counts the number of unique values that exist for a group.
- COUNTIF: Counts the number of values that meet the specified criterion.
- COUNTIFS: Counts the number of values that meet the specified criteria.
- GROUP_CONCAT: Concatenates a group of strings into a single string.
- GROUPCONCAT: Concatenates a group of strings into a single string. This function is a customizable version of GROUP_CONCAT.
- MAX: Returns the maximum value from each group of values.

- MEDIAN: Returns the median number out of a group of numbers.
- MIN: Returns the minimum value from each group of values.
- MODE: Returns the mode (the value that occurs most frequently) from a group of values.
- MODEPERCENT: Calculates the percentage of values in a group that belong to the mode.
- PERCENTILE_CONT: Calculates a percentile based on the continuous distribution of the specified group of values.
- PERCENTILE_DISC: Calculates a percentile based on the discrete distribution of the specified group of values.
- PRODUCT: Calculates the product of a group of numbers.
- SAMPLE: Returns an arbitrary value from the specified group of values.
- STDEV: Measures the standard deviation in a group of numbers.
- STDEVP: Calculates the product of the standard deviation for a group of numbers.
- SUM: Calculates the sum of the numbers within a group.
- SUMIF: Calculates the sum of the range of values that meet the specified criterion.
- SUMIFS: Calculates the sums of the ranges of values that meet the specified criteria.
- SUMPRODUCT: Multiplies the numbers in a group and adds the results.
- SUMSQ: Calculates the square root of each number in a group and adds the results.
- VAR: Calculates the unbiased (sample) variance of a group of numbers.
- VARP: Calculates the biased (population) variance of a group of numbers.
- WEIGHTEDAVERAGE: Calculates the weighted average of a group of values.

AVERAGEIF

This function calculates the average of the range of values that meet the specified criterion.

Tip
You can use AVERAGEIFS to specify multiple value ranges and conditions.

Syntax

```
AVERAGEIF(values_to_test, criterion [, range_of_values ])
```

Argument	Type	Description
values_to_test	RDF term	The literal, URI, or blank node value that defines the values to test against the `criteria`.
criterion	RDF term	The literal, URI, or blank node value that defines the condition to test values against.
range_of_values	numeric	An optional number that defines the range of values to average. When omitted, `values_to_test` is used.

Returns

Type	Description
number	The average value from the range of values that meet the criterion.

AVERAGEIFS

This function calculates the averages of the ranges of values that meet the specified criteria. Unlike AVERAGEIF, this function enables you to specify multiple ranges and multiple conditions.

Syntax

```
AVERAGEIFS(values_to_average, value_range1, criteria1,
           value_range2, criteria2
           [, value_rangeN, criteriaN ])
```

Argument	Type	Description
values_to_average	numeric	The numeric value that defines the overall range of values to evaluate.
value_range1–N	RDF term	The literal, URI, or blank node value that defines the range of values to test against the corresponding `criteria`.
criteria1–N	RDF term	The literal, URI, or blank node value that defines the condition to test the corresponding `value_range` against.

Returns

Type	Description
number	The average values from the ranges of values that meet the criteria.

AVG

This function calculates the average (arithmetic mean) value for a group of numbers.

Syntax

```
AVG(number)
```

Argument	Type	Description
number	numeric	The numeric value for which to calculate the average.

Returns

Type	Description
number	The arithmetic mean of the input values.

CHOOSE_BY_MAX

This function calculates the maximum value for one group and returns the value from another group that corresponds to the maximum from the first group.

Syntax

```
CHOOSE_BY_MAX(test, value)
```

Argument	Type	Description
test	RDF term	The group of literal, URI, or blank node values from which to find the maximum value.
value	RDF term	The group of literal, URI, or blank node values from which to return the value that corresponds to the maximum value of `test`.

Example

In a fictional ticket sales data set, the following statement returns the ID of the buyer who paid the most:

```
CHOOSE_BY_MAX(?totalPaid, ?buyerID)
```

Returns

Type	Description
RDF term	The term from the `value` group that corresponds to the maximum value from the `test` group.

CHOOSE_BY_MIN

This function calculates the minimum value for one group and returns the value from another group that corresponds to the minimum from the first group.

Syntax

```
CHOOSE_BY_MIN(test, value)
```

Argument	Type	Description
test	RDF term	The group of literal, URI, or blank node values from which to find the minimum value.
value	RDF term	The group of literal, URI, or blank node values from which to return the value that corresponds to the minimum value of `test`.

Example

In a fictional ticket sales data set, the following statement returns the ID of the seller who sold the least number of tickets:

```
CHOOSE_BY_MIN(?totalTickets, ?sellerID)
```

Returns

Type	Description
RDF term	The term from the `value` group that corresponds to the minimum value from the `test` group.

COUNT

This function counts the number of values that exist for a group.

Syntax

```
COUNT(value)
```

Argument	Type	Description
value	RDF term	The group of literal, URI, or blank node values to count.

Returns

Type	Description
long	The number of values in the group.

COUNT_DISTINCT

This function counts the number of unique values that exist for a group.

Syntax

```
COUNT_DISTINCT(value)
```

Argument	Type	Description
value	RDF term	The group of literal, URI, or blank node values for which to count the number of distinct values.

Returns

Type	Description
long	The number of unique values in the group.

COUNTIF

This function counts the number of values that meet the specified criterion.

> **Tip**
> You can use COUNTIFS to specify multiple conditions.

Syntax

```
COUNTIF(values_to_test, criterion)
```

Argument	Type	Description
values_to_test	RDF term	The literal, URI, or blank node value that defines the values to test against the `criterion`.
criterion	RDF term	The literal, URI, or blank node value that defines the condition to test values against.

Returns

Type	Description
long	The number of values that meet the criterion.

COUNTIFS

This function counts the number of values that meet the specified criteria. Unlike COUNTIF, this function enables you to specify multiple conditions.

Syntax

```
COUNTIFS(values_to_count, criteria1 [, criteria2 ] [, criteriaN ])
```

Argument	Type	Description
values_to_count	RDF term	The literal, URI, or blank node value to compare against the criteria.
criteria1–N	RDF term	A literal, URI, or blank node value that defines a condition to test the `values_to_count` against.

Returns

Type	Description
long	The number of values that meet the specified conditions.

GROUP_CONCAT

This function concatenates a group of strings into a single string. It is a simplified version of GROUPCONCAT as it takes only one argument.

Syntax

```
GROUP_CONCAT(text)
```

Argument	Type	Description
text	string	The string property whose values to concatenate into a single string.

Returns

Type	Description
string	The concatenated string.

GROUPCONCAT

This function concatenates a group of strings into a single string. Unlike GROUP_CONCAT, this function allows for customization of the separator to use as well as the configuration of limits and options like prefixes and suffixes.

Syntax

```
GROUPCONCAT(group1, [ group2, ..., groupN, ] group_value_separator, separator,
serialize,
            row_limit, value_limit, delimit_blanks [, prefix ] [, suffix ] [, max_
length ])
```

Argument	Type	Description
group1–N	string	The group(s) of strings to concatenate.
group_value_separator	string	The separator string to use between the groups of strings if you specified more than one `group`.

Argument	Type	Description
separator	string	The separator string to use between the values in a concatenated group of strings.
serialize	boolean	A boolean value that indicates whether returned values should be serialized with the value's data type.
row_limit	int	An integer that puts a maximum limit on the number of rows to retrieve for a group.
value_limit	int	An integer that puts a maximum limit on the number of values to retrieve from a group of rows.
delimit_blanks	boolean	A boolean value that indicates whether to delimit blanks with the `separator` value.
prefix	string	Optional string to add as a prefix to the resulting string.
suffix	string	Optional string to add as a suffix to the resulting string.
max_length	int	Optional integer that puts a maximum limit on the number of characters the resulting string can have.

Returns

Type	Description
string	The concatenated string.

MAX

This function returns the maximum value from each group of values.

Syntax

```
MAX(value1 [, value2 ] [, valueN ])
```

Argument	Type	Description
value1–N	RDF term	The group(s) of literal, URI, or blank node values for which to return the maximum value.

Returns

Type	Description
RDF term	The maximum value from each group.

MEDIAN

This function returns the median value from a group of numbers. The median is the number in the group where half of the numbers are greater than the number and half are less than the number.

Syntax

```
MEDIAN(number)
```

Argument	Type	Description
number	numeric	The group of numeric values for which to calculate the median.

Returns

Type	Description
number	The median for the group.

MIN

This function returns the minimum value from each group of values.

Syntax

```
MIN(value1 [, value2 ] [, valueN ])
```

Argument	Type	Description
value1–N	RDF term	The group(s) of literal, URI, or blank node values for which to return the minimum value.

Returns

Type	Description
RDF term	The minimum value from each group.

MODE

This function returns the mode from a group of values. The mode is the value that occurs most frequently in the group.

Syntax

```
MODE(value)
```

Argument	Type	Description
value	RDF term	The group of literal, URI, or blank node values for which to return the mode.

Returns

Type	Description
RDF term	The mode from the group.

MODEPERCENT

This function calculates the percentage of values in a group that belong to the mode.

Syntax

```
MODEPERCENT(value)
```

Argument	Type	Description
value	RDF term	The group of literal, URI, or blank node values for which to calculate the modepercent.

Returns

Type	Description
double	The percentage of values that belong to the mode.

PERCENTILE_CONT

This function calculates a percentile based on the continuous distribution of the specified group of values. The returned value is interpolated and may not be equal to any of the values in the group.

Syntax

```
PERCENTILE_CONT(percentile, value)
```

Argument	Type	Description
percentile	float	A float value that specifies the percentile to compute.
value	RDF term	The group of literal, URI, or blank node values for which to calculate the percentile.

Returns

Type	Description
RDF term	The interpolated percentile.

PERCENTILE_DISC

This function calculates a percentile based on the discrete distribution of the specified group of values.

Syntax

```
PERCENTILE_DISC(percentile, value)
```

Argument	Type	Description
percentile	float	A float value that specifies the percentile to compute.
value	RDF term	The group of literal, URI, or blank node values for which to calculate the percentile.

Returns

Type	Description
RDF term	The percentile based on the discrete distribution of the group.

PRODUCT

This function calculates the product of a group of numbers.

Syntax

```
PRODUCT(number)
```

Argument	Type	Description
number	numeric	The group of numbers to multiply.

Returns

Type	Description
number	The product of the group.

SAMPLE

This function returns an arbitrary value from the specified group of values.

Syntax

```
SAMPLE(value)
```

Argument	Type	Description
value	RDF term	The group of literal, URI, or blank node values from which to choose a sample value.

Returns

Type	Description
RDF term	The arbitrary value from the group.

STDEV

This function measures the standard deviation (amount of dispersion) of a group of numbers.

Syntax

```
STDEV(value)
```

Argument	Type	Description
value	numeric	The numeric value that defines the set of numbers for which to measure the standard deviation.

Returns

Type	Description
number	The standard deviation of the group.

STDEVP

This function calculates the product of the standard deviation for a group of numbers.

Syntax

```
STDEVP(value)
```

Argument	Type	Description
value	numeric	The numeric value that defines the set of numbers for which to measure the standard deviation and compute the product.

Returns

Type	Description
number	The product of the standard deviation for the group.

SUM

This function calculates the sum of the numbers within a group.

Syntax

```
SUM(number)
```

Argument	Type	Description
number	numeric	The group of numbers to sum.

Returns

Type	Description
number	The sum of the values in the group.

SUMIF

This function calculates the sum of the range of values that meet the specified criterion.

> **Tip**
> You can use SUMIFS to specify multiple value ranges and conditions.

Syntax

```
SUMIF(values_to_test, criterion [, range_of_values ])
```

Argument	Type	Description
values_to_test	RDF term	The literal, URI, or blank node value that defines the values to test against the `criterion`.
criterion	RDF term	The literal, URI, or blank node value that defines the condition to test values against.
range_of_values	numeric	An optional number that defines the range of values to sum. When omitted, `values_to_test` is used.

Returns

Type	Description
number	The sum of the range of values.

SUMIFS

This function calculates the sums of the ranges of values that meet the specified criteria. Unlike SUMIF, this function enables you to specify multiple ranges and multiple conditions.

Syntax

```
SUMIFS(values_to_sum, value_range1, criteria1,
       value_range2, criteria2
       [, value_rangeN, criteriaN ])
```

Argument	Type	Description
values_to_sum	numeric	The numeric value that defines the overall range of values to evaluate.
value_range1–N	RDF term	The literal, URI, or blank node value that defines the range of values to test against the corresponding `criteria`.
criteria1–N	RDF term	The literal, URI, or blank node value that defines the condition to test the corresponding `value_range` against.

Returns

Type	Description
number	The sums of the ranges of values.

SUMPRODUCT

This function multiplies the numbers in a group and adds the results.

Syntax

```
SUMPRODUCT(number)
```

Argument	Type	Description
number	numeric	The group of numbers to multiply and then sum the results.

Returns

Type	Description
number	The sum of the product of the numbers in the group.

SUMSQ

This function calculates the square root of each number in a group and adds the results.

Syntax

```
SUMSQ(number)
```

Argument	Type	Description
number	numeric	The group of numbers for which to calculate the square root and then sum the results.

Returns

Type	Description
number	The sum of the square root of the numbers in the group.

VAR

This function calculates the unbiased (sample) variance for a group of numbers.

Syntax

```
VAR(value)
```

Argument	Type	Description
value	numeric	The numeric value that defines the set of numbers for which to measure the variance.

Returns

Type	Description
number	The unbiased variance for the group.

VARP

This function calculates the biased (population) variance for a group of numbers.

Syntax

```
VARP(value)
```

Argument	Type	Description
value	number	The value that defines the set of numbers for which to measure the population variance.

Returns

Type	Description
decimal	The biased variance for the group.

WEIGHTEDAVERAGE

This function calculates the weighted average of a group of values.

Syntax

```
WEIGHTEDAVERAGE(value, weight)
```

Argument	Type	Description
value	decimal	The decimal value that defines the group of values for which to calculate the weighted average.
weight	decimal	The decimal value that defines the weight to use in the calculation.

Returns

Type	Description
decimal	The weighted average for the group.

Date and Time Functions

This topic describes the date, time, and duration functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- DATE: Returns an xsd:date value based on the specified year, month, and day.
- DATEPART: Returns the date part of a literal string, date, long, or dateTime value.
- DATETIME (or xsd:dateTime): Returns a dateTime value from the given string, long, or dateTime.
- DAY: Returns the day of the month from the specified date.
- DAYSFROMDURATION: Returns the days portion of a duration value.
- DUR_TO_MILLIS: Calculates the time in milliseconds from a duration or numeric value.
- DURATION: Returns an xsd:duration value from the given numeric or duration value.
- DURATIONFORMAT: Converts a duration or numeric value to a string in the specified duration format.
- DURATIONPERIODFORMAT: Calculates the duration between the given start and end values and returns a string in the specified duration format.
- FORMATDATE: Converts a numeric or date value into a string with the specified date format.
- HOUR: Returns the hour portion of the given dateTime value.

- MASKEDDATETIME: Replaces the year, month, day, hour, minute, second, and millisecond values for the given date or dateTime value with the new date and time values that you specify.
- MILLIS: Calculates the number of milliseconds in the given date or dateTime value.
- MINUTE: Returns the minutes portion of the given dateTime value.
- MONTH: Returns the month portion of the given dateTime value.
- NOW: Returns the current server date and time.
- NOWMILLIS: Returns the current server date and time in epoch milliseconds.
- PARSEDATETIME: Attempts to convert the given string or plain literal to a date, time, or dateTime value.
- SECOND: Returns the seconds portion of the given dateTime value.
- TIME: Returns an xsd:time value based on the specified hour, minute, and second values.
- TIMEPART: Returns the time part of a time or dateTime value.
- TIMEVALUE: Converts the specified RDF term type value to an xsd:time value.
- TIMEZONE: Returns as a duration the timezone part of a dateTime value.
- TODAY: Returns today's date based on the server date.
- TZ: Returns as a string the timezone from a dateTime value.
- WEEKDAY: Returns the day of the week from a date or dateTime value.
- WEEKNUM: Returns the week of the year in which the given date or dateTime occurs.
- xsd:date: Converts the specified string, date, or dateTime value to an xsd:date.
- YEAR: Returns the year portion of the given dateTime value.
- YEARMONTH: Returns the year and month (in the format "year-month") from the specified date or dateTime value.

DATE

This function returns an xsd:date value based on the specified year, month, and day values.

Syntax

```
DATE(year, month, day)
```

Argument	Type	Description
year	int	An integer that represents the year.
month	int	An integer that represents the month.
day	int	An integer that represents the day.

Returns

Type	Description
date	The date according to the input values.

DATEPART

This function returns the date part of a literal string, date, long, or dateTime value.

Syntax

```
DATEPART(value)
```

Argument	Type	Description
value	literal string, date, long, or dateTime	The literal string, date, long, or dateTime value from which to return the date.

Returns

Type	Description
date	The date part of the input values.

DATETIME (or xsd:dateTime)

This function returns a dateTime value from the given string, long, or dateTime.

Syntax

```
DATETIME(value)
```

Argument	Type	Description
value	string, long, dateTime	The string, long, or dateTime value from which to return a dateTime.

Returns

Type	Description
dateTime	The dateTime value.

DAY

This function returns the day of the month from the specified date value.

Syntax

```
DAY(value)
```

Argument	Type	Description
value	date	The date value from which to return the day of the month.

Returns

Type	Description
int	The day of the month.

DAYSFROMDURATION

This function returns the days portion of a duration value.

Syntax

```
DAYSFROMDURATION(value)
```

Argument	Type	Description
value	duration	The duration value from which to return the days.

Returns

Type	Description
long	The number of days in the duration.

DUR_TO_MILLIS

This function calculates the time in milliseconds from a duration or numeric value.

Syntax

```
DUR_TO_MILLIS(value)
```

Argument	Type	Description
value	duration, numeric	The duration or numeric value from which to calculate the time in milliseconds.

Returns

Type	Description
long	The number of milliseconds.

DURATION

This function returns an xsd:duration value from the given numeric or duration value.

Syntax

```
DURATION(value)
```

Argument	Type	Description
value	duration, numeric	The duration or numeric value from which to return an xsd:duration.

Returns

Type	Description
duration	The duration value.

DURATIONFORMAT

This function converts a duration or numeric value to a string in the specified duration format.

Syntax

```
DURATIONFORMAT(value [, format ])
```

Argument	Type	Description
value	duration, numeric	The duration or numeric value to format.

Argument	Type	Description
format	string	An optional value that specifies the format to use for the resulting duration string. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **H** for hours • **m** for minutes • **s** for seconds • **S** for milliseconds • **'text'** for arbitrary text content If `format` is not specified, `H:mm:ss.SSS` is used.

Returns

Type	Description
string	The duration as a string.

DURATIONPERIODFORMAT

This function calculates the duration between the given start and end dateTime or numeric values and returns a string in the specified duration format.

Syntax

```
DURATIONPERIODFORMAT(start, end [, format ])
```

Argument	Type	Description
start	dateTime, numeric	The dateTime or numeric value that is the start of the duration period.
end	dateTime, numeric	The dateTime or numeric value that is the end of the duration period.
format	string	An optional value that specifies the format to use for the resulting duration string. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **H** for hours • **m** for minutes • **s** for seconds • **S** for milliseconds • **'text'** for arbitrary text content If `format` is not specified, the default is `'P'yyyy'Y'M'M'd'DT'H'H'm'M's.SSS'S'`. The default value uses 'text' patterns with Pattern Tokens, which results in a string such as `P1Y3M4DT1H4M44.000S`.

Returns

Type	Description
string	The duration as a string.

FORMATDATE

This function converts a numeric or date value into a string with the specified date format.

Syntax

```
FORMATDATE(value, format)
```

Argument	Type	Description
value	date, numeric	The date or numeric value to convert to a string in the specified date `format`.
format	string	The format to use for the resulting date string. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **'text'** for arbitrary text content For example, `"yyyy.MM.dd"` or `"dd/MM/yyyy"`.

Returns

Type	Description
string	The date as a string.

HOUR

This function returns the hour portion of the given dateTime value.

Syntax

```
HOUR(value [, timezone ])
```

Argument	Type	Description
value	dateTime	The dateTime value from which to return the hour portion.
timezone	string	An optional value that specifies the timezone for the `value`.

Returns

Type	Description
int	The hour.

MASKEDDATETIME

This function replaces the year, month, day, hour, minute, second, and millisecond values for the given date or dateTime value with the new date and time values that you specify.

Syntax

```
MASKEDDATETIME(value, year, month, day, hour, minute, second, milliseconds)
```

Argument	Type	Description
value	date, dateTime	The date or dateTime for which to replace the year, month, date, hour, minute, second, and milliseconds values.
year	int	The year to include in the resulting dateTime value.
month	int	The month to include in the resulting dateTime value.
day	int	The day to include in the resulting dateTime value.
hour	int	The hour to include in the resulting dateTime value.
minute	int	The minutes value to include in the resulting dateTime value.

Argument	Type	Description
second	int	The seconds value to include in the resulting dateTime value.
milliseconds	int	The milliseconds value to include in the resulting dateTime value.

Returns

Type	Description
dateTime	The dateTime value with the specified input values.

MILLIS

This function calculates the number of milliseconds in the given date or dateTime value.

Syntax

```
MILLIS(value)
```

Argument	Type	Description
value	date, dateTime	The date or dateTime value for which to calculate the number of milliseconds.

Returns

Type	Description
long	The number of milliseconds.

MINUTE

This function returns the minutes portion of the given dateTime value.

Syntax

```
MINUTE(value)
```

Argument	Type	Description
value	dateTime	The dateTime value from which to return the minutes portion.

Returns

Type	Description
int	The minutes portion of the input value.

MONTH

This function returns the month portion of the given dateTime value.

Syntax

```
MONTH(value)
```

Argument	Type	Description
value	dateTime	The dateTime value from which to return the month portion.

Returns

Type	Description
int	The month number.

NOW

This function returns the current server date and time.

Syntax

```
NOW([ timezone ])
```

Argument	Type	Description
timezone	string	An optional value that specifies the timezone for which to return the current dateTime.

Returns

Type	Description
dateTime	The current server date and time.

NOWMILLIS

This function returns the current server date and time in epoch milliseconds.

Syntax

```
NOWMILLIS()
```

Returns

Type	Description
long	The current server date and time in milliseconds.

PARSEDATETIME

This function attempts to convert the given string or plain literal to a date, time, or dateTime value. For values that do not include a timezone, Anzo stores them in GMT. Values that include a timezone are stored as the appropriate value in GMT for that timezone.

Syntax

```
PARSEDATETIME(value [, output_type ] [, format ])
```

Argument	Type	Description
value	string,	The string or plain literal value to convert to a date, time, or dateTime.

Argument	Type	Description
	literal	
output_type	URI	An optional URI (`xsd:date`, `xsd:time`, or `xsd:dateTime`) that specifies the type of value to return. If `output_type` is not specified, dateTime is returned.
format	string	An optional string that species the format to use for the resulting date, time, or dateTime value. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **H** for hours • **m** for minutes • **s** for seconds • **S** for milliseconds • **'text'** for arbitrary text content For example, `"yyyy.MM.dd HH:mm"` or `"dd/MM/yyyy HH:mm:ss"`.

Returns

Type	Description
date, time, or dateTime	The conversion of the string to the desired type.

SECOND

This function returns the seconds portion of the given dateTime value.

Syntax

```
SECOND(value)
```

Argument	Type	Description
value	dateTime	The dateTime value from which to return the seconds portion.

Returns

Type	Description
int	The seconds portion of the input value.

TIME

This function returns an xsd:time value based on the specified hour, minute, and second values.

Syntax

```
TIME(hour, minute, second)
```

Argument	Type	Description
hour	int	An integer that represents the hour.
minute	int	An integer that represents the minute.
second	int	An integer that represents the seconds.

Returns

Type	Description
time	The time according to the input values.

TIMEPART

This function returns the time part of a time or dateTime value.

Syntax

```
TIMEPART(value)
```

Argument	Type	Description
value	time, dateTime	The time or dateTime value from which to return the time portion.

Returns

Type	Description
time	The time portion of the input value.

TIMEVALUE

This function converts the specified RDF term type value to an xsd:time value.

Syntax

```
TIMEVALUE(value)
```

Argument	Type	Description
value	RDF term	The literal, URI, or blank node value to convert to a time value.

Returns

Type	Description
time	The conversion of the term to a time value.

TIMEZONE

This function returns the timezone part of a dateTime value as a duration. An error is returned if the input value does not include the timezone.

Syntax

```
TIMEZONE(value)
```

Argument	Type	Description
value	dateTime	The dateTime value to return the timezone from.

Returns

Type	Description
duration	The timezone in duration format.

TODAY

This function returns today's date based on the server date.

Syntax

```
TODAY()
```

Returns

Type	Description
date	Today's date according to the server.

TZ

This function returns the timezone part of a dateTime value as a string.

Syntax

```
TZ(value)
```

Argument	Type	Description
value	dateTime	The dateTime value to return the timezone from.

Returns

Type	Description
string	The timezone as a string.

WEEKDAY

This function returns the day of the week from a date or dateTime value.

Syntax

```
WEEKDAY(value [, day_number_start ])
```

Argument	Type	Description
value	date, dateTime	The date or dateTime value from which to return the day of the week.
day_number_start	int	An optional value of **1**, **2**, or **3** that defines how the days of the week are represented as numbers. • **1** means Sunday is day 1. Saturday is day 7. • **2** means Monday is day 1. Sunday is day 7. • **3** means Monday is day 0. Sunday is day 6. If `day_number_start` is not specified, the default value is **1**.

Returns

Type	Description
int	The day of the week from the input values.

WEEKNUM

This function returns the week of the year in which the given date or dateTime occurs.

Syntax

```
WEEKNUM(value [, day_week_begins ])
```

Argument	Type	Description
value	date, dateTime	The date or dateTime value from which to return the week number.
day_week_begins	int	An optional value of **1** or **2** that defines which day the weeks start on. • **1** means a new week starts on Sunday. • **2** means a new week starts on Monday. If `day_week_begins` is not specified, the default value is **1**.

Returns

Type	Description
int	The week of the year the input value falls in.

xsd:date

This function converts the specified string, date, or dateTime value to an xsd:date.

Syntax

```
xsd:date(value)
```

Argument	Type	Description
value	string, date, dateTime	The string, date, or dateTime value to convert to an xsd:date.

Returns

Type	Description
date	The input values converted to dates.

YEAR

This function returns the year portion of the given dateTime value.

Syntax

```
YEAR(value)
```

Argument	Type	Description
value	dateTime	The dateTime value to return the year from.

Returns

Type	Description
int	The year portion of the input values.

YEARMONTH

This function returns the year and month (in the format "year-month") from the specified date or dateTime value.

Syntax

```
YEARMONTH(value)
```

Argument	Type	Description
value	literal date or dateTime	The value to return the year-month from.

Returns

Type	Description
gYearMonth	The year-month from the input values.

Casting Functions

This topic describes the functions that are available for coercing data types in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- BNODE: Creates a blank node.
- BOOLEAN: Casts a literal value to a boolean data type.
- BYTE: Casts a literal value to a byte data type.
- CONCATURL: Concatenates two or more strings and returns the result as a URI.
- DATATYPE: Returns the data type of the given value.
- DATETIME (or xsd:dateTime): Returns a dateTime value from the given string, long, or dateTime.
- DATEVALUE: Casts a string to a date.
- DECIMAL: Casts a literal value to a decimal data type.
- DOUBLE: Casts a literal value to a double data type.
- DURATION: Returns an xsd:duration value from the given numeric or duration value.
- DURATIONFORMAT: Converts a duration or numeric value to a string in the specified duration format.
- ENCODE_FOR_URI: Encodes the specified string as a URI.
- FLOAT: Casts a literal value to a float data type.

- FORMATDATE: Casts a numeric or date value to a string in the specified date format.
- FORMATFRACTION: Converts a numeric value into a fraction string.
- FORMATNUMBER: Casts a numeric value to a string in the specified format.
- INT: Casts a literal value to an int data type.
- INTEGER: Casts a literal value to an integer data type.
- LONG: Casts a literal value to a long data type.
- PARSEDATETIME: Attempts to convert the given string or plain literal to a date, time, or dateTime value.
- RAD: Converts to radians an angle value that is in degrees.
- SERIALIZE: Creates a literal value from the string representation of the specified RDF term.
- SHORT: Casts a literal value to a short data type.
- STR: Casts an RDF term type value to a string.
- TEXT: Casts a numeric or dateTime value to a string in the specified format.
- TIMEVALUE: Converts the specified RDF term type value to an xsd:time value.
- TOURI: Casts a string literal value to a URI.
- UUID: Generates a Universally Unique Identifier (UUID).
- xsd:date: Converts the specified string, date, or dateTime value to an xsd:date.

BNODE

This function creates a blank node.

Syntax

```
BNODE([ value ])
```

Argument	Type	Description
value	string	An optional string value from which to create the blank node.

Returns

Type	Description
blank node	The generated blank node.

BOOLEAN

This function casts a literal value to a boolean data type.

Syntax

```
BOOLEAN(value)
```

Argument	Type	Description
value	literal	The literal value to cast to a boolean type.

Returns

Type	Description
boolean	The input cast to boolean.

BYTE

This function casts a literal value to a byte data type.

Syntax

```
BYTE(value)
```

Argument	Type	Description
value	literal	The literal value to cast to a byte type.

Returns

Type	Description
byte	The input value cast to a byte.

CONCATURL

This function concatenates two or more strings and returns the result as a URI.

Syntax

```
CONCATURL(text1, text2 [, textN ])
```

Argument	Type	Description
text1–N	string	The strings that you want to concatenate to form a URI.

Returns

Type	Description
URI	The concatenated string as a URI.

DATATYPE

This function returns the data type of the given literal value.

Syntax

```
DATATYPE(value)
```

Argument	Type	Description
value	literal	The literal value for which to return the data type.

Returns

Type	Description
URI	The data type.

DATETIME (or xsd:dateTime)

This function returns a dateTime value from the given string, long, or dateTime.

Syntax

```
DATETIME(value)
```

Argument	Type	Description
value	string, long, dateTime	The string, long, or dateTime value from which to return a dateTime.

Returns

Type	Description
dateTime	The dateTime value.

DATEVALUE

This function casts the given string value to a date.

Syntax

```
DATEVALUE(value)
```

Argument	Type	Description
value	string	The string value from which to return the date.

Returns

Type	Description
date	The string cast to a date.

DECIMAL

This function casts a literal value to a decimal data type.

Syntax

```
DECIMAL(value)
```

Argument	Type	Description
value	literal	The literal value to convert to a decimal type.

Returns

Type	Description
decimal	The literal value cast to a decimal type.

DOUBLE

This function casts a literal value to a double data type.

Syntax

```
DOUBLE(value)
```

Argument	Type	Description
value	literal	The literal value to convert to a double type.

Returns

Type	Description
double	The literal value cast to a double type.

DURATION

This function returns an xsd:duration value from the given numeric or duration value.

Syntax

```
DURATION(value)
```

Argument	Type	Description
value	duration, numeric	The duration or numeric value from which to return an xsd:duration.

Returns

Type	Description
duration	The duration value.

DURATIONFORMAT

This function converts a duration or numeric value to a string in the specified duration format.

Syntax

```
DURATIONFORMAT(value [, format ])
```

Argument	Type	Description
value	duration, numeric	The duration or numeric value to format.

Argument	Type	Description
format	string	An optional value that specifies the format to use for the resulting duration string. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **H** for hours • **m** for minutes • **s** for seconds • **S** for milliseconds • **'text'** for arbitrary text content If `format` is not specified, `H:mm:ss.SSS` is used.

Returns

Type	Description
string	The duration as a string.

ENCODE_FOR_URI

This function encodes the specified string as a URI and returns a string in URI format.

Syntax

```
ENCODE_FOR_URI(text)
```

Argument	Type	Description
text	string	The string value to encode as a URI.

Returns

Type	Description
string	The string as a URI.

FLOAT

This function casts a literal value to a float data type.

Syntax

```
FLOAT(value)
```

Argument	Type	Description
value	literal	The literal value to convert to a float type.

Returns

Type	Description
float	The literal value cast to a float type.

FORMATDATE

This function converts a numeric or date value into a string with the specified date format.

Syntax

```
FORMATDATE(value, format)
```

Argument	Type	Description
value	date, numeric	The date or numeric value to convert to a string in the specified date `format`.
format	string	The format to use for the resulting date string. Anzo supports

Argument	Type	Description
		Pattern Tokens for defining the format: • **y** for year digits • **M** for months • **d** for days • **'text'** for arbitrary text content For example, `"yyyy.MM.dd"` or `"dd/MM/yyyy"`.

Returns

Type	Description
string	The date as a string.

FORMATFRACTION

This function converts a numeric value into a fraction string.

Syntax

```
FORMATFRACTION(value [, tolerance ] [, whole_number ])
```

Argument	Type	Description
value	numeric	The numeric value to convert to fraction text.
tolerance	double	An optional double value that specifies the precision of the fraction. The default value is `0.0001`. The resulting fractional representation is the original value + or - the `tolerance`. The smaller the tolerance, the more precise the fraction is. For example, `399/800` vs. `1/2`.

Argument	Type	Description
whole_number	boolean	An optional boolean value that specifies whether to include whole numbers in the result. For example, if `true`, the result would be formatted like `1 2/3` instead of `5/3`.

Returns

Type	Description
string	The fraction string.

FORMATNUMBER

This function casts a numeric value to a string in the specified format.

Syntax

```
TEXT(value, format)
```

Argument	Type	Description
value	numeric	The numeric value to convert to a string.
format	string	A text string that specifies the format to follow when converting the `value` to a string. Anzo supports Java Decimal Format.

Returns

Type	Description
string	The numeric value as a string.

INT

This function casts a literal value to an int data type.

Syntax

```
INT(value)
```

Argument	Type	Description
value	literal	The literal value to convert to an int type.

Returns

Type	Description
int	The literal value cast to an int type.

INTEGER

This function casts a literal value to an integer data type.

Syntax

```
INTEGER(value)
```

Argument	Type	Description
value	literal	The literal value to convert to an integer type.

Returns

Type	Description
integer	The literal value cast to an integer type.

LONG

This function casts a literal value to a long data type.

Syntax

```
LONG(value)
```

Argument	Type	Description
value	literal	The literal value to convert to a long type.

Returns

Type	Description
long	The literal value cast to a long type.

PARSEDATETIME

This function attempts to convert the given string or plain literal to a date, time, or dateTime value. For values that do not include a timezone, Anzo stores them in GMT. Values that include a timezone are stored as the appropriate value in GMT for that timezone.

Syntax

```
PARSEDATETIME(value [, output_type ] [, format ])
```

Argument	Type	Description
value	string, literal	The string or plain literal value to convert to a date, time, or dateTime.
output_type	URI	An optional URI (`xsd:date`, `xsd:time`, or `xsd:dateTime`) that specifies the type of value to return. If `output_type` is not specified, dateTime is returned.
format	string	An optional string that species the format to use for the resulting date, time, or dateTime value. Anzo supports Pattern Tokens for defining the format: • **y** for year digits • **M** for months

Argument	Type	Description
		• **d** for days • **H** for hours • **m** for minutes • **s** for seconds • **S** for milliseconds • **'text'** for arbitrary text content For example, `"yyyy.MM.dd HH:mm"` or `"dd/MM/yyyy HH:mm:ss"`.

Returns

Type	Description
date, time, or dateTime	The conversion of the string to the desired type.

RAD

This function converts to radians an angle value that is in degrees.

Syntax

```
RAD(angle)
```

Argument	Type	Description
angle	double	The angle value to convert to radians.

Returns

Type	Description
double	The angle in radians.

SERIALIZE

This function creates a literal value from the string representation of the specified RDF term type value.

Syntax

```
SERIALIZE(value)
```

Argument	Type	Description
value	RDF term	The literal, URI, or blank node value for which to generate a plain literal.

Returns

Type	Description
string	The string representation of the input term.

SHORT

This function casts a literal value to a short data type.

Syntax

```
SHORT(value)
```

Argument	Type	Description
value	literal	The literal value to convert to a short type.

Returns

Type	Description
short	The literal value cast to a short type.

STR

This function casts the specified RDF term type value to a string.

Syntax

```
STR(value)
```

Argument	Type	Description
value	RDF term	The literal, URI, or blank node value to convert to a string.

Returns

Type	Description
string	The term cast to a string type.

TEXT

This function casts a numeric or dateTime value to a string in the specified format.

Syntax

```
TEXT(value, format)
```

Argument	Type	Description
value	numeric, dateTime	The numeric or datetime value to convert to a string.
format	string	A text string that specifies the format to follow when converting the `value` to a string. For numeric values, Anzo supports Java Decimal Format. For dateTime values, Simple Date Format is supported.

Returns

Type	Description
string	The string in the specified format.

TIMEVALUE

This function converts the specified RDF term type value to an xsd:time value.

Syntax

```
TIMEVALUE(value)
```

Argument	Type	Description
value	RDF term	The literal, URI, or blank node value to convert to a time value.

Returns

Type	Description
time	The conversion of the term to a time value.

TOURI

This function casts a string literal value to a URI.

Syntax

```
TOURI(text)
```

Argument	Type	Description
text	string	The string literal to cast to a URI.

Returns

Type	Description
URI	The literal value as a URI.

UUID

This function generates a Universally Unique Identifier (UUID).

Syntax

```
UUID()
```

Returns

Type	Description
URI	The UUID.

xsd:date

This function converts the specified string, date, or dateTime value to an xsd:date.

Syntax

```
xsd:date(value)
```

Argument	Type	Description
value	string, date, dateTime	The string, date, or dateTime value to convert to an xsd:date.

Returns

Type	Description
date	The input values converted to dates.

Logical Functions

This topic describes the logical functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- AND: Evaluates two logical expressions and returns true if both expressions are true.
- BOUND: Evaluates whether an RDF term type is bound.
- CASE: Evaluates a series of conditions for the specified value and returns the matching result.
- COALESCE: Evaluates a number of expressions and returns the results for the first expression that is bound and does not raise an error.
- EQUAL: Evaluates whether two values are equal.
- IF: Evaluates a condition and returns the specified result depending on the outcome of the test.
- IFERROR: Synonym for COALESCE.
- IN: Evaluates whether the specified RDF term is found in any of the given test values.
- NOT: Evaluates whether the specified logical expression is not true.
- NOT_EQUAL: Evaluates whether two values are not equal.
- NOT_IN: Evaluates whether the specified RDF term is not found in any of the given test values.

- OR: Evaluates two logical expressions and returns true if at least one of the expressions is true.
- PARTITIONINDEX: Returns the zero-based index of the bucket in which the specified value falls.
- SAMETERM: Evaluates whether two RDF term type values are the same.
- UNBOUND: Extends the SPARQL UNDEF functionality to enable users to include an undefined value as a function argument.

AND

This function evaluates two logical expressions. If both expressions are true, the function returns `true`. If one or both arguments are false, the function returns `false`.

Syntax

```
AND(logical_expression1, logical_expression2)
```

Argument	Type	Description
logical_expression1	evaluates to boolean	The first logical expression to evaluate.
logical_expression2	evaluates to boolean	The second logical expression to evaluate.

Returns

Type	Description
boolean	`True` if both conditions are true and `false` if either condition is false.

BOUND

This function evaluates whether the specified RDF term has a value bound to it.

Syntax

```
BOUND(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate.

Returns

Type	Description
boolean	`True` if the term is bound and `false` if it is not.

CASE

This function evaluates a series of conditions for the specified value and returns the matching result. CASE acts like an IF-THEN-ELSE statement.

Syntax

```
CASE(test_value, condition1, [ condition2 ] [, conditionN ]
     result1 [, result2 ] [, resultN ] [, default ])
```

Argument	Type	Description
test_value	RDF term	The literal, URI, or blank node value to compare to the list of conditions (`condition1-N`).
condition1–N	RDF term	The conditions to be evaluated in the order that they are specified. Once a condition evaluates to true, the corresponding result is returned and the remaining conditions are not evaluated.
result1–N	RDF term	The results to return for the specified conditions.
default	RDF term	An optional value to be returned if none of the specified conditions pass.

Returns

Type	Description
RDF term	The specified result according to the evaluation of the conditions.

COALESCE

This function evaluates a number of expressions and returns the results for the first expression that is bound and does not raise an error.

Syntax

```
COALESCE(expression1 [, expression2 ] [, expressionN ] )
```

Argument	Type	Description
expression1–N	RDF term	The literal, URI, or blank node expressions to evaluate.

Returns

Type	Description
RDF term	The result of the first expression that is bound and does not error.

EQUAL

This function evaluates whether value1 is equal to value2.

Syntax

```
EQUAL(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`.

Argument	Type	Description
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if the values are equal and `false` if they are not.

IF

This function evaluates a condition and returns the specified result depending on the outcome of the test. If the condition evaluates to true, the first result is returned. If the condition evaluates to false, the second result is returned. And if the condition results in an error, the third result is returned.

Syntax

```
IF(logical_expression, true_result, false_result [, error_result ])
```

Argument	Type	Description
logical_ expression	evaluates to boolean	The condition that evaluates to true or false.
true_result	RDF term	The value that defines the result to return if the condition evaluates to true.
false_result	RDF term	The value that defines the result to return if the condition evaluates to false.
error_result	RDF term	An optional value that defines the result to return if the condition evaluates to an error. If the condition results in an error and `error_result` is not specified, `logical_expression (error)` is returned.

Returns

Type	Description
RDF term	The result based on the evaluation of the condition.

IFERROR

This function is a synonym for COALESCE.

IN

This function evaluates whether the specified RDF term type value is found in any of the given test values.

Syntax

```
IN(term, test_value1 [, test_value2 ] [, test_valueN])
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to look for in the test values.
test_value1–N	RDF term	The literal, URI, or blank node values to look for the specified `term` in.

Returns

Type	Description
boolean	`True` if the given term is found in the test values and `false` if it is not.

NOT

This function evaluates whether the specified logical expression is not true.

Syntax

```
NOT(logical_expression)
```

Argument	Type	Description
logical_expression	evaluates to boolean	The condition to evaluate.

Returns

Type	Description
boolean	`True` if the condition is false and `false` if it is true.

NOT_EQUAL

This function evaluates whether value1 is not equal to value2.

Syntax

```
NOT_EQUAL(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 is not equal to value2. `False` if they are equal.

NOT_IN

This function evaluates whether the specified RDF term type value is not found in any of the given test values.

Syntax

```
IN(term, test_value1 [, test_value2 ] [, test_valueN])
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to look for in the test values.
test_value1–N	RDF term	The literal, URI, or blank node values in which to look for the specified `term`.

Returns

Type	Description
boolean	`True` if the given term is not in the test values and `false` if it is found in the test values.

OR

This function evaluates two logical expressions. If at least one expression is true, the function returns `true`. If both expressions are false, the function returns `false`.

Syntax

```
OR(logical_expression1, logical_expression2)
```

Argument	Type	Description
logical_expression1	evaluates to boolean	The first logical expression to evaluate.

Argument	Type	Description
logical_expression2	evaluates to boolean	The second logical expression to evaluate.

Returns

Type	Description
boolean	`True` if one or both conditions are true and `false` if both conditions are false.

PARTITIONINDEX

This function returns the zero-based index of the bucket in which the specified value falls. Buckets start at the specified `start` value and are sized according to the specified `interval`. The first bucket is `[start, start+interval)`. That means it is closed on the low end and open on the high end. PARTITIONINDEX returns less than 0 if the value does not fall into any bucket, such as when the given `value` is less than `start` or if the comparison is indeterminate for date and time data types.

Syntax

```
PARTITIONINDEX(value, start, interval)
```

Argument	Type	Description
value	literal	The literal value for which to determine the zero-based index.
start	literal	The literal value that indicates the start of the first bucket.
interval	literal	The literal value that specifies the size of the bucket.

Returns

Type	Description
long	The zero-based index of the bucket in which the specified value exists.

SAMETERM

This function evaluates whether two RDF term type values are the same.

Syntax

```
SAMETERM(term1, term2)
```

Argument	Type	Description
term1	RDF term	The first literal, URI, or blank node value to compare.
term2	RDF term	The literal, URI, or blank node value to compare to `term1`.

Returns

Type	Description
boolean	`True` if the terms are the same and `false` if they are not.

UNBOUND

This function is like the SPARQL UNDEF keyword but extends that functionality to enable users to include an undefined value as a function argument, as UNDEF is only supported in VALUES clauses.

Syntax

```
UNBOUND()
```

Example

The following example statement incorporates UNBOUND to return null if the specified condition (`?x > 5`) fails:

```
BIND(IF(?x > 5 , "Win", UNBOUND()) as ?testResult)
```

In this case, ?testResult is bound if ?x is greater than 5. If ?x is not greater than 5, ?testResult is not bound.

Returns

Type	Description
RDF term	The specified result according to the evaluation of the condition.

Informational or Testing Functions

This topic describes the functions in Anzo that retrieve information from your values and let you ask questions about them or test whether the values match expectations.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- CONTAINS: Evaluates whether the specified string contains the given pattern.
- GE: Evaluates whether one value is greater than or equal to (>=) another value.
- GT: Evaluates whether one value is greater than (>) another value.
- ISBLANK: Evaluates whether the given RDF term is a blank node.
- ISDATATYPE: Evaluates whether the given literal value is typed as the specified data type.
- ISERROR: Tests whether the given RDF term evaluates to an error. **Only valid in queries against the Anzo System Datasource or other volume**.
- ISIRI: Evaluates whether the given RDF term is an IRI.
- ISLITERAL: Evaluates whether the given RDF term is a literal value.
- ISNUMERIC: Evaluates whether the given RDF term is a numeric literal value.
- ISURI: Evaluates whether the given RDF term is a URI.
- LANG: Returns any language tags that are included with strings.
- LANGMATCHES: Evaluates whether a string includes a language tag that matches the specified language range.

- LE: Evaluates whether one value is less than or equal to (<=) another value.
- LOCALNAME: Retrieves the local name from the given URI.
- LT: Evaluates whether one value is less than (<) another value.
- METADATAGRAPHURI: Returns the metadata graph URI for the given URI.
- NAMESPACE: Retrieves the namespace for the specified URI.
- SAMETERM: Evaluates whether two RDF term type values are the same.

CONTAINS

This function evaluates whether the specified strings contain the given pattern. Results are grouped under "true" or "false."

Syntax

```
CONTAINS(text, pattern)
```

Argument	Type	Description
text	string	The string value that you want to check against the specified pattern.
pattern	string	The string pattern that you want to look for in the supplied text.

Returns

Type	Description
boolean	`True` if the strings contain the pattern and `false` if they do not.

GE

This function evaluates whether value1 is greater than or equal to (>=) value2.

Syntax

```
GE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be checked to see if it is greater than or equal to value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 >= value2. `False` if not.

GT

This function evaluates whether value1 is greater than (>) value2.

Syntax

```
GE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be checked to see if it is greater than value2.
value2	numeric, boolean, dateTime,	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Argument	Type	Description
	RDF term	

Returns

Type	Description
boolean	`True` if value1 > value2. `False` if not.

ISBLANK

This function evaluates whether the given RDF term value is a blank node. It returns `true` if it is a blank node or `false` if it is not.

Syntax

```
ISBLANK(value)
```

Argument	Type	Description
value	RDF term	The literal, URI, or blank node value to test and determine if it is a blank node.

Returns

Type	Description
boolean	`True` if the term is a blank node and `false` if it is not.

ISDATATYPE

This function evaluates whether the given literal value is typed as the specified data type. It returns `true` if the value is typed as the provided type or `false` if it is not.

Syntax

```
ISDATATYPE(value, datatype_uri)
```

Argument	Type	Description
value	literal	The literal value that you want to test against the `datatype_uri`.
datatype_uri	URI	The URI for the data type that you want to test the `value` against.

Returns

Type	Description
boolean	`True` if the literal is typed as specified data type and `false` if it is not.

ISERROR

This function tests whether the given RDF term evaluates to an error. It returns `true` if it does evaluate to an error or `false` if it does not.

> **Note**
> The ISERROR function is only for use in queries that are run against the Anzo System Datasource or other volume. It is invalid for graphmart queries.

Syntax

```
ISERROR(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate for an error.

Returns

Type	Description
boolean	`True` if the term evaluates to an error and `false` if it does not.

ISIRI

This function evaluates whether the given RDF term type value is an IRI. It returns `true` if the value is an IRI or `false` if it is not.

Syntax

```
ISIRI(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate whether it is an IRI.

Returns

Type	Description
boolean	`True` if the term is an IRI and `false` if it is not.

ISLITERAL

This function evaluates whether the given RDF term type value is a literal value. It returns `true` if the value is a literal or `false` if it is not.

Syntax

```
ISLITERAL(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate whether it is a literal.

Returns

Type	Description
boolean	`True` if the term is a literal value and `false` if it is not.

ISNUMERIC

This function evaluates whether the given RDF term type value is a numeric literal. It returns `true` if the value is a numeric literal or `false` if it is not.

Syntax

```
ISNUMERIC(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate whether it is a numeric literal.

Returns

Type	Description
boolean	`True` if the term is a numeric literal and `false` if it is not.

ISURI

This function evaluates whether the given RDF term type value is a URI. It returns `true` if the value is a URI or `false` if it is not.

Syntax

```
ISURI(term)
```

Argument	Type	Description
term	RDF term	The literal, URI, or blank node value to evaluate whether it is a URI.

Returns

Type	Description
boolean	`True` if the term is a URI and `false` if it is not.

LANG

This function returns any language tags that are included in the string. The results are grouped by each language tag or by "blank" if a value does not have a language tag.

Syntax

```
LANG(text)
```

Argument	Type	Description
text	string	The string to search for language tags.

Returns

Type	Description
string	The found language tags.

LANGMATCHES

This function tests whether a string includes a language tag that matches the specified language range.

Syntax

```
LANGMATCHES(text, language_range)
```

Argument	Type	Description
text	string	The string to evaluate.
language_range	string	The language tag to match in the `text`.

Example

```
LANGMATCHES(LANG(?prop),"en")
```

Returns

Type	Description
boolean	`True` if strings include a language tag that matches the range and `false` if they do not.

LE

This function evaluates whether value1 is less than or equal to (<=) value2.

Syntax

```
LE(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value2`. This is the value that will be evaluated to see if it is less than or equal to value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, literal, URI, or blank node value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 <= value2. `False` if not.

LOCALNAME

This function retrieves the local name from the given URI.

Syntax

```
LOCALNAME(uri)
```

Argument	Type	Description
uri	URI	The URI from which to retrieve the local name.

Returns

Type	Description
string	The local name.

LT

This function evaluates whether value1 is less than (<) value2.

Syntax

```
LT(value1, value2)
```

Argument	Type	Description
value1	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, or RDF term type value to compare to `value2`. This is the value that will be evaluated to see if it is less than value2.
value2	numeric, boolean, dateTime, RDF term	The number, boolean, datetime, or RDF term type value to compare to `value1`.

Returns

Type	Description
boolean	`True` if value1 < value2. `False` if not.

METADATAGRAPHURI

This function returns the metadata graph URI for the specified URI.

Syntax

```
METADATAGRAPHURI(uri)
```

Argument	Type	Description
uri	URI	The URI for which you want to return the corresponding metadata graph URI.

Returns

Type	Description
URI	The metadata graph URI.

NAMESPACE

This function retrieves the namespace for the given URI.

Syntax

```
NAMESPACE(uri)
```

Argument	Type	Description
uri	URI	The URI from which to retrieve the namespace.

Returns

Type	Description
string	The namespace.

SAMETERM

This function evaluates whether two RDF term type values are the same.

Syntax

```
SAMETERM(term1, term2)
```

Argument	Type	Description
term1	RDF term	The first literal, URI, or blank node value to compare.
term2	RDF term	The literal, URI, or blank node value to compare to `term1`.

Returns

Type	Description
boolean	`True` if the terms are the same and `false` if they are not.

Hash Functions

This topic describes the hash functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- MD5: Returns the MD5 checksum of a string as a hexadecimal string.
- SHA1: Calculates the SHA-1 digest of a string value.
- SHA224: Calculates the SHA-224 digest of a string value.
- SHA256: Calculates the SHA-256 digest of a string value.
- SHA384: Calculates the SHA-384 digest of a string value.
- SHA512: Calculates the SHA-512 digest of a string value.

MD5

This function returns the MD5 checksum of a string as a hexadecimal string.

Syntax

```
MD5(text)
```

Argument	Type	Description
text	string	The string for which to return the MD5 checksum.

Returns

Type	Description
string	The hexadecimal string.

SHA1

This function calculates the SHA-1 digest of a string.

Syntax

```
SHA1(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-1 digest.

Returns

Type	Description
string	The SHA-1 digest.

SHA224

This function calculates the SHA-224 digest of a string.

Syntax

```
SHA224(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-224 digest.

Returns

Type	Description
string	The SHA-224 digest.

SHA256

This function calculates the SHA-256 digest of a string.

Syntax

```
SHA256(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-256 digest.

Returns

Type	Description
string	The SHA-256 digest.

SHA384

This function calculates the SHA-384 digest of a string.

Syntax

```
SHA384(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-384 digest.

Returns

Type	Description
string	The SHA-384 digest.

SHA512

This function calculates the SHA-512 digest of a string.

Syntax

```
SHA512(text)
```

Argument	Type	Description
text	string	The string for which to calculate the SHA-512 digest.

Returns

Type	Description
string	The SHA-512 digest.

Window Aggregate and Ranking Functions

Window aggregates operate on a particular partition or window of the result set. Unlike grouped aggregate functions that group the result set and return a single row, window aggregates retain the resulting rows and return a value for each row. For example, using the grouped aggregate SUM function to add the total number of tickets sold in a year returns one value: the total number of tickets sold for the year. By using WINDOW_SUM, the results could be partitioned by month so that the query returns 12 values: the sum of the number of tickets sold in each month of the year. This topic describes the window aggregate functions in Anzo.

Typographical Conventions

This documentation uses the following conventions in function syntax:

- `CAPS`: Although SPARQL is case-insensitive, function names and other keywords are written in uppercase for readability.
- `[ argument ]`: Brackets are used to indicate optional arguments. Arguments without brackets are required.

Functions

- WINDOW_AVG: Calculates the average value of each group of values.
- WINDOW_COUNT: Counts the number of values in each group of values.
- WINDOW_MAX: Calculates the maximum value of each group of values.
- WINDOW_MIN: Calculates the minimum value of each group of values.
- WINDOW_NTILE: Divides the rows in the partition into the specified number of ranked groups and returns the group that each value belongs to.
- WINDOW_PERCENTILE: Divides the rows in the partition into 100 ranked groups and returns the group that each value belongs to.
- WINDOW_PERCENTILE_CONT: Calculates a percentile based on the continuous distribution of the specified groups of values.

- WINDOW_PERCENTILE_DISC: Calculates a percentile based on the discrete distribution of the specified groups of values.
- WINDOW_PRODUCT: Calculates the product of each group of values.
- WINDOW_QUARTILE: Divides the rows in the partition into four ranked groups and returns the group that each value belongs to.
- WINDOW_SUM: Calculates the sum of each group of values.

WINDOW_AVG

This function calculates the average value of each group of values.

Syntax

```
WINDOW_AVG(value [, partition_over ] [, order_by ] [, order ]
          [, start_frame_type ] [, start_frame_value ]
          [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_`

Argument	Type	Description
		`value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value • FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The average values.

WINDOW_COUNT

This function counts the number of values in each group of values.

Syntax

```
WINDOW_COUNT(value [, partition_over ] [, order_by ] [, order ]
             [, start_frame_type ] [, start_frame_value ]
             [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value

Argument	Type	Description
		• FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The counts of values.

WINDOW_MAX

This function calculates the maximum value of each group of values.

Syntax

```
WINDOW_MAX(value [, partition_over ] [, order_by ] [, order ]
          [, start_frame ] [, start_frame_type ] [, start_frame_value ]
          [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value • FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values:

Argument	Type	Description
		• CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The maximum values.

WINDOW_MIN

This function calculates the minimum value of each group of values.

Syntax

```
WINDOW_MIN(value [, partition_over ] [, order_by ] [, order ]
          [, start_frame_type ] [, start_frame_value ]
          [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows

Argument	Type	Description
		within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value • FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value

Argument	Type	Description
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The minimum values.

WINDOW_NTILE

This function divides the rows in the partition into the specified number of ranked groups and returns the group that each value belongs to.

Syntax

```
WINDOW_NTILE(ntile, value, order_by [, partition_over ])
```

Argument	Type	Description
ntile	int	Required argument that specifies the number of ranking groups.
value	numeric	Required argument that defines the groups of values to operate on.
order_by	variable	Required argument that defines the order or sequence of rows within each partition.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.

Returns

Type	Description
int	The group that the values belong to.

WINDOW_PERCENTILE

This function divides the rows in the partition into 100 ranked groups and returns the group that each value belongs to.

Syntax

```
WINDOW_PERCENTILE(value, order_by [, partition_over ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
order_by	variable	Required argument that defines the order or sequence of rows within each partition.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.

Returns

Type	Description
int	The group that the values belong to.

WINDOW_PERCENTILE_CONT

This function calculates a percentile based on the continuous distribution of the specified groups of values. The returned value is interpolated and may not be equal to any of the values in the group.

Syntax

```
WINDOW_PERCENTILE_CONT(percentile, value, order_by [, partition_over ])
```

Argument	Type	Description
percentile	int	Required argument that specifies the percentile for the calculation.
value	numeric	Required argument that defines the groups of values to operate on.
order_by	variable	Required argument that defines the order or sequence of rows within each partition.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.

Returns

Type	Description
int	The interpolated percentiles.

WINDOW_PERCENTILE_DISC

This function calculates a percentile based on the discrete distribution of the specified groups of values.

Syntax

```
WINDOW_PERCENTILE_DISC(percentile, value, order_by [, partition_over ])
```

Argument	Type	Description
percentile	int	Required argument that specifies the percentile for the calculation.

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
order_by	variable	Required argument that defines the order or sequence of rows within each partition.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.

Returns

Type	Description
int	The percentiles based on the discrete distribution of the groups.

WINDOW_PRODUCT

This function calculates the product of each group of values.

Syntax

```
WINDOW_PRODUCT(value [, partition_over ] [, order_by ] [, order ]
              [, start_frame_type ] [, start_frame_value ]
              [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows

Argument	Type	Description
		within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value • FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value

Argument	Type	Description
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The products of the groups.

WINDOW_QUARTILE

This function divides the rows in the partition into four ranked groups and returns the group that each value belongs to.

Syntax

```
WINDOW_QUARTILE(value, order_by [, partition_over ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
order_by	variable	Required argument that defines the order or sequence of rows within each partition.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.

Returns

Type	Description
int	The group that the values belong to.

WINDOW_SUM

This function calculates the sum of each group of values.

Syntax

```
WINDOW_SUM(value [, partition_over ] [, order_by ] [, order ]
          [, start_frame_type ] [, start_frame_value ]
          [, end_frame_type ] [, end_frame_value ])
```

Argument	Type	Description
value	numeric	Required argument that defines the groups of values to operate on.
partition_over	variable	Optional argument that partitions the results into groups of rows. If you do not include `partition_over`, the partition becomes the entire set identified by `value`.
order_by	variable	Optional argument that defines the order or sequence of rows within each partition.
order	boolean	Optional argument that controls whether the order is ascending or descending. When `true`, the order is ascending. When `false`, the order is descending.
start_frame_type	string	When `order_by` is specified, the optional `start_frame_type`, `start_frame_value`, `end_frame_type`, and `end_frame_value` arguments can be included to refine the set of rows to include in the partitions or groups. The `start_frame_type` argument defines the starting row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED PRECEDING • PRECEDING start_frame_value

Argument	Type	Description
		• FOLLOWING start_frame_value
start_frame_value	int	Optional argument that specifies the starting row based on the `start_frame_type` value.
end_frame_type	string	This argument defines the ending row of the partition and can be one of the following values: • CURRENT ROW • UNBOUNDED FOLLOWING • PRECEDING end_frame_value • FOLLOWING end_frame_value
end_frame_value	int	Optional argument that specifies the ending row based on the `end_frame_type` value.

Returns

Type	Description
int	The sum of each group.

Example

The example below first creates data by running the following INSERT DATA query in a graphmart Query Step:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ex: <http://example.com/csi#>
INSERT DATA {
  GRAPH ${targetGraph} {
    ex:sale1 ex:date "2014-05-23T10:20:13"^^xsd:dateTime ; ex:volume 15 .
    ex:sale2 ex:date "2014-06-23T10:20:13"^^xsd:dateTime ; ex:volume 3 .
    ex:sale3 ex:date "2014-06-23T10:22:13"^^xsd:dateTime ; ex:volume 35 .
```

```
    ex:sale4 ex:date "2014-07-23T10:20:13"^^xsd:dateTime ; ex:volume 66 .
    ex:sale5 ex:date "2014-09-23T10:20:13"^^xsd:dateTime ; ex:volume 19 .
    ex:sale6 ex:date "2014-11-23T10:20:13"^^xsd:dateTime ; ex:volume 33 .
    ex:sale7 ex:date "2014-12-23T10:20:13"^^xsd:dateTime ; ex:volume 12 .
  }
}
```

The following query against the new data uses the WINDOW_SUM function to return the total volume of sales for each month:

```
PREFIX ex: <http://example.com/csi#>
SELECT DISTINCT ?Month ?Total_Volume
WHERE {
  {
    SELECT
      ?Month (WINDOW_SUM(?o, ?Month) AS ?Total_Volume)
    WHERE {
      ?s ex:volume ?o .
      ?s ex:date ?date .
      BIND (MONTH(?date) AS ?Month)
    }
  }
}
```

The query returns the following results:

```
Month | Total_Volume
------+-------------
5     |     15
6     |     38
7     |     66
9     |     19
11    |     33
12    |     12
```

Develop

The topics in this section provide information for developers.

In this section:

Anzo Rest API 1176

Anzo Java SDK 1233

Anzo Rest API

Anzo includes a REST API that you can use when developing applications or automation tasks. The API supports create, read, update, and delete operations on datasets, graphmarts, data layers, and steps. The interface also supports read and update operations for artifact access control lists, upload, download, and delete operations for models, run and cancel operations for unstructured pipelines, import and export operations for migration packages, and read operations for dynamic launch configurations and static AnzoGraph instances.

The topics in this section give an overview of the API, list the options for viewing the detailed documentation, provide instructions for enabling CORS, and describe the schemas for the data layer step types.

In this section:

Introduction to the API 1177

Viewing the API Documentation 1180

Enabling Cross-Origin Resource Sharing 1181

Step Type Schemas 1183

Introduction to the API

The API supports create, read, update, and delete operations on datasets, graphmarts, data layers, and steps. The interface also supports read and update operations for artifact access control lists, upload, download, and delete operations for models, run and cancel operations for unstructured pipelines, import and export operations for migration packages, and read operations for dynamic launch configurations and static AnzoGraph instances.

This topic gives a summary of the API request URL, endpoints and methods, URI encoding requirements, and error handling.

- Request URL
- Endpoints and Methods
- URI Encoding Requirements
- Error Handling

Request URL

The path in the URL that you use to access the REST API endpoints (as well as the documentation on your Anzo server) differs depending on the version of Anzo that is installed:

- **In 5.4.1 releases**, the URL is `https://<hostname>/api/v1/<endpoint>`.
- **In 5.4.2+ releases**, the URL is `https://<hostname>/api/<endpoint>`.

The `v1` in the path was removed in 5.4.2. The URL to access the documentation on your Anzo server was also changed from `https://<hostname>/api/v1/docs/index.html` in 5.4.1 to `https://<hostname>/api/docs/index.html` in 5.4.2.

Endpoints and Methods

There is an endpoint for each type of Anzo artifact or object (dataset, edition, model, graphmart, layer, step, etc.). Each endpoint supports a subset or all of the following methods:

- POST or PUT for create
- GET for reads
- PATCH for updates
- DELETE for delete

URI Encoding Requirements

When including a URI (e.g., a graphmart, layer, or step URI) in a request URL, the URI must be URL-encoded. The following example shows a layer URI, `http://cambridgesemantics.com/Layer/858c521bc7d84364a5a2112e38dc0b52`, that has been URL-encoded:

```
http%3A%2F%2Fcambridgesemantics.com%2FLayer%2F858c521bc7d84364a5a2112e38dc0b52
```

The encoded value should be used in the request URL. For example, the following request retrieves the status of the layer:

```
http://10.100.0.10:8080/api/layers/http%3A%2F%2Fcambridgesemantics.com%2FLayer%2F858c52
1bc7d84364a5a2112e38dc0b52/status
```

> **Note**
> URIs that appear in the body of a request do not need to be URL-encoded.

An error message such as the one below indicates that a request URL included a URI that was not URL-encoded:

```
{
  "summary": "Rest API Error: No handler found for GET
/api/graphmarts/http://cambridgesemantics.com/Graphmart/811ece67d61e436cb128a929797b68d
f",
  "detail": "No handler found for GET
/api/graphmarts/http://cambridgesemantics.com/Graphmart/811ece67d61e436cb128a929797b68d
f"
}
```

To resolve the error, encode the URI as a URL and then resend the request.

Error Handling

Errors returned from the API contain a summary and detailed message. Stack traces from the server are not included in API responses. Stack traces can be obtained via the server logs. The documentation describes the error codes and response details.

Viewing the API Documentation

An Open API definition is used to generate the API documentation. There are two ways to view the document:

- Access the document on your Anzo server at the following URL:

  ```
  https://<hostname>/api/docs/index.html
  ```

  > **Note**
  > In releases prior to 5.4.2, the URL to view the documentation on your server is `https://<hostname>/api/v1/docs/index.html`.

  Where <hostname> is the Anzo server DNS name or IP address. When you access the API documentation on your server, a **Try It Out** button is available for each request. Clicking Try It Out opens a request body that you can edit and execute against the server.

- Access the document on the Cambridge Semantics documentation website at the following URL:

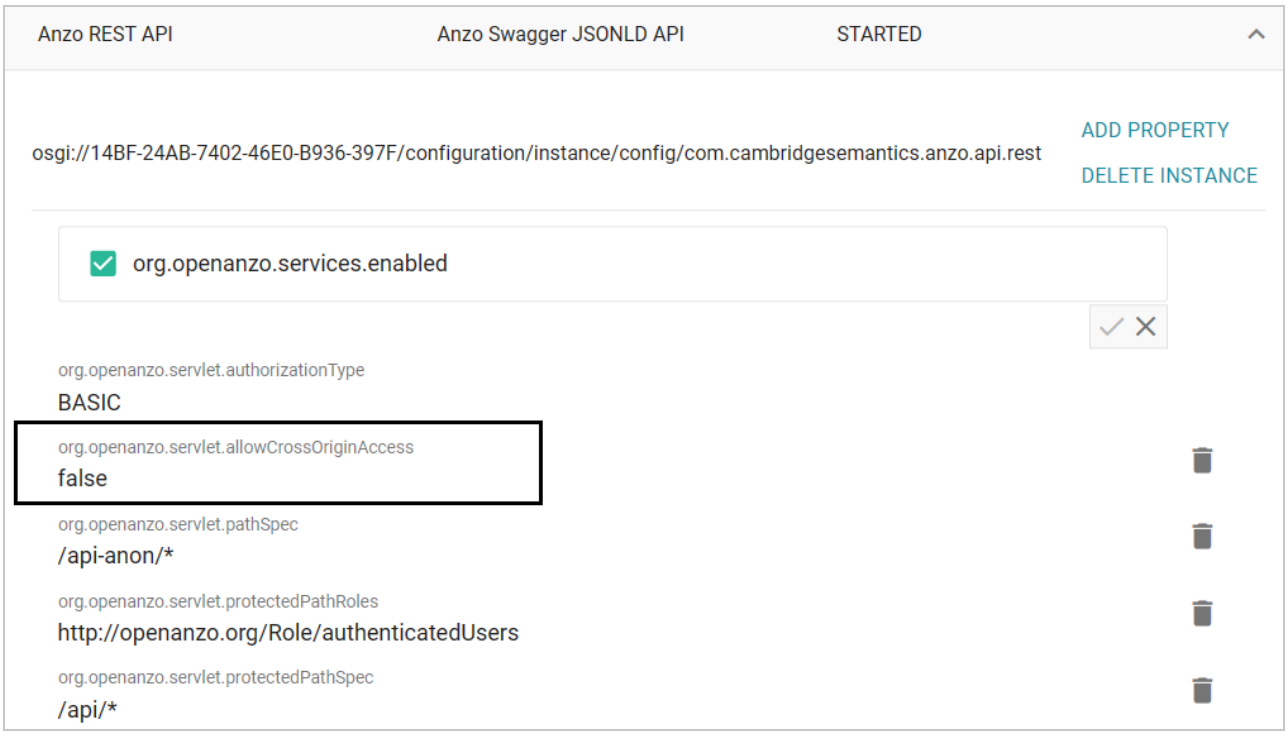
  https://docs.cambridgesemantics.com/anzo/v5.4/api/

  This version of the documentation is static and the Try It Out functionality is disabled. However, it remains available if you are unable to access the Anzo server.

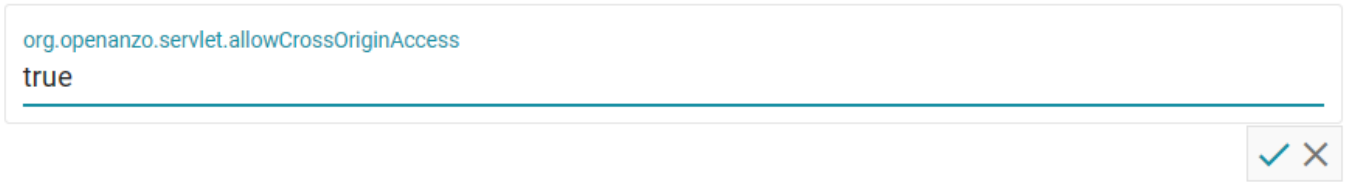
Enabling Cross-Origin Resource Sharing

By default, cross-origin resource sharing (CORS) is disabled for the Anzo REST API service. If you plan to access the API from a web client and need to enable cross-origin requests, follow the steps below.

1. In the Administration application, expand the **Servers** menu and click **Advanced Configuration**. Click **I understand and accept the risk**.
2. Search for the **Anzo REST API** bundle and view its details.
3. Click the **Services** tab and expand **Anzo REST API**.
4. Locate the **org.openanzo.servlet.allowCrossOriginAccess** property (shown in the image below).



5. Click the property to make it editable, and then change `false` to `true`.



6. Click the checkmark icon (✓) for that property to save the change.

7. Restart Anzo to apply the configuration change.

The Anzo API service is now configured to allow cross-origin requests. As an example of a CORS request, the following snippet from a React application reloads a graphmart via an API call. The request retrieves an authorization token and configures an Axios request object with the token and CORS-related headers:

```
export default async function reloadAnzo() {
  const authorization = getAnzoAuth(); //basicAuth
  console.log('Authorization', authorization);
  const config: AxiosRequestConfig = {
    headers: {
      Authorization: authorization,
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Credentials': true,
      'Access-Control-Allow-Methods': 'POST, OPTIONS',
    },
  };
  const url =
    'https://{{AnzoURL}}/api/v1/graphmarts/{{GraphmartIRI}}/reload';
  try {
    const success = await axios.post(url, config);
    console.log('return', success);
    return success;
  } catch (error) {
    console.log(error);
    return {error};
  }
}
```

Step Type Schemas

This section provides reference information for each type of step that can be created or updated via the Anzo REST API. For each step type in the list below, there is a JSON request that includes all of the step's body parameters, excluding the read-only options. Below the request is a description of the step's schema, including the read-only parameters.

In this section:

Direct Load Step 1183

Elasticsearch Indexing Step 1187

Elasticsearch Snapshot Step 1191

Export Step 1195

Load Dataset Step 1202

Pre-Compile Query Step 1207

Query-Driven Templated Step 1211

Query Step 1215

RDFS+ Inference Step 1219

Templated Step 1222

Validation Step 1227

Direct Load Step

This type of step loads data directly from an external source.

JSON Request

The following template shows the body of a JSON request that could be used in a Direct Load Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "title" : "string",
   "transformQuery" : "string",
   "incrementalData" : [ "string" ],
   "type" : "DirectLoadStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
              } ],
   "tagTitle" : [ "string" ]
}
```


Schema Details

The table below describes the Direct Load Step schema.

> **Tip**
> You can also see the Direct Load Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **DirectLoadStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.

Property	Format	Required?	Description
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as DirectLoadStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
title	"string"	Required	The name of the step.
transformQuery	"string"	Required	The SPARQL query to run.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "DirectLoadStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.

Property	Format	Require-d?	Description
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	This type of step automatically generates a managed model that is owned and managed by the data layer that contains the Direct Load Step. If you would like to associate additional models with this step, you can include the URIs for those models. For more information about managed models, see Managed Model Concepts.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.

Property	Format	Require-d?	Description
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Elasticsearch Indexing Step

This type of step creates an Elasticsearch index to associate with a layer.

JSON Request

The following template shows the body of a JSON request that could be used in an Elasticsearch Indexing Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "elasticsearchIndexingQuery" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "ElasticsearchIndexingStep",
```

```
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Elasticsearch Indexing Step schema.

> **Tip**
> You can also see the Elasticsearch Indexing Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **ElasticsearchIndexingStep**.

Property	Format	Requir-ed?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.

Property	Format	Requir-ed?	Description
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as ElasticsearchIndexingStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
elasticsearchIndexingQuery	"string"	Required	The SPARQL query to run for creating the Elasticsearch index.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "ElasticsearchIndexingStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.

Property	Format	Required?	Description
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.

Property	Format	Requir-ed?	Description
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Elasticsearch Snapshot Step

This type of step creates an Elasticsearch snapshot of the index associated with a layer.

JSON Request

The following template shows the body of a JSON request that could be used in an Elasticsearch Snapshot Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "gmLinkedDataset" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "ElasticsearchSnapshotStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
```

```
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Elasticsearch Snapshot Step schema.

> **Tip**
> You can also see the Elasticsearch Snapshot Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **ElasticsearchSnapshotStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.

Property	Format	Require-d?	Description
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as ElasticsearchSnapshotStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
gmLinkedDataset	"uri"	Required	The URI of the Linked Dataset Catalog entry that represents the target FLDS for the export. To get the catalog entry, you can retrieve data about the dataset and use the `catalogEntry` value.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "ElasticsearchSnapshotStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.

Property	Format	Require-d?	Description
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.

Property	Format	Require-d?	Description
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Export Step

This type of step exports the contents of a graphmart to an FLDS on disk.

JSON Request

The following template shows the body of a JSON request that could be used in an Export Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "doNotCreateEditionsOnExport" : true,
   "generateMetrics" : true,
   "exportBinaryStoreContents" : true,
   "overwriteFlds" : true,
   "alwaysMoveBinaryStore" : true,
   "gmLinkedDataset" : "string",
   "edition" : "string",
   "elasticsearchBulkSize" : 0,
   "elasticsearchBulkActions" : 0,
   "elasticsearchBulkMaxThreadsPerFlds" : 0,
   "elasticsearchBulkConcurrentRequests" : 0,
   "keepEsIndexOnline" : true,
   "elasticsearchBulkMaxFldsThreads" : 0,
   "maxComponentsInEdition" : 0,
   "elasticsearchIndexSettings" : "string",
   "exportElasticSearchContents" : true,
```

```
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "ExportStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Export Step schema.

> **Tip**
> You can also see the Export Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **ExportStep**.

Property	Forma-t	Requir-ed?	Description
uri (read-only)	"uri"	Auto-genera ted	The URI of the step.
creator (read-only)	"uri"	Auto-genera ted	The creator of the step.
created (read-only)	"dateTi	Auto-	The timestamp when the step was created.

Property	Format	Required?	Description
	me"	generated	
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as ExportStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
doNotCreateEditionsOnExport	boolean	Optional	Controls whether a new edition is created for the dataset each time the step is run.
generateMetrics	boolean	Optional	Controls whether a data profile is generated before the data is exported. If you load the exported files in the future, the data profile is also loaded.
exportBinaryStoreContents	boolean	Optional	Applies to exports of unstructured graphmarts and controls whether the binary store is exported along with the data.
overwriteFlds	boolean	Optional	Controls whether the existing FLDS is replaced with the exported files whenever the step is run or whether the exported files are added to the existing FLDS.

Property	Format	Required?	Description
			When `overwriteFlds` is `true`, Anzo archives the existing files in a new timestamped export subdirectory under the FLDS directory each time the step runs. If you add the exported dataset to a graphmart, only the latest version of the data will be loaded. When `overwriteFlds` is `false`, Anzo adds all of the exported datasets to a cumulative export directory under the FLDS directory. The dataset will contain the original files as well as all cumulative working editions. If you add this dataset to a graphmart, all of the data from all of the subdirectories will be loaded.
alwaysMoveBinaryStore	boolean	Optional	This option also applies to exports of unstructured graphmarts and controls whether the binary store is moved or copied during the export. Since the binary store can be large and have a nested structure, copying the data can take a very long time. Since moving the binary store is almost instantaneous, however, leaving this option set to `true` can reduce the time it takes to complete the export.
gmLinkedDataset	"uri"	Required	The URI of the Linked Dataset Catalog entry that represents the target FLDS for the export. To get the catalog entry, you can retrieve data about the dataset and use the `catalogEntry` value.

Property	Format	Required?	Description
edition	"uri"	Optional	The URI of the edition that will be created on export.
elasticsearchBulkSize	long	Optional	The maximum batch size in MB.
elasticsearchBulkActions	int	Optional	The maximum number of documents to include in each batch.
elasticsearchBulkMaxThreadsPerFlds	int	Optional	The maximum number of threads to use for indexing per FLDS.
elasticsearchBulkConcurrentRequests	int	Optional	The maximum number of bulk requests that can run concurrently.
keepEsIndexOnline	boolean	Optional	Controls whether the Elasticsearch index remains stored in Elasticsearch or is removed from Elasticsearch once it is exported.
elasticsearchBulkMaxFldsThreads	int	Optional	The maximum number of FLDSes to index concurrently.
maxComponentsInEdition	int	Optional	Controls the maximum number of components to retain in an edition. The default value is `0`, which means unlimited. If you specify a number in this field and the limit is reached, Anzo ages off the oldest components as new ones are created.
elasticsearchIndexSettings	"string"	Optional	A JSON-formatted list of any Elasticsearch-specific index settings to apply.

Property	Format	Required?	Description
exportElasticSearchContents	boolean	Optional	Indicates whether to export any Elasticsearch contents that are included in the graphmart.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "ExportStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the

Property	Format	Required?	Description
			data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.

Property	Format	Required?	Description
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Load Dataset Step

This type of step loads a system dataset or a dataset that is in the Datasets catalog.

JSON Request

The following template shows the body of a JSON request that could be used in a Load Dataset Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "skipEsLoadIfIndexExists" : true,
   "edition" : "string",
   "gmLinkedDataset" : "string",
   "watchFldsDirectory" : true,
   "maskedPredicate" : [ "string" ],
   "fldsComponent" : [ "string" ],
   "transformQuery" : "string",
   "title" : "string",
   "type" : "LoadDatasetStep",
   "incrementalData" : [ "string" ],
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
               "description" : "string",
```

```
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Load Dataset Step schema.

> **Tip**
> You can also see the Load Dataset Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **LoadDatasetStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as LoadDatasetStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.

Property	Format	Required?	Description
skipEsLoadIfIndexExists	boolean	Optional	This option applies to graphmarts with Elasticsearch indexes. It controls whether or not Anzo first checks to see if an index with the alias for the dataset already exists in Elasticsearch. If this setting is `true` and the index does exist, Anzo will not reload the index snaphsot into Elasticsearch.
edition	"uri"	Optional	If you want to load a certain edition in the dataset, you can include this property to specify the URI of the edition.
gmLinkedDataset	"uri"	Optional	The URI of the Linked Dataset Catalog entry that represents the target FLDS for the load. To get the catalog entry, you can retrieve data about the dataset and use the `catalogEntry` value.
watchFldsDirectory	boolean	Optional	Controls whether the FLDS directory is monitored for changes. If `true` and files change, Anzo marks the step (and layer) as needing a refresh.

Property	Format	Required?	Description
maskedPredicate	["uri", "..."]	Optional	To exclude certain triples from the load, you can specify a list of predicate URIs to filter out.
fldsComponent	["uri", "..."]	Optional	If you want to load specific components in the dataset, you can include this property to list the component URIs.
transformQuery	"string"	Optional	If you want to hand-pick the data to load, you can include this property to run a SPARQL query that inserts specific values or filters out certain values.
title	"string"	Required	The name of the step.
type	"string"	Required	The type of step: "LoadDatasetStep".
incrementalData	"string"	Optional	Incremental load data associated with the step.
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that

Property	Format	Required?	Description
			layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a

Property	Format	Required?	Description
			tag to the step without including a description.

Pre-Compile Query Step

This type of step runs the specified query immediately after the graphmart load so that the query is pre-compiled by AnzoGraph.

JSON Request

The following template shows the body of a JSON request that could be used in a Pre-Compile Query Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "validationFailsGraphmart" : true,
   "validationFailsLayer" : true,
   "validationQuery" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "ValidationStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Pre-Compile Query Step schema.

> **Tip**
> You can also see the Pre-Compile Query Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **PreCompileQueryStep**.

Property	Format	Requir-ed?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as PreCompileQueryStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
validationFailsGr	boolean	Optiona	If the query fails, this value controls whether the

Property	Format	Required?	Description
aphmart		l	entire graphmart load should fail.
validationFailsLayer	boolean	Optional	If the query fails, this value controls whether the layer should fail.
validationQuery	"string"	Required	The SPARQL query that the step should run.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "PreCompileQueryStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.

Property	Format	Requir-ed?	Description
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.

Property	Format	Requir-ed?	Description
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Query-Driven Templated Step

This type of step creates a reusable template for creating additional query steps.

JSON Request

The following template shows the body of a JSON request that could be used in a Query-Driven Templated Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "parametersTemplate" : [ "string" ],
   "template" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "QueryDrivenTemplatedStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
```

```
   "tags" : [ {
                "description" : "string",
                "title" : "string"
              } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Query-Driven Templated Step schema.

> **Tip**
> You can also see the Query-Driven Templated Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **QueryDrivenTemplatedStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as QueryDrivenTemplatedStep, Step, LayerChild, etc.

Property	Format	Require-d?	Description
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
parametersTemplate	"string"	Required	The query to run for determining the key-value pairs.
template	"string"	Required	The query template. In the query, include keys as parameters in the format `${key_name}`. The keys are replaced at runtime with the values defined for the key.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "QueryDrivenTemplatedStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.

Property	Format	Require-d?	Description
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.

Property	Format	Require-d?	Description
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Query Step

This type of step runs a SPARQL query that creates, cleans, conforms, or transforms data.

JSON Request

The following template shows the body of a JSON request that could be used in a Query Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "transformQuery" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "QueryStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
```

```
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Query Step schema.

> **Tip**
> You can also see the Query Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **QueryStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as QueryStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.

Property	Format	Require-d?	Description
transformQuery	"string"	Required	The SPARQL query to run.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "QueryStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Gr

Property	Format	Require-d?	Description
			aphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

RDFS+ Inference Step

This type of step uses RDFS and OWL rules to infer new information about your data based on the vocabularies in the existing data.

JSON Request

The following template shows the body of a JSON request that could be used in an RDFS+ Inference Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "inferenceRules" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "RDFSInferenceStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the RDFS+ Inference Step schema.

Tip

You can also see the RDFS+ Inference Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **RDFSInferenceStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as RDFSInferenceStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
inferenceRules	"string"	Optional	By default all inference rules are run. This property can be used to list a subset of rules to run or specific rules to exclude. For more information about the inference rules, see Infer New Data (RDFS+ Inference Step).

Property	Format	Require-d?	Description
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "RDFSInferenceStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored.

Property	Format	Require-d?	Description
			• "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Templated Step

This type of step creates a reusable query template based on key-value pairs.

JSON Request

The following template shows the body of a JSON request that could be used in a Templated Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "template" : "string",
   "templateParameters" : [
     {
       "type": "TemplateParameterSet",
       "templateParameter": [
         {
           "type": "TemplateParameter",
           "parameterName": "key",
           "parameterValue": [ "uri" ],
         }
       ]
     }
   ],
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "TemplatedStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```

Schema Details

The table below describes the Templated Step schema.

Tip

You can also see the Templated Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **TemplatedStep**.

Property	Format	Required?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as TemplatedStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
template	"string"	Required	The query template. In the query, include keys as parameters in the format `${key_name}`. The keys are replaced at runtime with the values defined for the key.

Property	Format	Require-d?	Description
templateParameters	Array of objects	Required	The list of objects that define the key-value pairs.
templateParameter	Array of objects	Required	The list of key-value pairs that will be substituted for the `${key_name}` parameters in the query template. Each `templateParameter` must include `type`, `parameterName`, and `parameterValue`. The following snippet shows the format of template parameters. `"templateParameter": [ { "type": "TemplateParameter", "parameterName": "key", "parameterValue": [ "uri" ], } ], "templateParameter": [ { ... } ]`
parameterName	"string"	Required	The name of the key that is used in the query template.
parameterValue	["uri"]	Required	The value for the corresponding key (parameterName).
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.

Property	Format	Require-d?	Description
type	"string"	Required	The type of step: "TemplatedStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the

Property	Format	Require-d?	Description
			graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Validation Step

This type of step runs a validation query to ensure that data conforms to expectations.

JSON Request

The following template shows the body of a JSON request that could be used in a Validation Step PUT or PATCH request. It lists all of the step's required and optional body parameters but excludes the read-only options. The default values for each parameter are shown. Below the request (in Schema Details) is a table that describes the complete schema, including the read-only parameters. Clicking a link in the template takes you to the schema details for that parameter.

```
{
   "validationFailsLayer" : true,
   "validationDatasourceUri" : "string",
   "validationQuery" : "string",
   "validationFailsGraphmart" : true,
   "resultVar" : "string",
   "validationOrConditional" : "string",
   "title" : "string",
   "incrementalData" : [ "string" ],
   "type" : "ValidationStep",
   "enabled" : true,
   "contextProvider" : [ "string" ],
   "description" : "string",
   "ontology" : [ "string" ],
   "source" : [ "string" ],
   "ignoreLoadErrors" : true,
   "disableLoadCounts" : true,
   "preGenerateStatistics" : true,
   "tags" : [ {
                "description" : "string",
                "title" : "string"
            } ],
   "tagTitle" : [ "string" ]
}
```


Schema Details

The table below describes the Validation Step schema.

> **Tip**
> You can also see the Validation Step schema by expanding **Schemas** at the bottom of the Anzo REST API document and viewing **ValidationStep**.

Property	Format	Requir-ed?	Description
uri (read-only)	"uri"	Auto-generated	The URI of the step.

Property	Format	Requir-ed?	Description
creator (read-only)	"uri"	Auto-generated	The creator of the step.
created (read-only)	"dateTime"	Auto-generated	The timestamp when the step was created.
modifier (read-only)	"uri"	Auto-generated	The user who modified the step.
alltypes (read-only)	Array of strings	Optional	A list of the types related to the step, such as ValidationStep, Step, LayerChild, etc.
contextAttribute (read-only)	Array of strings	Optional	A list of any context attributes that are used.
validationFailsLayer	boolean	Optional	If validationOrConditional is `validation`, and the validation fails, this value controls whether the layer should fail.
validationDatasourceUri	"uri"	Optional	The source to perform the validation on.
validationQuery	"string"	Required	The SPARQL query that the step should run.
validationFailsGraphmart	boolean	Optional	If validationOrConditional is `validation`, and the validation fails, this value controls whether the entire graphmart load should fail.

Property	Format	Requir-ed?	Description
resultVar	"string"	Optional	The variable name to use to store the result from the query. This variable becomes available when configuring an execution condition for a layer or step.
validationOrConditional	"string"	Optional	The type of check to perform: `validation` or `condition`. A validation check validates the data according to the defined query and takes the action configured with validationFailsLayer and validationFailsGraphmart. A condition check takes the results of the query and associates it with the specified resultVar.
title	"string"	Required	The name of the step.
incrementalData	"string"	Optional	Incremental load data associated with the step.
type	"string"	Required	The type of step: "ValidationStep".
enabled	boolean	Optional	Controls whether the step is enabled or disabled.
contextProvider	["uri", "..."]	Optional	A list of any referenced context providers (the data source URI). You can retrieve data for the parent layer to get a list of providers for that layer.
description	"string"	Optional	A brief description of the step.

Property	Format	Requir-ed?	Description
ontology	["uri", "..."]	Optional	A list of any models to associate with this step.
source	["uri", "..."]	Required	The source data for the step. Options are any combination of the following values: • "http://cambridgesemantics.com/ontologies/Graphmarts#Self": The source is the data that is in this step's layer. • "http://cambridgesemantics.com/ontologies/Graphmarts#AllPrevious": The source is the data from all of the successful layers that precede this step's layer. Failed layers are ignored. • "http://cambridgesemantics.com/ontologies/Graphmarts#Previous": The source is the data that is in the one layer that precedes this step's layer. • "layer_uri": The source is a specific layer in the graphmart.
ignoreLoadErrors	boolean	Optional	Controls whether to ignore errors and proceed with the load or fail the step if there is an error.
disableLoadCounts	boolean	Optional	Controls whether Anzo periodically queries AnzoGraph to count the total number of statements that are processed. Disabling the load count decreases the number of queries that run during activation.

Property	Format	Requir-ed?	Description
preGenerateStatistics	boolean	Optional	Controls whether AnzoGraph generates statistics on the data before the step is run.
tags	Array of objects	Optional	Any tags on the step.
tagTitle	["string"]	Optional	A virtual property that is available for all objects. It lists the tags associated with the step or can be used to add a tag to the step without including a description.

Anzo Java SDK

This topic provides instructions for setting up an Anzo development environment using the Anzo software development kit (SDK) and Eclipse integrated development environment (IDE). The sample instructions below deploy the Anzo SDK in a Windows environment with Eclipse IDE for Java Developers Version 4.12.0. Anzo SDK and Eclipse can also be deployed on Linux and Mac operating systems.

Requirements

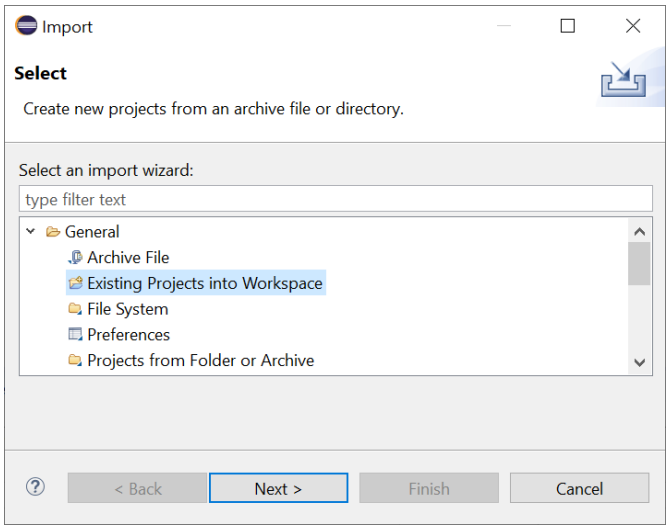
Make sure that the Anzo development server meets the requirements in Anzo Requirements in the Deployment Guide. In addition, install the following programs for working with the Anzo Java SDK:

- Eclipse for Java Developers Version 4.7.3+: Install the **Eclipse IDE for Java Developers** or **Eclipse IDE for Enterprise Java Developers**.
- Java Runtime Environment Version 8: Eclipse and the Anzo SDK require JDK version 8. Cambridge Semantics tests with jdk1.8.0_181.

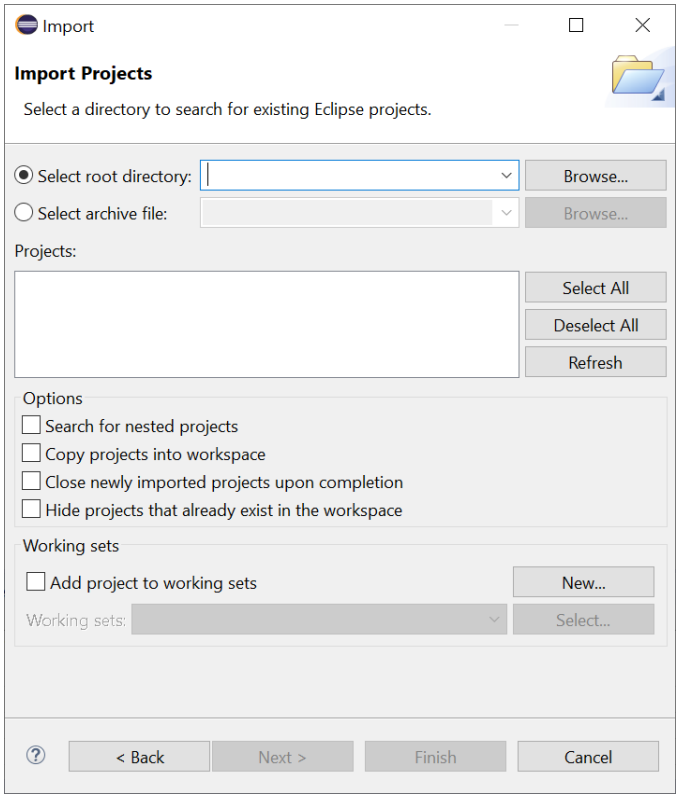
Deploying the Anzo SDK with Eclipse

Follow the instructions below to import the Anzo Java SDK to Eclipse and configure and test the environment.

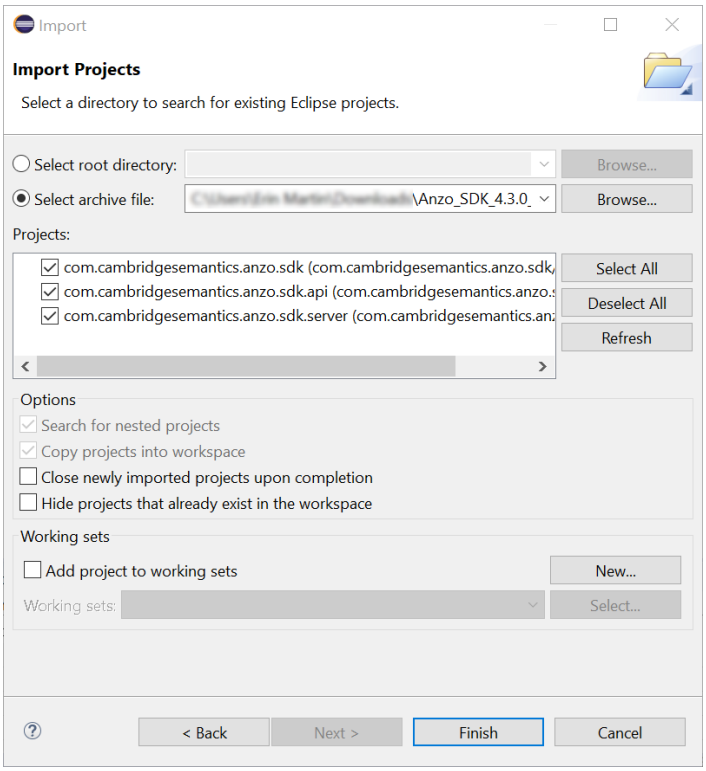
1. Download the Anzo SDK .zip file to the host server. Do not unpack the file.
2. In Eclipse, click the **File** menu and select **Import**. Eclipse opens the Import dialog box. For example:



3. In the Import dialog box, expand the **General** folder and select **Existing Projects into Workspace** and click **Next**. Eclipse opens the Import Projects dialog box. For example:



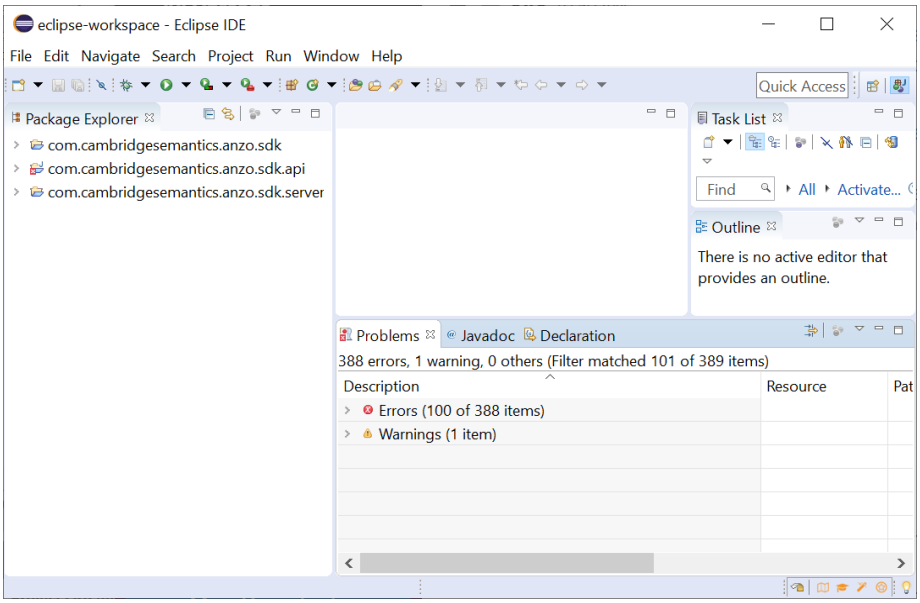
4. Select the **Select archive file** radio button and then browse to and select the Anzo SDK .zip file. Eclipse loads the .zip file and lists the contents in the Projects field. For example:



The Anzo SDK contains three projects:

- **com.cambridgesemantics.anzo.sdk**: This core project is required for creating solutions. It contains the Anzo libraries that provide the Anzo APIs and extension points as well as the libraries that enable Anzo to run in the development environment.
- **com.cambridgesemantics.anzo.sdk.server** This core project is required for creating solutions. It contains configuration files for running Anzo as well as a launcher for starting the Anzo server.
- **com.cambridgesemantics.anzo.sdk.api**: This is an example project that contains sample Java programs that illustrate several aspects of the Anzo client APIs. Each program is a simple example that demonstrates how to communicate with the Anzo server to read, write, and query data. See the comments in each example for an explanation of what each one demonstrates.

5. Click **Finish** to import the Anzo SDK .jar files. The process may take a few minutes. When the import is complete, Eclipse opens the workspace. At this point in the process, expect to see several errors in the workspace. For example:



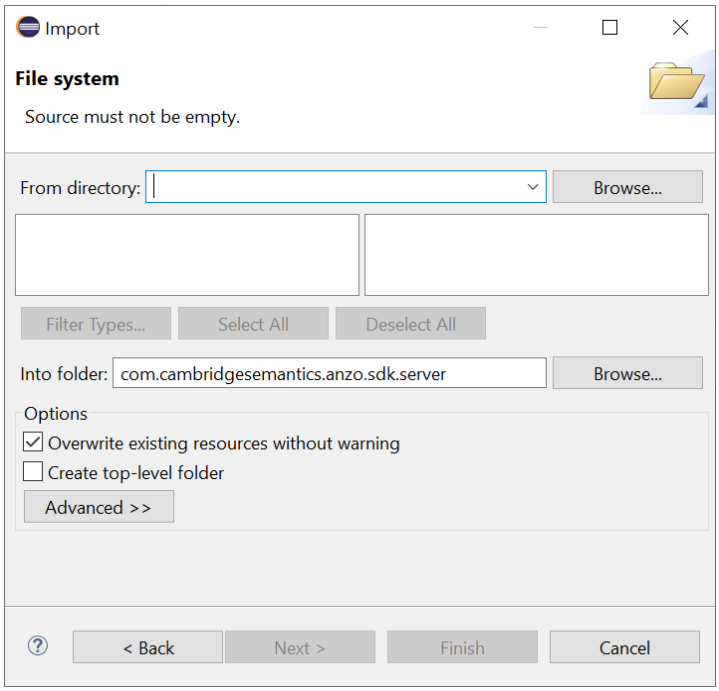
6. Import your Anzo license:

   a. Make sure that you have a copy of the Anzo license on the server. If necessary, you can view and download a copy from the Cambridge Semantics Support Center.

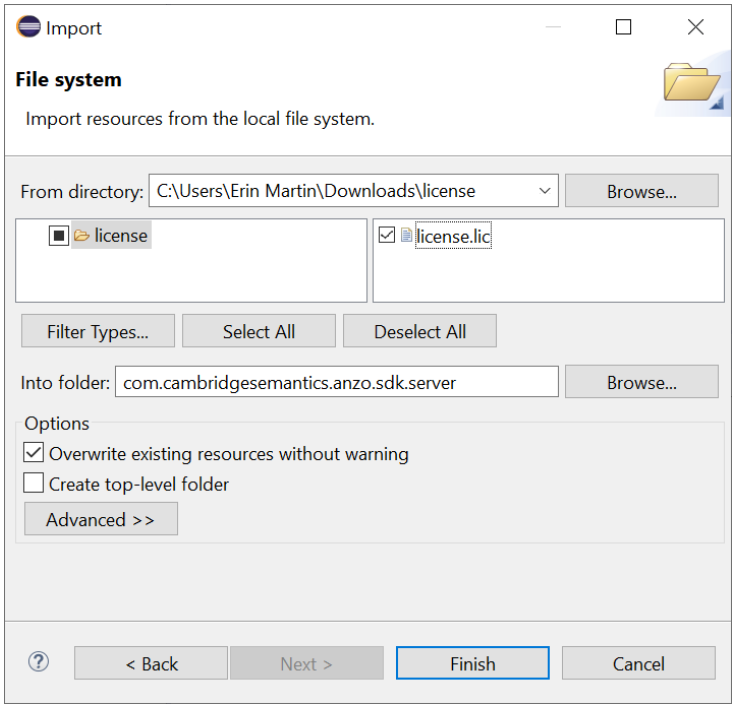
   b. Rename the license file so its file extension is .lic. For example, **license.lic**.

   c. In the Eclipse Package Explorer, right-click **com.cambridgesemantics.anzo.sdk.server** and select **Import**.

   d. In the Import dialog box, expand the **General** folder and select **File System**. Then click **Next**. Eclipse opens the File System Import dialog box. For example:



e. Click the **Browse** button next to the From directory field and select the directory that contains the license file. Eclipse displays the directory and its contents.

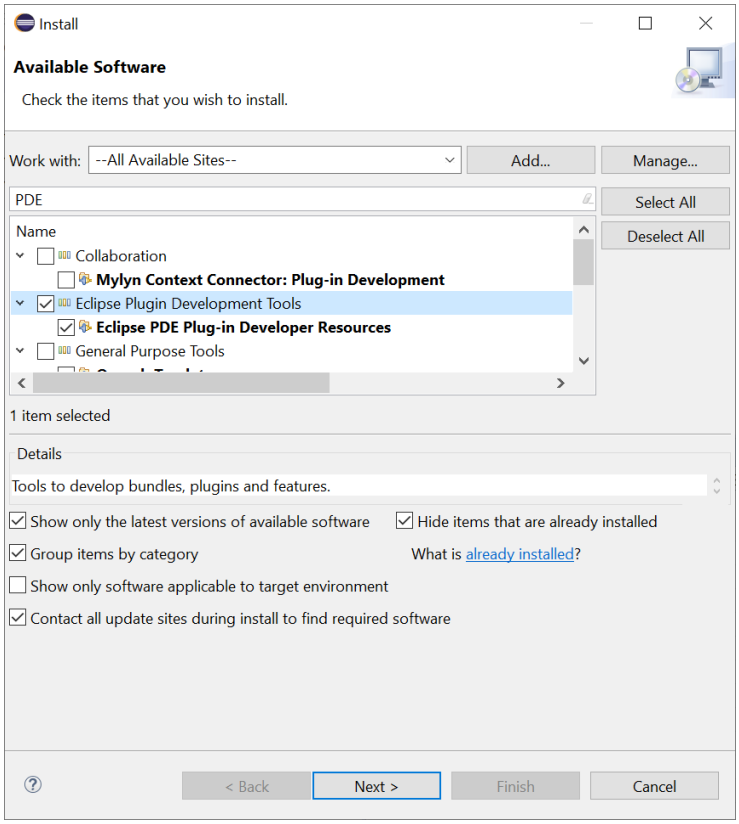


f. Select the license file in the right pane, and then click **Finish**.

7. Install the Eclipse Plugin Development Tools:

   a. Click the **Help** menu and select **Install New Software**. Eclipse opens the Install dialog box.

   b. In the Install dialog box, click the **Work with** drop-down list and select **All Available Sites**. In the search field below the Work with field, type "PDE" and wait for Eclipse to find the plugin tools. Select the checkbox next to **Eclipse Plugin Development Tools**, including **Eclipse PDE Plug-in Developer Resources**. For example:



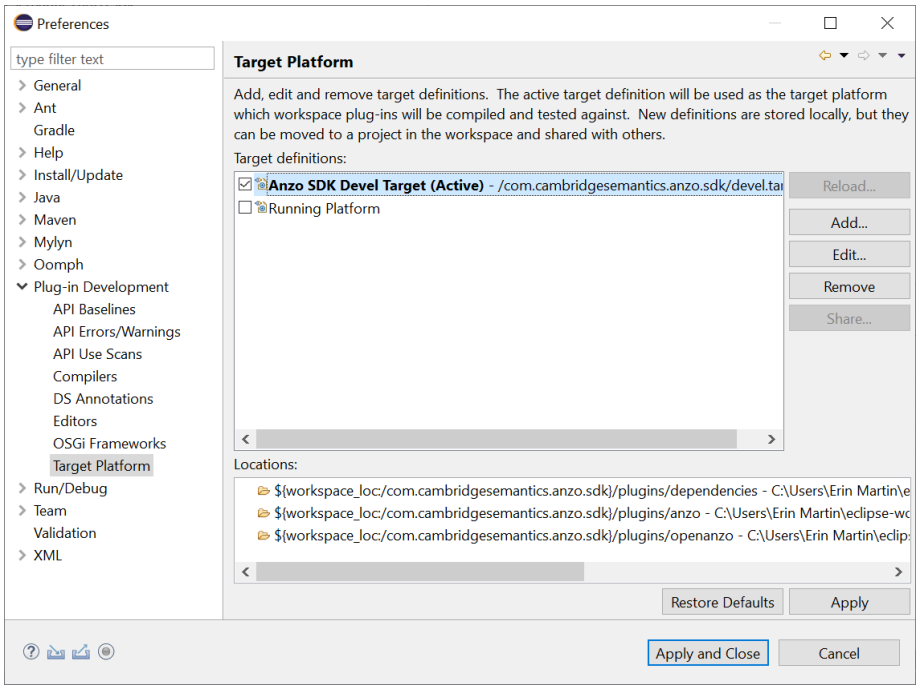
   c. Click **Next** and accept the license agreement, then click **Finish**. Eclipse installs the software and then prompts you to restart the application.

8. After restarting Eclipse, load the Anzo SDK Target Platform:

   a. Click the **Window** menu and select **Preferences**.

   b. In the Preferences dialog box, expand **Plug-in Development** and select **Target Platform**.

c. In the Target Platform definitions, select the **Anzo SDK Devel Target** checkbox. For example:

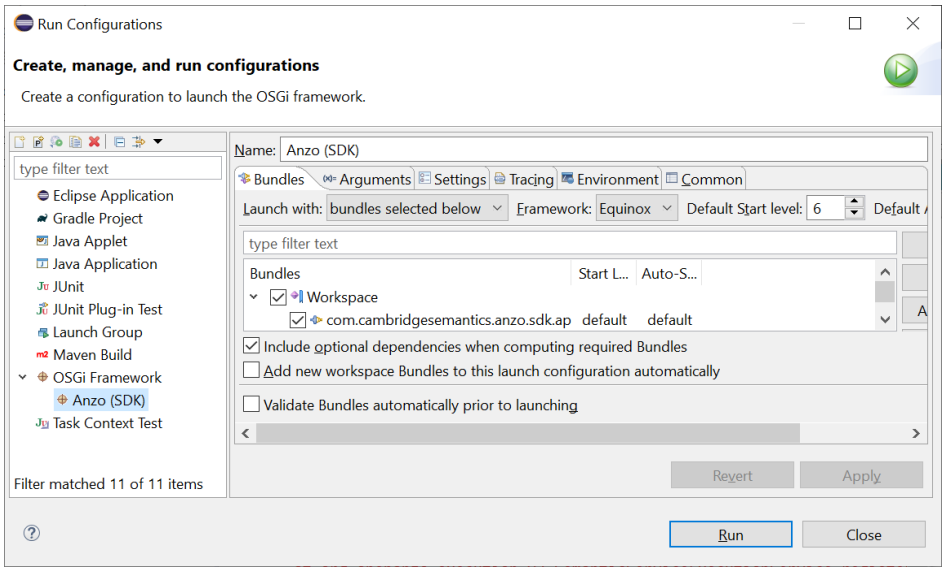


d. Click **Apply and Close**. Eclipse loads the Anzo SDK Target Platform.

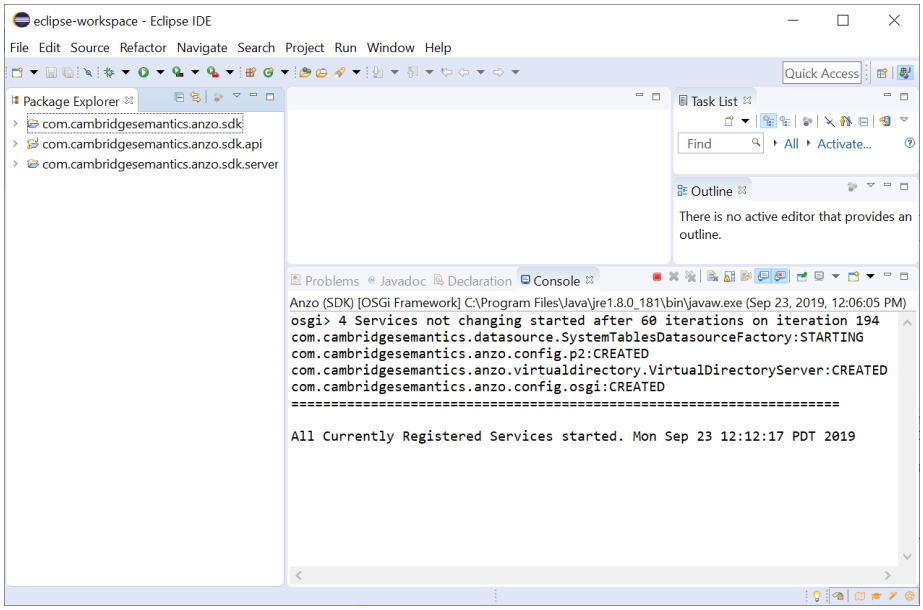
9. Test the environment:

a. In the Eclipse workspace, click the **Run** menu and select **Run Configurations**. Eclipse opens the Run Configurations dialog box.

b. On the left side of the dialog box, expand the **OSGi Framework** folder and select **Anzo (SDK)**. For example:

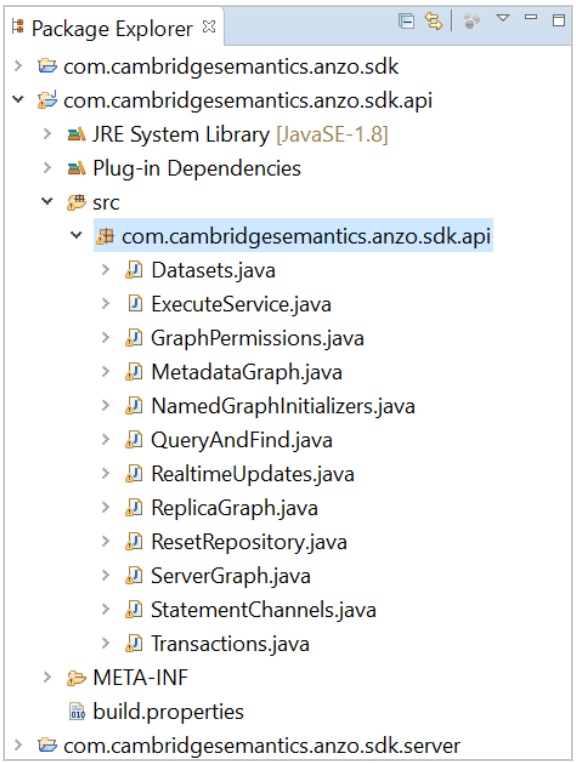


c. Click **Run** to run the Anzo SDK target platform. A Console tab opens in Eclipse and shows the status messages. When Anzo starts, the console displays the message "All Currently Registered Services started." For example:



If Anzo fails to start, one of the common reasons for the failure is that one or more of the Anzo ports are in use by other software. See Firewall Requirements in the Deployment Guide for information about the ports that Anzo uses.

To explore the sample Java programs that are included in the Anzo SDK, expand the **com.cambridgesemantics.anzo.sdk.api** package in the Package Explorer. In the package, expand the **src** directory and then the **com.cambridgesemantics.anzo.sdk.api** directory to see the list of sample programs. For example:



To run a program, right-click the .java file and select **Run As** > **Java Application**. For more information about using the Anzo SDK, see the **Anzo Java SDK Guide.pdf** that is distributed in the SDK .zip file.