Big Data Technologies

Agenda

- Hive

Q & A

- Schema-on-Read
- Hive OLAP
- Derby is single user database
- Hive-on-MR is deprecated.

Hive Execution Engine

- Hive supports multiple execution engine.
- Execution engine can be set in hive-site.xml or using SET command.
    - beeline> set hive.execution.engine=mr;
- Hadoop (default) -- deprecated
    - Hive QL --> Hive --> MR job --> Hadoop cluster
- Tez
    - Tez is another big data/distributed processing engine
    - Hive QL --> Hive --> Tez job --> Tez cluster
- Spark
    - Spark is one more distributed computing framework
    - Hive QL --> Hive --> Spark job --> Spark cluster

Serde

- SerDe is short for Serializer/Deserializer.
- Hive uses the SerDe interface for IO.

- Built-in SerDes
  - Avro (Hive 0.9.1 and later)
  - ORC (Hive 0.11 and later)
  - RegEx
  - Thrift
  - Parquet (Hive 0.13 and later)
  - CSV (Hive 0.14 and later)
  - JsonSerDe (Hive 0.12 and later in hcatalog-core)

Hive DML operations

- Hive is designed for OLAP (Analysis).
- However later versions (0.14+) added support for Update and Delete operation.
- DML operations are carried out by MR job and hence need considerable time.
- DML is possible in Hive tables, if following conditions are fulfilled.
  - In hive-site.xml, set transaction manager.
  - Table must be in ORC format.
  - Table should be bucketed (recommended).
  - Table must be transactional.

```
CREATE TABLE ncdc_orc(
yr SMALLINT,
temp INT,
quality TINYINT
)
STORED AS ORC
TBLPROPERTIES('transactional'='true');

INSERT INTO ncdc_orc(yr, temp, quality)
SELECT yr, temp, quality FROM ncdc_staging;

SELECT * FROM ncdc_orc
LIMIT 10;
```

```
-- update and delete operations are possible on this table.
SELECT * FROM ncdc_orc
WHERE temp = 9999;

DELETE FROM ncdc_orc
WHERE temp = 9999;
```

Hive Views

- In general, types of views
  - Simple view -- can also perform DML (in MySQL)
  - Complex view -- cannot perform DML
  - Inline view -- SELECT in projection
    - SELECT empno, ename, sal, (SELECT SUM(sal) FROM emp) total FROM emp;
  - Materialized view -- store result of view query.

Materialized View

- Can be created only on transactional tables.

```
CREATE MATERIALIZED VIEW mv_ncdc_avgtemp AS
SELECT yr, AVG(temp) AS avgtemp FROM ncdc_orc
WHERE temp != 9999 AND quality IN (0,1,2,4,5,9)
GROUP BY yr;

SELECT * FROM mv_ncdc_avgtemp
LIMIT 20;

(SELECT 'Coolest' AS category, yr, avgtemp FROM mv_ncdc_avgtemp
ORDER BY avgtemp ASC
LIMIT 1)
UNION
```

```
(SELECT 'Hotest' AS category, yr, avgtemp FROM mv_ncdc_avgtemp
ORDER BY avgtemp DESC
LIMIT 1);

SHOW MATERIALIZED VIEWS;

SHOW VIEWS;

SHOW TABLES;

SELECT * FROM ncdc_orc
WHERE quality = 2;

DELETE FROM ncdc_orc
WHERE quality = 2;

-- rebuild view i.e. reexecute view query and store results once again
ALTER MATERIALIZED VIEW mv_ncdc_avgtemp REBUILD;

SELECT * FROM mv_ncdc_avgtemp
LIMIT 20;

(SELECT 'Coolest' AS category, yr, avgtemp FROM mv_ncdc_avgtemp
ORDER BY avgtemp ASC
LIMIT 1)
UNION
(SELECT 'Hotest' AS category, yr, avgtemp FROM mv_ncdc_avgtemp
ORDER BY avgtemp DESC
LIMIT 1);

DROP MATERIALIZED VIEW mv_ncdc_avgtemp;
```


Assignment

- Create table "books_staging" and load books.csv in it.

- Create table "books_orc" as transactional table.
- Create a materialized view for summary -- Subjectwise average book price.
- Display a report that shows subject and average price in descending order -- on materialized view.
- Create a new file newbooks.csv.

```
20,Atlas Shrugged,Ayn Rand,Novel,723.90
21,The Fountainhead,Ayn Rand,Novel,923.80
22,The Archer,Paulo Cohelo,Novel,623.94
23,The Alchemist,Paulo Cohelo,Novel,634.80
```

- Upload the file into books_staging.
- Insert "new" records from books_staging into books_orc.
- Display a report that shows subject and average price in descending order -- on materialized view. -- Are new books visible in report?
- Rebuild the materialized view.
- Display a report that shows subject and average price in descending order -- on materialized view. -- Are new books visible in report?
- Increase price of all Java books by 10% in books_orc.
- Rebuild the materialized view.
- Display a report that shows subject and average price in descending order -- on materialized view. -- Are new price changes visible in report?
- Delete all Java books.
- Rebuild the materialized view.
- Display a report that shows subject and average price in descending order -- on materialized view. -- Are new price changes visible in report?

Hive Joins

```
CREATE TABLE ratings_staging(
userid INT,
movieid INT,
rating DOUBLE,
rtime BIGINT
)
ROW FORMAT DELIMITED
```

```
FIELDS TERMINATED BY ','
TBLPROPERTIES('skip.header.line.count'='1');

LOAD DATA LOCAL
INPATH '/tmp/movies/ratings.csv'
INTO TABLE ratings_staging;

SELECT * FROM  ratings_staging
LIMIT 10;

-- display top 10 movies (number of ratings)
SELECT movieid, COUNT(rating) FROM ratings_staging
GROUP BY movieid
ORDER BY 2 DESC
LIMIT 10;

SELECT m.title, COUNT(r.rating) cnt FROM
movies_staging m LEFT JOIN
ratings_staging r ON m.id = r.movieid
GROUP BY m.title
ORDER BY 2 DESC
LIMIT 10;

SELECT m.title, COUNT(r.rating) cnt FROM
movies_staging m LEFT JOIN
ratings_staging r ON m.id = r.movieid
GROUP BY m.title;

EXPLAIN
SELECT m.title, COUNT(r.rating) cnt FROM
movies_staging m LEFT JOIN
ratings_staging r ON m.id = r.movieid
GROUP BY m.title;

SET hive.auto.convert.join=false;
```

```
SELECT m.title, COUNT(r.rating) cnt FROM
movies_staging m LEFT JOIN
ratings_staging r ON m.id = r.movieid
GROUP BY m.title;

EXPLAIN
SELECT m.title, COUNT(r.rating) cnt FROM
movies_staging m LEFT JOIN
ratings_staging r ON m.id = r.movieid
GROUP BY m.title;

SET hive.auto.convert.join=true;
```

Movie Recommendation

```
CREATE TABLE ratings(
userid INT,
movieid INT,
rating DOUBLE,
rtime BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
TBLPROPERTIES('skip.header.line.count'='1');

LOAD DATA LOCAL
INPATH '/home/nilesh/sep22/dbda/bigdata/day09/ratings.csv'
INTO TABLE ratings;

SELECT * FROM ratings
LIMIT 10;

CREATE TABLE user_moview AS
SELECT r1.movieid m1, r2.movieid m2, r1.rating rt1, r2.rating rt2 FROM ratings r1
```

```
INNER JOIN ratings r2 ON r1.userid = r2.userid
WHERE r1.movieid < r2.movieid;

SELECT * FROM user_moview;

SELECT m1, m2, COUNT(rt1) cnt, CORR(rt1,rt2) cor
FROM user_moview
GROUP BY m1, m2;

DROP TABLE ratings;
```

```
SHOW TABLES;

CREATE TABLE movies_orc(
id INT,
title STRING,
genres STRING
)
STORED AS ORC
TBLPROPERTIES('transactional'='true');

INSERT INTO movies_orc
SELECT * FROM movies_staging;

CREATE TABLE ratings_orc(
userid INT,
movieid INT,
rating DOUBLE,
rtime BIGINT
)
STORED AS ORC
TBLPROPERTIES('transactional'='true');

INSERT INTO ratings_orc
```

```
SELECT * FROM ratings_staging;

SELECT * FROM movies_orc
LIMIT 15;

SELECT * FROM ratings_orc
LIMIT 15;

CREATE MATERIALIZED VIEW mv_user_movies AS
SELECT r1.movieid m1, r2.movieid m2, r1.rating rt1, r2.rating rt2 FROM ratings_orc r1
INNER JOIN ratings_orc r2 ON r1.userid = r2.userid
WHERE r1.movieid < r2.movieid;

SELECT * FROM mv_user_movies
LIMIT 5;

CREATE TABLE movies_corr AS
SELECT m1, m2, COUNT(rt1) cnt, CORR(rt1,rt2) cor
FROM mv_user_movies
GROUP BY m1, m2;

SELECT * FROM movies_corr
WHERE cnt > 50
ORDER BY cor DESC
LIMIT 20;

SELECT * FROM movies_orc WHERE id IN (858,1221);

-- Assignment
-- show top 5 recommended movies for given movie id.
SELECT m.id, m.title, mc.cnt, mc.cor FROM movies_orc m
INNER JOIN movies_corr mc ON (mc.m1 = m.id OR mc.m2 = m.id)
WHERE mc.cnt > 50 AND (mc.m1 = 260 OR mc.m2 = 260) AND (m.id != 260)
ORDER BY mc.cor DESC
LIMIT 10;
```

Hive JDBC connectivity

- Hive can be connected from Java using JDBC.
- Java App --> JDBC Driver --> RDBMS
- JDBC Driver is a component that converts Java requests into RDBMS understandable form and RDBMS response into Java understandable form.
- JDBC Driver are typically packaged as "JAR" -- set of classes inherited from JDBC interfaces.
- JDBC interfaces/objects:
  - Driver -- responsible for making database connection (socket)
  - Connection -- wrapper on socket connection and communicate with database.
  - Statement -- execute SQL query on server and get the result.
  - ResultSet -- represent result from SELECT query and access it row by row.
- JDBC steps
  - step 1: add JDBC driver into classpath. For maven project, add dependency in pom.xml.
  - step 2: load and register driver.
  - step 3: create connection object using DriverManager.
  - step 4: create statement object using Connection.
  - step 5: execute statement object and process the result.
  - step 6: close all (statement, connection).

Hive JDBC steps

- step 1: add JDBC driver into classpath.
  - Maven Project: Add hive-jdbc dependency.

```
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-jdbc</artifactId>
    <version>3.1.2</version>
</dependency>
```

- step 2: load and register driver (one time - static block).

```
String DB_DRIVER = "org.apache.hive.jdbc.HiveDriver";
Class.forName(DB_DRIVER);
```

- step 3: create connection object using DriverManager.

```
String DB_URL = "jdbc:hive2://localhost:10000/dbda";
String DB_USER = "nilesh";
String DB_PASSWORD = " ";
Connection con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
```

- step 4: create statement object using Connection.

```
String sql = "SELECT * FROM movies LIMIT 10";
PreparedStatement stmt = con.prepareStatement(sql);
```

- step 5: execute statement object (executeQuery() for SELECT or executeUpdate() for non-SELECT) and process the result.

```
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
    int id = rs.getInt("id");
    String title = rs.getString("title");
    String generes = rs.getString("genres");
    // ...
}
```

- step 6: close all (statement, connection) OR use try-with-resource.

```
rs.close();
stmt.close();
con.close();
```

- Refer demo code.
- If "Maven Project" is not working, create "Java Project" and add "$HIVE_HOME/jdbc/hive-jdbc-3.1.2-standalone.jar" as External jar.

SQL Injection

- Input user id and pin from user and login into the system.

```
String id = sc.nextLine();  // 1
String pin = sc.nextLine(); // 1234
String sql = "SELECT * FROM users WHERE id="+id+" AND pin=" + pin;
    // "SELECT * FROM users WHERE id=1 AND pin=1234";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
if(rs.next())
    // login into the system
else
    // invalid credentials
```

- If user inputs a malicious string like "1234 OR 10=10", then the resultant query will be compromised like "SELECT * FROM users WHERE id=1 AND pin=1234 OR 10=10". This is called as "SQL Injection".
- To avoid this never create queries by String concatenation.
- In Java, it is recommended to use PrepararedStatement.

```
String id = sc.nextLine();  // 1
String pin = sc.nextLine(); // 1234
String sql = "SELECT * FROM users WHERE id=? AND pin=?"
```

```
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setInt(1, Integer.parseInt(id));
stmt.setInt(2, Integer.parseInt(pin));
ResultSet rs = stmt.executeQuery();
if(rs.next())
    // login into the system
else
    // invalid credentials
```

Hive Python connectivity

- Refer demo code.
- terminal> sudo apt install python3-dev libsasl2-dev python3-pip libsasl2-modules
- terminal> pip install thrift sasl thrift_sasl
- terminal> pip install pyhive
- terminal> python /path/of/hive-python1.py
- Assignment: Implement movie recommendation in python + hive.