**Introduction**

Apache Trino, formerly known as PrestoSQL, is a highly versatile and powerful distributed SQL query engine designed to query large datasets across multiple data sources. One of the key features that make Trino so flexible is its extensive range of connectors, allowing users to query data from various storage systems and databases seamlessly. Among these connectors, the **memory** and **black hole** connectors stand out due to their unique functionalities and use cases. This tutorial will delve into these special connectors, providing a detailed description and practical examples to illustrate their applications.

**What is a Trino Connector?**

A Trino connector is a modular component within the Trino SQL query engine that allows Trino to interact with various data sources. These connectors abstract the underlying complexities of different data storage systems, enabling Trino to query data from a diverse range of databases, file systems, and other data sources using a uniform SQL interface.

**Key Features of Trino Connectors**

1. **Unified Query Interface**: Trino connectors provide a consistent SQL interface to interact with different data sources, making it easier to query data without needing to understand the specific APIs or query languages of each source.
2. **Data Source Abstraction**: Each connector abstracts the underlying storage and retrieval mechanisms of the data source it represents. This means users can perform SQL operations on a data source without worrying about the low-level details of how the data is stored or accessed.
3. **Seamless Integration**: Connectors allow Trino to integrate seamlessly with a wide variety of data sources, including traditional relational databases (like MySQL, PostgreSQL), big data storage systems (like Hadoop, Apache HDFS), cloud storage services (like Amazon S3, Google Cloud Storage), and specialized data systems (like Apache Kafka, Elasticsearch).
4. **Extensibility**: Trino’s architecture allows developers to create custom connectors to support additional data sources. This extensibility ensures that Trino can be tailored to meet the specific needs of different organizations and use cases.

To execute the Sql statements of this tutorial we will be using a Zeppelin note with the **trino**interpreter (**%trino**).

%trino

-- Show trino version

SELECT version()

**Types of Trino Connectors**

Trino supports a wide range of connectors, which can be broadly categorized into:

* **Relational Database Connectors**: For traditional SQL databases like MySQL, PostgreSQL, Oracle, and SQL Server.
* **Big Data Connectors**: For querying big data platforms like Apache Hive, Apache HBase, and Apache Cassandra.
* **Cloud Storage Connectors**: For accessing data stored in cloud services like Amazon S3, Google Cloud Storage, and Microsoft Azure Blob Storage.
* **Specialized Connectors**: For interacting with systems like Apache Kafka for streaming data, Elasticsearch for search and analytics, and Redis for in-memory data storage.

**Example of Using a Trino Connector**

To illustrate how a Trino connector works, let’s consider an example using the MySQL connector.

1. **Configuration**: First, you need to configure the MySQL connector in the Trino catalog properties file. This involves specifying the connector name and connection details:

connector.name=mysql

connection-url=jdbc:mysql://mysql-host:3306

connection-user=myuser

connection-password=mypassword

1. **Querying Data**: Once the connector is configured, you can use Trino to query data from the MySQL database using SQL:

SELECT \* FROM mysql.database\_name.table\_name WHERE condition;

In this example, Trino uses the MySQL connector to translate the SQL query into the appropriate operations for retrieving data from the MySQL database.

**Trino Special Connectors**

Trino comes with some special connectors dedicated for testing and benchmarking.

1. **Memory Connector**

The **memory** connector in Trino is designed for temporary, in-memory storage of data. This connector is particularly useful for testing, prototyping, and small-scale data processing tasks where persistence is not required. The data stored using the memory connector is ephemeral, meaning it is lost when the Trino server restarts, or the data is explicitly dropped. The **memory**connector allows you to create and manipulate tables that reside entirely in memory. This makes it an excellent tool for scenarios where you need fast access to data without the overhead of disk I/O. It supports typical SQL operations like SELECT, INSERT, UPDATE, and DELETE, making it functionally similar to other connectors but with the advantage of in-memory performance.

**Example**

The **memory** connector is already configured in your sandbox. The following code illustrates how to use it:

%trino

-- Create a new schema using the memory connector

CREATE SCHEMA memory.tutorial;

%trino

-- Create a table in this schema

CREATE TABLE memory.tutorial.example\_table (

id INTEGER,

name VARCHAR

);

%trino

-- Insert some data into the table

INSERT INTO memory.tutorial.example\_table (id, name) VALUES (1, 'Alice'), (2, 'Bob');

%trino

-- Show the data from the table

SELECT \* FROM memory.tutorial.example\_table;

In this example, we first create a schema named tutorial within the memory catalog. Then, we create a table example\_table with two columns: id and name. We insert a couple of rows into this table and then select all rows from it. The operations are performed entirely in memory, ensuring fast execution times.

1. **Black Hole Connector**

The **black hole** connector is another unique feature of Trino, primarily used for testing and benchmarking. As the name suggests, the **black hole** connector acts like a data sink where all data is discarded. This can be useful for testing the performance of queries without actually storing any data or when you want to validate the query logic without generating large outputs.

When you write data to a table defined using the **black hole** connector, Trino processes the data but does not store it. Similarly, any read operations on such tables will result in empty results. This can be useful for stress testing, query optimization, and verifying SQL logic without the need for persistent storage.

**Example**

The **black hole**connector is already configured in your sandbox. The following code illustrates how to use it:

%trino

-- Create a new schema

CREATE SCHEMA if not exists blackhole.tutorial;

%trino

-- Create a new table in this schema

CREATE TABLE blackhole.tutorial.example\_table (

id INTEGER,

name VARCHAR

);

%trino

-- Insert some data into the table

INSERT INTO blackhole.tutorial.example\_table (id, name) VALUES (1, 'Alice'), (2, 'Bob');

%trino

-- Show the data from the table - should retrun empty

SELECT \* FROM blackhole.tutorial.example\_table;

In this example, we create a schema named tutorial within the blackhole catalog and then create a table example\_table with columns id and name. We insert rows into this table, but since it’s a black hole table, the data is discarded. A SELECT query on this table will return no results.

1. **TPCDS connector**

The TPCDS connector provides a set of schemas to support the [TPC Benchmark™ DS (TPC-DS)](http://www.tpc.org/tpcds/). TPC-DS is a database benchmark used to measure the performance of complex decision support databases.

This connector can be used to test the capabilities and query syntax of Trino without configuring access to an external data source. When you query a TPCDS schema, the connector generates the data on the fly using a deterministic algorithm.

%trino

-- The TPCDS connector supplies several schemas

SHOW SCHEMAS FROM tpcds;

%trino

-- Show tables from the tiny schema

show tables from tpcds.tiny;

%trino

-- Show data from the customer table

select \* from tpcds.tiny.customer;

Every TPCDS schema provides the same set of tables. Some tables are identical in all schemas. The scale factor of the tables in a particular schema is determined from the schema name.

For example, the schema **sf1** corresponds to scale factor **1** and the schema **sf300** corresponds to scale factor **300**. Every unit in the scale factor corresponds to a gigabyte of data. For example, for scale factor **300**, a total of **300** gigabytes are generated. The tiny schema is an alias for scale factor **0.01**, which is a very small data set useful for testing.

1. **TPCH connector**

The TPCH connector provides a set of schemas to support the [TPC Benchmark™ H (TPC-H)](http://www.tpc.org/tpch/). TPC-H is a database benchmark used to measure the performance of highly-complex decision support databases.

This connector can be used to test the capabilities and query syntax of Trino without configuring access to an external data source. When you query a TPCH schema, the connector generates the data on the fly using a deterministic algorithm.

%trino

-- The TPCDS connector supplies several schemas

SHOW SCHEMAS FROM tpch;

%trino

-- Show tables from the tiny schema

show tables from tpch.tiny;

%trino

-- Show data from the customer table

select \* from tpch.tiny.customer;

**Summary**

Apache Trino’s **memory** and **black hole** connectors provide unique capabilities for in-memory data processing and performance testing. The memory connector allows for the creation and manipulation of ephemeral, in-memory tables, making it ideal for fast, temporary data storage and testing scenarios. On the other hand, the black hole connector serves as a data sink for testing and benchmarking, discarding all written data and returning empty results for read operations. These special connectors extend Trino’s versatility, enabling users to perform a wide range of tasks efficiently. Whether you’re testing SQL queries, prototyping applications, or benchmarking performance, these connectors offer valuable tools to enhance your data processing workflows. The TPCH / TPCDS schemas provide the same set of tables. Some tables are identical in all schemas. Other tables vary based on the *scale factor*, which is determined based on the schema name. These tables are used for testing and benchmarking.