Apache Presto

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Apache Presto is an open source distributed SQL engine. Presto originated at Facebook for data analytics needs and later was open sourced. Now, Teradata joins Presto community and offers support.

Apache Presto is very useful for performing queries even petabytes of data. Extensible architecture and storage plugin interfaces are very easy to interact with other file systems. Most of today's best industrial companies are adopting Presto for its interactive speeds and low latency performance.

This tutorial explores Presto architecture, configuration, and storage plugins. It discusses the basic and advanced queries and finally concludes with real-time examples.

Audience

This tutorial has been prepared for professionals aspiring to make a career in Big Data Analytics. This tutorial will give you enough understanding on Apache Presto.

Prerequisites

Before proceeding with this tutorial, you must have a good understanding of Core Java, DBMS and any of the Linux operating systems.

Disclaimer & Copyright

Table of Contents

1. Presto – Overview

Data analytics is the process of analyzing raw data to gather relevant information for better decision making. It is primarily used in many organizations to make business decisions. Well, big data analytics involves a large amount of data and this process is quite complex, hence companies use different strategies.

For example, Facebook is one of the leading data driven and largest data warehouse company in the world. Facebook warehouse data is stored in Hadoop for large scale computation. Later, when warehouse data grew to petabytes, they decided to develop a new system with low latency. In the year of 2012, Facebook team members designed "**Presto**" for interactive query analytics that would operate quickly even with petabytes of data.

What is Apache Presto?

Apache Presto is a distributed parallel query execution engine, optimized for low latency and interactive query analysis. Presto runs queries easily and scales without down time even from gigabytes to petabytes.

A single Presto query can process data from multiple sources like HDFS, MySQL, Cassandra, Hive and many more data sources. Presto is built in Java and easy to integrate with other data infrastructure components. Presto is powerful, and leading companies like Airbnb, DropBox, Groupon, Netflix are adopting it.

Presto – Features

Presto contains the following features:

- Simple and extensible architecture
- Pluggable connectors - Presto supports pluggable connector to provide metadata and data for queries.
- Pipelined executions - Avoids unnecessary I/O latency overhead
- User-defined functions - Analysts can create custom user-defined functions to migrate easily
- Vectorized columnar processing

Presto – Benefits

Here is a list of benefits that Apache Presto offers -

- Specialized SQL operations
- Easy to install and debug
- Simple storage abstraction
- Quickly scales petabytes data with low latency

Presto – Applications

Presto supports most of today's best industrial applications. Let's take a look at some of the notable applications.

- **Facebook**: Facebook built Presto for data analytics needs. Presto easily scales large velocity of data.
- **Teradata**: Teradata provides end-to-end solutions in Big Data analytics and data warehousing. Teradata contribution to Presto makes it easier for more companies to enable all analytical needs.
- **Airbnb**: Presto is an integral part of the Airbnb data infrastructure. Well, hundreds of employees are running queries each day with the technology.

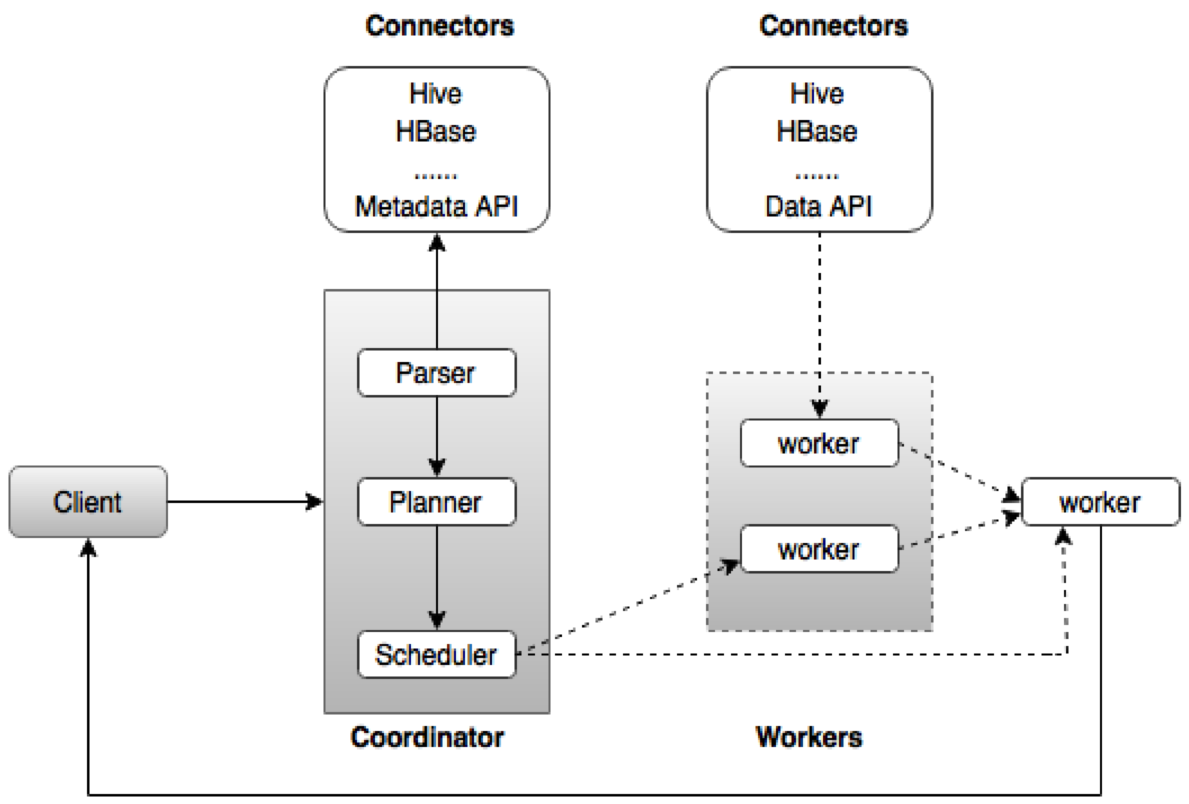
Why Presto?

Presto supports standard ANSI SQL which has made it very easy for data analysts and developers. Though it is built in Java, it avoids typical issues of Java code related to memory allocation and garbage collection. Presto has a connector architecture that is Hadoop friendly. It allows to easily plug in file systems.

Presto runs on multiple Hadoop distributions. In addition, Presto can reach out from a Hadoop platform to query Cassandra, relational databases, or other data stores. This cross-platform analytic capability allows Presto users to extract maximum business value from gigabytes to petabytes of data.

2. Presto – Architecture

The architecture of Presto is almost similar to classic MPP (massively parallel processing) DBMS architecture. The following diagram illustrates the architecture of Presto.



The above diagram consists of different components. Following table describes each of the component in detail.

Component	Description
Client	Client (Presto CLI) submits SQL statements to a coordinator to get the result.
Coordinator	Coordinator is a master daemon. The coordinator initially parses the SQL queries then analyzes and plans for the query execution. Scheduler performs pipeline execution, assigns work to the closest node and monitors progress.
Connector	Storage plugins are called as connectors. Hive, HBase, MySQL, Cassandra and many more act as a connector; otherwise you can also implement a custom one. The connector provides metadata and data for queries. The coordinator uses the connector to get metadata for building a query plan.

Worker	The coordinator assigns task to worker nodes. The workers get actual data from the connector. Finally, the worker node delivers result to the client.

Presto – Workflow

Presto is a distributed system that runs on a cluster of nodes. Presto's distributed query engine is optimized for interactive analysis and supports standard ANSI SQL, including complex queries, aggregations, joins, and window functions. Presto architecture is simple and extensible. Presto client (CLI) submits SQL statements to a master daemon coordinator.

The scheduler connects through execution pipeline. The scheduler assigns work to nodes which is closest to the data and monitors progress. The coordinator assigns task to multiple worker nodes and finally the worker node delivers the result back to the client. The client pulls data from the output process. Extensibility is the key design. Pluggable connectors like Hive, HBase, MySQL, etc., provides metadata and data for queries. Presto was designed with a "simple storage abstraction" that makes it easy to provide SQL query capability against these different kind of data sources.

Execution Model

Presto supports custom query and execution engine with operators designed to support SQL semantics. In addition to improved scheduling, all processing is in memory and pipelined across the network between different stages. This avoids unnecessary I/O latency overhead.

3. Presto – Installation

This chapter will explain how to install Presto on your machine. Let's go through the basic requirements of Presto,

- Linux or Mac OS
- Java version 8

Now, let's continue the following steps to install Presto on your machine.

Verifying Java installation

Hopefully, you have already installed Java version 8 on your machine right now, so you just verify it using the following command.

```
$ java -version
```

If Java is successfully installed on your machine, you could see the version of installed Java. If Java is not installed, follow the subsequent steps to install Java 8 on your machine.

Download JDK. Download the latest version of JDK by visiting the following link.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

The latest version is JDK 8u 92 and the file is "jdk-8u92-linux-x64.tar.gz". Please download the file on your machine.

After that, extract the files and move to the specific directory.

Then set Java alternatives. Finally Java will be installed on your machine.

Apache Presto Installation

Download the latest version of Presto by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-server/0.149/

Now the latest version of "presto-server-0.149.tar.gz" will be downloaded on your machine.

Extract tar Files

Extract the **tar** file using the following command:

```
$ tar  -zxf  presto-server-0.149.tar.gz
$ cd presto-server-0.149
```

Configuration Settings

Create "data" directory

Create a data directory outside the installation directory, which will be used for storing logs, metadata, etc., so that it is to be easily preserved when upgrading Presto. It is defined using the following code:

```
$ cd
$ mkdir data
```

To view the path where it is located, use the command "pwd". This location will be assigned in the next node.properties file.

Create "etc" directory

Create an etc directory inside Presto installation directory using the following code:

```
$ cd presto-server-0.149
$ mkdir etc
```

This directory will hold configuration files. Let's create each file one by one.

Node Properties

Presto node properties file contains environmental configuration specific to each node. It is created inside etc directory (etc/node.properties) using the following code:

```
$ cd etc
$ vi node.properties

node.environment=production
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff
node.data-dir=/Users/../workspace/Presto
```

After making all the changes, save the file, and quit the terminal. Here **node.data** is the location path of the above created data directory. **node.id** represents the unique identifier for each node.

JVM Config

Create a file "jvm.config" inside etc directory (etc/jvm.config). This file contains a list of command line options used for launching the Java Virtual Machine.

```
$ cd etc
$ vi jvm.config

-server
-Xmx16G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:OnOutOfMemoryError=kill -9 %p
```

After making all the changes, save the file, and quit the terminal.

Config Properties

Create a file "config.properties" inside etc directory(etc/config.properties). This file contains the configuration of Presto server. If you are setting up a single machine for testing, Presto server can function only as the coordination process as defined using the following code:

```
$ cd etc
$ vi config.properties

coordinator=true
node-scheduler.include-coordinator=true
http-server.http.port=8080
query.max-memory=5GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri= http://localhost:8080
```

Here,

- **coordinator**: master node.
- **node-scheduler.include-coordinator**: Allows scheduling work on the coordinator.

- **http-server.http.port**: Specifies the port for the HTTP server.
- **query.max-memory=5GB**: The maximum amount of distributed memory.
- **query.max-memory-per-node=1GB**: The maximum amount of memory per node.
- **discovery-server.enabled**: Presto uses the Discovery service to find all the nodes in the cluster.
- **discovery.uri**: The URI to the Discovery server.

If you are setting up multiple machine Presto server, Presto will function as both coordination and worker process. Use this configuration setting to test Presto server on multiple machines.

Configuration for Coordinator

```
$ cd etc
$ vi config.properties

coordinator=true
node-scheduler.include-coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery-server.enabled=true
discovery.uri= http://localhost:8080
```

Configuration for Worker

```
$ cd etc
$ vi config.properties

coordinator=false
http-server.http.port=8080
query.max-memory=50GB
query.max-memory-per-node=1GB
discovery.uri= http://localhost:8080
```

Log Properties

Create a file "log.properties" inside etc directory(etc/log.properties). This file contains minimum log level for named logger hierarchies. It is defined using the following code:

```
$ cd etc
$ vi log.properties

com.facebook.presto=INFO
```

Save the file and quit the terminal. Here, four log levels are used such as DEBUG, INFO, WARN and ERROR. Default log level is INFO.

Catalog Properties

Create a directory "catalog" inside etc directory(etc/catalog). This will be used for mounting data. For example, create **etc/catalog/jmx.properties** with the following contents to mount the **jmx connector** as the jmx catalog:

```
$ cd etc
$ mkdir catalog
$ cd catalog
$ vi jmx.properties

connector.name=jmx
```

Start Presto

Presto can be started using the following command,

```
$ bin/launcher start
```

Then you will see the response similar to this,

```
Started as 840
```

Run Presto

To launch Presto server, use the following command:

```
$ bin/launcher run
```

After successfully launching Presto server, you can find log files in "var/log" directory.

- **launcher.log**: This log is created by the launcher and is connected to the stdout and stderr streams of the server.
- **server.log**: This is the main log file used by Presto.
- **http-request.log**: HTTP request received by the server.

As of now, you have successfully installed Presto configuration settings on your machine. Let's continue the steps to install Presto CLI.

Install Presto CLI

The Presto CLI provides a terminal-based interactive shell for running queries.

Download the Presto CLI by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.149/

Now "presto-cli-0.149-executable.jar" will be installed on your machine.

Run CLI

After downloading the presto-cli, copy it to the location which you want to run it from. This location may be any node that has network access to the coordinator. First change the name of the Jar file to Presto. Then make it executable with **chmod+x** command using the following code:

```
$ mv presto-cli-0.149-executable.jar presto

$ chmod +x presto
```

Now execute CLI using the following command,

```
./presto --server localhost:8080 --catalog jmx --schema default


Here jmx(Java Management Extension) refers to catalog and default referes to
schema.
```

You will see the following response,

```
 presto:default>
```

Now type "jps" command on your terminal and you will see the running daemons.

Stop Presto

After having performed all the executions, you can stop the presto server using the following command:

```
$ bin/launcher stop
```

4. Presto – Configuration Settings

This chapter will discuss the configuration settings for Presto.

Presto Verifier

The Presto Verifier can be used to test Presto against another database (such as MySQL), or to test two Presto clusters against each other.

Create Database in MySQL

Open MySQL server and create a database using the following command.

```
create database test
```

Now you have created "test" database in the server. Create the table and load it with the following query.

```
CREATE TABLE verifier_queries(
    id INT NOT NULL AUTO_INCREMENT,
    suite VARCHAR(256) NOT NULL,
    name VARCHAR(256),
    test_catalog VARCHAR(256) NOT NULL,
    test_schema VARCHAR(256) NOT NULL,
    test_prequeries TEXT,
    test_query TEXT NOT NULL,
    test_postqueries TEXT,
    test_username VARCHAR(256) NOT NULL default 'verifier-test',
    test_password VARCHAR(256),
    control_catalog VARCHAR(256) NOT NULL,
    control_schema VARCHAR(256) NOT NULL,
    control_prequeries TEXT,
    control_query TEXT NOT NULL,
    control_postqueries TEXT,
    control_username VARCHAR(256) NOT NULL default 'verifier-test',
    control_password VARCHAR(256),
    session_properties_json TEXT,
   PRIMARY KEY (id)
);
```

Add Config Settings

Create a properties file to configure the verifier:

```
$ vi config.properties


suite=mysuite
query-database=jdbc:mysql://localhost:3306/tutorials?user=root&password=pwd
control.gateway=jdbc:presto://localhost:8080
test.gateway=jdbc:presto://localhost:8080
thread-count=1
```

Here, in the **query-database** field, enter the following details: mysql database name, user name, and password.

Download JAR File

Download Presto-verifier jar file by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-verifier/0.149/

Now the version "**presto-verifier-0.149-executable.jar**" is downloaded on your machine.

Execute JAR

Execute the JAR file using the following command,

```
$ mv presto-verifier-0.149-executable.jar verifier


$ chmod+x verifier
```

Run Verifier

Run the verifier using the following command,

```
$ ./verifier config.properties
```

Create Table

Let’s create a simple table in **“test”** database using the following query.

```
create table product(id int not null, name varchar(50))
```

Insert Table

After creating a table, insert two records using the following query,

```
insert into product values(1,’Phone')

insert into product values(2,’Television’)
```

Run Verifier Query

Execute the following sample query in the verifier terminal (./verifier config.propeties) to check the verifier result.

Sample Query:

```
insert into verifier_queries (suite, test_catalog, test_schema, test_query,
control_catalog, control_schema, control_query) values ('mysuite', 'mysql',
'default', 'select * from mysql.test.product', 'mysql', 'default', 'select *
from mysql.test.product');
```

Here, **select * from mysql.test.product** query refers to mysql catalog, **test** is database name and **product** is table name. In this way, you can access mysql connector using Presto server.

Here, two same select queries are tested against each other to see the performance. Similarly, you can run other queries to test the performance results. You can also connect two Presto clusters to check the performance results.

5. Presto – Administration Tools

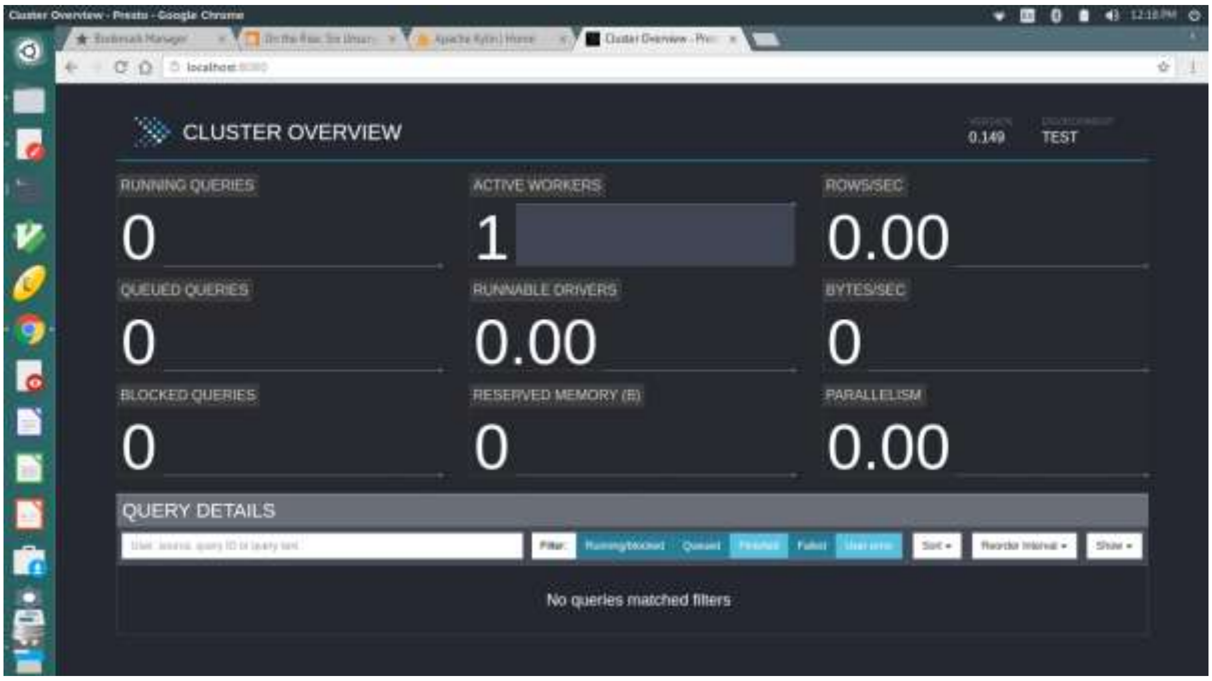
In this chapter, we will discuss the administration tools used in Presto. Let's start with the Web Interface of Presto.

Web Interface

Presto provides a web interface for monitoring and managing queries. It can be accessed from the port number specified in the coordinator Config Properties.

Start Presto server and Presto CLI. Then you can access the web interface from the following url:

http://localhost:8080



CLUSTER OVERVIEW

VERSION 0.149 ENVIRONMENT TEST

RUNNING QUERIES 0

ACTIVE WORKERS 1

ROWS/SEC 0.00

QUEUED QUERIES 0

RUNNABLE DRIVERS 0.00

BYTES/SEC 0

BLOCKED QUERIES 0

RESERVED MEMORY (B) 0

PARALLELISM 0.00

QUERY DETAILS

No queries matched filters

The output will be similar to the above screen.

Here, the main page has a list of queries along with information like unique query ID, query text, query state, percentage completed, username and source from which this query is originated. Latest queries are running first, then completed or not completed queries are displayed at the bottom.

Tuning the Performance on Presto

If Presto cluster is having any performance-related issues, change your default configuration settings to the following settings.

Config Properties

- **task. info -refresh-max-wait**: Reduces coordinator work load.
- **task.max-worker-threads**: Splits the process and assigns to each worker nodes.
- **distributed-joins-enabled**: Hash-based distributed joins.
- **node-scheduler.network-topology**: Sets network topology to scheduler.

JVM Settings

Change your default JVM settings to the following settings. This will be helpful for diagnosing garbage collection issues.

```
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintGCCause
-XX:+PrintGCDateStamps
-XX:+PrintGCTimeStamps
-XX:+PrintGCDetails
-XX:+PrintReferenceGC
-XX:+PrintClassHistogramAfterFullGC
-XX:+PrintClassHistogramBeforeFullGC
-XX:PrintFLSStatistics=2
-XX:+PrintAdaptiveSizePolicy
-XX:+PrintSafepointStatistics
-XX:PrintSafepointStatisticsCount=1
```

6. Presto – Basic SQL Operations

In this chapter, we will discuss how to create and execute queries on Presto. Let us go through Presto supported basic data types.

Basic Data Types

The following table describes the basic data types of Presto.

Data type	Description
VARCHAR	Variable length character data
BIGINT	A 64-bit signed integer
DOUBLE	A 64-bit floating point double precision value
DECIMAL	A fixed precision decimal number. For example DECIMAL(10,3) - 10 is precision, i.e. total number of digits and 3 is scale value represented as fractional point. Scale is optional and default value is 0
BOOLEAN	Boolean values true and false
VARBINARY	Variable length binary data
JSON	JSON data
DATE	Date data type represented as year-month-day
TIME, TIMESTAMP, TIMESTAMP with TIME ZONE	TIME - Time of the day (hour-min-sec-millisecond) TIMESTAMP - Date and time of the day TIMESTAMP with TIME ZONE - Date and time of the day with time zone from the value
INTERVAL	Stretch or extend date and time data types

ARRAY	Array of the given component type. For example, ARRAY[5,7]
MAP	Map between the given component types. For example, MAP(ARRAY[‘one’,’two’],ARRAY[5,7])
ROW	Row structure made up of named fields

Presto – Operators

Presto operators are listed in the following table.

Operator	Description
Arithmetic operator	Presto supports arithmetic operators such as +, -, *, /, %
Relational operator	<,>,<=,>=,=,<>
Logical operator	AND, OR, NOT
Range operator	Range operator is used to test the value in a specific range. Presto supports BETWEEN, IS NULL, IS NOT NULL, GREATEST and LEAST
Decimal operator	Binary arithmetic decimal operator performs binary arithmetic operation for decimal type Unary decimal operator: The **- operator** performs negation
String operator	The **‘||’ operator** performs string concatenation
Date and time operator	Performs arithmetic addition and subtraction operations on date and time data types
Array operator	Subscript operator[] - access an element of an array Concatenation operator || - concatenate an array with an array or an element of the same type
Map operator	Map subscript operator [] - retrieve the value corresponding to a given key from a map

Let us look at some simple examples on Presto operators.

Arithmetic Operators

Following sample queries are an example of arithmetic operators.

Query 1:

```
presto:default> select 4+5 as addition;
```

Result:

```
 addition
----------
        9
(1 row)
```

From the above result "as" is used for alias for the query.

Query 2:

```
presto:default> select 4.5-3.5 as sub;
```

Result:

```
 sub
-----
 1.0
(1 row)
```

The output is subtraction between two values.

Query 3:

```
presto:default> select 4%2 as modulus;
```

Result:

```
 modulus
---------
       0
(1 row)
```

The output is returned as the modulus for given values.

Relational Operators

Relational operators are used to compare between two values. Take a look at the following queries. They show how relational operators work.

Query 1:

```
presto:default> select 4 < 5 as lessthan;
```

Result:

```
 lessthan
----------
 true
(1 row)
```

The above condition is true, hence the result is true.

Query 2:

```
presto:default> select 25 >= 45 as greater;
```

Result:

```
 greater
---------
 false
(1 row)
```

The above condition is false, hence the result is false.

Query 3:

```
presto:default> select 3<>5 as notequal;
```

Result:

```
 notequal
----------
 true
```

Here 3 is not equal to 5, hence the result is returned as true.

Logical Operators

Logical operators work on Boolean operands and produce Boolean results. Let's take a few examples to see how logical operators work in Presto:

Query 1:

```
select 3 < 2 and 4 > 1 as logical;
```

Result:

```
 logical
---------
 false
```

Here, 4 > 1 is false so "AND" operator returns the result as false.

Query 2:

```
presto:default> select 3<2 or 4>1 as logical;
```

Result:

```
 logical
---------
 true
(1 row)
```

Both conditions are true, hence the result is true.

Query 3:

```
presto:default> select 3 not in (1,2) as not_operator;
```

Result:

```
 not_operator
--------------
 true
(1 row)
```

Here, 3 value is not in the given set (1,2) hence it produces true result.

Range Operator

Between operator is used to test the particular value, which exists from minimum to maximum range.

Query 1:

```
presto:default> select 30.5 between 10 and 40 as range;
```

Result:

```
 range
-------
 true
(1 row)
```

Query 2:

```
presto:default> select 4.5 is null;
```

Result:

```
 _col0
-------
 false
(1 row)
```

Here, 4.5 is a value and not null since it is checked with null, so the result is false.

Query 3:

```
presto:default> select 3 is not null;
```

Result:

```
 _col0
-------
 true
(1 row)
```

Greatest(x,y)

If the value of **x** is greater than **y**, then it returns **x**, otherwise **y**.

Query:

```
presto:default> select greatest(200,300) as greatest;
```

Result:

```
  greatest
----------
       300
(1 row)
```

The output is returned as the greatest of two values.

Least(x,y)

Returns the least value from the two given values.

Query:

```
presto:default> select least('a','b') as result;
```

Result:

```
 result
--------
 a
```

Here, the lowest character value is 'a' hence it is returned in the result.

Decimal Operator

Query 1:

```
presto:default> select decimal '123' + decimal ’22’ as decimal;
```

Result:

```
 decimal
-------
 145
```

The output is decimal addition.

Query 2:

```
presto:default> select decimal '123' - decimal ’22’ as unary_minus;
```

Result:

```
 unary_minus
-------
 101
```

Query 3:

```
presto:default> select decimal '123' between decimal '100' and decimal '200' as
decimal_range;
```

Result:

```
 decimal_range
---------------
 true
```

Here, 123 lies between the specified range.

String Operator

String operator “||” performs concatenation.

Query:

```
presto:default> select 'tutorials' || 'point' as string_concat;
```

Result:

```
 string_concat
----------------
 tutorialspoint
```

Here, the output is performed as a concatenation of the two given strings.

Date and Time Operator

Query 1:

```
presto:default> select date '2016-06-20' + interval '2' day as date_add;
```

Result:

```
   date_add
------------
 2016-06-22
```

The above output indicates two days are added from the specified date using an interval data type.

Query 2:

```
presto:default> select time '12:30' + interval '1' hour as time_add;
```

Result:

```
    time_add
--------------
 13:30:00.000
```

Here, one hour is added from the given time.

Query 3:

```
presto:default> select timestamp '2016-06-21 12:32' + interval '2' year as
timestamp_add;
```

Result:

```
       timestamp_add
-------------------------
 2018-06-21 12:32:00.000
```

Array Operator

Following are the types of Array operators.

Array subscript[] Operator

Query:

```
presto:default> select array[1,2,3] as array_elements;
```

Result:

```
 array_elements
----------------
 [1, 2, 3]
```

Array subscript operator returns array elements.

Array Concatenation Operator

Query:

```
presto:default> select array[1,2,3] || array[4] as array_concat;
```

Result:

```
 array_concat
--------------
 [1, 2, 3, 4]
```

Here, two arrays are concatenated with one array.

Map Operator

Following are the types of Map operators.

Map Subscript Operator

Query:

```
presto:default> select map(array[1,2],array[2,3]) as map;
```

Result:

```
  map
------------
 {1=2, 2=3}
```

Map operator returns array elements with keys and values.

7. Presto – SQL Functions

As of now we were discussing running some simple basic queries on Presto. This chapter will discuss the important SQL functions.

Math Functions

Math functions operate on mathematical formulas. Following table describes the list of functions in detail.

Functions	Description
abs(x)	Returns the absolute value of **x**
cbrt(x)	Returns the cube root of **x**
ceiling(x)	Returns the **x** value rounded up to the nearest integer
ceil(x)	Alias for ceiling(x)
degress(x)	Returns the degree value for **x**
e(x)	Returns the double value for Euler's number
exp(x)	Returns the exponent value for Euler's number
floor(x)	Returns **x** rounded down to the nearest integer
from_base(string,radix)	Returns the value of string interpreted as a base-radix number
ln(x)	Returns the natural logarithm of **x**
log2(x)	Returns the base 2 logarithm of **x**
log10(x)	Returns the base 10 logarithm of **x**
log(x,y)	Returns the base **y** logarithm of **x**
mod(n,m)	Returns the modulus (remainder) of **n** divided by **m**

pi()	Returns pi value. The result will be returned as a double value
power(x,p)	Returns power of value '**p**' to the **x** value
pow(x,p)	Alias for power(x,p)
radians(x)	converts the angle **x** in degree radians
rand()	Alias for radians()
random()	Returns the pseudo-random value
rand(n)	Alias for random()
round(x)	Returns the rounded value for x
round(x,d)	**x** value rounded for the '**d**' decimal places
sign(x)	Returns the signum function of x, i.e., 0 if the argument is 0 1 if the argument is greater than 0 -1 if the argument is less than 0 For double arguments, the function additionally returns: NaN if the argument is NaN 1 if the argument is +Infinity -1 if the argument is -Infinity
sqrt(x)	Returns the square root of **x**
to_base(x,radix)	Return type is archer. The result is returned as the base radix for **x**
truncate(x)	Truncates the value for **x**
width_bucket(x, bound1, bound2, n)	Returns the bin number of **x** specified bound1 and bound2 bounds and n number of buckets
width_bucket(x, bins)	Returns the bin number of **x** according to the bins specified by the array bins

Let us look at some simple examples on Math functions.

abs(x)

Query:

```
presto:default> select abs(4.65) as absolute;
```

Result:

```
 absolute
----------
     4.65
```

Here, the absolute value is returned for the given value.

cbrt(x)

Query:

```
presto:default> select cbrt(4) as cubic_root;
```

Result:

```
     cubic_root
--------------------
 1.5874010519681996
```

The cubic root of 4 is returned as an output.

ceiling(x)

Query:

```
presto:default> select ceiling(4.7) as ceiling;
```

Result:

```
 ceiling
---------
     5.0
```

Ceiling of 4.7 is 5.0, which is rounded to the next integer.

floor(x)

Query:

```
presto:default> select floor(4.8) as floor;
```

Result:

```
 floor
-------
   4.0
```

floor(4.8) result is 4.0

degrees(x)

Query:

```
presto:default> select degrees(2) as degree;
```

Result:

```
    degree
--------------------
 114.59155902616465
```

The output is the degree value for the integer 2.

e()

Query:

```
presto:default> select e() as exponent;
```

Result:

```
      exponent
-------------------
 2.718281828459045
```

The exponent value is returned as an output.

log2(x)

Query:

```
presto:default> select log2(5) as log_value;
```

Result:

```
   log_value
-------------------
 2.321928094887362
```

mod(n,m)

Query:

```
presto:default> select mod(2,4) as mod_value;
```

Result:

```
 mod_value
-----------
         2
```

Here, the result of (2,4) modulus is 2.

power(x,p)

Query:

```
presto:default> select power(2,3) as power;
```

Result:

```
 power
-------
   8.0
```

Here, the output return type is double.

radians(x)

Query:

```
presto:default> select radians(4) as radian_value;
```

Result:

```
    radian_value
---------------------
 0.06981317007977318
```

random(x)

Query:

```
presto:default> select random(6) as random_value;
```

Result:

```
 random_value
--------------
            2
```

A random number is generated as a result.

round(x)

Query:

```
presto:default> select round(5.9) as round_value;
```

Result:

```
 round_value
-------------
         6.0
```

The output is rounded as a result.

truncate(x)

Query:

```
presto:default> select truncate(5.9) as truncate_value;
```

Result:

```
 truncate_value
----------------
            5.0
```

Here, the result is truncated.

sqrt(x)

Query:

```
presto:default> select sqrt(3) as squareroot;
```

Result:

```
   squareroot
--------------------
 1.7320508075688772
```

The square root of 3 is 1.732.

to_base(x,radix)

Query:

```
presto:default> select to_base(3,4) as base;
```

Result:

```
 base
------
 3
```

width_bucket(x,bound1,bound2,n)

Query:

```
presto:default> select width_bucket(5,3,4,5) as width;
```

Result:

```
 width
-------
     6
```

From the above input, 3 and 4 are different bounds and returns the bucket width as 6.

width_bucket(x,bins)

Query:

```
presto:default> select width_bucket(6,array[1,2,3]) as width;
```

Result:

```
 width
-------
     3
```

Here, the array maximum value is set as bucket width.

Trigonometric Functions

Trigonometric functions arguments are represented as radians(). Following table lists out the functions.

Functions	Description
acos(x)	Returns the inverse cosine value(x)
asin(x)	Returns the inverse sine value(x)
atan(x)	Returns the inverse tangent value(x)
atan2(y,x)	Returns the inverse tangent value(y/x)

cos(x)	Returns the cosine value(x)
cosh(x)	Returns the hyperbolic cosine value(x)
sin(x)	Returns the sine value(x)
tan(x)	Returns the tangent value(x)
tanh(x)	Returns the hyperbolic tangent value(x)

acos(x)

Query:

```
presto:default> select acos(0.5) as inversecosine;
```

Result:

```
   inversecosine
--------------------
 1.0471975511965979
```

atan2(y,x)

Query:

```
presto:default> select atan2(2,3) as inverse_tangent;
```

Result:

```
 inverse_tangent
--------------------
 0.5880026035475675
```

sin(x)

Query:

```
presto:default> select sin(90) as sine;
```

Result:

```
     sine
--------------------
 0.8939966636005579
```

cosh(x)

Query:

```
presto:default> select cosh(1) as hyperbolic_cosine;
```

Result:

```
 hyperbolic_cosine
-------------------
 1.543080634815244
```

Bitwise Functions

The following table lists out the Bitwise functions.

Functions	Description
bit_count(x, bits)	Count the number of bits
bitwise_and(x,y)	Perform bitwise AND operation for two bits, **x** and **y**
bitwise_or(x,y)	Bitwise OR operation between two bits **x, y**
bitwise_not(x)	Bitwise Not operation for bit **x**
bitwise_xor(x,y)	XOR operation for bits **x, y**

bit_count(x,bits)

Query:

```
presto:default> select bit_count(1,2) as bitcounts;
```

Result:

```
 bitcounts
-----------
         1
```

Here, the total number of bit is 1.

bitwise_and(x,y)

Query:

```
presto:default> select bitwise_and(2,3) as bit_and;
```

Result:

```
 bit_and
---------
       2
```

The above and performed between two bits 2(010) and 3(011). Hence, the result is 2.

bitwise_or(x,y)

Query:

```
presto:default> select bitwise_or(2,3) as bit_or;
```

Result:

```
 bit_or
--------
      3
```

Here, the bits 2(010) and 3(011) OR operation result is 3.

bitwise_not(x)

Query:

```
presto:default> select bitwise_not(3) as bit_not;
```

Result:

```
 bit_not
---------
      -4
```

Here, the complement of 3 value is -4.

bitwise_xor(x,y)

Query:

```
presto:default> select bitwise_xor(2,3) as bit_xor;
```

Result:

```
 bit_xor
---------
       1
```

Here, XOR operation for bits 2 and 3 is 1.

String Functions

Following table lists out the String functions.

Functions	Description
concat(string1, ..., stringN)	Concatenate the given strings
length(string)	Returns the length of the given string
lower(string)	Returns the lowercase format for the string
upper(string)	Returns the uppercase format for the given string

lpad(string, size, padstring)	Left padding for the given string
ltrim(string)	Removes the leading whitespace from the string
replace(string, search, replace)	Replaces the string value
reverse(string)	Reverses the operation performed for the string
rpad(string, size, padstring)	Right padding for the given string
rtrim(string)	Removes the trailing whitespace from the string
split(string, delimiter)	Splits the string on delimiter and returns an array of size at the most limit
split_part(string, delimiter, index)	Splits the string on delimiter and returns the field index
strpos(string, substring)	Returns the starting position of the substring in the string
substr(string, start)	Returns the substring for the given string
substr(string, start, length)	Returns the substring for the given string with the specific length
trim(string)	Removes the leading and trailing whitespace from the string

concat(string1,string2,..n)

Query:

```
presto:default> select concat('tut','orials','point') as string_concat;
```

Result:

```
 string_concat
---------------
  tutorialspoint
```

The above output is concatenation of the given strings.

length(string)

Query:

```
presto:default> select length('tutorialspoint') as string_length;
```

Result:

```
 string_length
---------------
            14
```

The output is the length of the given string.

lower(string)

Query:

```
presto:default> select lower('TutorialspoinT') as string_lower;
```

Result:

```
  string_lower
----------------
 tutorialspoint
```

The output is the lowercase of the given string.

upper(string)

Query:

```
presto:default> select upper('tutorialspoint') as string_upper;
```

Result:

```
  string_upper
----------------
 TUTORIALSPOINT
```

The output is the uppercase of the given string.

lpad(string,size,padstring)

Query:

```
presto:default> select lpad(‘Presto',5,'o') as string_padding;
```

Result:

```
 string_padding
----------------
 Prest
```

The output is the left padding of the string size.

ltrim(string)

Query:

```
presto:default> select ltrim('Apache Presto') as string_ltrim;
```

Result:

```
string_ltrim
---------------
 Apache Presto
```

reverse(string)

Query:

```
presto:default> select reverse('Presto') as string_reverse;
```

Result:

```
 string_reverse
----------------
 otserP
```

replace(string)

Query:

```
presto:default> select replace('cat','c','r') as string_replace;
```

Result:

```
 string_replace
----------------
 rat
```

rpad(string,size,padstring)

Query:

```
presto:default> select rpad('presto',2,'o') as string_rpad;
```

Result:

```
 string_rpad
-------------
 pr
```

rtrim(string)

Query:

```
presto:default> select rtrim('apache presto') as string_rtrim;
```

Result:

```
 string_rtrim
---------------
 apache presto
```

split(string,delimiter)

Query:

```
presto:default> select split('apache presto','e') as string_split;
```

Result:

```
   string_split
-------------------
 [apach,  pr, sto]
```

split_part(string,delimiter,index)

Query:

```
presto:default> select split_part('apache presto’,'p',2);
```

Result:

```
 _col0
-------
 ache
```

strpos(string,substring)

Query:

```
presto:default> select strpos('apache','ap') as string_position;
```

Result:

```
 string_position
-----------------
        1
```

Query:

```
presto:default> select strpos('apache','c') as string_position;
```

Result:

```
 string_position
-----------------
               4
```

substr(string,start)

Query:

```
presto:default> select substr('tutorialspoint',10) as substring;
```

Result:

```
 substring
-----------
 point
```

substr(string,start,length)

Query:

```
presto:default> select substr('tutorialspoint',10,2) as substring;
```

Result:

```
 substring
-----------
 po
```

trim(string)

Query:

```
presto:default> select trim('Presto') as string_trim;
```

Result:

```
 string_trim
-------------
 Presto
```

Date and Time Functions

Following table lists out the Date and Time functions.

Functions	Description
current_date	Returns the current date
current_time	Returns the current time
current_timestamp	Returns the current timestamp
current_timezone()	Returns the current timezone
now()	Returns the current date,timestamp with the timezone
localtime	Returns the local time
localtimestamp	Returns the local timestamp

current_date

Query:

```
presto:default> select current_date as date;
```

Result:

```
    Date
------------
 2016-07-06
```

The query returns the current date.

current_time

Query:

```
presto:default> select current_time as time;
```

Result:

```
           time
---------------------------
 18:44:16.345 Asia/Kolkata
```

The query returns the current time.

current_timestamp

Query:

```
presto:default> select current_timestamp as timestamp;
```

Result:

```
              timestamp
--------------------------------------
 2016-07-06 13:14:51.911 Asia/Kolkata
```

The query returns the current timestamp.

current_timezone()

Query:

```
presto:default> select current_timezone() as timezone;
```

Result:

```
   timezone
--------------
 Asia/Kolkata
```

The query returns the current timezone.

now()

Query:

```
presto:default> select now() as today;
```

Result:

```
                today
-------------------------------------
 2016-07-06 13:15:49.704 Asia/Kolkata
```

The query returns the current date,timestamp with timezone.

localtime

Query:

```
presto:default> select localtime as local_time;
```

Result:

```
  local_time
--------------
 18:47:32.425
```

The query returns the local time.

localtimestamp

Query:

```
presto:default> select localtimestamp as local_timestamp;
```

Result:

```
     local_timestamp
-------------------------
 2016-07-06 13:17:59.347
```

The query returns the local timestamp.

Regular Expression Functions

The following table lists out the Regular Expression functions.

Functions	Description
regexp_extract_all(string, pattern)	Returns the string matched by the regular expression for the pattern
regexp_extract_all(string, pattern, group)	Returns the string matched by the regular expression for the pattern and the group
regexp_extract(string, pattern)	Returns the first substring matched by the regular expression for the pattern
regexp_extract(string, pattern, group)	Returns the first substring matched by the regular expression for the pattern and the group
regexp_like(string, pattern)	Returns the string matches for the pattern. If the string is returned, the value will be true otherwise false
regexp_replace(string, pattern)	Replaces the instance of the string matched for the expression with the pattern
regexp_replace(string, pattern, replacement)	Replace the instance of the string matched for the expression with the pattern and replacement
regexp_split(string, pattern)	Splits the regular expression for the given pattern

regexp_extract_all(string,pattern)

Query:

```
presto:default> SELECT regexp_extract_all('1a 2b 3c 6f', '\d+') as regularexp;
```

Result:

```
  regularexp
--------------
 [1, 2, 3, 6]
```

Here, the query returns the string matched by the regular expression for the pattern specified only in digits.

regexp_extract_all(string,pattern,group)

Query:

```
presto:default> SELECT regexp_extract_all('1a 2b 3c 6f', '(\d+)([a-z]+)', 2) as
regexp_group;
```

Result:

```
 regexp_group
--------------
 [a, b, c, f]
```

Here,

- First arg - string
- Second arg - pattern
- Third arg - 2 indicates two groups are used (d+ and a-z)

Hence, the query returns the string matched by the regular expression pattern (a-z) characters with the group.

regexp_extract(string,pattern)

Query:

```
presto:default> SELECT regexp_extract('1a 2b 3c 6f', '[a-z]+') as
regexp_pattern;
```

Result:

```
 regexp_pattern
----------------
 a
```

The query returns the first string matched by the expression.

regexp_extract(string,pattern,group)

Query:

```
presto:default> SELECT regexp_extract('1a 2b 3c 6f', '(\d+)', 1) as regexp;
```

Result:

```
 regexp
--------
 1
```

The query returns the first digit matched by the expression with one group.

regexp_like(string,pattern)

Query:

```
presto:default> SELECT regexp_like('1a 2b 3c 6f', '\d+c') as expression;
```

Result:

```
 expression
------------
 true
```

Here, the digit 3 has character **c**, hence the result is true.

Query:

```
presto:default> SELECT regexp_like('1a 2b 3c 6f', '\d+e') as expression;
```

Result:

```
 expression
------------
 false
```

Here, the character **e** is not in the regular expression.

regexp_replace(string,pattern)

Query:

```
presto:default> SELECT regexp_replace('1a 2b 3c 6f', '\d+[abc] ') as
expression;
```

Result:

```
 expression
------------
 6f
```

Replace the instance of the string matched for the expression with the pattern (d+[abc]).

regexp_replace(string,pattern,replacement)

Query:

```
presto:default> SELECT regexp_replace('1a 2b 3c 6f', '(\d+)([abc]) ', 'aa$2 ')
as expression;
```

Result:

```
   expression
----------------
 aaa aab aac 6f
(1 row)
```

Replace the instance of the string matched for the expression with the pattern and replacement string 'aa'.

regexp_split(string,pattern)

Query:

```
presto:default> SELECT regexp_split('1a2b3c6f', '\s*') as split;
```

Result:

```
             split
------------------------------
 [, 1, a, 2, b, 3, c, 6, f, ]
```

Split the instance of the string matched for the expression with pattern(s*).

JSON Functions

The following table lists out JSON functions.

Functions	Description
json_array_contains(json, value)	Check the value exists in a json array. If the value exists it will return true, otherwise false
json_array_get(json_array, index)	Get the element for index in json array
json_array_length(json)	Returns the length in json array
json_format(json)	Returns the json structure format
json_parse(string)	Parses the string as a json
json_size(json, json_path)	Returns the size of the value

json_array_contains

Query:

```
presto:default> SELECT json_array_contains('[11, 12, 13]', 11) as JSON;
```

Result:

```
 JSON
------
  true
```

Query:

```
presto:default> SELECT json_array_contains('[11, 12, 13]', 15) as JSON;
```

Result:

```
  JSON
-------
  false
```

json_array_get

Query 1:

```
presto:default> SELECT json_array_get('[11, 12, 13]', 0) as JSON;
```

Result:

```
 JSON
------
  11
```

Query 2:

```
presto:default> SELECT json_array_get('[11, 12, 13]', -1) as JSON;
```

Result:

```
  JSON
------
  13
```

Query 3:

```
presto:default> SELECT json_array_get('[11, 12, 13]', 5) as JSON;
```

Result:

```
 JSON
------
 NULL
```

json_array_length

Query:

```
presto:default> SELECT json_array_length('[11, 12, 13]') as JSON;
```

Result:

```
 JSON
------
    3
```

json_format

Query:

```
presto:default> select json_format(JSON '[11,12,13]') as JSON;
```

Result:

```
    JSON
------------
 [11,12,13]
```

json_parse

Query:

```
presto:default> select json_parse('[11,12,13]') as JSON;
```

Result:

```
  JSON
------------
 [11,12,13]
```

json_size

Query:

```
presto:default> SELECT json_size('{"a": [11, 12, 13]}', '$.a') as json;
```

Result:

```
 json
------
    3
```

URL Functions

The following table lists out the URL functions.

Functions	Description
url_extract_host(url)	Returns the URL's host
url_extract_path(url)	Returns the URL's path
url_extract_port(url)	Returns the URL's port
url_extract_protocol(url)	Returns the URL's protocol
url_extract_query(url)	Returns the URL's query string

url_extract_host(url)

Query:

```
presto:default> select url_extract_host('https://www.tutorialspoint.com') as
port;
```

Result:

```
          port
------------------------
 www.tutorialspoint.com
```

url_extract_path(url)

Query:

```
presto:default> select
url_extract_path('http://www.tutorialspoint.com/hive/hive_installation.htm') as
path;
```

Result:

```
            path
-----------------------------
 /hive/hive_installation.htm
```

url_extract_port(url)

Query:

```
presto:default> select url_extract_port('https://www.tutorialspoint.com:8080')
as port;
```

Result:

```
 port
------
 8080
```

url_extract_protocol(url)

Query:

```
presto:default> select url_extract_protocol('https://www.tutorialspoint.com')
as protocol;
```

Result:

```
 protocol
----------
 https
```

url_extract_query(url)

Query:

```
presto:default> select
url_extract_query('http://www.tutorialspoint.com/hive?title=hive&action=post')
as query;
```

Result:

```
         query
------------------------
 title=hive&action=post
```

Aggregate Functions

The following table lists out the Aggregate functions.

Functions	Description
avg(x)	Returns average for the given value
min(x,n)	Returns the minimum value from two values
max(x,n)	Returns the maximum value from two values
sum(x)	Returns the sum of value

count(*)	Returns the number of input rows
count(x)	Returns the count of input values
checksum(x)	Returns the checksum for **x**
arbitrary(x)	Returns the arbitrary value for **x**

min(x,n)

Query:

```
presto:default> select min(3,4) as min;
```

Result:

```
 min
-----
 [3]
```

max(x,n)

Query:

```
presto:default> select max(9,11) as max;
```

Result:

```
 max
-----
 [9]
```

sum(x)

Query:

```
presto:default> select sum(4+7+10) as sum;
```

Result:

```
 sum
-----
  21
```

count(x)

Query:

```
presto:default> select count(23) as count;
```

Result:

```
 count
-------
     1
```

count(*)

Query:

```
presto:default> select count(*) as count;
```

Result:

```
 count
-------
     1
```

checksum(x)

Query:

```
presto:default> select checksum(1) as count;
```

Result:

```
          count
-------------------------
 87 ca eb 85 b1 79 37 9e
```

Here checksum for given input 1 value is 87 ca eb 85 b1 79 37 9e

arbitrary(x)

Query:

```
presto:default> select arbitrary(1) as arbit_value;
```

Result:

```
 arbit_value
-------------
           1
```

Color Functions

Following table lists out the Color functions.

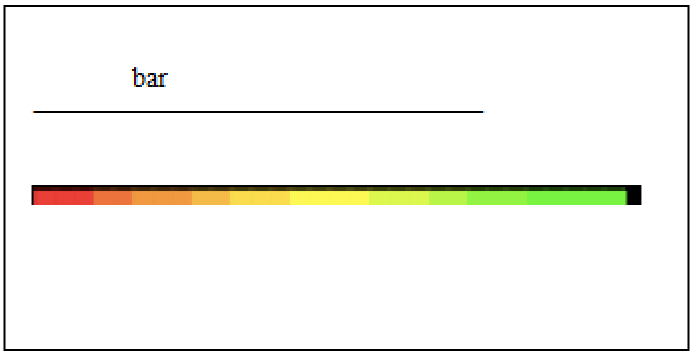
Functions	Description
bar(x, width)	Renders a single bar using rgb low_color and high_color
bar(x, width, low_color, high_color)	Renders a single bar for the specified width
color(string)	Returns the color value for the entered string
render(x, color)	Renders value **x** using the specific color using ANSI color codes
render(b)	Accepts boolean value **b** and renders a green true or a red false using ANSI color codes
rgb(red, green, blue)	Returns a color value capturing the RGB value of three component color values supplied as int parameters ranging from 0 to 255

bar(x, width)

Query:

```
presto:default> select bar(20,30) as bar;
```

Result:

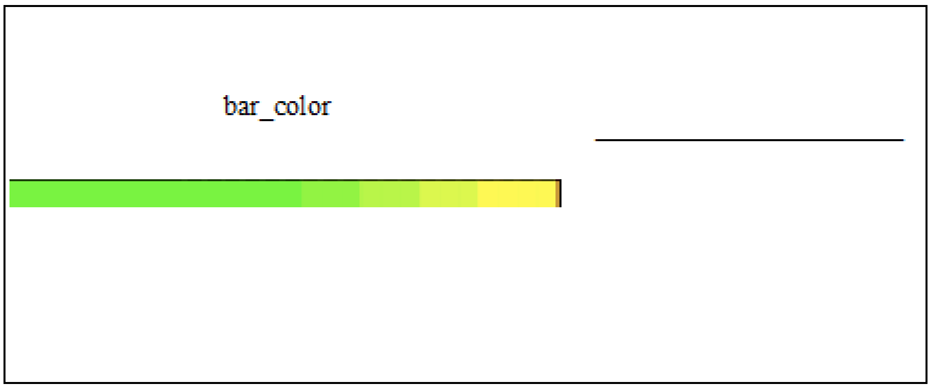

bar

bar(x, width, low_color, high_color)

Query:

```
presto:default> select bar(1,20,rgb(0,255,0),rgb(255,255,0)) as bar_color;
```

Result:


bar_color

color(string)

Query:

```
presto:default> select color('yellow') as colorvalue;
```

Result:

```
 colorvalue
-------------
 c9 e9 65 a3
```

render(x,color)

Query:

```
presto:default> select render(7,rgb(255,0,0)) as render_color;
```

Result:

```
 render_color
--------------
 7
```

render(b)

Query:

```
presto:default> select render(true) as boolean_render;
```

Result:

```
 boolean_render
----------------
 ✓
```

Boolean value true is returned, hence the result color is green.

Array Functions

The following table lists out the Array functions.

Functions	Description
array_max(x)	Finds the max element in an array
array_min(x)	Finds the min element in an array
array_sort(x)	Sorts the elements in an array
array_remove(x,element)	Removes the specific element from an array
concat(x,y)	Concatenates two arrays
contains(x,element)	Finds the given elements in an array. True will be returned if it is present, otherwise false

array_position(x,element)	Find the position of the given element in an array
array_intersect(x,y)	Performs an intersection between two arrays
element_at(array,index)	Returns the array element position
slice(x,start,length)	Slices the array elements with the specific length

array_max(x)

Query:

```
presto:default> select array_max(array[1,2,3]) as maxelement;
```

Result:

```
 maxelement
------------
          3
```

array_min(x)

Query:

```
presto:default> select array_min(array[1,2,3]) as minelement;
```

Result:

```
 minelement
------------
          1
```

array_sort(x)

Query:

```
presto:default> select array_sort(array[23,12,34]) as sort;
```

Result:

```
     sort
--------------
 [12, 23, 34]
```

array_remove(x,element)

Query:

```
presto:default> select array_remove(array[1,2,3],2) as remove;
```

Result:

```
 remove
--------
 [1, 3]
```

concat(x,y)

Query:

```
presto:default> select concat(array[1,2],array[3,4,5]) as concat;
```

Result:

```
     concat
-----------------
 [1, 2, 3, 4, 5]
```

contains(x,element)

Query:

```
presto:default> select contains(array[1,2],5) as array_contains;
```

Result:

```
 array_contains
----------------
 false
```

array_position(x,element)

Query:

```
presto:default> select array_position(array[1,2],1) as position;
```

Result:

```
 position
----------
        1
```

array_intersect(x,y)

Query:

```
presto:default> select array_position(array[11,12,13,14],2) as position;
```

Result:

```
 position
----------
        0
```

element_at(array,index)

Query:

```
presto:default> select array_intersect(array[1,2,3],array[2,4]) as
intersection;
```

Result:

```
 intersection
--------------
 [2]
```

element_at(array,index)

Query:

```
presto:default> select element_at(array[1,2,3],2) as element_position;
```

Result:

```
 element_position
------------------
       2
```

slice(x,start,length)

Query:

```
presto:default> select slice(array[1,2,3],1,2) as array_slice;
```

Result:

```
 array_slice
-------------
 [1, 2]
```

Teradata Functions

The following table lists out Teradata functions.

Functions	Description
index(string,substring)	Returns the index of the string with the given substring
substring(string,start)	Returns the substring of the given string. You can specify the start index here
substring(string,start,length)	Returns the substring of the given string for the specific start index and length of the string

index(string,substring)

Query:

```
presto:default> select index('tutorialspoint','point') as index;
```

Result:

```
 index
-------
    10
```

Here, the index position 10 for the given string 'tutorialspoint' is point.

substring(string,start)

Query:

```
presto:default> select substring('tutorialspoint',3) as substring_start;
```

Result:

```
 substring_start
-----------------
 torialspoint
```

Here, the substring of starting position 3 is 'torialspoint'.

substring(string,start,length)

Query:

```
presto:default> select substring('tutorialspoint',10,5) as substring;
```

Result:

```
 substring
-----------
 point
```

Here, the substring of starting index 10 with length 5 is 'point'.

8. Presto – MySQL Connector

The MySQL connector is used to query an external MySQL database.

Prerequisites

MySQL server installation.

Configuration Settings

Hopefully you have installed mysql server on your machine. To enable mysql properties on Presto server, you must create a file "**mysql.properties**" in "**etc/catalog**" directory. Issue the following command to create a mysql.properties file.

```
$ cd etc

$ cd catalog

$ vi mysql.properties



connector.name=mysql

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

connection-user=root

connection-password= pwd
```

Save the file and quit the terminal. In the above file, you must enter your mysql password in connection-password field.

Create Database in MySQL Server

Open MySQL server and create a database using the following command.

```
create database tutorials
```

Now you have created "tutorials" database in the server. To enable database type, use the command "use tutorials" in the query window.

Create Table

Let's create a simple table on "tutorials" database.

```
create table author(auth_id int not null, auth_name varchar(50),topic
varchar(100))
```

Insert Table

After creating a table, insert three records using the following query.

```
insert into author values(1,'Doug Cutting','Hadoop')
insert into author values(2,’James Gosling','java')
insert into author values(3,'Dennis Ritchie’,'C')
```

Select Records

To retrieve all the records, type the following query.

Query:

```
select * from author
```

Result:

```
auth_id        auth_name         topic


1             Doug Cutting       Hadoop
2             James Gosling       java
3             Dennis Ritchie       C
```

As of now, you have queried data using MySQL server. Let’s connect Mysql storage plugin to Presto server.

Connect Presto CLI

Type the following command to connect MySql plugin on Presto CLI.

```
 ./presto --server localhost:8080 --catalog mysql --schema tutorials
```

You will receive the following response.

```
presto:tutorials>
```

Here “**tutorials**” refers to schema in mysql server.

List Schemas

To list out all the schemas in mysql, type the following query in Presto server.

Query:

```
presto:tutorials> show schemas from mysql;
```

Result:

```
Schema
--------------------
 information_schema
 performance_schema
 sys
 tutorials
```

From this result, we can conclude the first three schemas as predefined and the last one as created by yourself.

List Tables from Schema

Following query lists out all the tables in tutorials schema.

Query:

```
presto:tutorials> show tables from mysql.tutorials;
```

Result:

```
 Table
--------
 author
```

We have created only one table in this schema. If you have created multiple tables, it will list out all the tables.

Describe Table

To describe the table fields, type the following query.

Query:

```
presto:tutorials> describe mysql.tutorials.author;
```

Result:

```
  Column   |      Type      | Comment
-----------+----------------+---------
 auth_id   | integer        |
 auth_name | varchar(50)    |
 topic     | varchar(100)   |
```

Show Columns from Table

Query:

```
presto:tutorials> show columns from mysql.tutorials.author;
```

Result:

```
  Column   |      Type      | Comment
-----------+----------------+---------
 auth_id   | integer        |
 auth_name | varchar(50)    |
 topic     | varchar(100)   |
```

Access Table Records

To fetch all the records from mysql table, issue the following query.

Query:

```
presto:tutorials> select * from mysql.tutorials.author;
```

Result:

```
 auth_id |    auth_name    | topic
---------+-----------------+--------
       1 | Doug Cutting    | Hadoop
       2 | James Gosling   | java
       3 | Dennis Ritchie  | C
```

From this result, you can retrieve mysql server records in Presto.

Create Table Using as Command

Mysql connector doesn’t support create table query but you can create a table using **as** command.

Query:

```
presto:tutorials> create table mysql.tutorials.sample as select * from
mysql.tutorials.author;
```

Result:

```
CREATE TABLE: 3 rows
```

You can’t insert rows directly because this connector has some limitations. It cannot support the following queries:

- create
- insert
- update
- delete
- drop

To view the records in the newly created table, type the following query.

Query:

```
presto:tutorials> select * from mysql.tutorials.sample;
```

Result:

```
 auth_id |   auth_name    | topic
---------+----------------+--------
       1 | Doug Cutting   | Hadoop
       2 | James Gosling  | java
       3 | Dennis Ritchie | C
```

9. Presto – JMX Connector

Java Management Extensions (JMX) gives information about the Java Virtual Machine and software running inside JVM. The JMX connector is used to query JMX information in Presto server.

As we have already enabled "**jmx.properties**" file under "**etc/catalog**" directory. Now connect Prest CLI to enable JMX plugin.

Presto CLI

Query:

```
 $ ./presto --server localhost:8080 --catalog jmx --schema jmx
```

Result:

You will receive the following response.

```
presto:jmx>
```

JMX Schema

To list out all the schemas in "jmx", type the following query.

Query:

```
presto:jmx> show schemas from jmx;
```

Result:

```
 Schema
--------------------
 information_schema
 current
```

Show Tables

To view the tables in the "current" schema, use the following command.

Query 1:

```
presto:jmx> show tables from jmx.current;
```

Result:

```
                                   Table
------------------------------------------------------------------------------
 com.facebook.presto.execution.scheduler:name=nodescheduler
 com.facebook.presto.execution:name=queryexecution
 com.facebook.presto.execution:name=querymanager
 com.facebook.presto.execution:name=remotetaskfactory
 com.facebook.presto.execution:name=taskexecutor
 com.facebook.presto.execution:name=taskmanager
 com.facebook.presto.execution:type=queryqueue,name=global,expansion=global
………………
……………….
```

Query 2:

```
presto:jmx> select * from jmx.current."java.lang:type=compilation";
```

Result:

```
node                 | compilationtimemonitoringsupported |      name    |
objectname           | totalcompilationti
--------------------------------------+--------------------------------------+---
------------------------------+-----------------------------+-------------------
ffffffff-ffff-ffff-ffff-ffffffffffff | true | HotSpot 64-Bit Tiered Compilers |
java.lang:type=Compilation |       1276
```

Query 3:

```
presto:jmx> select * from
jmx.current."com.facebook.presto.server:name=taskresource";
```

Result:

```
                 node                 | readfromoutputbuffertime.alltime.count
| readfromoutputbuffertime.alltime.max | readfromoutputbuffertime.alltime.maxer

--------------------------------------+---------------------------------------
+--------------------------------------+---------------------------------------

 ffffffff-ffff-ffff-ffff-ffffffffffff |                                   92.0
|                          1.009106149 |
```

10. Presto – Hive Connector

The Hive connector allows querying data stored in a Hive data warehouse.

Prerequisites

- Hadoop
- Hive

Hopefully you have installed Hadoop and Hive on your machine. Start all the services one by one in the new terminal. Then, start hive metastore using the following command,

```
hive --service metastore
```

Presto uses Hive metastore service to get the hive table's details.

Configuration Settings

Create a file "**hive.properties**" under "**etc/catalog**" directory. Use the following command.

```
$ cd etc
$ cd catalog
$ vi hive.properties

connector.name=hive-cdh4
hive.metastore.uri=thrift://localhost:9083
```

After making all the changes, save the file and quit the terminal.

Create Database

Create a database in Hive using the following query:

Query:

```
hive> CREATE SCHEMA tutorials;
```

After the database is created, you can verify it using the "**show databases**" command.

Create Table

Create Table is a statement used to create a table in Hive. For example, use the following query.

```
hive> create table author(auth_id int, auth_name varchar(50), topic
varchar(100) STORED AS SEQUENCEFILE;
```

Insert Table

Following query is used to insert records in hive’s table.

```
hive> insert into table author values (1,’ Doug Cutting’,Hadoop),(2,’ James
Gosling’,java),(3,’ Dennis Ritchie’,C);
```

Start Presto CLI

You can start Presto CLI to connect Hive storage plugin using the following command.

```
$ ./presto --server localhost:8080 --catalog hive –schema tutorials;
```

You will receive the following response.

```
presto:tutorials >
```

List Schemas

To list out all the schemas in Hive connector, type the following command.

Query:

```
presto:tutorials > show schemas from hive;
```

Result:

```
default


tutorials
```

List Tables

To list out all the tables in “tutorials” schema, use the following query.

Query:

```
presto:tutorials > show tables from hive.tutorials;
```

Result:

```
author
```

Fetch Table

Following query is used to fetch all the records from hive’s table.

Query:

```
presto:tutorials > select * from hive.tutorials.author;
```

Result:

```

 auth_id |   auth_name     | topic
---------+----------------+--------
       1 | Doug Cutting   | Hadoop
       2 | James Gosling  | java
       3 | Dennis Ritchie | C
```

11. Presto – Kafka Connector

The Kafka Connector for Presto allows to access data from Apache Kafka using Presto.

Prerequisites

Download and install the latest version of the following Apache projects.

- Apache ZooKeeper
- Apache Kafka

Start ZooKeeper

Start ZooKeeper server using the following command.

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Now, ZooKeeper starts port on 2181.

Start Kafka

Start Kafka in another terminal using the following command.

```
$ bin/kafka-server-start.sh config/server.properties
```

After kafka starts, it uses the port number 9092.

TPCH Data

Download tpch-kafka

```
$  curl -o kafka-tpch
https://repo1.maven.org/maven2/de/softwareforge/kafka_tpch_0811/1.0/kafka_tpch_
0811-1.0.sh
```

Now you have downloaded the loader from Maven central using the above command. You will get a similar response as the following.

```
% Total     % Received % Xferd  Average Speed   Time    Time     Time  Current

                                  Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0


  5 21.6M    5 1279k    0     0  83898      0  0:04:30  0:00:15  0:04:15  129k
  6 21.6M    6 1407k    0     0  86656      0  0:04:21  0:00:16  0:04:05  131k


 24 21.6M   24 5439k    0     0   124k      0  0:02:57  0:00:43  0:02:14  175k
 24 21.6M   24 5439k    0     0   124k      0  0:02:58  0:00:43  0:02:15  160k
 25 21.6M   25 5736k    0     0   128k      0  0:02:52  0:00:44  0:02:08  181k
 ………………………..
```

Then, make it executable using the following command,

```
$ chmod 755 kafka-tpch
```

Run tpch-kafka

Run the kafka-tpch program to preload a number of topics with tpch data using the following command.

Query:

```
$ ./kafka-tpch load --brokers localhost:9092 --prefix tpch. --tpch-type tiny
```

Result:

```
2016-07-13T16:15:52.083+0530    INFO  main   io.airlift.log.Logging    Logging
to stderr
2016-07-13T16:15:52.124+0530    INFO  main   de.softwareforge.kafka.LoadCommand
     Processing tables: [customer, orders, lineitem, part, partsupp, supplier,
nation, region]
2016-07-13T16:15:52.834+0530    INFO  pool-1-thread-1
     de.softwareforge.kafka.LoadCommand     Loading table 'customer' into
topic 'tpch.customer'...
2016-07-13T16:15:52.834+0530    INFO  pool-1-thread-2
     de.softwareforge.kafka.LoadCommand     Loading table 'orders' into topic
'tpch.orders'...
2016-07-13T16:15:52.834+0530    INFO  pool-1-thread-3
     de.softwareforge.kafka.LoadCommand     Loading table 'lineitem' into
topic 'tpch.lineitem'...
2016-07-13T16:15:52.834+0530    INFO  pool-1-thread-4
     de.softwareforge.kafka.LoadCommand     Loading table 'part' into topic
'tpch.part'...
………………………
……………………….
```

Now, Kafka tables customers,orders,supplier, etc., are loaded using tpch.

Add Config Settings

Let's add the following Kafka connector configuration settings on Presto server.

```
connector.name=kafka


kafka.nodes=localhost:9092


kafka.table-names =
tpch.customer,tpch.orders,tpch.lineitem,tpch.part,tpch.partsupp,

tpch.supplier,tpch.nation,tpch.region


kafka.hide-internal-columns=false
```

In the above configuration, Kafka tables are loaded using Kafka-tpch program.

Start Presto CLI

Start Presto CLI using the following command,

```
$ ./presto --server localhost:8080 --catalog kafka —schema tpch;
```

Here "**tpch**" is a schema for Kafka connector and you will receive a response as the following.

```
presto:tpch>
```

List Tables

Following query lists out all the tables in "**tpch**" schema.

Query:

```
presto:tpch> show tables;
```

Result:

```
Table
----------
 customer
 lineitem
 nation
 orders
```

```
part
partsupp
region
supplier
```

Describe Customer Table

Following query describes "**customer**" table.

Query:

```
presto:tpch> describe customer;
```

Result:

```
      Column       |  Type   |                   Comment
-------------------+---------+---------------------------------------------
 _partition_id     | bigint  | Partition Id
 _partition_offset | bigint  | Offset for the message within the partition
 _segment_start    | bigint  | Segment start offset
 _segment_end      | bigint  | Segment end offset
 _segment_count    | bigint  | Running message count per segment
 _key              | varchar | Key text
 _key_corrupt      | boolean | Key data is corrupt
 _key_length       | bigint  | Total number of key bytes
 _message          | varchar | Message text
 _message_corrupt  | boolean | Message data is corrupt
 _message_length   | bigint  | Total number of message bytes
```

12. Presto – JDBC Interface

Presto’s JDBC interface is used to access Java application.

Prerequisites

Install presto-jdbc-0.150.jar

You can download the JDBC jar file by visiting the following link,

https://repo1.maven.org/maven2/com/facebook/presto/presto-jdbc/0.150/

After the jar file has been downloaded, add it to the class path of your Java application.

Create a Simple Application

Let’s create a simple java application using JDBC interface.

Coding: PrestoJdbcSample.java

```
import java.sql.*;
import com.facebook.presto.jdbc.PrestoDriver;
//import presto jdbc driver packages here.

public class PrestoJdbcSample {

  public static void main(String[] args) {

    Connection connection = null;
    Statement statement = null;

    try {
      Class.forName("com.facebook.presto.jdbc.PrestoDriver");

      connection = 
DriverManager.getConnection("jdbc:presto://localhost:8080/mysql/tutorials", 
"tutorials", “");
     //connect mysql server tutorials database here
```

```
      statement = connection.createStatement();
      String sql;

      sql = "select auth_id, auth_name from mysql.tutorials.author”;
      //select mysql table author table two columns

      ResultSet resultSet = statement.executeQuery(sql);

      while(resultSet.next()){

        int id  = resultSet.getInt("auth_id");
        String name = resultSet.getString(“auth_name");

        System.out.print("ID: " + id + ";\nName: " + name + "\n");
      }

      resultSet.close();
      statement.close();
      connection.close();
    }catch(SQLException sqlException){
      sqlException.printStackTrace();
    }catch(Exception exception){
      exception.printStackTrace();
    }
  }
}
```

Save the file and quit the application. Now, start Presto server in one terminal and open a new terminal to compile and execute the result. Following are the steps:

Compilation

```
~/Workspace/presto/presto-jdbc $ javac -cp presto-jdbc-0.149.jar
PrestoJdbcSample.java
```

Execution

```
~/Workspace/presto/presto-jdbc $ java -cp .:presto-jdbc-0.149.jar
PrestoJdbcSample
```

Output

```
INFO: Logging initialized @146ms


ID: 1;

Name: Doug Cutting

ID: 2;

Name: James Gosling

ID: 3;

Name: Dennis Ritchie
```

13. Presto – Custom Function Application

Create a Maven project to develop Presto custom function.

SimpleFunctionsFactory.java

Create SimpleFunctionsFactory class to implement FunctionFactory interface.

```
package com.tutorialspoint.simple.functions;

import com.facebook.presto.metadata.FunctionFactory;
import com.facebook.presto.metadata.FunctionListBuilder;
import com.facebook.presto.metadata.SqlFunction;
import com.facebook.presto.spi.type.TypeManager;

import java.util.List;

public class SimpleFunctionFactory
        implements FunctionFactory
{
    private final TypeManager typeManager;

    public SimpleFunctionFactory(TypeManager typeManager)
    {
        this.typeManager = typeManager;
    }

    @Override
    public List<SqlFunction> listFunctions()
    {
        return new FunctionListBuilder(typeManager)
                .scalar(SimpleFunctions.class)
                .getFunctions();
    }
}
```

SimpleFunctionsPlugin.java

Create a SimpleFunctionsPlugin class to implement Plugin interface.

```
package com.tutorialspoint.simple.functions;


import com.facebook.presto.metadata.FunctionFactory;
import com.facebook.presto.spi.Plugin;
import com.facebook.presto.spi.type.TypeManager;
import com.google.common.collect.ImmutableList;


import javax.inject.Inject;
import java.util.List;
import static java.util.Objects.requireNonNull;


public class SimpleFunctionsPlugin
        implements Plugin
{


    private TypeManager typeManager;
    @Inject
    public void setTypeManager(TypeManager typeManager)
    {
        this.typeManager = requireNonNull(typeManager, "typeManager is null”);
        //Inject TypeManager class here
    }


    @Override
    public <T> List<T> getServices(Class<T> type)
    {
        if (type == FunctionFactory.class) {
            return ImmutableList.of(type.cast(new
SimpleFunctionFactory(typeManager)));
        }
        return ImmutableList.of();
    }
}
```

Add Resource File

Create a resource file which is specified in the implementation package.

```
(com.tutorialspoint.simple.functions.SimpleFunctionsPlugin)
```

Now move to the resource file location @ /path/to/resource/

Then add the changes,

```
com.facebook.presto.spi.Plugin
```

pom.xml

Add the following dependencies to pom.xml file.

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                         http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.tutorialspoint.simple.functions</groupId>
    <artifactId>presto-simple-functions</artifactId>

    <packaging>jar</packaging>

    <version>1.0</version>
    <name>presto-simple-functions</name>
    <description>Simple test functions for Presto</description>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

    <dependencies>
        <dependency>
         <groupId>com.facebook.presto</groupId>
         <artifactId>presto-spi</artifactId>
```

```
          <version>0.149</version>
        </dependency>

        <dependency>
          <groupId>com.facebook.presto</groupId>
          <artifactId>presto-main</artifactId>
          <version>0.149</version>
        </dependency>

        <dependency>
          <groupId>javax.inject</groupId>
          <artifactId>javax.inject</artifactId>
          <version>1</version>
        </dependency>

        <dependency>
             <groupId>com.google.guava</groupId>
             <artifactId>guava</artifactId>
          <version>19.0</version>
        </dependency>
    </dependencies>

    <build>
     <finalName>presto-simple-functions</finalName>

     <plugins>

     <!-- Make this jar executable -->
     <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.3.2</version>
     </plugin>
     </plugins>
    </build>
</project>
```

SimpleFunctions.java

Create SimpleFunctions class using Presto attributes.

```
package com.tutorialspoint.simple.functions;


import com.facebook.presto.operator.Description;

import com.facebook.presto.operator.scalar.ScalarFunction;

import com.facebook.presto.operator.scalar.StringFunctions;

import com.facebook.presto.spi.type.StandardTypes;

import com.facebook.presto.type.LiteralParameters;

import com.facebook.presto.type.SqlType;


public final class SimpleFunctions

{

    private SimpleFunctions()

    {

    }


    @Description("Returns summation of two numbers")

    @ScalarFunction(“mysum")

    //function name

    @SqlType(StandardTypes.BIGINT)

    public static long sum(@SqlType(StandardTypes.BIGINT) long num1,
@SqlType(StandardTypes.BIGINT) long num2)

    {

        return num1 + num2;

    }

}
```

After the application is created compile and execute the application. It will produce the JAR file. Copy the file and move the JAR file into the target Presto server plugin directory.

Compilation

```
mvn compile
```

Execution

```
 mvn package
```

Now restart Presto server and connect Presto client. Then execute the custom function application as explained below,

```
$ ./presto --catalog mysql --schema default
```

Query:

```
presto:default> select mysum(10,10);
```

Result:

```
 _col0
-------
    20
```