Author: Madhurima Rawat

Hive Queries for Customers and Orders

Cloudera Installation Steps Using Oracle VM VirtualBox

To set up Cloudera on Windows using Oracle VM VirtualBox, follow these steps:

1. **Download and Install Oracle VM VirtualBox**:
   - Visit the Oracle VM VirtualBox website and download the latest version for Windows.
   - Install VirtualBox by following the on-screen instructions.
2. **Download Cloudera QuickStart VM**:
   - Go to the Cloudera Downloads page and download the QuickStart VM (typically in `.ova` format).
3. **Import the Cloudera QuickStart VM**:
   - Open Oracle VM VirtualBox.
   - Click on **File** > **Import Appliance**.
   - Select the downloaded `.ova` file and follow the prompts to import the VM.
4. **Configure the VM Settings** (optional):
   - Adjust the number of processors and RAM to allocate more resources for better performance (recommended: at least 4GB RAM).
5. **Start the Cloudera QuickStart VM**:
   - Select the imported VM and click **Start**.
   - Once the VM is running, log in with the credentials:
     - **Username**: `cloudera`
     - **Password**: `cloudera`
6. **Access Cloudera Manager**:
   - Open a web browser in the VM and navigate to `http://localhost:7180`.
   - Log in using the same credentials (`cloudera/cloudera`).

Start hive in Cloudera using the following command:

```
hive
```

Database Management Commands

a. Show Databases

```
SHOW DATABASES;
```

Output:

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

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

b. Create a Database

```
CREATE DATABASE madhurima_database;
```

Output:

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

Explanation: This command creates a new database named `madhurima_database`.

c. Use the Database

```
USE madhurima_database;
```

Output:

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

Explanation: This command sets the current working database to `madhurima_database`.

Dataset Explanation

Customers Table

The **Customers** table contains information about the customers, including:

- `customer_id` : Unique identifier for each customer.
- `name` : Name of the customer.
- `city` : City where the customer resides.

Example Records:

customer_id	name	city
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	Chicago
4	Dave	New York
5	Eve	Miami
6	Frank	Boston

Orders Table

The **Orders** table contains details about the orders placed by customers, including:

- `order_id` : Unique identifier for each order.
- `customer_id` : ID of the customer who placed the order (foreign key).
- `amount` : Total amount of the order.
- `order_date` : Date when the order was placed.

Example Records:

order_id	customer_id	amount	order_date
101	1	100.50	2024-10-01
102	2	200.00	2024-10-02
103	1	50.75	2024-10-03
104	3	80.00	2024-10-03

order_id	customer_id	amount	order_date
105	4	300.00	2024-10-04
106	5	120.00	2024-10-05
107	2	150.00	2024-10-05
108	6	250.00	2024-10-06

1. Sample Data

a. Customers Table Creation

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

Output:

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

Explanation: This command create the table customers in the current Hive database.

b. Orders Table Creation

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

Output:

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

Explanation: This command create the table orders in the current Hive database.

c. Show Tables

```
SHOW TABLES;
```

Output:

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

Explanation: This command displays all the tables in the current Hive database.

d. Describe Customers Table in madhurima_database

```
DESCRIBE madhurima_database.customers;
```

Output:

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

Explanation: This command describes the structure of the `customers` table in the `madhurima_database`, showing the column names and data types.

e. Describe Orders Table in madhurima_database

```
DESCRIBE madhurima_database.orders;
```

Output:

```
OK
order_id             int
customer_id          int
amount               float
date                 string
Time taken: 0.530 seconds, Fetched: 4 row(s)
```

Explanation: This command describes the structure of the `orders` table in the `madhurima_database`, showing the column names and data types.

2. Loading Data into the Table

a. 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: This command loads data from a local file into the `customers` table. Ensure the path to the file is correct.

b. 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: This command loads data from a local file into the `orders` table.

c. Select All from Customers Table

```
SELECT * FROM customers;
```

Output:

customer_id	name	city
1	Alice	New York
2	Bob	Los Angeles
3	Charlie	Chicago
4	Dave	New York
5	Eve	Miami

customer_id	name	city
6	Frank	Boston

Explanation: This command retrieves all records from the `customers` table.

d. Select All from Orders Table

```
SELECT * FROM orders;
```

Output:

order_id	customer_id	amount	order_date
101	1	100.50	2024-10-01
102	2	200.00	2024-10-02
103	1	50.75	2024-10-03
104	3	80.00	2024-10-03
105	4	300.00	2024-10-04
106	5	120.00	2024-10-05
107	2	150.00	2024-10-05
108	6	250.00	2024-10-06

Explanation: This command retrieves all records from the `orders` table.

3. Filtering Queries

a. Filter Orders Based on City

```
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:

order_id	amount	name	city
101	100.50	Alice	New York
105	300.00	Dave	New York

Explanation: This command retrieves order details for customers residing in New York.

b. Filter Orders Above a Certain Amount

```
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:

order_id	amount	name
102	200.00	Bob
105	300.00	Dave
108	250.00	Frank

Explanation: This command fetches order details for orders that exceed 150 in amount.

4. Join Queries

a. Inner Join (Orders and Customers)

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

Output:

name	order_id	amount
Alice	101	100.50
Bob	102	200.00

name	order_id	amount
Alice	103	50.75
Charlie	104	80.00
Dave	105	300.00
Eve	106	120.00
Bob	107	150.00
Frank	108	250.00

Explanation: This command retrieves all orders along with the customer names using an inner join.

b. Left Join (Customers with their Orders)

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

Output:

name	order_id	amount
Alice	101	100.50
Bob	102	200.00
Alice	103	50.75
Charlie	104	80.00
Dave	105	300.00
Eve	106	120.00
Bob	107	150.00
Frank	108	250.00

Explanation: This command retrieves all customers and their orders, including those without orders (order details will be null).

c. Right Join (All Orders, Matching Customers)

This query retrieves all orders, including those without matching customer records.

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

Output:

name	order_id	amount
Alice	101	100.50
Bob	102	200.00
Alice	103	50.75
Charlie	104	80.00
Dave	105	300.00
Eve	106	120.00
Bob	107	150.00
Frank	108	250.00

Explanation: This command retrieves all orders along with the customer names, including orders that may not have a corresponding customer record.

d. Full Outer Join (All Customers and Orders)

This query retrieves all customers and all orders, showing null values for missing data.

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

Output:

name	order_id	amount
Alice	101	100.50
Bob	102	200.00
Alice	103	50.75
Charlie	104	80.00

name	order_id	amount
Dave	105	300.00
Eve	106	120.00
Bob	107	150.00
Frank	108	250.00

Explanation: This command retrieves all customers and orders, including those that may not have matching records, with nulls displayed for missing data.

5. Aggregation Queries

a. Total Amount Spent by Each Customer

This query calculates the total amount each customer has spent on orders.

```
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;
```

Output:

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

Explanation: This command sums the amounts from orders associated with each customer. It groups the results by customer name to provide the total spent by each customer.

b. Count Number of Orders per Customer

This query counts the total number of orders placed by each customer.

```
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;
```

Output:

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

Explanation: This command counts the number of orders linked to each customer. It groups the results by customer name to show how many orders each customer has placed.

c. Average Order Amount by City

This query calculates the average order amount for customers from each 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;
```

Output:

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

Explanation: This command averages the order amounts for customers in each city. It groups the results by city to provide the average order amount for customers residing in that city.

6. Combining Filtering with Aggregation

a. Total Spending for Customers Who Spent Over 200

This query retrieves customers whose total spending exceeds 200.

```
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;
```

Output:

name	total_spent
Bob	350.00
Dave	300.00
Frank	250.00

Explanation: This command sums the total amounts spent by each customer. The `HAVING` clause filters the results to include only those customers who spent more than 200.

b. Find the Number of Orders Greater than 100 by Each Customer

This query counts the number of orders placed by each customer that exceed 100.

```
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;
```

Output:

name	high_value_orders
Alice	1

name	high_value_orders
Bob	2
Dave	1
Eve	1
Frank	1

Explanation: This command counts how many orders each customer placed that have an amount greater than 100. The results show the number of high-value orders for each customer.