Apache Tajo

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Apache Tajo is an open-source distributed data warehouse framework for Hadoop. Tajo was initially started by Gruter, a Hadoop-based infrastructure company in south Korea. Later, experts from Intel, Etsy, NASA, Cloudera, Hortonworks also contributed to the project.

Tajo refers to an ostrich in Korean language. In the year March 2014, Tajo was granted a top-level open source Apache project. This tutorial will explore the basics of Tajo and moving on, it will explain cluster setup, Tajo shell, SQL queries, integration with other big data technologies and finally conclude with some examples.

Audience

Before proceeding with this tutorial, you must have a sound knowledge on core Java, any of the Linux OS, and DBMS.

Prerequisites

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 Tajo.

Disclaimer & Copyright

Table of Contents

1. Apache Tajo — Introduction

Distributed Data Warehouse System

Data warehouse is a relational database that is designed for query and analysis rather than for transaction processing. It is a subject-oriented, integrated, time-variant, and non-volatile collection of data. This data helps analysts to take informed decisions in an organization but relational data volumes are increased day by day.

To overcome the challenges, distributed data warehouse system shares data across multiple data repositories for the purpose of Online Analytical Processing(OLAP). Each data warehouse may belong to one or more organizations. It performs load balancing and scalability. Metadata is replicated and centrally distributed.

Apache Tajo is a distributed data warehouse system which uses Hadoop Distributed File System (HDFS) as the storage layer and has its own query execution engine instead of MapReduce framework.

Overview of SQL on Hadoop

Hadoop is an open-source framework that allows to store and process big data in a distributed environment. It is extremely fast and powerful. However, Hadoop has limited querying capabilities so its performance can be made even better with the help of SQL on Hadoop. This allows users to interact with Hadoop through easy SQL commands.

Some of the examples of SQL on Hadoop applications are Hive, Impala, Drill, Presto, Spark, HAWQ and Apache Tajo.

What is Apache Tajo

Apache Tajo is a relational and distributed data processing framework. It is designed for low latency and scalable ad-hoc query analysis.

- Tajo supports standard SQL and various data formats. Most of the Tajo queries can be executed without any modification.

- Tajo has **fault-tolerance** through a restart mechanism for failed tasks and extensible query rewrite engine.

- Tajo performs the necessary **ETL (Extract Transform and Load process)** operations to summarize large datasets stored on HDFS. It is an alternative choice to Hive/Pig.

The latest version of Tajo has greater connectivity to Java programs and third-party databases such as Oracle and PostGreSQL.

Features of Apache Tajo

Apache Tajo has the following features:

- Superior scalability and optimized performance
- Low latency
- User-defined functions
- Row/columnar storage processing framework.
- Compatibility with HiveQL and Hive MetaStore
- Simple data flow and easy maintenance.

Benefits of Apache Tajo

Apache Tajo offers the following benefits:

- Easy to use
- Simplified architecture
- Cost-based query optimization
- Vectorized query execution plan
- Fast delivery
- Simple I/O mechanism and supports various type of storage.
- Fault tolerance

Use Cases of Apache Tajo

The following are some of the use cases of Apache Tajo:

Data warehousing and analysis

Korea's SK Telecom firm ran Tajo against 1.7 terabytes worth of data and found it could complete queries with greater speed than either Hive or Impala.

Data discovery

The Korean music streaming service Melon uses Tajo for analytical processing. Tajo executes ETL (extract-transform-load process) jobs 1.5 to 10 times faster than Hive.

Log analysis

Bluehole Studio, a Korean based company developed TERA — a fantasy multiplayer online game. The company uses Tajo for game log analysis and finding principal causes of service quality interrupts.

Storage and Data Formats

Apache Tajo supports the following data formats:

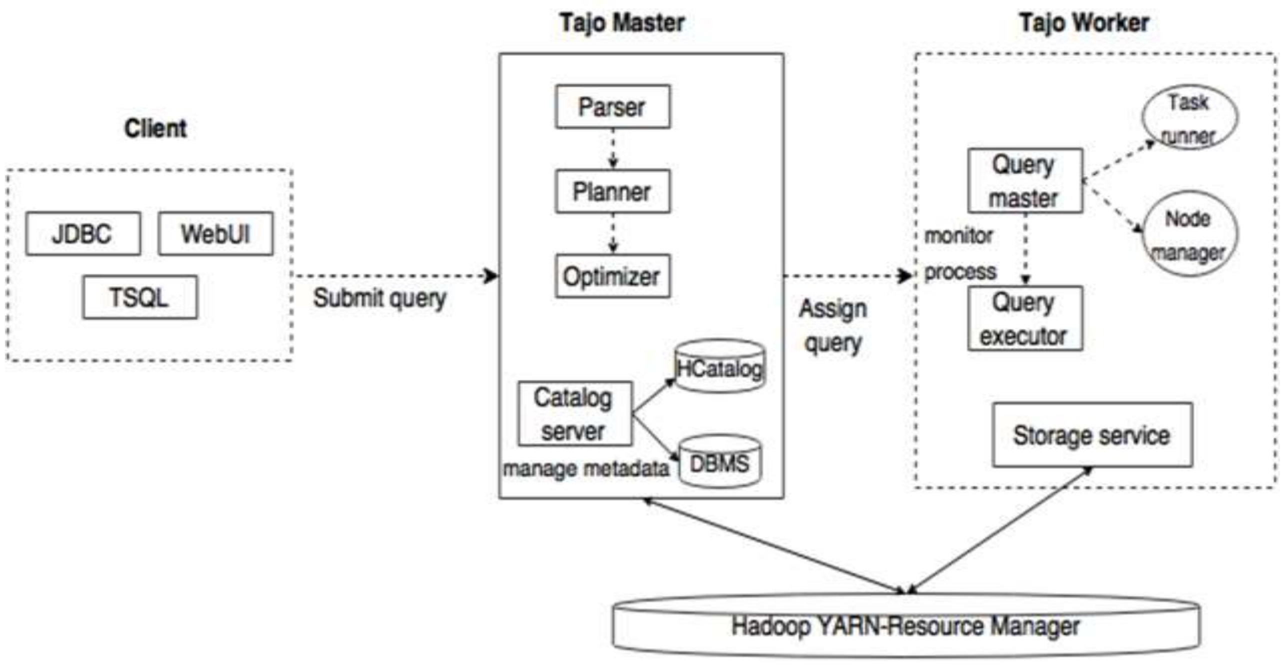
- JSON
- Text file(CSV)
- Parquet
- Sequence File
- AVRO
- Protocol Buffer
- Apache Orc

Tajo supports the following storage formats:

- HDFS
- JDBC
- Amazon S3
- Apache HBase
- Elasticsearch

2. Apache Tajo — Architecture

The following illustration depicts the architecture of Apache Tajo.



The following table describes each of the components in detail.

Component	Description
Client	**Client** submits the SQL statements to the Tajo Master to get the result.
Master	Master is the main daemon. It is responsible for query planning and is the coordinator for workers.
Catalog server	Maintains the table and index descriptions. It is embedded in the Master daemon. The catalog server uses Apache Derby as the storage layer and connects via JDBC client.
Worker	Master node assigns task to worker nodes. TajoWorker processes data. As the number of TajoWorkers increases, the processing capacity also increases linearly.

Query Master	Tajo master assigns query to the Query Master. The Query Master is responsible for controlling a distributed execution plan. It launches the TaskRunner and schedules tasks to TaskRunner. The main role of the Query Master is to monitor the running tasks and report them to the Master node.
Node Managers	Manages the resource of the worker node. It decides on allocating requests to the node.
TaskRunner	Acts as a local query execution engine. It is used to run and monitor query process. The TaskRunner processes one task at a time. It has the following three main attributes: • Logical plan - An execution block which created the task. • A fragment - an input path, an offset range, and schema. • Fetches URIs
Query Executor	It is used to execute a query.
Storage service	Connects the underlying data storage to Tajo.

Workflow

Tajo uses Hadoop Distributed File System (HDFS) as the storage layer and has its own query execution engine instead of the MapReduce framework. A Tajo cluster consists of one master node and a number of workers across cluster nodes.

The master is mainly responsible for query planning and the coordinator for workers. The master divides a query into small tasks and assigns to workers. Each worker has a local query engine that executes a directed acyclic graph of physical operators.

In addition, Tajo can control distributed data flow more flexible than that of MapReduce and supports indexing techniques.

The web-based interface of Tajo has the following capabilities:

- Option to find how the submitted queries are planned
- Option to find how the queries are distributed across nodes
- Option to check the status of the cluster and nodes

3. Apache Tajo — Installation

To install Apache Tajo, you must have the following software on your system:

- Hadoop version 2.3 or greater
- Java version 1.7 or higher
- Linux or Mac OS

Let us now continue with the following steps to install Tajo.

Verifying Java installation

Hopefully, you have already installed Java version 8 on your machine. Now, you just need to proceed by verifying it.

To verify, use the following command:

```
$ java -version
```

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

Download JDK

Download the latest version of JDK by visiting the following link and then, download the latest version.

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. Following this, extract the files and move them to a specific directory. Now, set the Java alternatives. Finally, Java is installed on your machine.

Verifying Hadoop Installation

You have already installed **Hadoop** on your system. Now, verify it using the following command:

```
$ hadoop version
```

If everything is fine with your setup, then you could see the version of Hadoop. If Hadoop is not installed, download and install Hadoop by visiting the following link:

http://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.6.2/hadoop-2.6.2.tar.gz

Apache Tajo Installation

Apache Tajo provides two execution modes — local mode and fully distributed mode. After verifying Java and Hadoop installation proceed with the following steps to install Tajo cluster on your machine. A local mode Tajo instance requires very easy configurations.

Download the latest version of Tajo by visiting the following link:

http://www.apache.org/dyn/closer.cgi/tajo

Now you can download the file **"tajo-0.11.3.tar.gz"** from your machine.

Extract Tar File

Extract the tar file by using the following command:

```
$ cd opt/
$ tar tajo-0.11.3.tar.gz
$ cd tajo-0.11.3
```

Set Environment Variable

Add the following changes to **"conf/tajo-env.sh"** file

```
$ cd tajo-0.11.3
$ vi conf/tajo-env.sh

# Hadoop home. Required
export HADOOP_HOME=/Users/path/to/Hadoop/hadoop-2.6.2

# The java implementation to use.  Required.
export JAVA_HOME=/path/to/jdk1.8.0_92.jdk/
```

Here, you must specify Hadoop and Java path to **"tajo-env.sh"** file. After the changes are made, save the file and quit the terminal.

Start Tajo Server

To launch the Tajo server, execute the following command:

```
$ bin/start-tajo.sh
```

You will receive a response similar to the following:

```
Starting single TajoMaster
starting master, logging to /Users/path/to/Tajo/tajo-0.11.3/bin/../
localhost: starting worker, logging to /Users/path/toe/Tajo/tajo-0.11.3/bin/../logs/


Tajo master web UI: http://local:26080
Tajo Client Service: local:26002
```

Now, type the command “jps” to see the running daemons.

```
$ jps

1010 TajoWorker
1140 Jps
933 TajoMaster
```

Launch Tajo Shell (Tsql)

To launch the Tajo shell client, use the following command:

```
$ bin/tsql
```

You will receive the following output:

```
welcome to

   _____ ___  _____ ___
  /_  _/ _  |/_  _/   /
   / // /_| |_/ // / /
  /_//_/ /_/___/ \__/  0.11.3

Try \? for help.
```

Quit Tajo Shell

Execute the following command to quit Tsql:

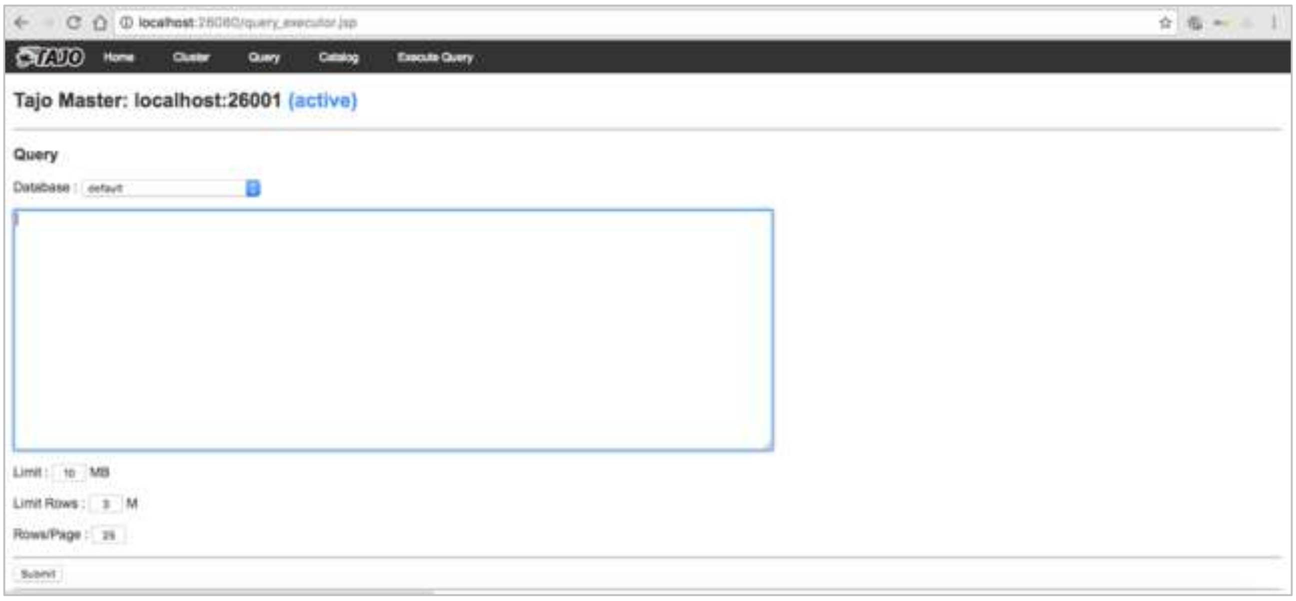
```
default> \q
bye!
```

Here, the default refers to the catalog in Tajo.

Web UI

Type the following URL to launch the Tajo web UI: http://localhost:26080

You will now see the following screen which is similar to the ExecuteQuery option.



Tajo Master: localhost:26001 (active)

Query

Database : default

Limit : 10 MB

Limit Rows : 3 M

Rows/Page : 25

Stop Tajo

To stop the Tajo server, use the following command:

```
$ bin/stop-tajo.sh
```

You will get the following response:

```
localhost: stopping worker
stopping master
```

4. Apache Tajo — Configuration Settings

Tajo's configuration is based on Hadoop's configuration system. This chapter explains Tajo configuration settings in detail.

Basic Settings

Tajo uses the following two config files:

- catalog-site.xml — configuration for the catalog server.
- tajo-site.xml — configuration for other Tajo modules.

Distributed Mode Configuration

Distributed mode setup runs on Hadoop Distributed File System (HDFS). Let's follow the steps to configure Tajo distributed mode setup.

tajo-site.xml

This file is available **@ /path/to/tajo/conf** directory and acts as configuration for other Tajo modules. To access Tajo in a distributed mode, apply the following changes to **"tajo-site.xml"**.

```
<property>
  <name>tajo.rootdir</name>
  <value>hdfs://hostname:port/tajo</value>
</property>

<property>
  <name>tajo.master.umbilical-rpc.address</name>
  <value>hostname:26001</value>
</property>

<property>
  <name>tajo.master.client-rpc.address</name>
  <value>hostname:26002</value>
</property>

<property>
  <name>tajo.catalog.client-rpc.address</name>
  <value>hostname:26005</value>
```

```
</property>
```

Master Node Configuration

Tajo uses HDFS as a primary storage type. The configuration is as follows and should be added to **"tajo-site.xml".**

```
<property>
  <name>tajo.rootdir</name>
  <value>hdfs://namenode_hostname:port/path</value>
</property>
```

Catalog Configuration

If you want to customize the catalog service, copy **$path/to/Tajo/conf/catalog-site.xml.template** to **$path/to/Tajo/conf/catalog-site.xml** and add any of the following configuration as needed.

For example, if you use **"Hive catalog store"** to access Tajo, then the configuration should be like the following:

```
<property>
  <name>tajo.catalog.store.class</name>
  <value>org.apache.tajo.catalog.store.HCatalogStore</value>
</property>
```

If you need to store **MySQL** catalog, then apply the following changes:

```
<property>
  <name>tajo.catalog.store.class</name>
  <value>org.apache.tajo.catalog.store.MySQLStore</value>
</property>
<property>
  <name>tajo.catalog.jdbc.connection.id</name>
  <value><mysql user name></value>
</property>
<property>
  <name>tajo.catalog.jdbc.connection.password</name>
  <value><mysql user password></value>
</property>
  <property>
  <name>tajo.catalog.jdbc.uri</name>
  <value>jdbc:mysql://<mysql host name>:<mysql port>/<database name for
tajo>?createDatabaseIfNotExist=true</value>
</property>
```

Similarly, you can register the other Tajo supported catalogs in the configuration file.

Worker Configuration

By default, the TajoWorker stores temporary data on the local file system. It is defined in the "tajo-site.xml" file as follows:

```
<property>
  <name>tajo.worker.tmpdir.locations</name>
  <value>/disk1/tmpdir,/disk2/tmpdir,/disk3/tmpdir</value>
</property>
```

To increase the capacity of running tasks of each worker resource, choose the following configuration:

```
<property>
  <name>tajo.worker.resource.cpu-cores</name>
  <value>12</value>
</property>

 <property>
  <name>tajo.task.resource.min.memory-mb</name>
  <value>2000</value>
</property>

<property>
  <name>tajo.worker.resource.disks</name>
  <value>4</value>
</property>
```

To make the Tajo worker run in a dedicated mode, choose the following configuration:

```
<property>
  <name>tajo.worker.resource.dedicated</name>
  <value>true</value>
</property>
```

5. Apache Tajo — Shell Commands

In this chapter, we will understand the Tajo Shell commands in detail.

To execute the Tajo shell commands, you need to start the Tajo server and the Tajo shell using the following commands:

Start server

```
$ bin/start-tajo.sh
```

Start Shell

```
$ bin/tsql
```

The above commands are now ready for execution.

Meta Commands

Let us now discuss the **Meta Commands**. Tsql meta commands start with a backslash **('\').**

Help Command

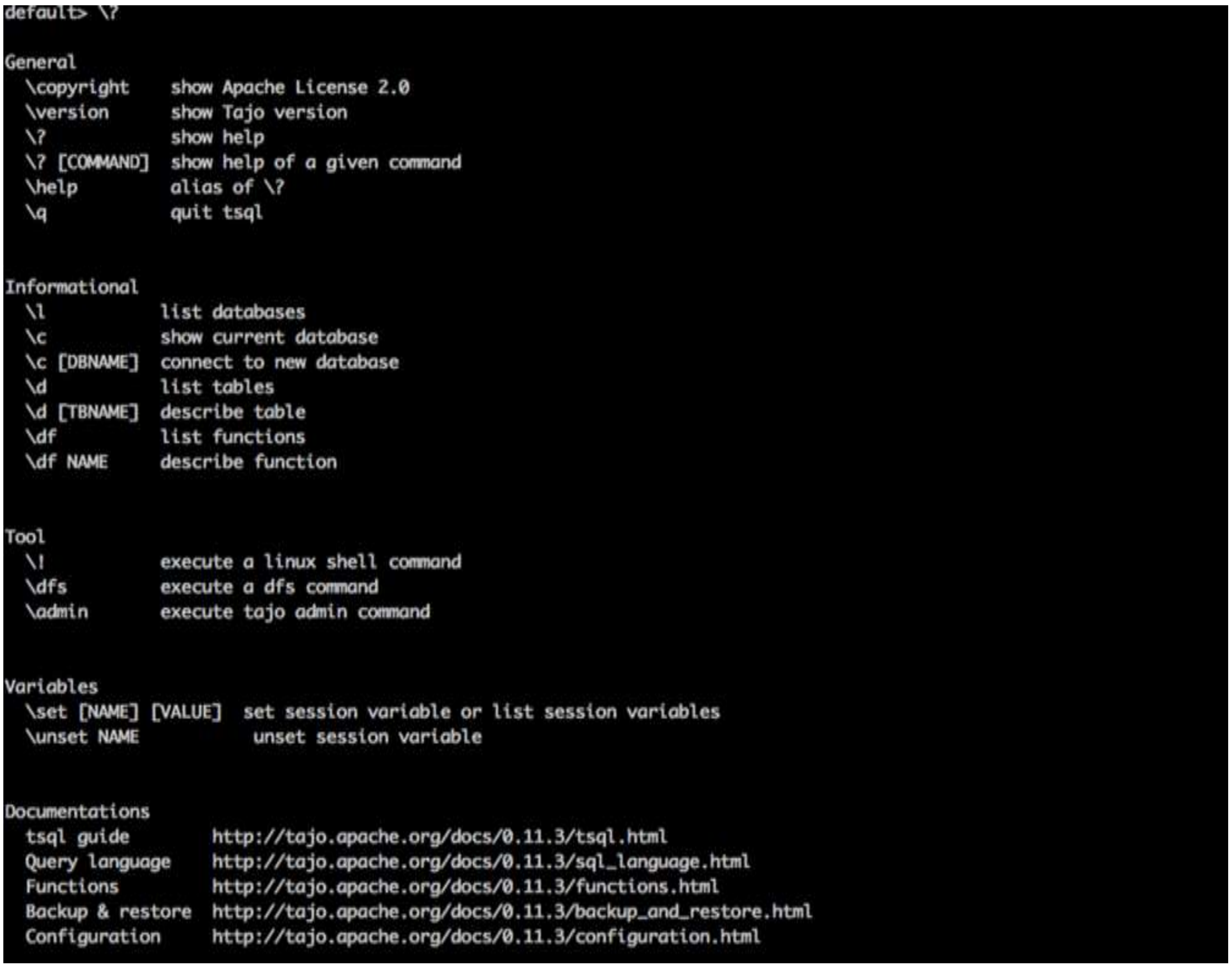
"\?" Command is used to show the help option.

Query

```
default> \?
```

Result

The above **\?** Command list out all the basic usage options in Tajo. You will receive the following output:



```
default> \?

General
  \copyright    show Apache License 2.0
  \version      show Tajo version
  \?            show help
  \? [COMMAND]  show help of a given command
  \help         alias of \?
  \q            quit tsql


Informational
  \l           list databases
  \c           show current database
  \c [DBNAME]  connect to new database
  \d           list tables
  \d [TBNAME]  describe table
  \df          list functions
  \df NAME     describe function


Tool
  \!           execute a linux shell command
  \dfs         execute a dfs command
  \admin       execute tajo admin command


Variables
  \set [NAME] [VALUE]  set session variable or list session variables
  \unset NAME           unset session variable


Documentations
  tsql guide        http://tajo.apache.org/docs/0.11.3/tsql.html
  Query language    http://tajo.apache.org/docs/0.11.3/sql_language.html
  Functions         http://tajo.apache.org/docs/0.11.3/functions.html
  Backup & restore  http://tajo.apache.org/docs/0.11.3/backup_and_restore.html
  Configuration     http://tajo.apache.org/docs/0.11.3/configuration.html
```

List Database

To list out all the databases in Tajo, use the following command:

Query

```
default> \l
```

Result

You will receive the following output:

```
information_schema

default
```

At present, we have not created any database so it shows two built in Tajo databases.

Current Database

\c option is used to display the current database name.

Query

```
default> \c
```

Result

You are now connected to database "default" as user “username”.

List out Built-in Functions

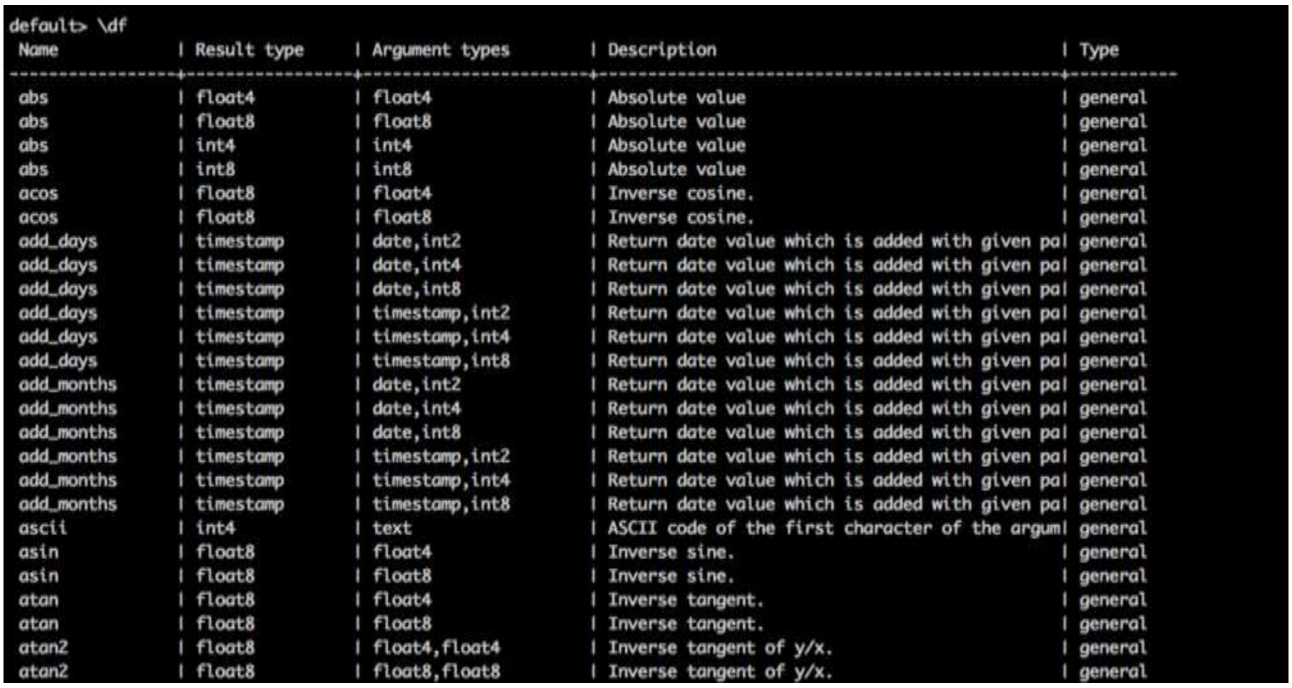
To list out all the built-in function, type the query as follows:

Query

```
default> \df
```

Result

You will receive the following output:



```
default> \df
 Name            | Result type     | Argument types        | Description                                    | Type
-----------------+-----------------+-----------------------+------------------------------------------------+----------
 abs             | float4          | float4                | Absolute value                                 | general
 abs             | float8          | float8                | Absolute value                                 | general
 abs             | int4            | int4                  | Absolute value                                 | general
 abs             | int8            | int8                  | Absolute value                                 | general
 acos            | float8          | float4                | Inverse cosine.                                | general
 acos            | float8          | float8                | Inverse cosine.                                | general
 add_days        | timestamp       | date,int2             | Return date value which is added with given pa| general
 add_days        | timestamp       | date,int4             | Return date value which is added with given pa| general
 add_days        | timestamp       | date,int8             | Return date value which is added with given pa| general
 add_days        | timestamp       | timestamp,int2        | Return date value which is added with given pa| general
 add_days        | timestamp       | timestamp,int4        | Return date value which is added with given pa| general
 add_days        | timestamp       | timestamp,int8        | Return date value which is added with given pa| general
 add_months      | timestamp       | date,int2             | Return date value which is added with given pa| general
 add_months      | timestamp       | date,int4             | Return date value which is added with given pa| general
 add_months      | timestamp       | date,int8             | Return date value which is added with given pa| general
 add_months      | timestamp       | timestamp,int2        | Return date value which is added with given pa| general
 add_months      | timestamp       | timestamp,int4        | Return date value which is added with given pa| general
 add_months      | timestamp       | timestamp,int8        | Return date value which is added with given pa| general
 ascii           | int4            | text                  | ASCII code of the first character of the argum| general
 asin            | float8          | float4                | Inverse sine.                                  | general
 asin            | float8          | float8                | Inverse sine.                                  | general
 atan            | float8          | float4                | Inverse tangent.                               | general
 atan            | float8          | float8                | Inverse tangent.                               | general
 atan2           | float8          | float4,float4         | Inverse tangent of y/x.                        | general
 atan2           | float8          | float8,float8         | Inverse tangent of y/x.                        | general
```

Describe Function

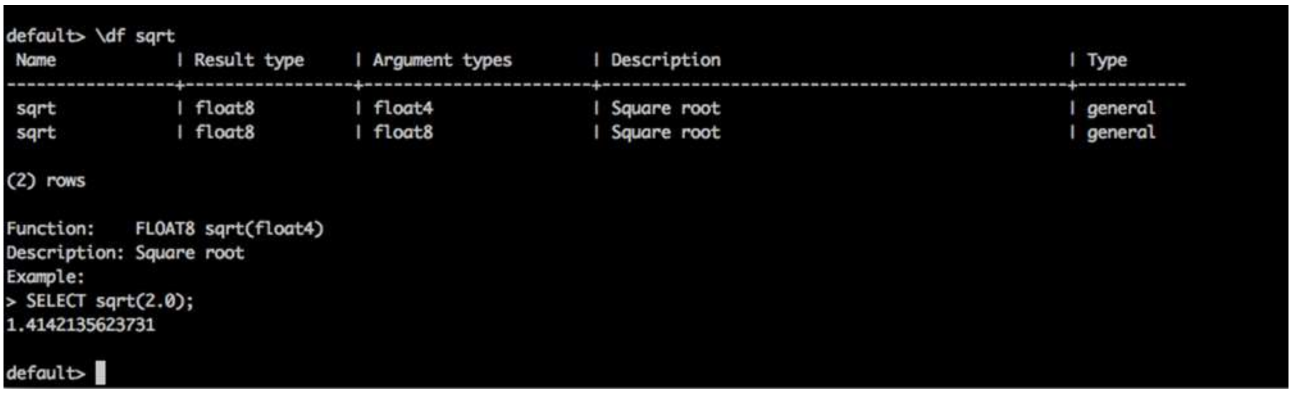
\df function name - This query returns the complete description of the given function.

Query

```
default> \df sqrt
```

Result

You will receive the following output:



```
default> \df sqrt
 Name           | Result type     | Argument types        | Description                                   | Type
----------------+-----------------+-----------------------+-----------------------------------------------+-----------
 sqrt           | float8          | float4                | Square root                                   | general
 sqrt           | float8          | float8                | Square root                                   | general

(2) rows

Function:    FLOAT8 sqrt(float4)
Description: Square root
Example:
> SELECT sqrt(2.0);
1.4142135623731

default>
```

Quit Terminal

To quit the terminal, type the following query:

Query

```
default> \q
```

Result

You will receive the following output:

```
bye!
```

Admin Commands

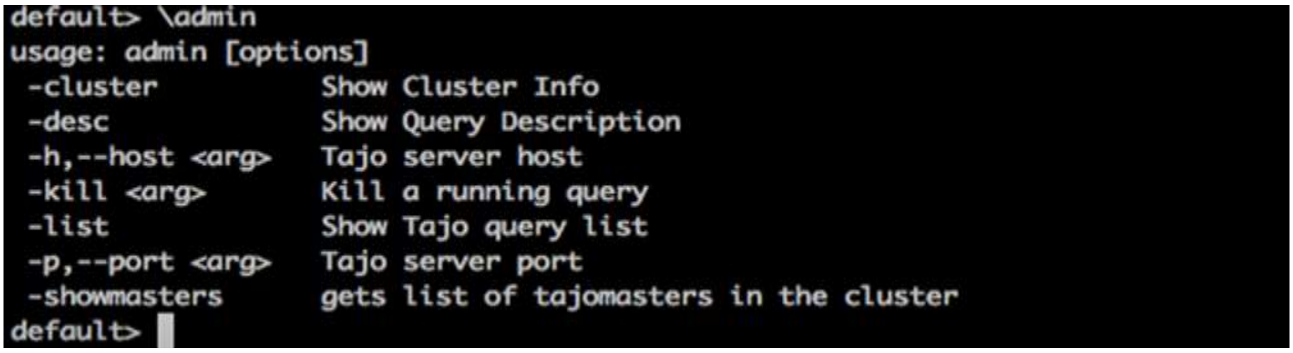
Tajo shell provides **\admin** option to list out all the admin features.

Query

```
default> \admin
```

Result

You will receive the following output:



```
default> \admin
usage: admin [options]
 -cluster          Show Cluster Info
 -desc             Show Query Description
 -h,--host <arg>   Tajo server host
 -kill <arg>       Kill a running query
 -list             Show Tajo query list
 -p,--port <arg>   Tajo server port
 -showmasters      gets list of tajomasters in the cluster
default>
```

Cluster Info

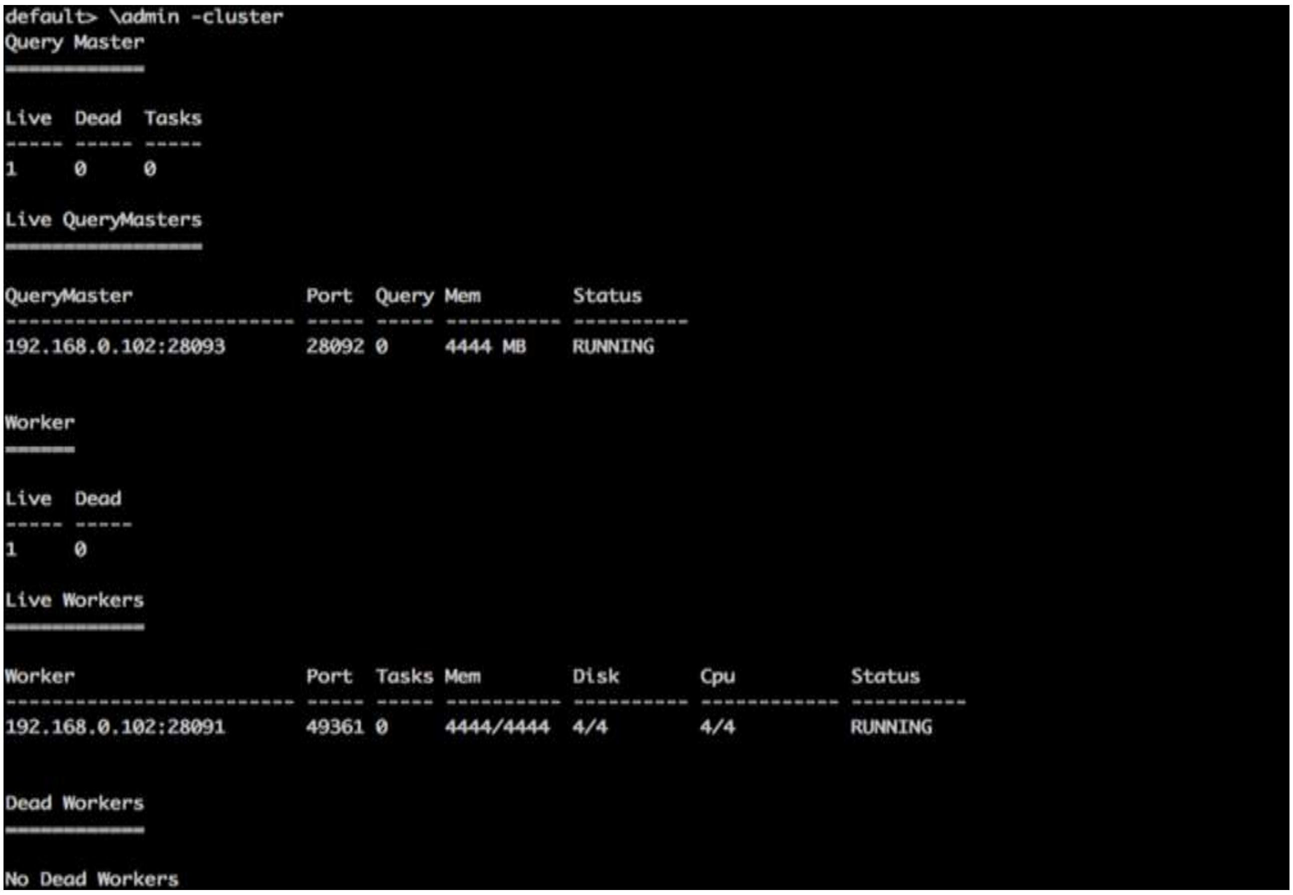
To display the cluster information in Tajo, use the following query

Query

```
default> \admin -cluster
```

Result

You will receive the following output:



```
default> \admin -cluster
Query Master
============

Live  Dead  Tasks
----- ----- -----
1     0     0

Live QueryMasters
=================

QueryMaster               Port  Query Mem        Status
------------------------- ----- ----- ---------- ----------
192.168.0.102:28093       28092 0     4444 MB    RUNNING


Worker
======

Live  Dead
----- -----
1     0

Live Workers
============

Worker                    Port  Tasks Mem        Disk       Cpu          Status
------------------------- ----- ----- ---------- ---------- ------------ ----------
192.168.0.102:28091       49361 0     4444/4444  4/4        4/4          RUNNING


Dead Workers
============

No Dead Workers
```

Show master

The following query displays the current master information.

Query

```
default> \admin -showmasters
```

Result

```
localhost
```

Similarly, you can try other admin commands.

Session Variables

The Tajo client connects to the Master via a unique session id. The session is live until the client is disconnected or expires.

The following command is used to list out all session variables.

Query

```
default> \set
```

Result

```
'SESSION_LAST_ACCESS_TIME'='1470206387146'
'CURRENT_DATABASE'='default'
‘USERNAME’='user'
'SESSION_ID'='c60c9b20-dfba-404a-822f-182bc95d6c7c'
'TIMEZONE'='Asia/Kolkata'
'FETCH_ROWNUM'='200'
‘COMPRESSED_RESULT_TRANSFER'='false'
```

The **\set key val** will set the session variable named **key** with the value **val**. For example,

Query

```
default> \set ‘current_database’='default'
```

Result

```
usage: \set [[NAME] VALUE]
```

Here, you can assign the key and value in the **\set** command. If you need to revert the changes then use the **\unset** command.

6. Apache Tajo — Data Types

To execute a query in a Tajo shell, open your terminal and move to the Tajo installed directory and then type the following command:

```
$ bin/tsql
```

You will now see the response as shown in the following program:

```
default>
```

You can now execute your queries. Otherwise you can run your queries through web console application to the following URL:

http://localhost:26080

Primitive Data Types

Apache Tajo supports the following list of primitive data types:

Data type	Description
integer	Used for storing integer value with 4 bytes storage.
tinyint	Tiny integer value is 1 byte
smallint	Used for storing small size integer 2 bytes value.
bigint	Big range integer value has 8 bytes storage.
boolean	Returns true/false
real	Used for storing real value. Size is 4 bytes.
float	Floating point precision value which has 4 or 8 bytes storage space.
double	Double point precision value stored in 8 bytes.

7.

char[(n)]	Character value.
varchar[(n)]	Variable-length non-Unicode data.
number	Decimal values.
binary	Binary values.
date	Calendar date (year, month, day). **Example**: DATE '2016-08-22'
time	Time of day (hour, minute, second, millisecond) without a time zone. Values of this type are parsed and rendered in the session time zone.
timezone	Time of day (hour, minute, second, millisecond) with a time zone. Values of this type are rendered using the time zone from the value. **Example**: TIME '01:02:03.456 Asia/kolkata'
timestamp	Instant in time that includes the date and time of day without a time zone. **Example**: TIMESTAMP '2016-08-22 03:04:05.321'
text	Variable-length Unicode text.

7. Apache Tajo – Operators

The following operators are used in Tajo to perform desired operations.

Operators	Description
Arithmetic operator	Presto supports arithmetic operators such as +, -, *, /, %.
Relational operator	<,>,<=,>=,=,<>
Logical operator	AND, OR, NOT
String operator	The ‘||’ operator performs string concatenation.
Range operator	Range operator is used to test the value in a specific range. Tajo supports BETWEEN, IS NULL, IS NOT NULL operators.

Let us perform a few operations using the above operators.

Arithmetic Operators

The following sample queries are examples of arithmetic operators:

Query 1

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

Result

The above query will generate the following output:

```
 addition
-------------------------------
9
```

The query performs addition of two given values.

Query 2

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

Result

The above query will generate the following output.

```
 modulus
-------------------------------
0
```

Output is returned as the modulus for given values. Similarly, you can perform other arithmetic operations.

Relational Operators

Relational operators are used to compare two values and return a Boolean result. Following queries are an examples of relational operators.

Query 1

```
default> select 50<=25 as lessthanequal;
```

Result

The above query will generate the following output:

```
lessthanequal
-------------------------------
false
```

The above condition is incorrect so the result is false.

Query 2

```
default> select 20>=10 as greater;
```

Result

```
greater
-------------------------------
true
```

Here 20 is greater than 10. So, the result is true.

Query 3

```
default> select 50=25 as equal;
```

Result

```
equal

-------------------------------

false
```

Here, 50 is not equal to 25. So, the result is false.

Query 4

```
default> select 50<>25 as notequal;
```

Result

```
notequal

-------------------------------

true
```

From the above result, both the values are not equal so the result is 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 Tajo.

Query 1

```
default> select 3<2 and 4>1 as logical_and;
```

Result

```


 logical_and

-------------------------------

false
```

The **AND** operator returns true only if both conditions are true; otherwise it returns false. Here, **4>1** condition is false. So, the "AND" operator returns false.

Query 2

```
default> select 3<2 or 4>1 as logical_or;
```

Result

The above query will generate the following result.

```
logical_or
-------------------------------
true
```

Here, the first condition is true and the second condition is false. One condition is satisfied so the result is true.

Query 3

```
default> select 3 not in (1,2) as logical_not;
```

Result

The above query will generate the following output:

```
logical_not
-------------------------------
true
```

3 is not in the given range. Therefore, the result is true.

String Operator

String operator "||" performs concatenation.

Query 1

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

Result

The above query will generate the following output:

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

The **String "||"** operator concatenates the given two strings **tutorials** and **point**. The output is "**tutorialspoint**".

Range Operator

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

Query 1

```
default> select 100 between 10 and 150 as range_between;
```

Result

```
range_between

-------------------------------

true
```

Here, 100 lies between the given range so the result is true.

Query 2

```
default> select 10 is null as range;
```

Result

```
range

-------------------------------

false
```

The given value 10 is compared with the null value so the result is false.

Let us now check with a not null operator and then see what the result would be like.

Query 3

```
default> select 10 is not null as range;
```

Result

The above query will generate the following output:

```
range

-------------------------------

true
```

8. Apache Tajo – SQL Functions

As of now, you were aware of running simple basic queries on Tajo. In the next subsequent few chapters, we will discuss the following SQL functions:

- Math Functions
- String Functions
- DateTime Functions
- JSON Functions

9. Apache Tajo – Math Functions

Math functions operate on mathematical formulae. The 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.
ceil(x)	Returns x value rounded up to the nearest integer.
floor(x)	Returns x rounded down to the nearest integer.
pi()	Returns pi value. Result will be returned as double value.
radians(x)	converts the angle x in degree radians.
degrees(x)	Returns degree value for x.
pow(x,p)	Returns power of value ‘p’ to the x value.
div(x,y)	Returns the division result for the given two x,y integer values.
exp(x)	Returns Euler’s number **e** raised to the power of a number
sqrt(x)	Returns the square root of x.
sign(x)	Returns the signum function of x, that is: • 0 if the argument is 0 • 1 if the argument is greater than 0 • 1 if the argument is less than 0

mod(n,m)	Returns the modulus (remainder) of n divided by m.
round(x)	Returns the rounded value for x.
cos(x)	Returns the cosine value(x).
asin(x)	Returns the inverse sine value(x).
acos(x)	Returns the inverse cosine value(x).
atan(x)	Returns the inverse tangent value(x).
atan2(y,x)	Returns the inverse tangent value(y/x).

Let us now work out a few examples on Math Operations.

abs(x)

Let us now check the **abs(x)** function with the following query.

Query

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

Result

The above query will generate the following result.

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

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

cbrt(x)

Let us now check the **cbrt(x)** function with the following query.

Query

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

Result

The above query will generate the following result.

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

The cubic root of 4 is returned as output.

ceil(x)

Let us now check the **ceil(x)** function with the following query.

Query

```
default> select ceil(4.7) as ceiling;
```

Result

The above query will generate the following result:

```
ceiling
---------
     5
```

Ceil of 4.7 is 5 which is rounded to the next integer. The **Ceiling()** function also returns the same result. This function acts as an alias for **ceil()**.

floor(x)

Let us now check the **floor(x)** function with the following query.

Query

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

Result

The above query will generate the following result.

```
 floor
-------
   4
```

Here, the given value 4.8 is rounded down to the previous integer value 4.

pi()

Let us now check the **pi(x)** function with the following query.

Query

```
default>select pi();
```

Result

The above query will generate the following result.

```
?pi
-------------------------------
3.141592653589793
```

Here, the pi value is 3.1459.

radians(x)

Let us now check the **radians(x)** function with the following query.

Query

```
default> select radians(4) as radian;
```

Result

The above query will generate the following result.

```
radian
-------------------------------
0.06981317007977318
```

Here, the angle 4 value is converted to radian as 0.069813.

degrees(x)

Let us now check **degrees(x)** with the following query.

Query

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

Result

The above query will generate the following result.

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

The degree value for the integer 2 is 114.59155.

pow(x,y)

Let us now check **pow(x,y)** with the following query.

Query

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

Result

The above query will generate the following result.

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

The query returns the result for 2 power 3, and the value is 8.

div(x,y)

Let us now check the **div(x,y)** function with the following query.

Query

```
default> select div(7,4) as division;
```

Result

The above query will generate the following result.

```
division
-------------------------------
1
```

The output is the division result for the given two integer values 7/4 is 1.

exp(x)

Let us now check the **erp(x)** function with the following query.

Query

```
default> select exp(5) as euler_number;
```

Result

The above query will generate the following result.

```
euler_number
-------------------------------
148.4131591025766
```

Here, Euler’s number is raised to the power value 5. exp(5) result is 148.4131591.

sqrt(x)

Let us now check the **sqrt(x)** function with the following query.

Query

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

Result

The above query will generate the following result.

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

Square root of 3 value is 1.732.

sign(x)

Let us now check the **sign(x)** function with the following query.

Query

```
default> select sign(4) as signum;
```

Result

The above query will generate the following result.

```
signum
-------------------------------
1.0
```

Here, the signum value is 4 which is greater than 0. So, the result is 1. If the value is <0 then result will be -1.

mod(n,m)

Let us now check the **mod(n,m)** function with the following query.

Query

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

Result

The above query will generate the following result.

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

Here (2,4) modulus result is 2.

round(x)

Let us now check the **round(x)** function with the following query.

Query

```
default> select round(7.8) as round;
```

Result

The above query will generate the following result.

```
round

-------------------------------

8
```

Here, 7.8 value is rounded by 8 using the **round()** function.

cos(x)

Let us now check the **cos(x)** function with the following query.

Query

```
default> select cos(30) as cosine;
```

Result

The above query will generate the following result.

```
cosine

-------------------------------

0.15425144988758405
```

The value of Cosine angle at 30 degree is 0.154251.

asin(x)

Let us now check the **asin(x)** function with the following query.

Query

```
default> select asin(0.4) as inverse_sine;
```

Result

The above query will generate the following result.

```
inverse_sine

-------------------------------

0.41151684606748806
```

Inverse sine(asine) value for 0.4 is 0.4115168.

acos(x)

Let us now check the **acos(x)** function with the following query.

Query

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

Result

The above query will generate the following result.

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

Inverse cosine(acos) value for 0.5 is 1.047197.

atan(x)

Let us now check the **atan(x)** function with the following query.

Query

```
default> select atan(0.5) as inversetangent;
```

Result

The above query will generate the following result.

```
inversetangent
-------------------------------
0.4636476090008061
```

Inverse tangent(atan) value for 0.5 is 0.4636476.

atan2(y/x)

Let us now check the **atan2(y/x)** function with the following query.

Query

```
default> select atan2(1.7,0.5) as tangent;
```

Result

The above query will generate the following result.

```
tangent
-------------------------------
1.2847448850775784
```

The output produces inverse tangent value of (y/x). Here, the given Inverse tangent value(1.7/0.5) is 1.2847448.

Data Type Functions

The following table lists out the data type functions available in Apache Tajo.

Functions	Description
to_bin(x)	Returns the binary representation of integer.
to_char(int,text)	Converts integer to string.
to_hex(x)	Converts the x value into hexadecimal.

to_bin(x)

Let us now check the **to_bin(x)** function with the following query.

Query

```
default> select to_bin(8) as binary_conversion;
```

Result

The above query will generate the following result.

```
binary_conversion
-------------------------------
1000
```

The query returns binary value "1000" for the given input value 8 using the **to_bin()** conversion function.

to_char(int,text)

Let us now check the **to_char(int,text)** function with the following query.

Query

```
default> select to_char(2,'011') as char_conversion;
```

Result

The above query will generate the following result.

```
char_conversion

-------------------------------

211
```

The above query performs character conversion. Here, the integer 2 is converted to string '011'. The result for this conversion is 211.

to_hex(x)

Let us now check the **to_hex(x)** function with the following query.

Query

```
default> select to_hex(11) as hex_conversion;
```

Result

The above query will generate the following result.

```
hex_conversion

-------------------------------

b
```

The query returns the hexadecimal value for the integer 11 and the value is b. Similarly, you can try other values.

10. Apache Tajo – String Functions

The following table lists out the string functions in Tajo.

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.
ascii(string text)	Returns the ASCII code of the first character of the text.
bit_length(string text)	Returns the number of bits in a string.
char_length(string text)	Returns the number of characters in a string.
octet_length(string text)	Returns the number of bytes in a string.
digest(input text, method text)	Calculates the **Digest** hash of string. Here, the second arg method refers to the hash method.
initcap(string text)	Converts the first letter of each word to upper case.
md5(string text)	Calculates the **MD5** hash of string.
left(string text, int size)	Returns the first n characters in the string.
right(string text, int size)	Returns the last n characters in the string.
locate(source text, target text, start_index)	Returns the location of specified substring.

strposb(source text, target text)	Returns the binary location of specified substring.
substr(source text, start index, length)	Returns the substring for the specified length.
trim(string text[, characters text])	Removes the characters (a space by default) from the start/end/both ends of the string.
split_part(string text, delimiter text, field int)	Splits a string on delimiter and returns the given field (counting from one).
regexp_replace(string text, pattern text, replacement text)	Replaces substrings matched to a given regular expression pattern.
reverse(string)	Reverse operation performed for the string.

concat (string1, ..., stringN)

Let us now check the **concat(string1, ..., stringN)** function with the following query.

Query

```
default>select concat('tutorials','point') as string_concat;
```

Result

The above query will generate the following result.

```
string_concat

-------------------------------

tutorialspoint
```

The above query returns the concatenation of two strings.

length (string)

Let us now check the **length(string)** function with the following query.

Query

```
default> select length('tutorialspoint') as length;
```

Result

The above query will generate the following result.

```
length
-------------------------------
14
```

Here, the length of the given string "tutorialspoint" is 14.

lower (string)

Let us now check the **lower(string)** function with the following query.

Query

```
default> select lower('Apache Tajo') as lower;
```

Result

The above query will generate the following result.

```
lower
-------------------------------
apache tajo
```

The query returns the lower case format of the given string "Apache Tajo".

upper (string)

Let us now check the **upper(string)** function with the following query.

Query

```
default> select upper('apache tajo') as upper;
```

Result

The above query will generate the following result.

```
upper
-------------------------------
APACHE TAJO
```

The query returns the upper case format of the given string "apache tajo".

asci (char)

Let us now check the **ascii(char)** function with the following query.

Query

```
default> select ascii('a') as ASCII;
```

Result

The above query will generate the following result.

```
ascii

-------------------------------

97
```

The ASCII value for char ‘a’ is 97.

bit_length (string)

Let us now check the **bit_length(string)** function with the following query.

Query

```
default> select bit_length('tutorialspoint') as bitslength;
```

Result

The above query will generate the following result.

```
bitslength

-------------------------------

112
```

The number of bits in “tutorialspoint” is 112.

char_length (string)

Let us now check the **char_length(string)** function with the following query.

Query

```
default> select char_length('tutorialspoint') as charlength;
```

Result

The above query will generate the following result.

```
charlength
-------------------------------
14
```

Here, the given string “tutorialspoint” length is 14.

octet_length (string text)

Let us now check the **octet_length(string text)** function with the following query.

Query

```
default> select octet_length('tajo') as octet;
```

Result

The above query will generate the following result.

```
octet
-------------------------------
4
```

The query returns the number of bytes for the given string Tajo is 4.

digest (string, method)

Let us now check the **digest(string,method)** function with the following query.

Query

```
default> select digest('tutorialspoint','sha1') as SHA;
```

Result

The above query will generate the following result.

```
sha
-------------------------------
43aeef3e6b34a52281c6f7ce49f43a94e009dda5
```

The query returns the Digest hash value for the string “tutorialspoint”. **Sha1** is a hash method. This can be used in hashing techniques.

initcap (string)

Let us now check the **initcap(string)** function with the following query.

Query

```
default> select initcap('tutorialspoint') as firstcap;
```

Result

The above query will generate the following result.

```
firstcap
-------------------------------
Tutorialspoint
```

Here, the first char of the given string is capitalised using the **initcap**() method.

md5 (string)

Let us now check the **md5(string)** function with the following query.

Query

```
default> select md5('tutorialspoint') as hash;
```

Result

The above query will generate the following result.

```
hash
-------------------------------
6c60b3cfe5124f982eb629e00a98f01f
```

The query returns md5 hash value for the given input string.

left (string, size)

Let us now check the **left(string,size)** function with the following query.

Query

```
default> select left('tutorialspoint',2) as leftshift;
```

Result

The above query will generate the following result.

```
leftshift

-------------------------------

tu
```

The query returns left padding for the string with the given size.

right (string, size)

Let us now check the **right(string,size)** function with the following query.

Query

```
default> select right('tutorialspoint',2) as rightshift;
```

Result

The above query will generate the following result.

```
rightshift

-------------------------------

nt
```

The output returns right padding for the string "tutorialspoint" with size 2 . The result is "nt".

locate (string, source text, target text, start_index)

Let us now check the **locate(string,source text, target text, start_index)** function with the following query.

Query

```
default> select locate('tutorialspoint','point',10) as string_position;
```

Result

The above query will generate the following result.

```
string_position

-------------------------------

10
```

The output returns the location of the specified string "point" from the given input string "tutorialspoint". The position is 10.

strposb (source, target)

Let us now check the **strposb(source,target)** function with the following query.

Query

```
default> select strposb('tutorialspoint','tutorial') as location;
```

Result

The above query will generate the following result.

```
location
-------------------------------
1
```

The output returns the binary string position for the given string.

substr (string, start, length)

Let us now check **substr(string,start,length)** with the following query.

Query

```
default> select substr('tutorialspoint',2,5) as substring;
```

Result

The above query will generate the following output.

```
substring
-------------------------------
utori
```

The output returns the substring of (tutorialspoint,2,5) is utori. Here, 2 is the index position and 5 is the length.

trim(string,char)

Let us now check the **trim(string,char)** function with the following query.

Query

```
default> select trim('tutorialspoint','t') as trim_char;
```

Result

The above query will generate the following result.

```
trim_char
-------------------------------
utorialspoin
```

Here, the char 't' is trimmed from 'tutorialspoint'.

split_part (string, delimiter, index)

Let us now check the **split_part(string,delimiter,index)** function with the following query.

Query

```
default> select split_part('tutorialspoint','t', 3) as split;
```

Result

The above query will generate the following result.

```
split
-------------------------------
orialspoin
```

The **split_part()** method splits a string 'tutorialspoint' from the delimiter 't' with the index position 3. The result is "orialspoin".

regexp_replace (string text, pattern text, replacement text)

Let us now check the **regexp_replace(string text, pattern text, replacement text)** function with the following query.

Query

```
default> select regexp_replace('abcdef', '(ˆab|ef$)', '–');
```

Result

The above query generates the following result.

```
–cd–
```

The query replaces the substrings matched to a given regular expression pattern.

reverse(string)

Let us now check the **reverse(string)** function with the following query.

Query

```
default> select reverse('tutorialspoint') as string_reverse;
```

Result

The above query will generate the following result.

```
string_reverse

-------------------------------

tniopslairotut
```

The query returns reverse of the given string.

11. Apache Tajo – DateTime Functions

Apache Tajo supports the following DateTime functions.

Functions	Description
add_days(date date or timestamp, int day)	Returns date added by the given day value.
add_months(date date or timestamp, int month)	Returns date added by the given month value.
current_date()	Returns today’s date
current_time()	Returns today’s time.
extract(century from date/timestamp)	Extracts century from the given parameter.
extract(day from date/timestamp)	Extracts day from the given parameter.
extract(decade from date/timestamp)	Extracts decade from the given parameter.
extract(day dow date/timestamp)	Extracts day of week from the given parameter.
extract(doy from date/timestamp)	Extracts day of year from the given parameter.
select extract(hour from timestamp)	Extracts hour from the given parameter.
select extract(isodow from timestamp)	Extracts day of week from the given parameter. This is identical to dow except for Sunday. This matches the ISO 8601 day of the week numbering.
select extract(isoyear from date)	Extracts ISO year from the specified date. ISO year may be different from the Gregorian year.

extract(microseconds from time)	Extracts microseconds from the given parameter. The seconds field, including fractional parts, multiplied by 1 000 000;
extract(extract(millennium from timestamp)	Extracts millennium from the given parameter.one millennium corresponds to 1000 years. Hence, the third millennium started January 1, 2001.
extract(milliseconds from time)	Extracts milliseconds from the given parameter.
extract(minute from timestamp)	Extracts minute from the given parameter.
extract(quarter from timestamp)	Extracts quarter of the year(1 - 4) from the given parameter.
date_part(field text, source date or timestamp or time)	Extracts date field from text.
now()	Returns current timestamp
to_char(timestamp, format text)	Converts timestamp to text.
to_date(src text, format text)	Converts text to date.
to_timestamp(src text, format text)	Converts text to timestamp

current_date()

Let us now check the **current_date()** function with the following query.

Query

```
default> select current_date() as today_date;
```

Result

The above query will generate the following result.

```
today_date
-------------------------------
2016-08-11
```

The output shows today’s date.

add_days(date,day)

Let us now check the **add_days(date,day)** function with the following query.

Query

```
default> select add_days(date '2016-08-11',2) as add;
```

Result

The above query will generate the following result.

```
add

-------------------------------

2016-08-13 00:00:00
```

The above query returns the given date by adding two days.

Let us now see how the query progresses further.

Query

```
default> select add_days(timestamp '2016-08-11 17:04:00',2) as add;
```

Result

The above query will generate the following result.

```
add

-------------------------------

2016-08-13 17:04:00
```

The query returns the given timestamp by adding two days to it.

add_months(date,month)

Let us now check the **add_months(date,month)** function with the following query.

Query

```
default> select add_months(date '2016-08-11',1) as months;
```

Result

The above query will generate the following result.

```
months
-------------------------------
2016-09-11 00:00:00
```

Here, one month is added from the given date.

current_time()

Let us now check the **current_time()** function with the following query.

Query

```
default> select current_time() as time;
```

Result

The above query will generate the following result.

```
time
-------------------------------
17:05:50.516999
```

The query returns the current time.

extract(century)

Let us now check the **extract(century)** function with the following query.

Query

```
default> select extract(century from date '2016-08-11') as century;
```

Result

The above query will generate the following result.

```
century
-------------------------------
21.0
```

The query extracts century from the given date. 2016 belongs to the twenty-first century so, the result is 21.

extract(day)

Let us now check the **extract(day)** function with the following query.

Query

```
default> select extract(day from date '2016-08-11') as day;
```

Result

The above query will generate the following result.

```
day

-------------------------------

11.0
```

The query extracts a day from the given day.

extract(decade)

Let us now check the **extract(decade)** function with the following query.

Query

```
default> select extract(decade from date '2016-08-11') as decade;
```

Result

The above query will generate the following output.

```
decade

-------------------------------

201.6
```

The query extracts a decade from the specified date. One decade is equal to 10 years so, 2016 years equals to 201.6 decades.

extract (day of week)

Let us now check the **extract(day of week)** function with the following query.

Query

```
default> select extract(dow from date '2016-08-11') as dayofweek;
```

Result

The above query will generate the following output.

```
dayofweek
-------------------------------
4.0
```

The query extracts the day of the week.

extract (day of year)

Let us now check the **extract(day of year)** function with the following query.

Query

```
default> select extract(doy from date '2016-08-11') as dayofyear;
```

Result

The above query will generate the following result.

```
dayofyear
-------------------------------
224.0
```

The query extracts the day of the year.

extract(hour)

Let us now check the **extract(hour)** function with the following query.

Query

```
default> select extract(hour from timestamp '2016-08-11 17:17:00') as hour_field;
```

Result

The above query will generate the following result.

```
hour_field
-------------------------------
17.0
```

The query extracts an hour from the given timestamp.

extract (isodow)

Let us now check the **extract(isodow)** function with the following query.

Query

```
default> select extract(isodow from timestamp '2016-08-11 17:22:00') as dayofweek;
```

Result

The above query will generate the following result

```
dayofweek

-------------------------------

4.0
```

The query extracts ISO day of the week.

extract (isoyear)

Let us now check the **extract(isoyear)** function with the following query.

Query

```
default> select extract(isoyear from date '2016-08-11') as year;
```

Result

The above query will generate the following result.

```
year

-------------------------------

2016.0
```

The query extracts ISO year from the specified date.

extract (microseconds)

Let us now check the **extract(microseconds)** function with the following query.

Query

```
default> select extract(microseconds from time '17:25:15') as seconds;
```

Result

The above query will generate the following result.

```
seconds
-------------------------------
1.5E7
```

The query extracts microseconds from the given time.

extract (millennium)

Let us now check the **extract(millenium)** function with the following query.

Query

```
default> select extract(millennium from timestamp '2016-08-11 17:25:15') as
millennium_year;
```

Result

The above query will generate the following result.

```
millennium_year
-------------------------------
3.0
```

The query returns the millennium for the given year 2016 and it is 3. One millennium is equal to 1000 years.

extract (milliseconds)

Let us now check the **extract(milliseconds)** function with the following query.

Query

```
default> select extract(milliseconds from time '17:25:15') as seconds;
```

Result

The above query will generate the following result.

```
seconds
-------------------------------
15000.0
```

The query extracts milliseconds from the specified time.

extract (minute)

Let us now check the **extract(minute)** function with the following query.

Query

```
default> select extract(minute from time '17:25:15') as minutes;
```

Result

The above query will generate the following result.

```
minutes

-------------------------------

25.0
```

The query extracts minute from the given time.

extract (quarter)

Let us now check the **extract(quarter)** function with the following query.

Query

```
default> select extract(quarter from date '2016-08-11') as quarter_month;
```

Result

The above query will generate the following result.

```
quarter_month

-------------------------------

3.0
```

The query extracts a quarter of the month from the given month. The given month August comes under the third quarter of the year.

date_part (month,date)

Let us now check the **date_part(month,date)** function with the following query.

Query

```
default> select date_part('month', date '2016-08-11') as month;
```

Result

The above query will generate the following result.

```
month
-------------------------------
8.0
```

The query returns a month from the specified date.

now()

Let us now check the **now()** function with the following query.

Query

```
default> select now() as datetime;
```

Result

The above query generates the following result.

```
datetime
-------------------------------
2016-08-11 17:37:42.45
```

The query returns today’s date with time.

to_char (timestamp,format)

Let us now check the **to_char(timestamp,format)** function with the following query.

Query

```
default> select to_char(current_timestamp,'dd-mm-yyyy') as convert_char;
```

Result

The above query will generate the following result.

```
convert_char
-------------------------------
11-08-2016
```

The query converts timestamp to character format.

12.

to_date (date, format)

Let us now check the **to_date(date,format)** function with the following query.

Query

```
default> select to_date('2014-01-04', ‘YYYY-MM-DD');
```

Result

The above query will generate the following result.

```
?to_date

-------------------------------

2014-01-04
```

The query returns the date conversion.

to_timestamp (date,format)

Let us now check the **to_timestamp(date,format)** function with the following query.

Query

```
default> select to_timestamp(412312345);
```

Result

The above query generates the following result.

```
1983-01-25 03:12:25
```

The query converts to timestamp.

12. Apache Pajo – JSON Functions

The JSON functions are listed in the following table:

Functions	Description
json_extract_path_text(json text, json_path text)	Extracts JSON string from a JSON string based on json path specified
json_array_get(json_array text, index int4)	Returns the element at the specified index into the JSON array.
json_array_contains(json_array text, value any)	Determine if the given value exists in the JSON array.
json_array_length(json_array text)	Returns the length of json array.

json_extract_path_text (json text, json_path text)

Let us now check the **json_extract_path_text(json text, json_path text)** function with the following query.

Query

```
default> select json_extract_path_text('{"tutorial" : {"key" :
"tajo"}}','$.tutorial.key') as path;
```

Result

The above query will generate the following result.

```
path
-------------------------------
tajo
```

The query extracts the JSON string "tutorial" from a JSON string based on the JSON path.

json_array_get(json_array text, index int4)

Let us now check the **json_array_get(json_array text, index int4)** function with the following query.

Query

```
default> select json_array_get('[10, 20, 30]', 2) as array;
```

Result

The above query will generate the following result.

```
array

-------------------------------

30
```

The query returns the element at the specified index into the JSON array. Here, 2 index has 20 array element.

json_array_contains (json_array text, value any)

Let us now check the **json_array_contains(json_array text, value any)** function with the following query.

Query

```
default> select json_array_contains('[10, 20, 30]', 10) as array_contains;
```

Result

The above query will generate the following result.

```
array_contains

-------------------------------

true
```

The query determines if the given value exists in the JSON array or not. Here, the given value 10 is present in the array so, the result is true.

json_array_length(json_array text)

Let us now check the **json_array_length(json_array text)** function with the following query.

Query

```
default> select json_array_length('[10, 20, 30]') as length;
```

Result

The above query will generate the following result.

```
length
-------------------------------
3
```

The query returns the length of the array. The above array contains 3 elements so, the length is 3.

13. Apache Tajo – Database Creation

This section explains the Tajo DDL commands. Tajo has a built-in database named **default**.

Create Database Statement

Create Database is a statement used to create a database in Tajo. The syntax for this statement is as follows:

```
CREATE DATABASE [IF NOT EXISTS] <database_name>
```

Query

```
default> default> create database if not exists test;
```

Result

The above query will generate the following result.

```
OK
```

Database is the namespace in Tajo. A database can contain multiple tables with a unique name.

Show Current Database

To check the current database name, issue the following command:

Query

```
default> \c
```

Result

The above query will generate the following result.

```
You are now connected to database "default" as user “user1".

default>
```

Connect to Database

As of now, you have created a database named “test”. The following syntax is used to connect the “test” database.

```
\c <database name>
```

Query

```
default> \c test
```

Result

The above query will generate the following result.

```
You are now connected to database "test" as user “user1”.
test>
```

You can now see the prompt changes from default database to test database.

Drop Database

To drop a database, use the following syntax:

```
DROP DATABASE <database-name>
```

Query

```
test> \c default
You are now connected to database "default" as user “user1".
default> drop database test;
```

Result

The above query will generate the following result.

```
OK
```

14. Apache Tajo — Table Management

A table is a logical view of one data source. It consists of a logical schema, partitions, URL, and various properties. A Tajo table can be a directory in HDFS, a single file, one HBase table, or a RDBMS table.

Tajo supports the following two types of tables:

- external table
- internal table

External Table

External table needs the location property when the table is created. For example, if your data is already there as Text/JSON files or HBase table, you can register it as Tajo external table.

The following query is an example of external table creation.

```
create external table sample(col1 int,col2 text,col3 int) location
‘hdfs://path/to/table';
```

Here,

- **External keyword** – This is used to create an external table. This helps to create a table in the specified location.
- Sample refers to the table name
- **Location** - It is a directory for HDFS,Amazon S3, HBase or local file system. To assign a location property for directories, use the below URI examples:
  - HDFS - hdfs://localhost:port/path/to/table
  - Amazon S3 - s3://bucket-name/table
  - local file system - file:///path/to/table
  - Openstack Swift - swift://bucket-name/table

Table Properties

An external table has the following properties:

- **TimeZone** - Users can specify a time zone for reading or writing a table.
- **Compression format** - Used to make data size compact. For example, the text/json file uses **compression.codec** property.

Managed Table

A **managed table** is also called an internal table. It is created in a pre-defined physical location called the **Tablespace**.

Syntax

```
create table table1(col1 int,col2 text);
```

By default, Tajo uses "tajo.warehouse.directory" located in "conf/tajo-site.xml" . To assign new location for the table, you can use Tablespace configuration.

Tablespace

Tablespace is used to define locations in the storage system. It is supported for only internal tables. You can access the tablespaces by their names. Each tablespace can use a different storage type. If you don't specify tablespaces then, Tajo uses the default tablespace in the root directory.

Tablespace Configuration

You have **"conf/tajo-site.xml.template"** in Tajo. Copy the file and rename it to **"storage-site.json"**. This file will act as a configuration for Tablespaces. Tajo data formats uses the following configuration:

HDFS Configuration

```
$ vi conf/storage-site.json


{
  "spaces": {


    "${tablespace_name}": {


      "uri": "hdfs://localhost:9000/path/to/Tajo"


    }
  }
}
```

HBase Configuration

```
$ vi conf/storage-site.json
{
  "spaces": {

    "${tablespace_name}": {

      "uri": “hbase:zk://quorum1:port,quorum2:port/"

    }
  }
}
```

Text File Configuration

```
$ vi conf/storage-site.json



{
  "spaces": {

    "${tablespace_name}": {

      “uri”: “hdfs://localhost:9000/path/to/Tajo”
              }
          }
}
```

Tablespace Creation

Tajo’s internal table records can be accessed from another table only. You can configure it with tablespace.

Syntax

```
CREATE TABLE [IF NOT EXISTS] <table_name> [(column_list)] [TABLESPACE
tablespace_name]
[using <storage_type> [with (<key> = <value>, ...)]] [AS <select_statement>]
```

Here,

- **IF NOT EXISTS** — This avoids an error if the same table has not been created already.
- **TABLESPACE** — This clause is used to assign the tablespace name.
- **Storage type** — Tajo data supports formats like text,JSON,HBase,Parquet,Sequencefile and ORC.
- **AS select statement** — Select records from another table.

Configure Tablespace

Start your Hadoop services and open the file **"conf/storage-site.json",** then add the following changes:

```
$ vi conf/storage-site.json


{
  "spaces": {


    “space1”: {


      "uri": “hdfs://localhost:9000/path/to/Tajo"
    }
  }
}
```

Here, Tajo will refer to the data from HDFS location and **space1** is the tablespace name. If you do not start Hadoop services, you can't register tablespace.

Query

```
default> create table table1(num1 int,num2 text,num3 float) tablespace space1;
```

The above query creates a table named "table1" and "space1" refers to the tablespace name.

Data formats

Tajo supports data formats. Let's go through each of the formats one by one in detail.

Text

A character-separated values' plain text file represents a tabular data set consisting of rows and columns. Each row is a plain text line.

Creating Table

```
default> create external table customer(id int,name text,address text,age int) using
text with('text.delimiter'=',') location ‘file:/Users/workspace/Tajo/customers.csv’;
```

Here, “**customers.csv**” file refers to a comma separated value file located in the Tajo installation directory.

To create internal table using text format, use the following query:

```
default> create table customer(id int,name text,address text,age int) using text;
```

In the above query, you have not assigned any tablespace so it will take Tajo’s default tablespace.

Properties

A text file format has the following properties:

- text.delimiter — This is a delimiter character. Default is ‘|’.
- compression.codec — This is a compression format. By default, it is disabled. you can change the settings using specified algorithm.
- timezone — The table used for reading or writing.
- text.error-tolerance.max-num — The maximum number of tolerance levels.
- text.skip.headerlines — The number of header lines per skipped.
- text.serde — This is serialization property.

JSON

Apache Tajo supports JSON format for querying data. Tajo treats a JSON object as SQL record. One object equals one row in a Tajo table. Let’s consider “array.json” as follows:

```
$ hdfs dfs -cat /json/array.json


{
"num1" : 10,
"num2" : "simple json array",
"num3" : 50.5
}
```

After you create this file, switch to the Tajo shell and type the following query to create a table using the JSON format.

Query

```
default> create external table sample (num1 int,num2 text,num3 float) using json
location 'json/array.json';
```

Always remember that the file data must match with the table schema. Otherwise, you can omit the column names and use * which doesn't require columns list.

To create an internal table, use the following query:

```
default> create table sample (num1 int,num2 text,num3 float) using json;
```

Parquet

Parquet is a columnar storage format. Tajo uses Parquet format for easy, fast and efficient access.

Table creation

The following query is an example for table creation:

```
CREATE TABLE parquet (num1 int,num2 text,num3 float) USING PARQUET;
```

Parquet file format has the following properties:

parquet.block.size — size of a row group being buffered in memory.

parquet.page.size — The page size is for compression.

parquet.compression — The compression algorithm used to compress pages.

parquet.enable.dictionary — The boolean value is to enable/disable dictionary encoding.

RCFile

RCFile is the Record Columnar File. It consists of binary key/value pairs.

Table creation

The following query is an example for table creation:

```
CREATE TABLE Record(num1 int,num2 text,num3 float) USING RCFILE;
```

RCFile has the following properties:

- rcfile.serde — custom deserializer class.
- compression.codec — compression algorithm.
- rcfile.null — NULL character

SequenceFile

SequenceFile is a basic file format in Hadoop which consists of key/value pairs.

Table creation

The following query is an example for table creation:

```
CREATE TABLE seq(num1 int,num2 text,num3 float) USING sequencefile;
```

This sequence file has Hive compatibility. This can be written in Hive as,

```
CREATE TABLE table1 (id int, name string, score float, type string)
STORED AS sequencefile;
```

ORC

ORC (Optimized Row Columnar) is a columnar storage format from Hive.

Table creation

The following query is an example for table creation.

```
CREATE TABLE optimized(num1 int,num2 text,num3 float) USING ORC;
```

The ORC format has the following properties:

- orc.max.merge.distance — ORC file is read, it merges when the distance is lower.
- orc.stripe.size — This is the size of each stripe.
- orc.buffer.size — The default is 256KB.
- orc.rowindex.stride — This is the ORC index stride in number of rows.

15. Apache Tajo — SQL Statements

In the previous chapter, you have understood how to create tables in Tajo. This chapter explains about the SQL statement in Tajo.

Create Table Statement

Before moving to create a table, create a text file "students.csv" in Tajo installation directory path as follows:

students.csv

Id	Name	Address	Age	Marks
1	Adam	23 New Street	21	90
2	Amit	12 Old Street	13	95
3	Bob	10 Cross Street	12	80
4	David	15 Express Avenue	12	85
5	Esha	20 Garden Street	13	50
6	Ganga	25 North Street	12	55
7	Jack	2 Park Street	12	60
8	Leena	24 South Street	12	70
9	Mary	5 West Street	12	75
10	Peter	16 Park Avenue	12	95

After the file has been created, move to the terminal and start the Tajo server and shell one by one.

Create Database

Create a new database using the following command:

Query

```
default> create database sampledb;
OK
```

Connect to the database "sampledb" which is now created.

```
default> \c sampledb

You are now connected to database "sampledb" as user “user1”.
```

Then, create a table in “sampledb” as follows:

Query

```
sampledb>  create external table mytable(id int,name text,address text,age int,mark
int) using text with('text.delimiter'=',') location
‘file:/Users/workspace/Tajo/students.csv’;
```

Result

The above query will generate the following result.

```
OK
```

Here, the external table is created. Now, you just have to enter the file location. If you have to assign the table from hdfs then use hdfs instead of file.

Next, the **“students.csv”** file contains comma separated values. The **text.delimiter** field is assigned with ‘,’.

You have now created “mytable” successfully in “sampledb”.

Show Table

To show tables in Tajo, use the following query.

Query

```
sampledb> \d


mytable


sampledb> \d mytable
```

Result

The above query will generate the following result.

```
table name: sampledb.mytable

table uri: file:/Users/workspace/Tajo/students.csv

store type: TEXT

number of rows: unknown

volume: 261 B
```

```
Options:
'timezone'='Asia/Kolkata'
'text.null'='\\N'
'text.delimiter'=','

schema:
id INT4
name TEXT
address TEXT
age INT4
mark INT4
```

List table

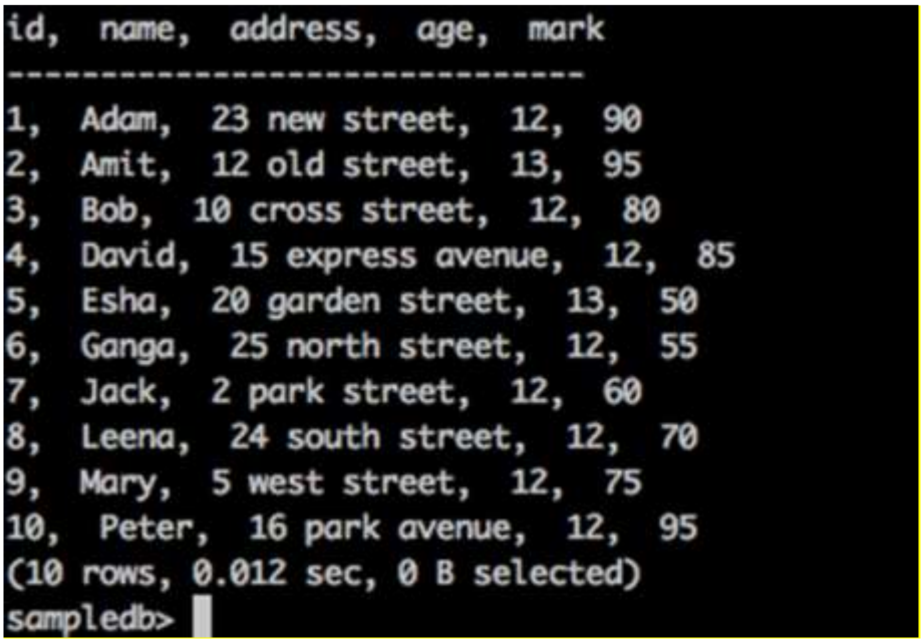
To fetch all the records in the table, type the following query:

Query

```
sampledb> select * from mytable;
```

Result

The above query will generate the following result.



```
id,  name,  address,  age,  mark
-------------------------------
1,  Adam,  23 new street,  12,  90
2,  Amit,  12 old street,  13,  95
3,  Bob,  10 cross street,  12,  80
4,  David,  15 express avenue,  12,  85
5,  Esha,  20 garden street,  13,  50
6,  Ganga,  25 north street,  12,  55
7,  Jack,  2 park street,  12,  60
8,  Leena,  24 south street,  12,  70
9,  Mary,  5 west street,  12,  75
10,  Peter,  16 park avenue,  12,  95
(10 rows, 0.012 sec, 0 B selected)
sampledb>
```

Insert Table Statement

Tajo uses the following syntax to insert records in table.

Syntax

```
create table table1 (col1 int8, col2 text, col3 text);
--schema should be same for target table schema

Insert overwrite into table1 select * from table2;

                         (or)

Insert overwrite into LOCATION '/dir/subdir' select * from table;
```

Tajo’s insert statement is similar to the **INSERT INTO SELECT** statement of SQL.

Query

Let’s create a table to overwrite table data of an existing table.

```
sampledb> create table test(sno int,name text,addr text,age int,mark int);

OK
sampledb> \d
```

Result

The above query will generate the following result.

```
mytable
test
```

Insert Records

To insert records in the “test” table, type the following query.

Query

```
sampledb> insert overwrite into test select * from mytable;
```

Result

The above query will generate the following result.

```
Progress: 100%, response time: 0.518 sec
```

Here, “mytable" records overwrite the “test” table. If you don’t want to create the “test” table, then straight away assign the physical path location as mentioned in an alternative option for insert query.

Fetch records

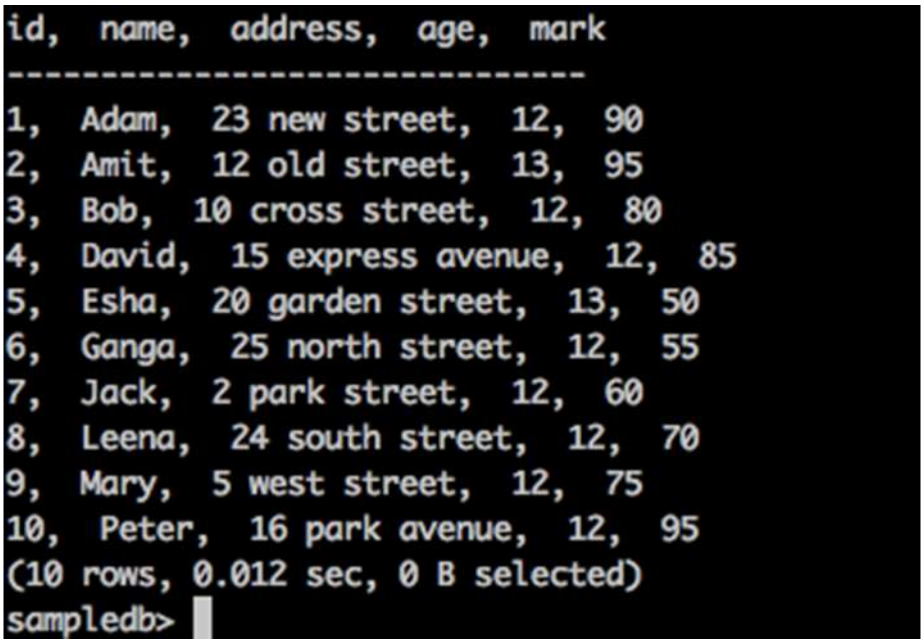
Use the following query to list out all the records in the “test” table:

Query

```
sampledb> select * from test;
```

Result

The above query will generate the following result.



```
id,  name,  address,  age,  mark
-------------------------------
1,  Adam,  23 new street,  12,  90
2,  Amit,  12 old street,  13,  95
3,  Bob,  10 cross street,  12,  80
4,  David,  15 express avenue,  12,  85
5,  Esha,  20 garden street,  13,  50
6,  Ganga,  25 north street,  12,  55
7,  Jack,  2 park street,  12,  60
8,  Leena,  24 south street,  12,  70
9,  Mary,  5 west street,  12,  75
10,  Peter,  16 park avenue,  12,  95
(10 rows, 0.012 sec, 0 B selected)
sampledb>
```

This statement is used to add, remove or modify columns of an existing table.

To rename the table use the following syntax:

```
Alter table table1 RENAME TO table2;
```

Query

```
sampledb> alter table test rename to students;
```

Result

The above query will generate the following result.

```
OK
```

To check the changed table name, use the following query.

```
sampledb> \d


mytable
students
```

Now the table “test” is changed to “students” table.

Add Column

To insert new column in the “students” table, type the following syntax:

```
Alter table <table_name> ADD COLUMN <column_name> <data_type>
```

Query

```
sampledb> alter table students add column grade text;
```

Result

The above query will generate the following result.

```
OK
```

Set Property

This property is used to change the table’s property.

Query

```
sampledb> ALTER TABLE students SET PROPERTY
'compression.type'='RECORD','compression.codec'='org.apache.hadoop.io.compress.Snappy
Codec' ;

OK
```

Here, compression type and codec properties are assigned.

To change the text delimiter property, use the following:

Query

```
ALTER TABLE students  SET PROPERTY ‘text.delimiter'=',';


OK
```

Result

The above query will generate the following result.

```
sampledb> \d students


table name: sampledb.students
table uri: file:/tmp/tajo-user1/warehouse/sampledb/students
store type: TEXT
number of rows: 10
volume: 228 B
Options:
'compression.type'='RECORD'
'timezone'='Asia/Kolkata'
'text.null'='\\N'
'compression.codec'='org.apache.hadoop.io.compress.SnappyCodec'
'text.delimiter'=','


schema:
id INT4
name TEXT
addr TEXT
age INT4
mark INT4
grade TEXT
```

The above result shows that the table’s properties are changed using the “SET” property.

Select Statement

The SELECT statement is used to select data from a database.

The syntax for the Select statement is as follows:

```
SELECT [distinct [all]] * | <expression> [[AS] <alias>] [, ...]
  [FROM <table reference> [[AS] <table alias name>] [, ...]]
  [WHERE <condition>]
  [GROUP BY <expression> [, ...]]
  [HAVING <condition>]
  [ORDER BY <expression> [ASC|DESC] [NULLS (FIRST|LAST)] [, …]]
```

Where Clause

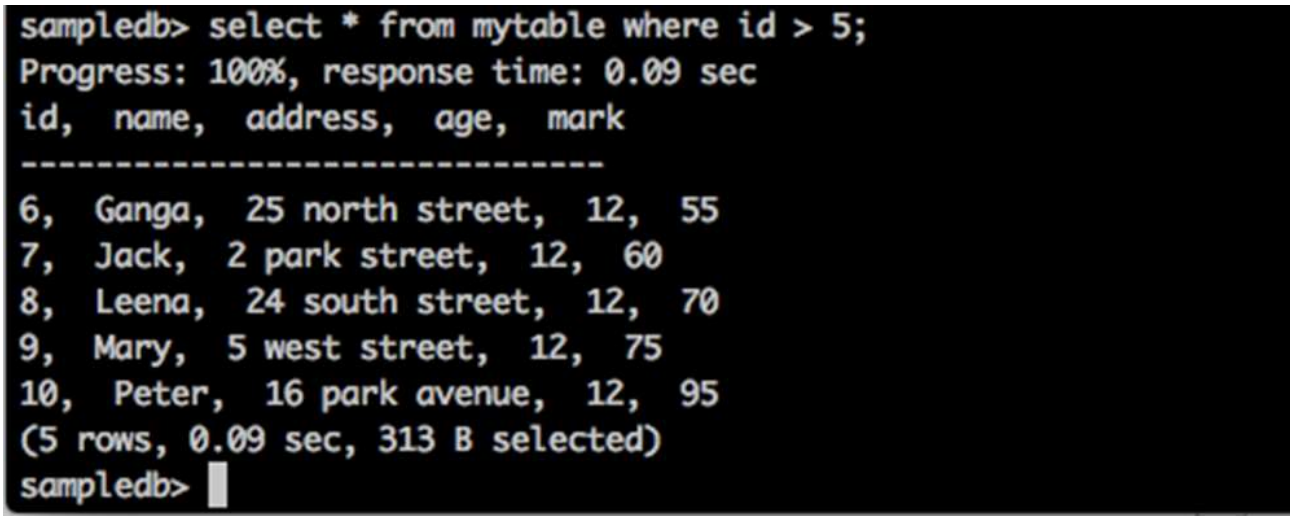
The Where clause is used to filter records from the table.

Query

```
sampledb> select * from mytable where id > 5;
```

Result

The above query will fetch the following result.



```
sampledb> select * from mytable where id > 5;
Progress: 100%, response time: 0.09 sec
id,  name,  address,  age,  mark
-------------------------------
6,  Ganga,  25 north street,  12,  55
7,  Jack,  2 park street,  12,  60
8,  Leena,  24 south street,  12,  70
9,  Mary,  5 west street,  12,  75
10,  Peter,  16 park avenue,  12,  95
(5 rows, 0.09 sec, 313 B selected)
sampledb>
```

The query returns the records of those students whose id is greater than 5.

Query

```
sampledb> select * from mytable where name = ‘Peter’;
```

Result

The above query generates the following result.

```
Progress: 100%, response time: 0.117 sec


id,  name,  address      ,   age
-------------------------------
10,  Peter,  16 park avenue , 12
```

The result filters Peter's records only.

Distinct Clause

A table column may contain duplicate values. The DISTINCT keyword can be used to return only distinct (different) values.

Syntax

```
SELECT DISTINCT column1,column2 FROM table_name;
```

Query

```
sampledb> select distinct age from mytable;
```

Result

The above query will generate the following result.

```
Progress: 100%, response time: 0.216 sec


age
-------------------------------
13
12
```

The query returns the distinct age of students from **mytable**.

Group By Clause

The GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

Syntax

```
SELECT column1, column2 FROM table_name WHERE [ conditions ] GROUP BY column1,
column2;
```

Query

```
select age,sum(mark) as sumofmarks from mytable group by age;
```

Result

The above query will generate the following result.

```
age,  sumofmarks
-------------------------------
13,  145
12,  610
```

Here, the "mytable" column has two types of ages — 12 and 13. Now the query groups the records by age and produces the sum of marks for the corresponding ages of students.

Having Clause

The HAVING clause enables you to specify conditions that filter which group results appear in the final results. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on the groups created by the GROUP BY clause.

Syntax

```
SELECT column1, column2 FROM table1 GROUP BY column HAVING [ conditions ]
```

Query

```
sampledb> select age from mytable group by age  having  sum(mark) > 200;
```

Result

The above query generates the following result.

```
age
-------------------------------
12
```

The query groups the records by age and returns the age when the condition result sum(mark)>200.

Order By Clause

The ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. The Tajo database sorts query results in ascending order by default.

Syntax

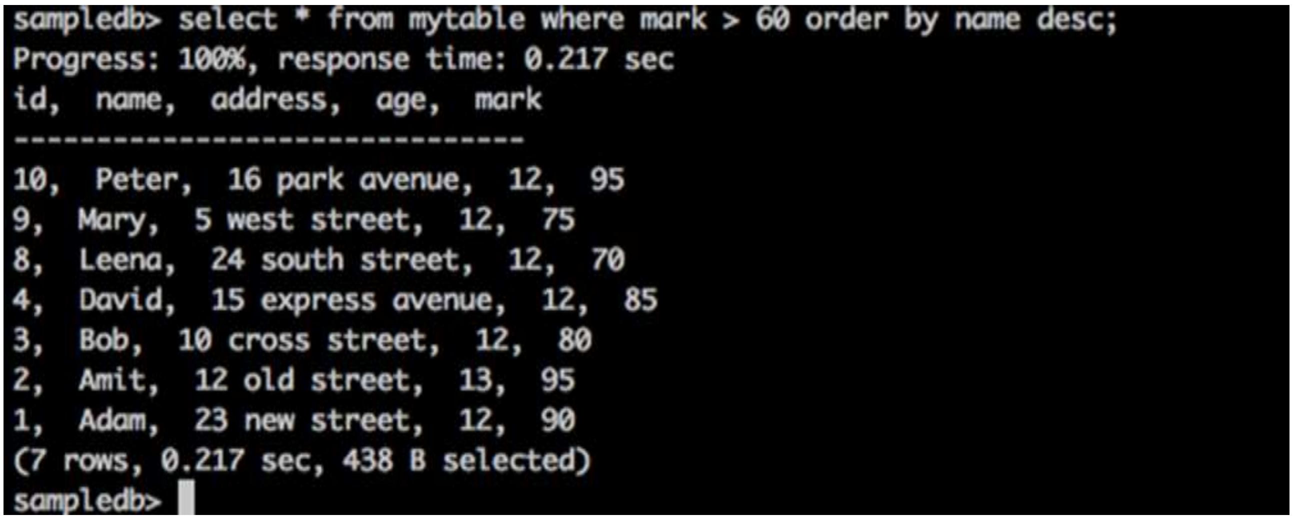
```
SELECT column-list FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

Query

```
sampledb> select * from mytable where mark > 60 order by name desc;
```

Result

The above query generates the following result.



```
sampledb> select * from mytable where mark > 60 order by name desc;
Progress: 100%, response time: 0.217 sec
id,  name,  address,  age,  mark
-------------------------------
10,  Peter,  16 park avenue,  12,  95
9,  Mary,  5 west street,  12,  75
8,  Leena,  24 south street,  12,  70
4,  David,  15 express avenue,  12,  85
3,  Bob,  10 cross street,  12,  80
2,  Amit,  12 old street,  13,  95
1,  Adam,  23 new street,  12,  90
(7 rows, 0.217 sec, 438 B selected)
sampledb>
```

The query returns the names of those students in descending order whose marks are greater than 60.

Create Index Statement

The CREATE INDEX statement is used to create indexes in tables. Index is used for fast retrieval of data. Current version supports index for only plain TEXT formats stored on HDFS.

Syntax

```
CREATE INDEX [ name ] ON table_name ( { column_name | ( expression ) }
```

Query

```
create index student_index on mytable(id);
```

Result

```
id
_______________
```

To view assigned index for the column, type the following query.

```
default> \d mytable

table name: default.mytable
table uri: file:/Users/deiva/workspace/Tajo/students.csv
store type: TEXT
number of rows: unknown

volume: 307 B
Options:
     'timezone'='Asia/Kolkata'
     'text.null'='\\N'
     'text.delimiter'=','

schema:
id    INT4
name  TEXT
address     TEXT
age   INT4
mark  INT4



Indexes:

"student_index" TWO_LEVEL_BIN_TREE (id ASC NULLS LAST )
```

Here, TWO_LEVEL_BIN_TREE method is used by default in Tajo.

Drop Table Statement

The Drop Table Statement is used to drop a table from the database.

Syntax

```
drop table table name;
```

Query

```
sampledb> drop table mytable;
```

To check if the table has been dropped from the table, type the following query.

```
sampledb> \d mytable;
```

Result

```
ERROR: relation 'mytable' does not exist
```

You can also check the query using “\d” command to list out the available Tajo tables.

16. Apache Tajo – Aggregate & Window Functions

This chapter explains the aggregate and window functions in detail.

Aggregation Functions

Aggregate functions produce a single result from a set of input values. The following table describes the list of aggregate functions in detail.

Function	Description
AVG(expression)	Averages a column of all records in a data source.
CORR(expression1, expression2)	Returns the coefficient of correlation between a set of number pairs.
COUNT()	Returns the number rows.
MAX(expression)	Returns the largest value of the selected column.
MIN(expression)	Returns the smallest value of the selected column.
SUM(expression)	Returns the sum of the given column.
LAST_VALUE(expression)	Returns the last value of the given column.

Let us now proceed by performing simple examples using aggregate functions.

Avg(exp)

Let us now check **Avg(exp)** function with the following query.

Query

```
sampledb> select avg(mark) as avg from mytable;
```

Result

The above query will generate the following result.

```
avg
-------------------------------
75.5
```

The above query returns the average of marks from mytable.

corr (exp1,exp2)

Let us now check **corr(exp1,exp2)** function with the following query.

Query

```
sampledb> select corr(age,mark) as correlation from mytable;
```

Result

The above query will generate the following result.

```
correlation
-------------------------------
-0.09637388493048485
```

The query returns the correlation coefficient value between the two columns age and mark.

count()

Let us now check **count()** function with the following query.

Query

```
sampledb> select count() from mytable;
```

Result

The above query will generate the following result.

```
?count
-------------------------------
10
```

The query returns the count of the total number of rows. The table contains 10 rows so, the count result is 10.

max (exp)

Let us now check **max(exp)** function with the following query.

Query

```
sampledb> select max(mark) as max from mytable;
```

Result

The above query will generate the following result.

```
max
-------------------------------
95
```

The query returns the maximum of marks from **mytable**. The maximum mark is 95.

min(exp)

Let us now check **min(exp)** function with the following query.

Query

```
sampledb> select min(mark) as min from mytable;
```

Result

The above query will generate the following result.

```
min
-------------------------------
50
```

The query returns the minimum of marks from **mytable**. The minimum mark is 50.

sum(exp)

Let us now check **sum(exp)** function with the following query.

Query

```
sampledb> select sum(mark) as sum from mytable;
```

Result

The above query will generate the following result.

```
sum

-------------------------------

755
```

The query returns the summation of marks from **mytable**. Total summation of mark column is 755.

last_value(exp)

Let us now check **last_value(exp)** function with the following query.

Query

```
sampledb> select last_value(mark) as lastvalue from mytable;
```

Result

The above query will generate the following result.

```
lastvalue

-------------------------------

95
```

The query returns the last value of mark column from **mytable**.

Window Function

The Window functions execute on a set of rows and return a single value for each row from the query. The term window has the meaning of set of row for the function.

The Window function in a query, defines the window using the OVER() clause.

The **OVER()** clause has the following capabilities:

- Defines window partitions to form groups of rows. (PARTITION BY clause)
- Orders rows within a partition. (ORDER BY clause)

The following table describes the window functions in detail.

Function	Return type	Description
rank()	int	Returns rank of the current row with gaps.
row_number()	int	Returns the current row within its partition, counting from 1.
lead(value[, offset integer[, default any]])	Same as input type	Returns value evaluated at the row that is offset rows after the current row within the partition. If there is no such row, default value will be returned.
lag(value[, offset integer[, default any]])	Same as input type	Returns value evaluated at the row that is offset rows before the current row within the partition.
first_value(value)	Same as input type	Returns the first value of input rows.
last_value(value)	Same as input type	Returns the last value of input rows.

rank()

Let us now check the **rank()** function with the following query.

Query

```
select rank() over (order by name) as rank from mytable;
```

Result

The above query generates the following result.

```
rank
-------------------------------
1
2
3
4
5
6
```

```
7
8
9
10
```

The query returns the rank of the current row which is order by name.

row_number()

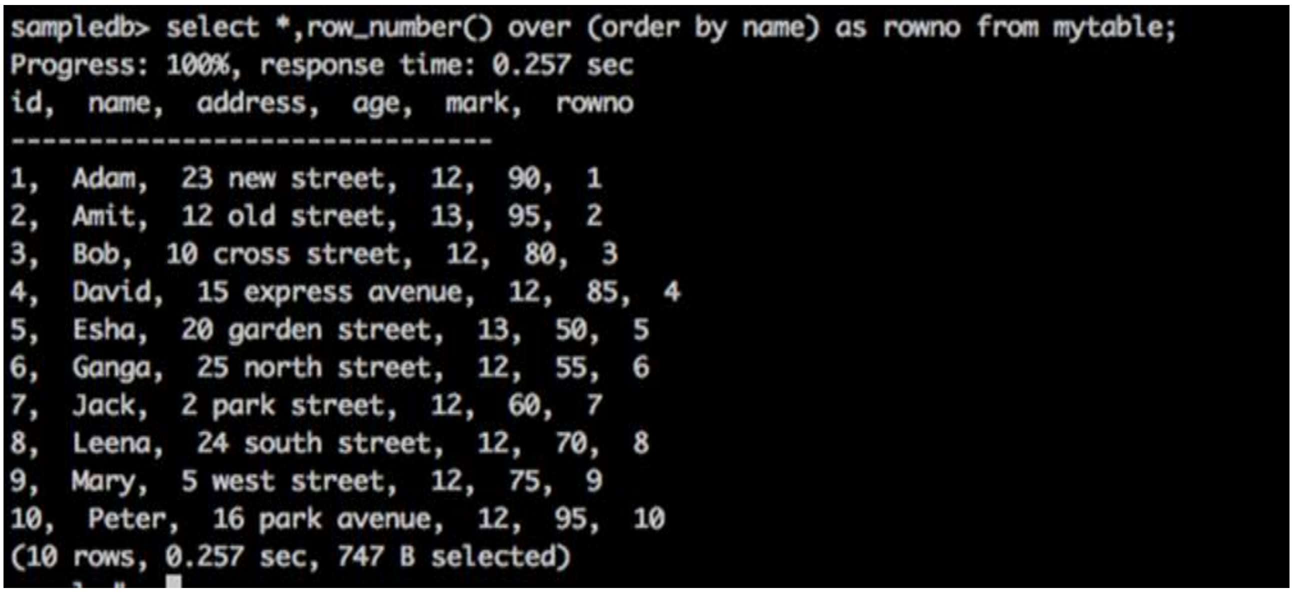
Let us now check the **row_number()** function with the following query.

Query

```
sampledb> select *,row_number() over (order by name) as rowno from mytable;
```

Result

The above query will generate the following result.



```
sampledb> select *,row_number() over (order by name) as rowno from mytable;
Progress: 100%, response time: 0.257 sec
id,  name,  address,  age,  mark,  rowno
-------------------------------
1,  Adam,  23 new street,  12,  90,  1
2,  Amit,  12 old street,  13,  95,  2
3,  Bob,  10 cross street,  12,  80,  3
4,  David,  15 express avenue,  12,  85,  4
5,  Esha,  20 garden street,  13,  50,  5
6,  Ganga,  25 north street,  12,  55,  6
7,  Jack,  2 park street,  12,  60,  7
8,  Leena,  24 south street,  12,  70,  8
9,  Mary,  5 west street,  12,  75,  9
10,  Peter,  16 park avenue,  12,  95,  10
(10 rows, 0.257 sec, 747 B selected)
```

The query assigns the row number for each row in the table.

lead (value, offset, default)

Let us now check the **lead(value,offset,default)** function with the following query.

Query

```
sampledb> select lead(mark,3,2) over (partition by age) as leadvalue from mytable;
```

Result

The above query will generate the following result.

```
leadvalue

------------------------------

55

60

70

75

95

2

2

2

2

2
```

The query returns the value evaluated at the row that is offset rows after the current row within the partition. If no such rows are present, then the given default value 2 is replaced.

lag (value, offset, default)

Let us now check the **lag(value,offset,default)** function with the following query.

Query

```
sampledb> select lag(mark,3,2) over (partition by age) as lagvalue from mytable;
```

Result

The above query will generate the following result.

```
lagvalue

------------------------------

2

2

2

90

80

85

55

60
```

```
2
2
```

The query returns the value evaluated at the row that is offset rows before the current row within the partition. Whenever the rows are not matching, then the given default value 2 is replaced.

first_value (value)

Let us now check the **first_value(value)** function with the following query.

Query

```
sampledb> select first_value(mark) over (order by name) as firstvalue from mytable;
```

Result

The above query will generate the following result.

```
firstvalue
-------------------------------
90
90
90
90
90
90
90
90
90
90
```

The first value 90 in the Mark column is replaced in all rows using the **first_value()** function.

last_value(value)

Let us now check the **last_value(value)** function with the following query.

Query

```
sampledb> select last_value(mark) over (order by name) as lastvalue from mytable;
```

Result

The above query will generate the following result.

```
lastvalue
-------------------------------
95
95
95
95
95
95
95
95
95
95
```

The last value 95 in the Mark column is replaced in all rows using the **last_value()** function.

17. Apache Tajo — SQL Queries

This chapter explains about the following significant Queries.

- Predicates
- Explain
- Join

Let us proceed and perform the queries.

Predicates

Predicate is an expression which is used to evaluate true/false values and UNKNOWN. Predicates are used in the search condition of WHERE clauses and HAVING clauses and other constructs where a Boolean value is required.

IN predicate

Determines whether the value of expression to test matches any value in the subquery or the list. Subquery is an ordinary SELECT statement that has a result set of one column and one or more rows. This column or all expressions in the list must have the same data type as the expression to test.

Syntax

```
IN::=
<expression to test> [NOT] IN (<subquery>)
| (<expression1>,...)
```

Query

```
select id,name,address from mytable where id in(2,3,4);
```

Result

The above query will generate the following result.

```
id,  name,   address
-------------------------------
2,  Amit,  12 old street
3,  Bob,   10 cross street
```

```
4,  David, 15 express avenue
```

The query returns records from mytable for the students id 2,3 and 4.

Query

```
select id,name,address from mytable where id not in(2,3,4);
```

Result

The above query will generate the following result.

```
id,  name,  address
------------------------------
1,  Adam,   23 new street
5,  Esha,   20 garden street
6,  Ganga,  25 north street
7,  Jack,   2 park street
8,  Leena,  24 south street
9,  Mary,   5 west street
10, Peter,  16 park avenue
```

The above query returns records from **mytable** where students is not in 2,3 and 4.

Like Predicate

The LIKE predicate compares the string specified in the first expression for calculating the string value, which is refered to as a value to test, with the pattern that is defined in the second expression for calculating the string value.

The pattern may contain any combination of wildcards such as:

- Underline symbol (_), which can be used instead of any single character in the value to test.
- Percent sign (%), which replaces any string of zero or more characters in the value to test.

Syntax

```
LIKE::=
<expression for calculating the string value>
[NOT] LIKE
<expression for calculating the string value>
[ESCAPE <symbol>]
```

Query

```
select * from mytable where name like ‘A%';
```

Result

The above query will generate the following result.

```
id,  name,  address,     age,  mark
-------------------------------
1,  Adam,  23 new street,  12,  90
2,  Amit,  12 old street,  13,  95
```

The query returns records from mytable of those students whose names are starting with ‘A’.

Query

```
select * from mytable where name like ‘_a%';
```

Result

The above query will generate the following result.

```
id,  name,  address,           age,  mark
——————————————————————————————————————-
4,  David,  15 express avenue,  12,  85
6,  Ganga,  25 north street,    12,  55
7,  Jack,  2 park street,       12,  60
9,  Mary,  5 west street,       12,  75
```

The query returns records from **mytable** of those students whose names are starting with ‘a’ as the second char.

Using NULL Value in Search Conditions

Let us now understand how to use NULL Value in the search conditions.

Syntax

```
Predicate


IS [NOT] NULL
```

Query

```
select name from mytable where name is not null;
```

Result

The above query will generate the following result.

```
name
-------------------------------
Adam
Amit
Bob
David
Esha
Ganga
Jack
Leena
Mary
Peter

(10 rows, 0.076 sec, 163 B selected)
```

Here, the result is true so it returns all the names from table.

Query

Let us now check the query with NULL condition.

```
default> select name from mytable where name is null;
```

Result

The above query will generate the following result.

```
name
-------------------------------
(0 rows, 0.068 sec, 0 B selected)
```

Explain

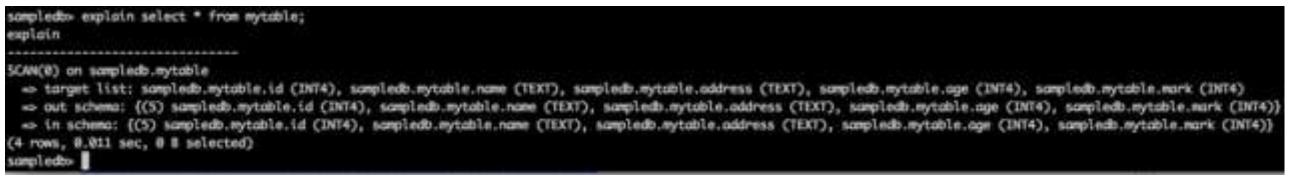
Explain is used to obtain a query execution plan. It shows a logical and global plan execution of a statement.

Logical Plan Query

```
explain select * from mytable;


explain

-------------------------------



 => target list: default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)
  => out schema: {(5) default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)}
  => in schema: {(5) default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)}
```

Result

The above query will generate the following result.



```
sampledb> explain select * from mytable;
explain
-------------------------------
SCAN(0) on sampledb.mytable
  => target list: sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)
  => out schema: {(5) sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)}
  => in schema: {(5) sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)}
(4 rows, 0.011 sec, 0 B selected)
sampledb>
```

The query result shows a logical plan format for the given table. The Logical plan returns the following three results:

- Target list
- Out schema
- In schema

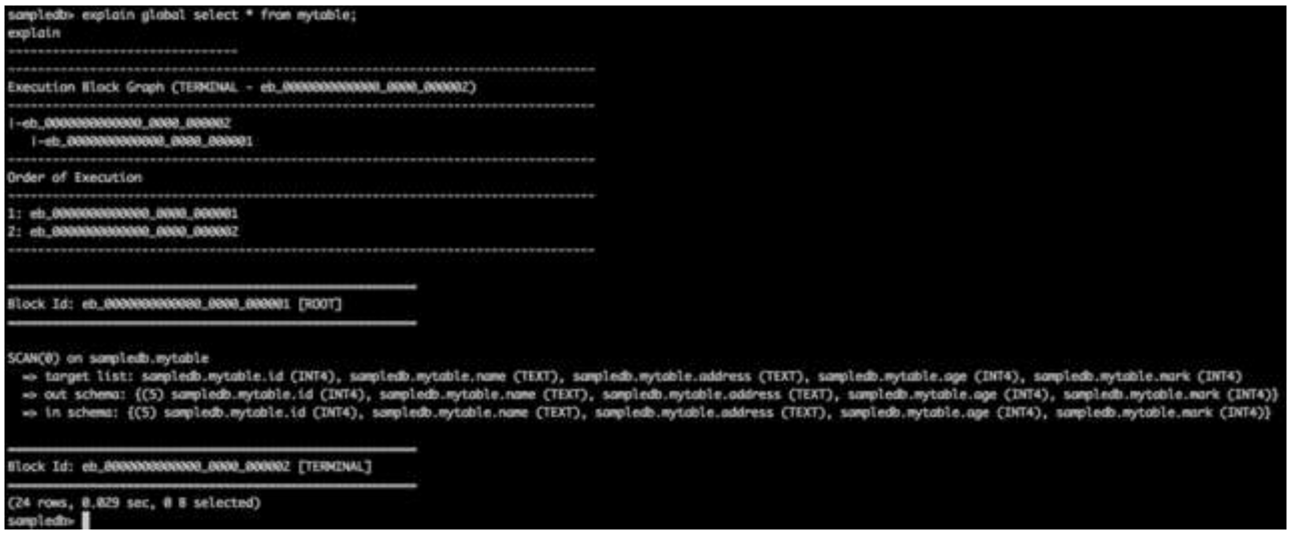
Global Plan Query

```
explain global select * from mytable;


explain

-------------------------------

-------------------------------------------------------------------------------

Execution Block Graph (TERMINAL - eb_0000000000000_0000_000002)
```

```
-------------------------------------------------------------------------------
|-eb_0000000000000_0000_000002
   |-eb_0000000000000_0000_000001
-------------------------------------------------------------------------------
Order of Execution
-------------------------------------------------------------------------------
1: eb_0000000000000_0000_000001
2: eb_0000000000000_0000_000002
-------------------------------------------------------------------------------


=======================================================
Block Id: eb_0000000000000_0000_000001 [ROOT]
=======================================================


SCAN(0) on default.mytable
  => target list: default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)
  => out schema: {(5) default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)}
  => in schema: {(5) default.mytable.id (INT4), default.mytable.name (TEXT),
default.mytable.address (TEXT), default.mytable.age (INT4), default.mytable.mark
(INT4)}


=======================================================
Block Id: eb_0000000000000_0000_000002 [TERMINAL]
=======================================================
(24 rows, 0.065 sec, 0 B selected)
```

Result

The above query will generate the following result.



```
sampledb> explain global select * from mytable;
explain
-------------------------------
-------------------------------------------------------------------------------
Execution Block Graph (TERMINAL - eb_0000000000000_0000_000002)
-------------------------------------------------------------------------------
|-eb_0000000000000_0000_000002
   |-eb_0000000000000_0000_000001
-------------------------------------------------------------------------------
Order of Execution
-------------------------------------------------------------------------------
1: eb_0000000000000_0000_000001
2: eb_0000000000000_0000_000002
-------------------------------------------------------------------------------

=======================================================
Block Id: eb_0000000000000_0000_000001 [ROOT]
=======================================================

SCAN(0) on sampledb.mytable
  => target list: sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)
  => out schema: {(5) sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)}
  => in schema: {(5) sampledb.mytable.id (INT4), sampledb.mytable.name (TEXT), sampledb.mytable.address (TEXT), sampledb.mytable.age (INT4), sampledb.mytable.mark (INT4)}

=======================================================
Block Id: eb_0000000000000_0000_000002 [TERMINAL]
=======================================================
(24 rows, 0.029 sec, 0 B selected)
sampledb>
```

Here, Global plan shows execution block ID, order of execution and its information.

Joins

SQL joins are used to combine rows from two or more tables. The following are the different types of SQL Joins:

- Inner join
- { LEFT | RIGHT | FULL } OUTER JOIN
- Cross join
- Self join
- Natural join

Consider the following two tables to perform joins operations.

Table1 − Customers

Id	Name	Address	Age
1	Customer 1	23 Old Street	21
2	Customer 2	12 New Street	23
3	Customer 3	10 Express Avenue	22
4	Customer 4	15 Express Avenue	22
5	Customer 5	20 Garden Street	33
6	Customer 6	21 North Street	25

Table2 − customer_order

Id	Order Id	Emp Id
1	1	101
2	2	102
3	3	103
4	4	104
5	5	105

Let us now proceed and perform the SQL joins operations on the above two tables.

Inner Join

The Inner join selects all rows from both the tables when there is a match between the columns in both tables.

Syntax

```
SELECT column_name(s) FROM table1 INNER JOIN table2 ON
table1.column_name=table2.column_name;
```

Query

```
default> select c.age,c1.empid from customers c inner join customer_order c1 on
c.id=c1.id;
```

Result

The above query will fetch the following result.

```
age,  empid

------------------------------

21,  101

23,  102

22,  103

22,  104

33,  105
```

The query matches five rows from both the tables. Hence, it returns the matched rows age from the first table.

Left Outer Join

A left outer join retains all of the rows of the “left” table, regardless of whether there is a row that matches on the “right” table or not.

Query

```
 select c.name,c1.empid from customers c left outer join customer_order c1 on
c.id=c1.id;
```

Result

The above query generates the following result.

```
name,          empid

------------------------------

customer1,  101

customer2,  102

customer3,  103

customer4,  104

customer5,  105

customer6,
```

Here, the left outer join returns name column rows from the customers(left) table and empid column matched rows from the customer_order(right) table.

Right Outer Join

A right outer join retains all of the rows of the “right” table, regardless of whether there is a row that matches on the “left” table.

Query

```
select c.name,c1.empid from customers c right outer join customer_order c1 on
c.id=c1.id;
```

Result

The above query will generate the following result.

```
name,      empid
------------------------------
customer1,  101
customer2,  102
customer3,  103
customer4,  104
customer5,  105
```

Here, the Right Outer Join returns the empid rows from the customer_order(right) table and the name column matched rows from customers table.

Full Outer Join

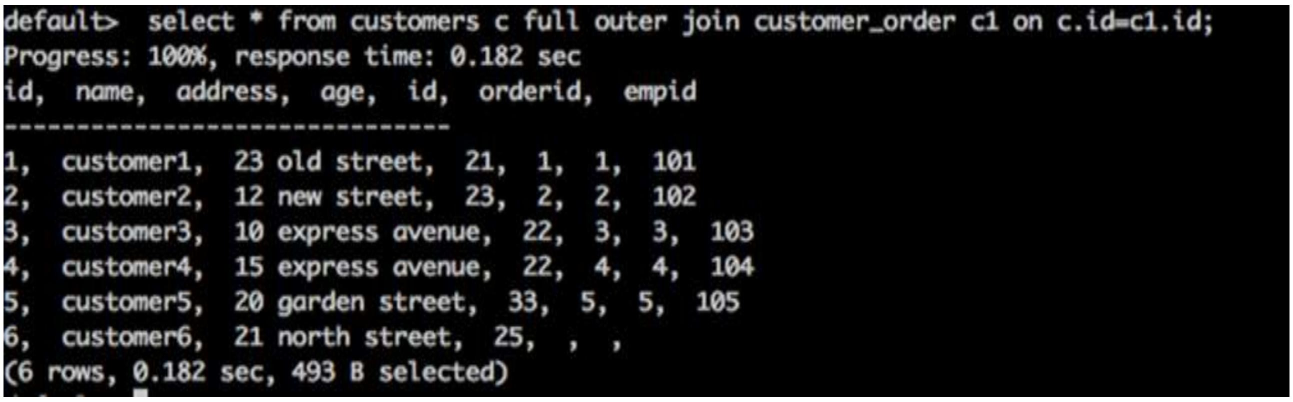
The Full Outer Join retains all rows from both the left and the right table.

Query

```
 select * from customers c full outer join customer_order c1 on c.id=c1.id;
```

Result

The above query will generate the following result.



```
default>  select * from customers c full outer join customer_order c1 on c.id=c1.id;
Progress: 100%, response time: 0.182 sec
id,  name,  address,  age,  id,  orderid,  empid
-------------------------------
1,  customer1,  23 old street,  21,  1,  1,  101
2,  customer2,  12 new street,  23,  2,  2,  102
3,  customer3,  10 express avenue,  22,  3,  3,  103
4,  customer4,  15 express avenue,  22,  4,  4,  104
5,  customer5,  20 garden street,  33,  5,  5,  105
6,  customer6,  21 north street,  25,  ,  ,
(6 rows, 0.182 sec, 493 B selected)
```

The query returns all the matching and non-matching rows from both the customers and the customer_order tables.

Cross Join

This returns the Cartesian product of the sets of records from the two or more joined tables.

Syntax

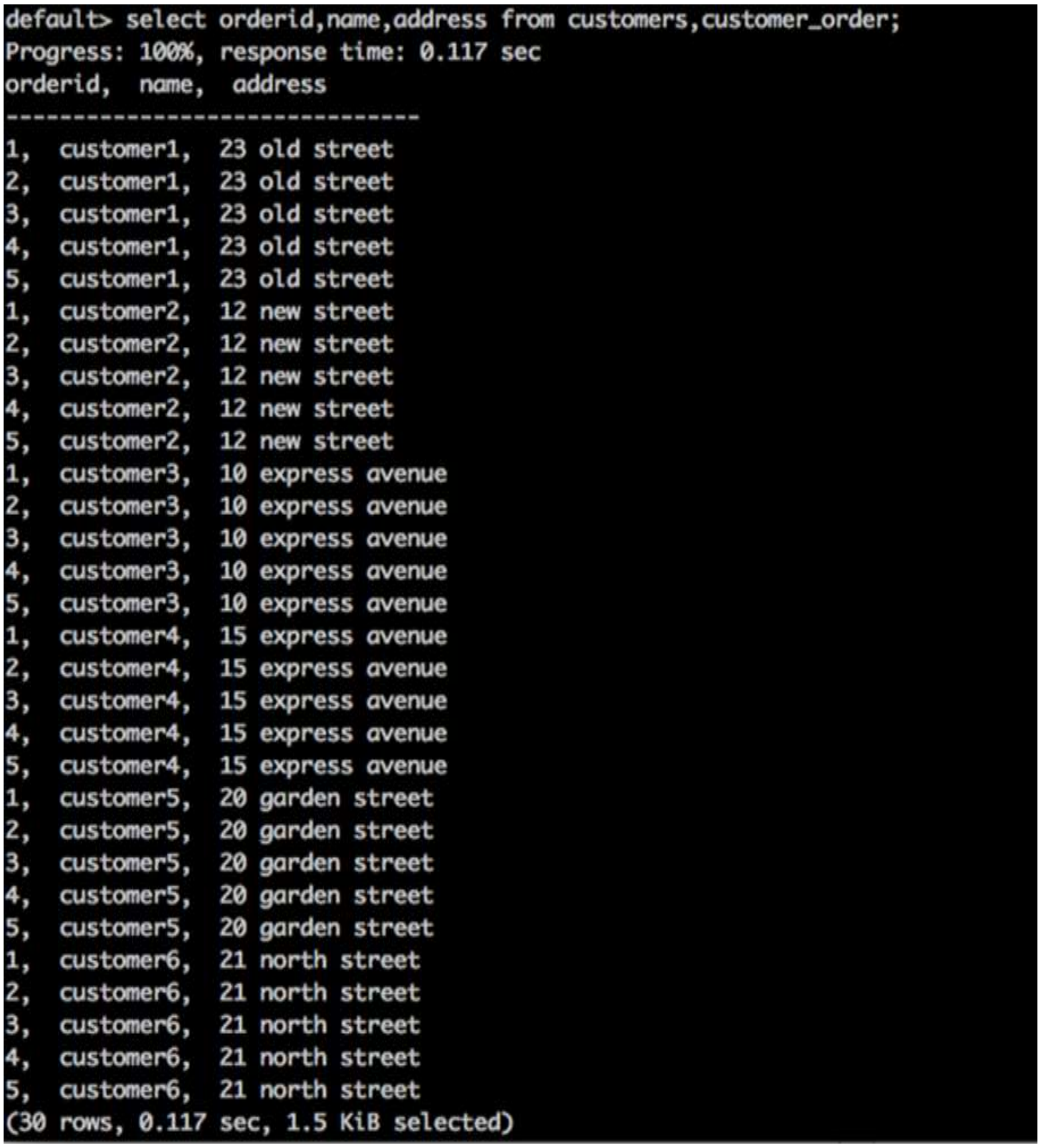
```
SELECT *  FROM table1  CROSS JOIN table2;
```

Query

```
select orderid,name,address from customers,customer_order;
```

Result

The above query will generate the following result.



```
default> select orderid,name,address from customers,customer_order;
Progress: 100%, response time: 0.117 sec
orderid,  name,  address
-------------------------------
1,  customer1,  23 old street
2,  customer1,  23 old street
3,  customer1,  23 old street
4,  customer1,  23 old street
5,  customer1,  23 old street
1,  customer2,  12 new street
2,  customer2,  12 new street
3,  customer2,  12 new street
4,  customer2,  12 new street
5,  customer2,  12 new street
1,  customer3,  10 express avenue
2,  customer3,  10 express avenue
3,  customer3,  10 express avenue
4,  customer3,  10 express avenue
5,  customer3,  10 express avenue
1,  customer4,  15 express avenue
2,  customer4,  15 express avenue
3,  customer4,  15 express avenue
4,  customer4,  15 express avenue
5,  customer4,  15 express avenue
1,  customer5,  20 garden street
2,  customer5,  20 garden street
3,  customer5,  20 garden street
4,  customer5,  20 garden street
5,  customer5,  20 garden street
1,  customer6,  21 north street
2,  customer6,  21 north street
3,  customer6,  21 north street
4,  customer6,  21 north street
5,  customer6,  21 north street
(30 rows, 0.117 sec, 1.5 KiB selected)
```

The above query returns the Cartesian product of the table.

Natural Join

A Natural Join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between the two relations.

Syntax

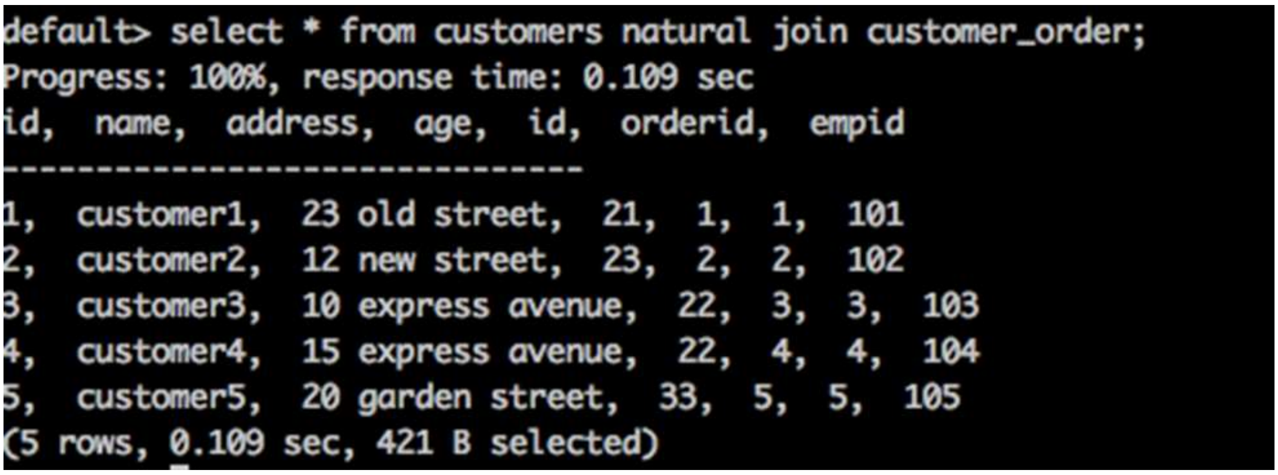
```
SELECT * FROM table1 NATURAL JOIN table2;
```

Query

```
select * from customers natural join customer_order;
```

Result

The above query will generate the following result:



```
default> select * from customers natural join customer_order;
Progress: 100%, response time: 0.109 sec
id,  name,  address,  age,  id,  orderid,  empid
-------------------------------
1,  customer1,  23 old street,  21,  1,  1,  101
2,  customer2,  12 new street,  23,  2,  2,  102
3,  customer3,  10 express avenue,  22,  3,  3,  103
4,  customer4,  15 express avenue,  22,  4,  4,  104
5,  customer5,  20 garden street,  33,  5,  5,  105
(5 rows, 0.109 sec, 421 B selected)
```

Here, there is one common column id that exists between two tables. Using that common column, the **Natural Join** joins both the tables.

Self Join

The SQL SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field
```

Query

```
default> select c.id,c1.name from customers c, customers c1 where c.id=c1.id;
```

Result

The above query will fetch the following result.

```
id,   name
-------------------------------
1,   customer1
2,   customer2
3,   customer3
4,   customer4
5,   customer5
6,   customer6
```

The query joins a customer table to itself.

18. Apache Tajo — Storage Plugins

Tajo supports various storage formats. To register storage plugin configuration, you should add the changes to the configuration file “storage-site.json”.

storage-site.json

The structure is defined as follows:

```
{
  "storages": {
    “storage plugin name“: {
      "handler": "${class name}”,
      "default-format": “plugin name"
    }
  }
}
```

Each storage instance is identified by URI.

PostgreSQL Storage Handler

Tajo supports PostgreSQL storage handler. It enables user queries to access database objects in PostgreSQL. It is the default storage handler in Tajo so you can easily configure it.

configuration

```
{
  "spaces": {

    "postgre": {

      "uri": "jdbc:postgresql://hostname:port/database1"

      "configs": {

        "mapped_database": “sampledb”

        "connection_properties": {

          "user":     “tajo",


          "password": "pwd"
        }
      }
    }
  }
}
```

Here, **“database1”** refers to the **postgreSQL** database which is mapped to the database **“sampledb**” in Tajo.

19. Apache Tajo — Integration with HBase

Apache Tajo supports HBase integration. This enables us to access HBase tables in Tajo. HBase is a distributed column-oriented database built on top of the Hadoop file system. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System. The following steps are required to configure HBase integration.

Set Environment Variable

Add the following changes to "conf/tajo-env.sh" file.

```
$ vi conf/tajo-env.sh


# HBase home directory. It is opitional but is required mandatorily to use HBase.
# export HBASE_HOME= path/to/HBase
```

After you have included the HBase path, Tajo will set the HBase library file to the classpath.

Create an External Table

Create an external table using the following syntax:

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] <table_name> [(<column_name> <data_type>, ...
)]
USING hbase WITH ('table'='<hbase_table_name>'
, 'columns'=':key,<column_family_name>:<qualifier_name>, ...'
, 'hbase.zookeeper.quorum'='<zookeeper_address>'
, 'hbase.zookeeper.property.clientPort'='<zookeeper_client_port>')
[LOCATION 'hbase:zk://<hostname>:<port>/'] ;
```

To access HBase tables, you must configure the tablespace location.

Here,

- **Table** : Set hbase origin table name. If you want to create an external table, the table must exists on HBase.

- **Columns** : Key refers to the HBase row key. The number of columns entry need to be equal to the number of Tajo table columns.

- **hbase.zookeeper.quorum** : Set zookeeper quorum address.

- **hbase.zookeeper.property.clientPort** : Set zookeeper client port.

Query

```
CREATE EXTERNAL TABLE students (rowkey text,id int,name text)
USING hbase WITH (
  'table'='students'
  , 'columns'=':key,info:id,content:name')
LOCATION 'hbase:zk://<hostname>:<port>/';
```

Here, the Location path field sets the zookeeper client port id. If you don’t set the port, Tajo will refer the property of hbase-site.xml file.

Create Table in HBase

You can start the HBase interactive shell using the “hbase shell” command as shown in the following query.

Query

```
/bin/hbase shell
```

Result

The above query will generate the following result.

```
hbase(main):001:0>
```

Steps to Query HBase

To query HBase, you should complete the following steps:

Step 1: Pipe the following commands to the HBase shell to create a “tutorial” table.

Query

```
hbase(main):001:0> create ‘students’,{NAME=>’info’},{NAME=>’content’}
put 'students', ‘row-01', 'content:name', 'Adam'
put 'students', ‘row-01', 'info:id', '001'
put 'students', ‘row-02', 'content:name', 'Amit'
put 'students', ‘row-02', 'info:id', '002'
put 'students', ‘row-03', 'content:name', 'Bob'
put 'students', ‘row-03', 'info:id', ‘003'
```

Step 2: Now, issue the following command in hbase shell to load the data into a table.

```
main):001:0> cat ../hbase/hbase-students.txt | bin/hbase shell
```

Step 3: Now, return to the Tajo shell and execute the following command to view the metadata of the table:

```
default> \d students;


table name: default.students
table path:
store type: HBASE
number of rows: unknown
volume: 0 B
Options:
        'columns'=':key,info:id,content:name'
        'table'='students'


schema:
rowkey  TEXT
id  INT4
name TEXT
```

Step 4: To fetch the results from the table, use the following query:

Query

```
default> select * from students
```

Result

The above query will fetch the following result:

```
rowkey,  id,  name
-------------------------------
row-01,  001,  Adam
row-02,  002,  Amit
row-03   003,  Bob
```

20. Apache Tajo — Integration with Hive

Tajo supports the HiveCatalogStore to integrate with Apache Hive. This integration allows Tajo to access tables in Apache Hive.

Set Environment Variable

Add the following changes to "conf/tajo-env.sh" file.

```
$ vi conf/tajo-env.sh


export HIVE_HOME=/path/to/hive
```

After you have included the Hive path, Tajo will set the Hive library file to the classpath.

Catalog Configuration

Add the following changes to the "conf/catalog-site.xml" file.

```
$ vi conf/catalog-site.xml


<property>
  <name>tajo.catalog.store.class</name>
  <value>org.apache.tajo.catalog.store.HiveCatalogStore</value>
</property>
```

Once HiveCatalogStore is configured, you can access Hive's table in Tajo.

21. Apache Tajo — OpenStack Swift Integration

Swift is a distributed and consistent object/blob store. Swift offers cloud storage software so that you can store and retrieve lots of data with a simple API. Tajo supports Swift integration.

The following are the prerequisites of Swift Integration:

- Swift
- Hadoop

Core-site.xml

Add the following changes to the hadoop "core-site.xml" file:

```
<property>
  <name>fs.swift.impl</name>
  <value>org.apache.hadoop.fs.swift.snative.SwiftNativeFileSystem</value>
  <description>File system implementation for Swift</description>
</property>

<property>
  <name>fs.swift.blocksize</name>
  <value>131072</value>
  <description>Split size in KB</description>
</property>
```

This will be used for Hadoop to access the Swift objects. After you made all the changes move to the Tajo directory to set Swift environment variable.

conf/tajo-env.h

Open the Tajo configuration file and add set the environment variable as follows:

```
$ vi conf/tajo-env.h

export TAJO_CLASSPATH=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-openstack-x.x.x.jar
```

Now, Tajo will be able to query the data using Swift.

Create Table

Let’s create an external table to access Swift objects in Tajo as follows:

```
default> create external table swift(num1 int, num2 text, num3 float) using text with
('text.delimiter'='|') location 'swift://bucket-name/table1';
```

After the table has been created, you can run the SQL queries.

22. Apache Tajo — JDBC Interface

Apache Tajo provides JDBC interface to connect and execute queries. We can use the same JDBC interface to connect Tajo from our Java based application. Let us now understand how to connect Tajo and execute the commands in our sample Java application using JDBC interface in this section.

Download JDBC Driver

Download the JDBC driver by visiting the following link:

http://apache.org/dyn/closer.cgi/tajo/tajo-0.11.3/tajo-jdbc-0.11.3.jar

Now, "tajo-jdbc-0.11.3.jar" file has been downloaded on your machine.

Set Class Path

To make use of the JDBC driver in your program, set the class path as follows:

```
CLASSPATH=path/to/tajo-jdbc-0.11.3.jar:$CLASSPATH
```

Connect to Tajo

Apache Tajo provides a JDBC driver as a single jar file and it is available **@ /path/to/tajo/share/jdbc-dist/tajo-jdbc-0.11.3.jar**.

The connection string to connect the Apache Tajo is of the following format:

```
jdbc:tajo://host/

jdbc:tajo://host/database

jdbc:tajo://host:port/

jdbc:tajo://host:port/database
```

Here,

- **host** - The hostname of the TajoMaster.
- **port** - The port number that server is listening. Default port number is 26002.
- **database** - The database name. The default database name is default.

Java Application

Let us now understand Java application.

Coding

```
import java.sql.*;
import org.apache.tajo.jdbc.TajoDriver;

public class TajoJdbcSample {

  public static void main(String[] args) {

    Connection connection = null;
    Statement statement = null;

    try {
      Class.forName("org.apache.tajo.jdbc.TajoDriver");

      connection = DriverManager.getConnection(“jdbc:tajo://localhost/default");



      statement = connection.createStatement();
      String sql;

      sql = "select * from mytable”;
      // fetch records from mytable.

      ResultSet resultSet = statement.executeQuery(sql);

      while(resultSet.next()){

        int id  = resultSet.getInt("id");
        String name = resultSet.getString("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();
    }
  }
}
```

The application can be compiled and run using the following commands.

Compilation

```
javac -cp /path/to/tajo-jdbc-0.11.3.jar:. TajoJdbcSample.java
```

Execution

```
java -cp /path/to/tajo-jdbc-0.11.3.jar:. TajoJdbcSample
```

Result

The above commands will generate the following result:

```
ID: 1;
Name: Adam

ID: 2;
Name: Amit

ID: 3;
Name: Bob

ID: 4;
Name: David

ID: 5;
Name: Esha

ID: 6;
Name: Ganga
```

```

ID: 7;
Name: Jack

ID: 8;
Name: Leena

ID: 9;
Name: Mary

ID: 10;
Name: Peter
```

23. Apache Tajo — Custom Functions

Apache Tajo supports the custom / user defined functions (UDFs). The custom functions can be created in python.

The custom functions are just plain python functions with decorator **“@output_type(<tajo sql datatype>)”** as follows:

```
@ouput_type(“integer”)
def sum_py(a, b):
    return a + b;
```

The python scripts with UDFs can be registered by adding the below configuration in **“tajo-site.xml”**.

```
<property>
  <name>tajo.function.python.code-dir</name>
  <value>file:///path/to/script1.py,file:///path/to/script2.py</value>
</property>
```

Once the scripts are registered, restart the cluster and the UDFs will be available right in the SQL query as follows:

```
select sum_py(10, 10) as pyfn;
```

Apache Tajo supports user defined aggregate functions as well but does not support user defined window functions.