Big Data Technologies

Agenda

- Apache Hive

Apache Hive

Hive Architecture

- HiveServer2: Accept queries and execute them
  - ThriftService = Accept queries (port=10000)
  - Hive Driver = Parser + Planner + Optimizer
  - Execution Engine
- Hive Execution engine
  - MR or Tez or Spark
  - Hive 2.x -- Map-Reduce is deprecated.
- Hive Metastore
  - Store table structure & other metadata
    - Database names
    - Table names
    - Column names and Data types
    - Views
    - etc
  - Embedded mode
    - Easy to configure
    - Derby is single user RDBMS and by default available with Java (JDK)
    - e.g. hive-site.xml --> javax.jdo.option.ConnectionURL=jdbc:derby:;databaseName=/home/nilesh/setup/bigdata/apache-hive-3.1.2-bin/metastore_db;create=true
  - Local/Remote mode

        - Can be accessed from local or remote machine
        - Usually MySQL or PostgreSQL RDBMS is preferred.
- Hive Clients
    - Hive CLI - Deprecated (because no security)
    - Hive Beeline - Client application that connects to HiveServer2 using JDBC.
    - Java/Python programs
        - Java programs --> JDBC --> HiveServer2
        - Python programs --> Thrift --> HiveServer2

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
    - SET hive.execution.engine=mr;
    - Hive QL --> Hive --> MR job --> Hadoop cluster
- Tez
    - SET hive.execution.engine=tez;
    - Tez is another big data/distributed processing engine
    - Hive QL --> Hive --> Tez job --> Tez cluster
- Spark
    - SET hive.execution.engine=spark;
    - Spark is one more distributed computing framework
    - Hive QL --> Hive --> Spark job --> Spark cluster

Managed Table vs External Tables

- Managed Table
    - CREATE TABLE statement
    - Located in HDFS warehouse directory
    - Loading data explicitly into the table (HDFS) after table creation.

    - Drop table operation drop table data (from HDFS) as well as table structure (from metastore).
- External Table
    - CREATE EXTERNAL TABLE statement.
    - Located in HDFS directory -- LOCATION '/hdfs/dir/path';
    - Drop table operation drop only table structure (from metastore). Data in HDFS is intact.
    - When data is already present in HDFS and need to process it using HiveQL.
    - Multiple tables (metadata) can refer to the same data files in HDFS.
    - DML operations and transactional properties not supported.

Hive Functions

- Scalar functions / Single row functions
    - input n rows ---> output n rows
    - e.g. SIZE(), ARRAY_CONTAINS(), CONCAT(), FROM_UNIXTIME(), YEAR(), ABS(), etc.
- Group functions / Aggergate functions / Multi row functions
    - input n rows ---> output m rows (where m <= n)
    - e.g. SUM(), AVG(), MAX(), MIN(), COUNT(), CORR(), STDDEV(), COLLECT_LIST(), etc.
- Table valued functions
    - input n rows ---> output m rows (where m >= n)
    - e.g. EXPLODE(array)
        - Input row=[1,2,3], Then EXPLODE() Output= row1=1, row2=2, row3=3
- Hive user-defined functions
    - UDF -- User Defined Function
    - UDAF -- User Defined Aggregate Function
    - UDTF -- User Defined Table Function
    - Implemented in Java/PL-SQL.
- https://cwiki.apache.org/confluence/display/Hive/LanguageManual

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.

```
<!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc -->
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-jdbc</artifactId>
    <version>3.1.2</version>
</dependency>
```

  - step 2: load and register driver.

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

  - step 3: create connection object using DriverManager.

```
Connection con = DriverManager.getConnection("jdbc:hive2://localhost:10000/classwork", "nilesh", "");
```

  - step 4: create statement object using Connection.

```

```

```
Statement stmt = con.createStatement();
```

- step 5: execute statement object and process the result.

```
String sql = "SELECT * FROM books_json";
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    String author = rs.getString("author");
    String subject = rs.getString("subject");
    double price = rs.getDouble("price");
    System.out.println(id + ", " + name + ", " + author + ", " + subject + ", " + price);
}
```

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

Hive Python connectivity

- Example code

```
# hive config
host_name = 'localhost'
port = 10000
user = 'nilesh'
password = ''
db_name = 'classwork'

# get hive connection
conn = hive.Connection(host=host_name, port=port, username=user, password=password, database=db_name,
auth='CUSTOM')
```

```
# get the cursor object
cur = conn.cursor()

# execute the sql query using cursor
sal = input('Enter minimum salary: ')
sql = "SELECT * FROM emp_staging WHERE sal > " + str(sal)
cur.execute(sql)

# collect/process result
result = cur.fetchall()
for row in result:
    print(row)

# close the connection
conn.close()
```

- terminal> sudo apt install python3-dev libsasl2-dev python3-pip libsasl2-modules
- terminal> pip3 install thrift sasl thrift_sasl
- terminal> pip3 install pyhive
- terminal> python /path/of/hive-python1.py

Assignment

1. Implement Movie recommendation system.
    - Example Input Data

```
userId,movieId,rating,rtime
17,70,3,0
35,21,1,0
49,19,2,0
49,21,1,0
49,70,4,0
```

```
87,19,1,0
87,21,2,0
98,19,2,0
```

- Create pairs of movies rated by same user.

```
userId,movie1,rating1,movie2,rating2
49,21,1.0,70,4.0
49,19,2.0,21,1.0
49,19,2.0,70,4.0
87,19,1.0,21,2.0
```

- Create correlation table.

```
movie1,movie2,cnt,cor
19,21,2,-1.0
19,70,1,0.0
21,70,1,0.0
```

- Predict Similar movies for given movie Id. Get the recommended movies titles from movies table.
- Hints
  - Start with above small data tables to test accuracy of the steps.
  - You will need to create new intermediate tables to store results of earlier queries.
  - For main data use ORC format to speed-up the queries.
  - You may need to change reducer tasks memory for quicker execution and avoid OutOfMemory errors.
    - SET mapreduce.reduce.memory.mb = 4096;
    - SET mapreduce.reduce.java.opts = -Xmx4096m;

2. Execute following queries on "emp" and "dept" dataset.
   1. Create table "emp_staging" and load data from emp.csv in it.

2. Create table "dept_staging" and load data from dept.csv in it.
3. Display dept name and number of emps in each dept.
4. Display emp name and his dept name.
5. Display all emps (name, job, deptno) with their manager (name, job, deptno), who are not in their department.
6. Display all manager names with list of all dept names (where they can work).
7. Display job-wise total salary along with total salary of all employees.
8. Display dept-wise total salary along with total salary of all employees.
9. Display per dept job-wise total salary along with total salary of all employees.
10. Display number of employees recruited per year in descending order of employee count.
11. Display unique job roles who gets commission.
12. Display dept name in which there is no employee (using sub-query).
13. Display emp-name, dept-name, salary, total salary of that dept (using sub-query).
14. Display all managers and presidents along with number of (immediate) subbordinates.

3. Execute following queries on "emp" and "dept" dataset using CTE.
    1. Find emp with max sal of each dept.
    2. Find avg of deptwise total sal.
    3. Compare (show side-by-side) sal of each emp with avg sal in his dept and avg sal for his job.
    4. Divide emps by category -- Poor < 1500, 1500 <= Middle <= 2500, Rich > 2500. Hint: CASE ... WHEN. Count emps for each category.
    5. Display emps with category (as above), empno, ename, sal and dname.
    6. Count number of emps in each dept for each category (as above).
4. Execute following queries for books.csv dataset.
    1. Create table "books_staging" and load books.csv in it.
    2. Create table "books_orc" as transactional table.
    3. Create a materialized view for summary -- Subjectwise average book price.
    4. Display a report that shows subject and average price in descending order -- on materialized view.
    5. 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
```

6. Upload the file newbooks.csv into books_staging.
7. Insert "new" records from books_staging into books_orc.
8. Display a report that shows subject and average price in descending order -- on materialized view. -- Are new books visible in report?
9. Rebuild the materialized view.
10. Display a report that shows subject and average price in descending order -- on materialized view. -- Are new books visible in report?
11. Increase price of all Java books by 10% in books_orc.
12. Rebuild the materialized view.
13. Display a report that shows subject and average price in descending order -- on materialized view. -- Are new price changes visible in report?
14. Delete all Java books.
15. Rebuild the materialized view.
16. Display a report that shows subject and average price in descending order -- on materialized view. -- Are new price changes visible in report?

5. Execute following queries for movies dataset.
    1. Upload movies data (movies_caret.csv) into HDFS directory (not in hive warehouse).
    2. Create external table movies1 with schema - `id INT, title STRING, genres STRING`.
        - Find number of 'Action' movies.
    3. Create external table movies2 with schema - `id INT, title STRING, genres ARRAY<STRING>`.
        - Find number of movies having single genre.
6. Upload busstops.json data into HDFS directory. Then create hive external table to fetch data using JsonSerDe.

```
{"_id":{"$oid":"5a0720b478597fc11004d951"},"stop":"Non-BRTS","code":"103B-D-
04","seq":4.0,"stage":1.0,"name":"Aranyeshwar Corner","location":{"type":"Point","coordinates":
[73.857675,18.486381]}}
```

```
location STRUCT<type:STRING, coordinates:ARRAY<DOUBLE>>
```

```
When column-name have special charatcters like _ or $, they should be encapsulated in `back-quotes`.
```