**Introduction**

SQL is powerful and widely understood querying language but has limitations in the world of “Big Data”, where data engineers and scientists often need to mix in custom code with their queries. Platforms and tools such as Apache Hadoop, Apache Hive, and Apache Spark are widely used to query and manipulate large data volumes.

The ability to query many disparate data source in the same system with the same SQL greatly simplifies analytics that require understanding the large picture of all your data. Federated queries in Trino can access your object storage, your main relational databases, and your new streaming or NoSQL system, all in the same query. Trino completely changes what is possible in this central data consumption layer.

**What is Trino?**

**Trino** is a tool designed to efficiently query vast amounts of data using **distributed queries**. When we work with terabytes or petabytes of data, we are likely using tools that interact with Hadoop and HDFS. Trino was designed as an alternative to tools that query HDFS using pipelines of MapReduce jobs, such as Hive, but Trino is **not** limited to accessing HDFS. Trino can be and has been extended to operate over different kinds of data sources, including traditional relational databases and other data sources such as MySQl, Cassandra, etc.

Trino was designed to handle data warehousing and analytics: data analysis, aggregating large amounts of data and producing reports. These workloads are often classified as Online Analytical Processing (OLAP).

**Features of Trino**

Trino offers several characteristics that make it a crucial tool for data analysts and businesses trying to effectively manage and analyze large amounts of data. In this section, we’ll go over some of Trino’s most vital attributes, such as its capacity for massive data queries, data integration skills, connection, and federated query capabilities.

**Querying Huge Amounts of Data**

Trino’s capacity for speedy and effective data querying is one of its most important characteristics. Trino has been designed to efficiently query huge amounts of data using distributed queries over numerous nodes. This makes it well suited for studying large volumes of data. Due to Trino’s distributed design, queries may be processed more quickly and with more scalability by being parallelized over numerous nodes.

**Data Integration**

The Hadoop Distributed File System (HDFS), Amazon S3, Cassandra, MySQL, and MongoDB are just a few data sources that Trino may combine. Data analysts may query data from several sources using a single SQL interface thanks to Trino’s data integration features. This simplifies the analysis of complicated datasets. Trino users may execute federated queries across several data sources, providing a complete picture of the data. It’s simpler to access and analyze data from many sources because of Trino’s connection to a wide variety of data sources. Trino can connect to Hive, Microsoft SQL Server, Oracle, MySQL, PostgreSQL, and other databases and data warehouses. Trino can also connect to cloud storage services like Google Cloud Storage and Amazon S3.

**Federated Query Capabilities**

Trino can query big data from several data sources, such as Hive, Cassandra, and relational databases. This is due to its federated query capabilities. Thanks to this capability, it’s simple to evaluate data from many sources without having to create intricate ETL pipelines. Trino’s federated queries enable complicated aggregations, joins, and data combining from several sources.

For example, When Trino is connected to Hive, users can use Trino’s SQL syntax to query data from both Hive and other data sources that Trino supports, such as MySQL, Cassandra, and MongoDB. This makes it easier for users to perform complex analytics tasks that require data from multiple sources.

**Trino Architecture**

Trino is a distributed query engine that processes data in parallel across multiple servers. There are two types of Trino servers, coordinators and workers. The following sections describe these servers and other components of Trino’s architecture.

1. **Cluster**

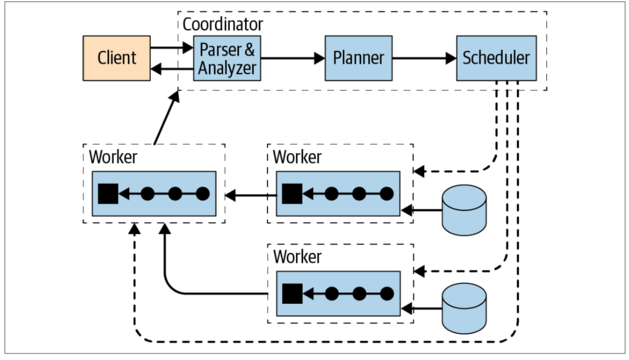
A Trino cluster consists of a coordinator and many workers. Users connect to the coordinator with their SQL query tool. The coordinator collaborates with the workers. The coordinator and the workers access the connected data sources. This access is configured in catalogs. Processing each query is a stateful operation. The workload is orchestrated by the coordinator and spread parallel across all workers in the cluster. Each node runs Trino in one JVM instance, and processing is parallelized further using threads.

1. **Coordinator**

The Trino coordinator is the server that is responsible for parsing statements, planning queries, and managing Trino worker nodes. It is the “brain” of a Trino installation and is also the node to which a client connects to submit statements for execution. Every Trino installation must have a Trino coordinator alongside one or more Trino workers. For development or testing purposes, a single instance of Trino can be configured to perform both roles. The coordinator keeps track of the activity on each worker and coordinates the execution of a query. The coordinator creates a logical model of a query involving a series of stages, which is then translated into a series of connected tasks running on a cluster of Trino workers. Coordinators communicate with workers and clients using a REST API.

1. **Worker**

A Trino worker is a server in a Trino installation, which is responsible for executing tasks and processing data. Worker nodes fetch data from connectors and exchange intermediate data with each other. The coordinator is responsible for fetching results from the workers and returning the final results to the client. When a Trino worker process starts up, it advertises itself to the discovery server in the coordinator, which makes it available to the Trino coordinator for task execution. Workers communicate with other workers and Trino coordinators using a REST API.

[](http://localhost/wp-content/uploads/2024/01/Trino_architecture.png)

**Data sources**

While using Trino, you need to learn some terms such as connector, catalog, schema, and table. These fundamental concepts cover Trino’s model of a particular data source and are described in the following section.

1. **Connector**

A connector adapts Trino to a data source such as Hive or a relational database. You can think of a connector the same way you think of a driver for a database.

Trino contains several built-in connectors: a connector for Hive, a TPCH connector designed to serve TPC-H benchmark data and many others. Many third-party developers have contributed connectors so that Trino can access data in a variety of data sources.

Every catalog is associated with a specific connector. It is possible to have more than one catalog use the same connector to access two different instances of a similar database. For example, if you have two Hive clusters, you can configure two catalogs in a single Trino cluster that both use the Hive connector, allowing you to query data from both Hive clusters, even within the same SQL query.

1. **Catalog**

A Trino catalog contains schemas and references a data source via a connector. For example, you can configure a Hive catalog to provide access to Hive metastore via the Hive connector. When you run SQL statements in Trino, you are running them against one or more catalogs. Other examples of catalogs include the MySQL catalog to connect to a Hive data source. When addressing a table in Trino, the fully-qualified table name is always rooted in a catalog. For example, a fully-qualified table name of **hive.test\_data.test** refers to the **test**table in the **test\_data** schema in the **hive** catalog.

1. **Schema**

Schemas are a way to organize tables. Together, a catalog and schema define a set of tables that can be queried. When accessing Hive or a relational database such as MySQL with Trino, a schema translates to the same concept in the target database. Other types of connectors may choose to organize tables into schemas in a way that makes sense for the underlying data source.

1. **Table**

A table is a set of unordered rows, which are organized into named columns with types. This is the same as in any relational database. The mapping from source data to tables is defined by the connector.

**Trino Special Connectors**

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

1. **Black Hole Connector**

Primarily Black Hole connector is designed for high performance testing of other components. It has some features that allow testing Trino in a controlled manner. Metadata for any tables created via this connector is kept in memory on the coordinator and discarded when Trino restarts. Created tables are by default always empty, and any data written to them is ignored and data reads return no rows. During table creation, a desired rows number can be specified. In such cases, writes behave in the same way, but reads always return the specified number of some constant rows. You shouldn’t rely on the content of such rows.

1. **Memory connector**

The Memory connector stores all data and metadata in RAM on workers and both are discarded when Trino restarts.

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.

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.8

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()

**Trino SQL Language**

Trino is an ANSI SQL compliant query engine. This standard compliance allows Trino users to integrate their favorite data tools, including BI and ETL tools with any underlying data source. Trino validates and translates the received SQL statements into the necessary operations on the connected data source. In this section, we will run some SQL queries to retrieve, filter rows from the World sample database which is already installed in the sandbox. The World database has three tables: City, Country and CountryLanguage. The following figure illustrates the database schema.

[A screenshot of a computer

AI-generated content may be incorrect.](http://localhost/wp-content/uploads/2023/08/world-dm.png)

Let’s start by writing some queries to explore the World sample database.

This database is installed on MySQL Server. For that we need to use the Trino MySQL connector (already installed in the sandbox) to access and query this database.

1. **Show Catalogs**

The **Show Catalogs**statement is used to list all the available catalogs in Trino current installation.

%trino

-- Show all available catalogs

show catalogs;

1. **Show Schemas**

The **Show Schemas**statement is used to list the schemas in a particular catalog or in the current catalog.

%trino

-- Show schemas (databases) in MySql catalog

show schemas in mysql;

1. **Show Tables**

The **Show Tables**statement is used to list the tables in schema or in the current schema.

%trino

-- Show tables in a schema (database)

show tables in mysql.world;

Before start to query the World sample database, let’s show the World database tables information. This can be done using the **describe**statement.

1. **Describe Tables**

The **Describe Table**statement is used to list the columns in a table along with their data type and other attributes.

%trino

-- Show informations of the country table

describe mysql.world.country ;

%trino

-- Show informations of the city table

describe mysql.world.city;

%trino

-- Show informations of the countrylanguage table

describe mysql.world.countrylanguage;

**Writing Some Queries**

Trino executes SQL statements and turns these statements into queries, that are executed across a distributed cluster of coordinator and workers.

**Query 1**

Write a query to get all the countries that speak French.

%trino

-- # 1. Get all the countries that speak French

select c.name, l.language, l.percentage

from mysql.world.country c

join mysql.world.countrylanguage l on c.code = l.countrycode

where l.language = 'French'

order by l.percentage desc;

**Query 2**

Write a query to display the total number of cities for each country.

%trino

-- # 2. Display the total number of cities for each country

select count(ct.id) as num\_cities, c.name

from mysql.world.country c

join mysql.world.city ct on c.code = ct.countrycode

group by c.name

order by num\_cities desc;

**Query 3**

Write a query to get all the cities in Mexico with a population of greater than 500,000.

%trino

-- # 3. Get all the cities in Mexico with a population of greater than 500,000

select ct.population, ct.name

from mysql.world.country c

join mysql.world.city ct on c.code = ct.countrycode

where c.name = 'Mexico' and ct.population > 500000

order by ct.population desc;

**Query 4**

Write a query to get all languages in each country with a percentage greater than 89%.

%trino

-- # 4. Get all languages in each country with a percentage greater than 89%

select c.name, l.language, l.percentage

from mysql.world.country c

join mysql.world.countrylanguage l on c.code = l.countrycode

where l.percentage > 89

order by l.percentage desc;

**Query 5**

Write a query to get all the countries with Surface Area below 501 and Population greater than 100,000.

%trino

-- # 5. Get all the countries with Surface Area below 501 and Population greater than 100,000

select c.name, c.surfacearea, c.population

from mysql.world.country c

where c.surfacearea < 501 and c.population > 100000

order by c.population desc;

**Query 6**

Write a query to get countries with only Constitutional Monarchy with a capital greater than 200 and a life expectancy greater than 75 years.

%trino

-- # 6. Get countries with only Constitutional Monarchy with a capital greater than 200 and a life expectancy greater than 75 years

select c.name, c.governmentform, c.capital, c.lifeexpectancy

from mysql.world.country c

where c.governmentform = 'Constitutional Monarchy' and c.capital > 200 and c.lifeexpectancy > 75

order by c.lifeexpectancy;

**Query 7**

Write a query to get all the cities of Argentina inside the Buenos Aires district and have the population greater than 500, 000.

%trino

-- # 7. Get all the cities of Argentina inside the Buenos Aires district and have the population greater than 500, 000

select c.name, ct.name, ct.district, ct.population

from mysql.world.country c

join mysql.world.city ct on c.code = ct.countrycode

where c.name = 'Argentina' and ct.population > 500000

order by ct.population desc;

**Query 8**

Write a query to summarize the number of countries in each region.

%trino

-- # 8. Summarize the number of countries in each region

select count(c.code) as count, c.region

from mysql.world.country c

group by(c.region)

order by count desc;

**Summary**

In this tutorial we presented what Trino is and why it’s used. You’ve also seen some of Trino’s strongest attributes, such as its data integration and federated query capabilities. We also covered at a high level Trino’s architecture and introduced Trino terminology such as connectors, catalogs and schemas. Finaly we wrote some queries to practice Trino SQL statements.

Open Zeppelin Note

[Getting Started With Trino](http://localhost:19995/#/notebook/2JKRCX4VM)