Author: Madhurima Rawat

Hive Database and Table Operations

This document outlines the steps taken to create databases, tables, load data, and execute queries in Hive, including the outputs of each command.

Start hive in Cloudera using the following command:

```
hive
```

1. Database Management Commands

1. Show Databases

```
SHOW DATABASES;
```

Output:

```
OK
default
madhurima_bda_database
Time taken: 8.147 seconds, Fetched: 2 row(s)
```

Explanation: This command lists all databases available in the Hive environment.

2. Create Database

```
CREATE DATABASE madhurima_database;
```

Output:

```
OK
Time taken: 6.817 seconds
```

Explanation: A new database named `madhurima_database` is created.

3. Show Databases Again

```
SHOW DATABASES;
```

Output:

```
OK
default
madhurima_database
madhurima_bda_database
Time taken: 1.488 seconds, Fetched: 3 row(s)
```

Explanation: The updated list of databases shows the newly created database.

4. Use Database

```
USE madhurima_database;
```

Output:

```
OK
Time taken: 0.53 seconds
```

Explanation: This command sets the context to the `madhurima_database` for subsequent operations.

2. Table Creation and data Loading Commands

5. Create Customers Table

```
CREATE TABLE Customers(customer_id INT, name STRING,
city STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Output:

```
OK
Time taken: 0.881 seconds
```

Explanation: The `Customers` table is created with specified columns and data format.

6. Create Orders Table

```
CREATE TABLE Orders(order_id INT, customer_id INT, amount FLOAT,
date STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Output:

```
OK
Time taken: 4.882 seconds
```

Explanation: The `Orders` table is created with specified columns and data format.

7. Show Tables

```
SHOW TABLES;
```

Output:

```
OK
customers
orders
Time taken: 1.318 seconds, Fetched: 2 row(s)
```

Explanation: Lists all tables within the current database.

8. Describe Customers Table

```
DESCRIBE madhurima_database.customers;
```

Output:

```
OK
customer_id             int
name                    string
city                    string
Time taken: 0.967 seconds, Fetched: 3 row(s)
```

Explanation: Provides the schema of the `customers` table.

9. Describe Orders Table

```
DESCRIBE madhurima_database.orders;
```

Output:

```
OK
order_id                int
customer_id             int
amount                  float
date                    string
Time taken: 0.53 seconds, Fetched: 4 row(s)
```

Explanation: Provides the schema of the `orders` *table.*

10. Load Data into Customers Table

```
LOAD DATA LOCAL INPATH 'file:///home/cloudera/Downloads/Customers.txt'
INTO TABLE customers;
```

Output:

```
Loading data to table madhurima_database.customers
Table madhurima_database.customers stats: [numFiles=1, totalSize=96]
OK
Time taken: 9.524 seconds
```

Explanation: Loads data from a local file into the `customers` *table.*

11. Load Data into Orders Table

```
LOAD DATA LOCAL INPATH 'file:///home/cloudera/Downloads/Orders.txt'
INTO TABLE orders;
```

Output:

```
Loading data to table madhurima_database.orders
Table madhurima_database.orders stats: [numFiles=1, totalSize=190]
OK
Time taken: 0.854 seconds
```

Explanation: Loads data from a local file into the `orders` table.

3. Select Commands

12. Select All from Customers Table

```
SELECT * FROM customers;
```

Output:

```
OK
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	Chicago
4	Dave	New York
5	Eve	Miami
6	Frank	Boston
Time taken: 1.465 seconds, Fetched: 6 row(s)
```

Explanation: Retrieves all records from the `customers` table.

13. Select All from Orders Table

```
SELECT * FROM orders;
```

Output:

```
OK
101	1	100.5	2024-10-01
102	2	200.0	2024-10-02
103	1	50.75	2024-10-03
104	3	80.0	2024-10-03
105	4	300.0	2024-10-04
106	5	120.0	2024-10-05
107	2	150.0	2024-10-05
108	6	250.0	2024-10-06
Time taken: 0.446 seconds, Fetched: 8 row(s)
```

Explanation: Retrieves all records from the `orders` table.

4. Filtering Records

14. Select with Join Condition

```
SELECT o.order_id, o.amount, c.name, c.city
FROM orders o
JOIN customers c
ON o.customer_id = c.customer_id
WHERE c.city = 'New York';
```

Output:

```
OK
101 100.5    Alice    New York
103 50.75    Alice    New York
105 300.0    Dave     New York
Time taken: 94.098 seconds, Fetched: 3 row(s)
```

Explanation: Joins the `orders` and `customers` tables to retrieve order details for customers from New York.

15. Query to Select Orders Above 150

```
SELECT o.order_id, o.amount, c.name
FROM orders o
JOIN customers c
ON o.customer_id = c.customer_id
WHERE o.amount > 150;
```

Output:

```
102 200.0    Bob
105 300.0    Dave
108 250.0    Frank
```

This query retrieves order IDs, amounts, and customer names for orders with an amount greater than 150.

5. Join Operations(Join the `orders` and `customers` tables)

16. Query to Select All Customers with Inner Join on Orders

```
SELECT c.name, o.order_id, o.amount
FROM customers c
```

```
JOIN orders o
ON c.customer_id = o.customer_id;
```

Output:

```
Alice   101 100.5
Bob 102 200.0
Alice   103 50.75
Charlie 104 80.0
Dave    105 300.0
Eve     106 120.0
Bob     107 150.0
Frank   108 250.0
```

This query retrieves customer names along with their corresponding order IDs and amounts.

17. Query to Select All Customers with Left Join on Orders

```
SELECT c.name, o.order_id, o.amount
FROM customers c
LEFT JOIN orders o
ON c.customer_id = o.customer_id;
```

Output:

```
Alice   101 100.5
Alice   103 50.75
Bob     102 200.0
Bob     107 150.0
Charlie 104 80.0
Dave    105 300.0
Eve     106 120.0
Frank   108 250.0
```

This query retrieves all customer names along with their order IDs and amounts, including customers without any orders.

18. Query to Select All Customers with Right Join on Orders

```
SELECT c.name, o.order_id, o.amount
FROM customers c
RIGHT JOIN orders o
ON c.customer_id = o.customer_id;
```

- This query retrieves all orders along with the customers who placed them. If an order is placed by a customer who doesn't exist in the `customers` table, that order will still appear with `NULL` for the customer's name.
- **Output:**

```
Alice    101    100.5
Bob      102    200.0
Alice    103    50.75
Charlie  104    80.0
Dave     105    300.0
Eve      106    120.0
Bob      107    150.0
Frank    108    250.0
```

19. Query to Select All Customers with Outer Join on Orders

```sql
SELECT c.name, o.order_id, o.amount
FROM customers c
FULL OUTER JOIN orders o
ON c.customer_id = o.customer_id;
```

- This query retrieves all customers and all orders. If a customer has no orders, the order fields will be `NULL`, and if an order has no associated customer, the customer fields will be `NULL`.
- **Output:**

```
Alice    103    50.75
Alice    101    100.5
Bob      107    150.0
Bob      102    200.0
Charlie  104    80.0
Dave     105    300.0
Eve      106    120.0
Frank    108    250.0
```

6. Aggregation Queries

20. Total Amount Spent by Each Customer

```sql
SELECT c.name, SUM(o.amount) AS total_spent
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.name;
```

- This query sums up the total amount spent by each customer by joining the two tables and grouping the results by customer name.
- **Output:**

name	total_spent
Alice	151.25
Bob	350.00
Charlie	80.00
Dave	300.00
Eve	120.00
Frank	250.00

The Hive queries you've executed successfully retrieve information about customer orders and their average order amounts based on city. Here's a summary of each query along with the results:

21. Customer Order Count

```
SELECT c.name, COUNT(o.order_id) AS order_count
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.name;
```

Description:

- The query processed successfully and ran a local task followed by a MapReduce job.
- The execution logs show the task was completed with 1 mapper and 1 reducer.

Output:

name	order_count
Alice	2
Bob	2
Charlie	1
Dave	1
Eve	1
Frank	1

22. Average Order Amount by City

```
SELECT c.city, AVG(o.amount) AS avg_order_amount
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.city;
```

Description:

- The query also executed successfully with a similar processing approach.
- The logs indicated successful completion of the MapReduce job with 1 mapper and 1 reducer.

Output:

city	avg_order_amount
Boston	250.0
Chicago	80.0
Los Angeles	175.0
Miami	120.0
New York	150.42

6. Combining Filtering with Aggregation

23: Total Spent by Customers

```
SELECT c.name, SUM(o.amount) AS total_spent
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.name
HAVING SUM(o.amount) > 200;
```

Explanation:

1. **Purpose**: This query calculates the total amount spent by each customer, filtering for those who have spent more than $200.

2. **Tables Involved**:

   - `customers` : Contains customer information, such as names and IDs.
   - `orders` : Contains order details, including amounts and the corresponding customer IDs.

3. **JOIN Operation**:
   - The `JOIN` clause combines records from `customers` and `orders` where the `customer_id` matches in both tables, allowing the query to relate customers to their respective orders.
4. **Aggregation**:
   - The `SUM(o.amount)` function computes the total spending for each customer.
5. **Grouping**:
   - The `GROUP BY c.name` clause ensures that the results are aggregated per customer name.
6. **Filtering**:
   - The `HAVING SUM(o.amount) > 200` clause filters the results to include only those customers whose total spending exceeds $200.

Output:

```
Bob     350.0
Dave    300.0
Frank   250.0
```

- This output shows that:
  - Bob spent a total of $350.
  - Dave spent $300.
  - Frank spent $250.

24: Count of High-Value Orders

```sql
SELECT c.name, COUNT(o.order_id) AS high_value_orders
FROM customers c
JOIN orders o
ON c.customer_id = o.customer_id
WHERE o.amount > 100
GROUP BY c.name;
```

Explanation:

1. **Purpose**: This query counts the number of high-value orders (orders greater than $100) made by each customer.
2. **Tables Involved**: Same as the previous query, using the `customers` and `orders` tables.
3. **JOIN Operation**:

   - Similar to the first query, it combines the `customers` and `orders` tables based on matching `customer_id`.

4. **Filtering**:

   - The `WHERE o.amount > 100` clause filters orders to include only those whose amounts exceed $100.

5. **Aggregation**:

   - The `COUNT(o.order_id)` function counts the number of high-value orders for each customer.

6. **Grouping**:

   - The `GROUP BY c.name` clause groups the results by customer name.

Output:

```
Alice  1
Bob    2
Dave   1
Eve    1
Frank  1
```

- This output indicates:
  - Alice made 1 high-value order.
  - Bob made 2 high-value orders.
  - Dave, Eve, and Frank each made 1 high-value order.

Summary Table

Explanation of Columns

- **Command No**: A unique identifier for the command.
- **Command**: The Hive query executed.
- **No. of Rows Fetched**: The number of rows returned by the query.
- **No. of Columns**: The number of columns in the result set.
- **Effect**: A brief description of what the command does.
- **Mapper**: The number of mapper tasks used in the job.
- **Reducer**: The number of reducer tasks used in the job.
- **Time Taken**: The total time taken to execute the command.

Summary Table

Command No	Command	No. of Rows Fetched	No. of Columns	Effect	Mapper	Reducer	Time Taken
1	`SHOW DATABASES;`	2	1	List databases	0	0	8.147 seconds
2	`CREATE DATABASE madhurima_database;`	0	0	Create database	0	0	6.817 seconds
3	`SHOW DATABASES;`	3	1	List databases	0	0	1.488 seconds
4	`USE madhurima_database;`	0	0	Use database	0	0	0.53 seconds
5	`CREATE TABLE Customers(...);`	0	0	Create table	0	0	0.881 seconds
6	`CREATE TABLE Orders(...);`	0	0	Create table	0	0	4.882 seconds
7	`SHOW TABLES;`	2	1	List tables	0	0	1.318 seconds
8	`DESCRIBE madhurima_database.customers;`	3	3	Describe table	0	0	0.967 seconds
9	`DESCRIBE madhurima_database.orders;`	4	4	Describe table	0	0	0.53 seconds
10	`LOAD DATA ... INTO TABLE customers;`	0	0	Load data	0	0	9.524 seconds
11	`LOAD DATA ... INTO TABLE orders;`	0	0	Load data	0	0	0.854 seconds
12	`SELECT * FROM customers;`	6	3	Select records	1	0	1.465 seconds
13	`SELECT * FROM orders;`	8	4	Select records	1	0	0.446 seconds
16	`SELECT o.order_id, o.amount, c.name, c.city ...;`	3	4	Inner Join tables	1	0	94.098 seconds
17	`SELECT c.name, o.order_id, o.amount FROM customers c LEFT JOIN orders o ON c.customer_id = o.customer_id;`	8	3	MapJoin with left join	1	0	28.115 sec
18	`SELECT c.name, o.order_id, o.amount FROM customers c RIGHT JOIN orders o ON c.customer_id = o.customer_id;`	8	3	MapJoin with right join	1	0	28.4 sec
19	`SELECT c.name, o.order_id, o.amount FROM customers c FULL OUTER JOIN orders o ON c.customer_id = o.customer_id;`	8	3	MapReduce with full outer join	2	1	46.554 sec
20	`SELECT c.name, SUM(o.amount) AS total_spent FROM customers c JOIN orders o ON c.customer_id = o.customer_id GROUP BY c.name;`	6	2	MapReduce with aggregation	1	1	29.606 sec
21	`SELECT c.name, COUNT(o.order_id) AS order_count`	6	2	Group by customer	1	1	35.988 seconds
22	`SELECT c.city, AVG(o.amount) AS avg_order_amount`	5	2	Group by city	1	1	34.327 seconds

Command No	Command	No. of Rows Fetched	No. of Columns	Effect	Mapper	Reducer	Time Taken
23	`SELECT c.name, SUM(o.amount) AS total_spent FROM customers c JOIN orders o ON c.customer_id = o.customer_id GROUP BY c.name HAVING SUM(o.amount) > 200;`	3	2	Fetch total spent > 200	1	1	40.735 seconds
24	`SELECT c.name, COUNT(o.order_id) AS high_value_orders FROM customers c JOIN orders o ON c.customer_id = o.customer_id WHERE o.amount > 100 GROUP BY c.name;`	5	2	Count orders > 100	1	1	37.594 seconds