LAB 9

Aim: Install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

Theory:

Apache Hive is a data warehouse infrastructure that facilitates querying and managing large data sets which resides in distributed storage system. It is built on top of Hadoop and developed by Facebook. Hive provides a way to query the data using a SQL-like query language called HiveQL(Hive query Language). Internally, a compiler translates HiveQL statements into MapReduce jobs, which are then submitted to Hadoop framework for execution. Hive looks very much similar like traditional database with SQL access. However, because Hive is based on Hadoop and MapReduce operations, there are several key differences: As Hadoop is intended for long sequential scans and Hive is based on Hadoop, you would expect queries to have a very high latency. It means that Hive would not be appropriate for those applications that need very fast response times, as you can expect with a traditional RDBMS database.

Procedure:

Prerequisites:

1. Java
2. Hadoop

1. Download the Hive Files from Apache.

2. Extract the files to a convenient location. (/usr/local).

3. Edit the system variable to include the Pig files.

4. Check Pig version to check if its working properly.

5. Create Hive directories within HDFS and give them read/write permissions. The directory 'warehouse' is the location to store the table or data related to hive.

6. Set Hadoop path in hive-env.sh

7. Edit the hive-site.xml file.

```
<configuration>
    <!-- MySQL Connection URL -->
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost:3306/metastore?createDatabaseIfNotExist=true
</value>
```

```xml
        <description>JDBC connection URL for the MySQL metastore database.</description>
    </property>

    <!-- MySQL Driver -->
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.cj.jdbc.Driver</value>
        <description>Driver class for MySQL database.</description>
    </property>

    <!-- Database User -->
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
        <description>Username for the MySQL database.</description>
    </property>

    <!-- Database Password -->
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>password</value>
        <description>Password for the MySQL database.</description>
    </property>

    <!-- Default Warehouse Directory -->
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
        <description>Default directory for tables created in Hive.</description>
    </property>

    <!-- Metastore Thrift Service Port -->
    <property>
        <name>hive.metastore.uris</name>
        <value>thrift://localhost:9083</value>
        <description>Thrift URI for the metastore service.</description>
    </property>
</configuration>
```

8. By default, Hive uses Derby database. Initialize Derby database using: bin/schematool -initSchema -dbType derby

9. Launch Hive

Hive Operations:

```
-- Create a database
hive> CREATE DATABASE sample_db;
OK
Database sample_db created successfully.

-- Use the created database
hive> USE sample_db;
OK
Database changed to sample_db.

-- Create a table
hive> CREATE TABLE employee (
    id INT,
    name STRING,
    age INT,
    salary FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
OK
Table employee created successfully.

-- Insert sample data into the table
hive> INSERT INTO TABLE employee VALUES (1, 'John Doe', 30, 50000),
(2, 'Jane Smith', 25, 60000);
OK
2 rows inserted into employee.

-- Alter the table by adding a new column
hive> ALTER TABLE employee ADD COLUMNS (department STRING);
OK
Table employee modified successfully.

-- Rename the table
hive> ALTER TABLE employee RENAME TO staff;
OK
Table employee renamed to staff.

-- Create a view based on the table
hive> CREATE VIEW senior_employees AS
SELECT name, age FROM staff WHERE age > 28;
OK
View senior_employees created successfully.

-- Drop the view
hive> DROP VIEW senior_employees;
OK
View senior_employees dropped successfully.

-- Add a custom function (UDF) from a JAR (replace with your UDF JAR
file path)
hive> ADD JAR /path/to/custom-udf.jar;
```

```
OK

-- Create the UDF in Hive
hive> CREATE FUNCTION to_upper AS 'com.example.udf.ToUpper';
OK
Function to_upper created successfully.

-- Drop the function
hive> DROP FUNCTION to_upper;
OK
Function to_upper dropped successfully.

-- Create an index on the salary column
hive> CREATE INDEX idx_salary
ON TABLE staff (salary)
AS 'COMPACT'
WITH DEFERRED REBUILD;
OK
Index idx_salary created successfully.

-- Drop the index
hive> DROP INDEX idx_salary ON staff;
OK
Index idx_salary dropped successfully.

-- Drop the table
hive> DROP TABLE staff;
OK
Table staff dropped successfully.

-- Drop the database with CASCADE to remove all associated objects
hive> DROP DATABASE sample_db CASCADE;
OK
Database sample_db dropped successfully.

```

Conclusion:

In this experiment we learnt about the Hive and its installation process and also we learn to implement HiveQL by creating, altering and deleting the database and table.