

AllegroGraph Semantic Layer for Databricks (Delta Lake)

This AllegroGraph tutorial is available on our Github Examples page.

Databricks is a popular choice for hosting lakehouses – a new architecture that unifies data storage, analytics, and AI on one platform. On the other hand, as an enterprise knowledge graph platform, AllegroGraph provides quick semantic layer integration with Databricks transparently through our advanced VKG (virtual knowledge graph) interface.

In this tutorial, we will show you how to load RDF triples directly from your Delta Tables that are hosted in Databricks, and we assume the readers have prior experience with AllegroGraph and our agtool facility.

For users starting with the open-source Delta Lake but not hosted on Databricks, this tutorial may still apply, as long as your platform exposes a JDBC connection and enables SQL as (one of) its query interface.

Requirements

Obviously, you will need to have a running cluster or a SQL warehouse in your Databricks workspace as well as an AllegroGraph server. This tutorial uses a cluster to demonstrate.

Note that Databricks provides trial clusters and one can start from here. If all is successfully set up, the cluster's dashboard should look similar to this:

+

FranzInc's Demo Cluster ✔ 🔒

More ...

Terminate

Edit

Configuration

Notebooks (0)

Libraries

Event log

Spark UI

Driver logs

Metrics

Apps

Spark cluster UI - Master

Policy ?

Unrestricted

Multi node

Single node

Access mode ?

Single user access ?

Single user

Tianyu Gu (gty@franz.com)

Performance

Databricks Runtime Version

12.2 LTS (includes Apache Spark 3.3.2, Scala 2.12)

Use Photon Acceleration ?

Worker type ?

Min workers

Max workers

Current

i3.xlarge

30.5 GB Memory, 4 Cores

2

8

2

Driver type

i3.xlarge

30.5 GB Memory, 4 Cores

Enable autoscaling ?

Enable autoscaling local storage ?

Terminate after

120

minutes of inactivity ?

Instance profile ?

None

Tags ?

No custom tags

> Automatically added tags

▶ Advanced options

Summary

2-8 Workers

61-244 GB Memory

8-32 Cores

1 Driver

30.5 GB Memory, 4 Cores

Runtime

12.2.x-scala2.12

Photon

i3.xlarge

6-18 DBU/h

Create a table and load a sample dataset

We use a sample dataset called *people10m* for this tutorial. As documented by Databricks, we can load it into a table by executing this SQL query:

```
CREATE TABLE default.people10m OPTIONS (PATH 'dbfs:/databricks-datasets/learning-spark-v2/people/people-10m.delta')
```

After being successfully loaded, you can find the table in the Data Explorer:

Data Explorer

[hive_metastore](#) > [default](#)

Filter tables...

[people10m](#)

[Table](#) [Delta](#) [Not set](#) 211MiB, 8 files [Add comment](#) [Create](#)

Columns Sample Data Details Permissions History

Filter columns...

Column	Type	Comment
id	int	
firstName	string	
middleName	string	
lastName	string	
gender	string	
birthDate	timestamp	
ssn	string	
salary	int	

as well as a few sample data rows:

Data Explorer

+ Add FranzInc's De... 91.5 GB, 12 Cores ▾

default >

default.people10m

Table	Delta	Not set ▾	211MiB, 8 files	Add comment	:	Create ▾
-------	-------	-----------	-----------------	-------------	---	----------

Columns

Sample Data

Details

Permissions

History

id	firstName	middleName	lastName	gender	birthDate	ssn	salary
3766824	Hisako	Isabella	Malitrott	F	1961-02-12T05:00:00.000+0000	938-80-1874	58862
3766825	Daisy	Merissa	Fibben	F	1998-05-19T04:00:00.000+0000	971-14-3755	66221
3766826	Caren	Blossom	Henner	F	1962-08-06T04:00:00.000+0000	954-19-8973	54376
3766827	Darleen	Gertie	Goodinson	F	1980-03-12T05:00:00.000+0000	981-65-5269	69954
3766828	Kyle	Lu	Habben	F	1974-02-15T04:00:00.000+0000	936-95-3240	56681
3766829	Melia	Kristy	Bonhill	F	1970-09-13T04:00:00.000+0000	960-91-9232	73995
3766830	Yvette	Faye	Bebbell	F	1972-09-07T04:00:00.000+0000	987-72-3701	92888
3766831	Delpha	Kenisha	Gillison	F	1979-06-25T04:00:00.000+0000	962-66-5404	51206
3766832	Mikaela	Jenifer	Hallan	F	1973-05-23T04:00:00.000+0000	911-38-3114	98887
3766833	Cindi	Renita	Cousin	F	1979-03-19T05:00:00.000+0000	666-50-3216	63646
3766834	Rosana	Sari	Griston	F	1956-10-25T04:00:00.000+0000	998-70-9023	19255
3766835	Leena	Vanita	Cajkler	F	1971-08-15T04:00:00.000+0000	954-81-8367	74760
3766836	Eleonore	Stephany	Goldhill	F	1987-12-15T05:00:00.000+0000	978-64-1528	67942
3766837	Olga	Jona	Klos	F	1978-05-26T04:00:00.000+0000	961-27-8570	52484
3766838	Chandra	Sherrill	Kelsey	F	1962-11-18T05:00:00.000+0000	910-53-2697	65929
3766839	Vania	Lashon	Clixby	F	1975-10-31T05:00:00.000+0000	985-43-1411	73756
3766840	Dagmar	Misti	Poole	F	1962-05-30T04:00:00.000+0000	916-94-8252	85176
3766841	Marta	Joel	Copelli	F	1953-01-23T05:00:00.000+0000	929-52-3074	82347
3766842	Danae	Darline	Mithun	F	1993-08-07T04:00:00.000+0000	957-79-3027	62469
3766843	Fredda	Shawn	Denness	F	1956-06-03T04:00:00.000+0000	946-20-3144	92998

>

1 2 3 4 5 > 20 / page ▾

default >


default.people10m

Table

 Delta

 Not set

211MiB, 8 files

 Add comment

...

Create

Columns

Sample Data

Details

Permissions

History

id	firstName	middleName	lastName	gender	birthDate	ssn	salary
3766824	Hisako	Isabella	Malitrott	F	1961-02-12T05:00:00.000+0000	938-80-1874	58862
3766825	Daisy	Merissa	Fibben	F	1998-05-19T04:00:00.000+0000	971-14-3755	66221
3766826	Caren	Blossom	Henner	F	1962-08-06T04:00:00.000+0000	954-19-8973	54376
3766827	Darleen	Gertie	Goodinson	F	1980-03-12T05:00:00.000+0000	981-65-5269	69954
3766828	Kyle	Lu	Habben	F	1974-02-15T04:00:00.000+0000	936-95-3240	56681
3766829	Melia	Kristy	Bonhill	F	1970-09-13T04:00:00.000+0000	960-91-9232	73995
3766830	Yvette	Faye	Bebbell	F	1972-09-07T04:00:00.000+0000	987-72-3701	92888
3766831	Delpha	Kenisha	Gillison	F	1979-06-25T04:00:00.000+0000	962-66-5404	51206
3766832	Mikaela	Jenifer	Hallan	F	1973-05-23T04:00:00.000+0000	911-38-3114	98887
3766833	Cindi	Renita	Cousin	F	1979-03-19T05:00:00.000+0000	666-50-3216	63646
3766834	Rosana	Sari	Griston	F	1956-10-25T04:00:00.000+0000	998-70-9023	19255
3766835	Leena	Vanita	Cajkler	F	1971-08-15T04:00:00.000+0000	954-81-8367	74760
3766836	Eleonore	Stephany	Goldhill	F	1987-12-15T05:00:00.000+0000	978-64-1528	67942
3766837	Olga	Jona	Klos	F	1978-05-26T04:00:00.000+0000	961-27-8570	52484
3766838	Chandra	Sherrill	Kelsey	F	1962-11-18T05:00:00.000+0000	910-53-2697	65929
3766839	Vania	Lashon	Clixby	F	1975-10-31T05:00:00.000+0000	985-43-1411	73756
3766840	Dagmar	Misti	Poole	F	1962-05-30T04:00:00.000+0000	916-94-8252	85176
3766841	Marta	Joel	Copelli	F	1953-01-23T05:00:00.000+0000	929-52-3074	82347
3766842	Danae	Darline	Mithun	F	1993-08-07T04:00:00.000+0000	957-79-3027	62469
3766843	Fredda	Shawn	Denness	F	1956-06-03T04:00:00.000+0000	946-20-3144	92998

1

2

:

4

5

>

20 / page

Now we need to prepare the Databricks JDBC connection details. You may follow these steps to retrieve the JDBC URL, which may look similar to:

```
jdbc:databricks://dbc-0bf1f204-2226.cloud.databricks.com:443/default;transportMode=http;ssl=1;httpPath=sql/protocolv1/o/326754737859861/0405-070225-tumf7a9c;AuthMech=3;UID=token;PWD=<personal-access-token>
```

A personal access token is needed, see [here](#) for how to generate one.

Last but not least, we will need to download the Databricks JDBC driver from [here](#). This tutorial uses version 2.6.32. Both the URL and the driver are needed by AllegroGraph's virtual knowledge graph interface, as we will see later.

vload – Load RDF triples from Databricks

The vload facility of agtool is able to load data from relational databases as RDF triples. For a tutorial for vload itself, please refer to [this page](#).

To configure vload, we need 2 files:

demo.properties

This file contains information about the Databricks JDBC connection details as what we have shown in the previous section:

```
jdbc.url=<your-JDBC-url>
jdbc.driver=com.databricks.client.jdbc.Driver
```

Note that the downloaded Databricks JDBC driver also needs to be properly installed. See more details [here](#).

demo.mapping.obda

This file defines the rules of how to map the columns from the people10m table between our expected RDF triples. As the target and source sections indicate, we will map id, firstName, lastName, gender, and salary into RDF triples by executing a SQL query.

```
[PrefixDeclaration]
:          http://example.org/
rdf:       http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:      http://www.w3.org/2000/01/rdf-schema#
```

owl: http://www.w3.org/2002/07/owl#
xsd: http://www.w3.org/2001/XMLSchema#
obda: https://w3id.org/obda/vocabulary#

```
[MappingDeclaration] @collection [[
```

```
mappingId      people10m
target         :{id} a :Person ; rdfs:label "{firstName}
{lastName}" ; :gender "{gender}"; :salary "{salary}"^^xsd:int
.
source                                     SELECT      *      FROM
`hive_metastore`.`default`.`people10m` LIMIT 1000

]]
```

By using this mapping, a row of such data:

id	firstName	middleName	lastName	gender	birthDate	ssn	salary
3766824	Hisako	Isabella	Malitrott	F	1961-02-12T05:00:00.000+0000	938-80-1874	58863

will be mapped to these RDF triples (in Turtle syntax):

```
@prefix :      <http://example.org/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .

:3766824 a      :Person ;
        rdfs:label "Hisako Malitrott" ;
        :gender   "F" ;
        :salary   "58863"^^xsd:int .
```

For more details on creating mappings, please refer to this page.

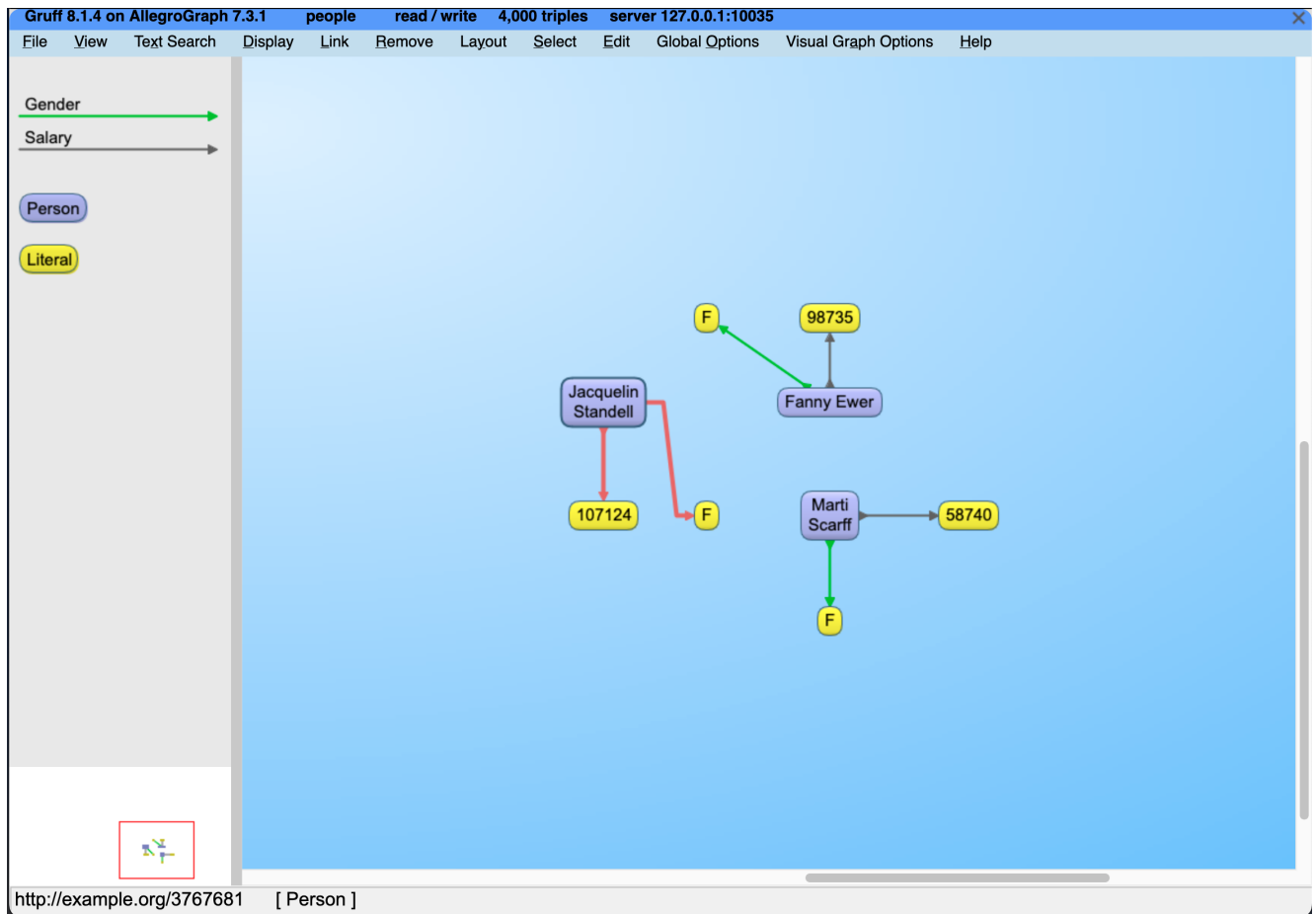
Finally, we can start vloading by running this command:

```
agtool vload --ontop-home /path/to/ontop --properties
/path/to/your/demo.properties --mapping
/path/to/your/demo.mapping.obda people
```

2023-04-12T19:04:13| Creating a temporary workspace

```
2023-04-12T19:04:13| Temporary workspace successfully created:
"/tmp/agtool-vload-1dfd731d-4862-e844-fde6-0242164d5260/"
2023-04-12T19:04:13| Mapping file is given, skip bootstrapping
2023-04-12T19:04:13| Starting materialization
2023-04-12T19:04:16| Materialization - OBTAINED FROM SPARK
JDBC DRIVER: hive_metastore, default
2023-04-12T19:04:18| Materialization - 19:04:18.398 |-INFO in
i.u.i.o.a.r.impl.QuestQueryProcessor - Ontop has completed the
setup and it is ready for query answering!
2023-04-12T19:04:30| Materialization - WARNING:
sun.reflect.Reflection.getCallerClass is not supported. This
will impact performance.
2023-04-12T19:04:31| Materialization - NR of TRIPLES: 1000
2023-04-12T19:04:31| Materialization - Elapsed time to
materialize: 13218 {ms}
2023-04-12T19:04:34| Materialization - NR of TRIPLES: 1000
2023-04-12T19:04:34| Materialization - Elapsed time to
materialize: 2786 {ms}
2023-04-12T19:04:35| Materialization - NR of TRIPLES: 1000
2023-04-12T19:04:35| Materialization - Elapsed time to
materialize: 1339 {ms}
2023-04-12T19:04:36| Materialization - NR of TRIPLES: 1000
2023-04-12T19:04:36| Materialization - Elapsed time to
materialize: 1210 {ms}
2023-04-12T19:05:36| Materialization successfully exited
2023-04-12T19:05:36| Start loading triples
2023-04-12T19:05:37| Load finished 4 sources in 78ms (0.08
seconds). Triples added: 4,000, Average Rate:
51,282 tps.
```

It will load the RDF triples into the people repository. You may display a few sample instances through Gruff:



Now let's try to query all the information of 10 top-paid people:

```
agtool query --output-format table people - <<EOF
PREFIX : <http://example.org/>
SELECT ?person ?name ?gender ?salary {
  ?person a :Person ;
    rdfs:label ?name ;
    :gender ?gender ;
    :salary ?salary .
}
ORDER BY DESC(?salary)
LIMIT 10
EOF
```

person	name	gender	salary
:3767538	Shameka Mitcham	F	135931
:3767690	Adelia Salters	F	134145

:3767101 Eldora Welbeck F 134099
:3767137 Rosalie Challenger F 129091
:3767409 Hassie Sides F 127972
:3767659 Bridget Inwood F 126424
:3767771 Lovie Dorn F 124903
:3767631 Latoya Stogill F 120098
:3766922 Dot Murkus F 119509
:3767736 Ima Adnam F 119195

Query information:

time : output: 0.001829, overall: 0.045899, parse: 0.000000, plan: 0.020477, query: 0.005075, system: 0.000072, total: 0.027381, user: 0.042787

memory : consCells: 5829080, majorPageFaults: 0, maximumChunk: 5200000, maximumMap: 10131448, minorPageFaults: 2787

other : generation: 2, info: "bindings-set", rowCount: 10

Summary

This tutorial has shown AllegroGraph's capability of creating a Semantic Layer for the Databricks lakehouse platform.

Adding a semantic layer, via AllegroGraph, ascribes business meaning to data so end users can better understand their data and associated metadata. A semantic layer provides a number of advantages in terms of Enterprise-wide data management. Users can define business concepts and connections which add meaning to their desired use-case. Some specific advantages of a semantic layer include: improved data integration, enhanced data accessibility, improved data governance, enhanced data quality, and enhanced data security.

What is a Semantic Layer?

There are several reasons why the notion of semantic layers has reached the forefront of today's data management conversations. The analyst community is championing the data fabric tenet. The data mesh and data lake house architectures are gaining traction. Data lakes are widely deployed. Even architectural-agnostic business intelligence tooling seeks to harmonize data across sources.

Each of these frameworks requires a semantic layer to ascribe business meaning to data – via metadata – so end users can understand data for their purposes and streamline data integration. This layer sits between users and sources, so the former can comprehend data without knowing the underlying data formats.

What are the advantages of a semantic layer in your data infrastructure?

A semantic layer is an intermediate layer in an Enterprise architecture that sits between the data sources and the applications that use the data. It provides a number of advantages in terms of data management, integration, and accessibility. Some specific advantages of a semantic layer include:

- 1. Improved data integration:** A semantic layer can help to integrate data from multiple sources by providing a common data model and set of APIs that can be used to access the data. This makes it easier to build applications that work with data from multiple sources.

- 2. Enhanced data accessibility:** A semantic layer can provide a higher level of abstraction over the data sources, making it easier for users to access and work with the data. This can be particularly useful for users who are

not technical experts or who do not have in-depth knowledge of the underlying data sources.

3. Better data governance: A semantic layer can help to enforce data governance policies by providing a centralized point of control for data access and management. This can help to ensure that data is used in a consistent and controlled manner.

4. Upgraded data quality: A semantic layer can help to improve the quality of the data by providing tools and processes for data cleansing, validation, and transformation. This can help to ensure that the data is accurate and consistent.

5. Advanced data security: A semantic layer can provide an additional layer of security by controlling access to the data sources and enforcing security policies. This can help to protect sensitive data and ensure that it is only accessed by authorized users.

Overall, a semantic layer can provide a number of benefits in terms of data integration, accessibility, governance, quality, and security, making it a valuable component of a data infrastructure.

Semantic Layers with W3C's Semantic Technologies

Semantic Technology refers to a set of tools and technologies that are used to represent, store, and manipulate data in a way that allows it to be understood and interpreted by computers. Some examples of semantic technology include graph databases, ontologies, and semantic web standards such as RDF and OWL.

While semantic technology is the preferred way to implement a semantic layer, some other players have tried other technologies including traditional relational databases, data warehousing tools, or even flat files. The key is to provide a

common data model and set of APIs that can be used to access the data in a consistent and predictable manner.

That being said, Standards based W3C Semantic Technology, like that offered by AllegroGraph, has a huge advantage when it comes to implementing a semantic layer. In particular, Semantic Technology is well-suited for representing complex, interconnected data relationships, and it can provide a high level of flexibility and adaptability when it comes to working with different data sources and structures. As such, semantic technology can be a particularly useful choice for organizations that need to integrate and work with large volumes of complex data.

There are rare cases where a proprietary semantic layer may work and the organization might not mind getting locked into the ecosystem of a vendor for their metadata management needs. But for the majority of use cases, the best way to future-proof the enterprise is to adopt a standardized semantic layer with semantic technologies. This method provides a seamless business understanding of data that complements any current or future IT needs, while reinforcing data integration, analytics, and data governance.