**EXPERIMENT 5: Hive Database and Table Operations**

**STEP 1: Start Hive**

In the terminal, type:

hive

**What this does:**

* Launches the **Hive shell (CLI)** — a command-line interface to interact with Hive.
* You’ll now see a prompt like:

hive>

**PART 1: Basic Datastore (Database) Operations in Hive**

**1. Create a Database**

CREATE DATABASE bdaplab;

**Explanation:**

| **Part** | **Meaning** |
| --- | --- |
| CREATE DATABASE | A HiveQL command to make a new database |
| bdaplab | The name of your database (you can use any valid name) |

This creates an **empty workspace** called bdaplab to store tables.

**2. List All Databases**

SHOW DATABASES;

**Explanation:**

* This lists **all available databases** in your Hive system.
* You’ll see names like:

default

bdaplab

**3. Switch to the bdaplab Database**

USE bdaplab;

**Explanation:**

| **Part** | **Meaning** |
| --- | --- |
| USE | Command to set which database you want to work inside |
| bdaplab | The name of the database to switch into |

This ensures all tables you create next go into this database.

**4. Check Current Database**

First, enable showing current DB:

SET hive.cli.print.current.db=true;

Then, check which DB you're in:

SELECT current\_database();

**Explanation:**

| **Command** | **Meaning** |
| --- | --- |
| SET hive.cli.print.current.db=true | Displays the name of the active DB in the CLI prompt |
| SELECT current\_database(); | Returns the name of the database you are currently using |

Useful to **verify you are working in the correct database**.

**5. Drop the Database (optional)**

If you want to delete the database:

DROP DATABASE bdaplab;

If the database is not empty:

DROP DATABASE bdaplab CASCADE;

**Explanation:**

| **Command** | **Meaning** |
| --- | --- |
| DROP DATABASE | Command to delete a database |
| CASCADE | Force delete the database and all tables inside it |

Use CASCADE carefully — it deletes everything inside.

**PART 2: Create Tables in Hive using HiveQL**

We are going to create the following tables:

1. sales\_order
2. product
3. customer (with nested STRUCT and partitioning)

Make sure you're inside Hive shell (hive>) and using the correct database (bdaplab).

**STEP 1: Create sales\_order Table**

**HiveQL Command:**

CREATE TABLE sales\_order (

order\_id INT,

customer\_id INT,

product\_id INT,

order\_date TIMESTAMP,

order\_amount DOUBLE,

quantity INT,

discount DOUBLE,

tax DOUBLE,

total\_amount DOUBLE

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**Explanation:**

| **Part** | **Meaning** |
| --- | --- |
| CREATE TABLE sales\_order (...) | Creates a new table named sales\_order |
| INT, DOUBLE, TIMESTAMP | Data types: integer, decimal, date-time |
| ROW FORMAT DELIMITED | Specifies that data is plain text with delimiters |
| FIELDS TERMINATED BY ',' | Columns are separated by **commas** in input files |
| STORED AS TEXTFILE | Table data will be stored in simple .txt files in HDFS |

You’ll use this format when loading .csv data later.

**STEP 2: Create product Table**

**HiveQL Command:**

CREATE TABLE product (

product\_id INT,

product\_name STRING,

category STRING,

price DOUBLE,

manufacturer STRING,

date\_added TIMESTAMP

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**Explanation:**

| **Column** | **Meaning** |
| --- | --- |
| product\_id | Unique ID for each product |
| product\_name | Name of the product |
| category | E.g., electronics, furniture |
| price | Price per unit |
| manufacturer | Company that made the product |
| date\_added | When it was added to catalog |

Other syntax is same as sales\_order.

**STEP 3: Create customer Table (with STRUCT and Partition)**

**HiveQL Command:**

CREATE TABLE customer (

customer\_id INT,

customer\_name STRING,

email STRING,

phone STRING,

address STRUCT<

street:STRING,

city:STRING,

state:STRING,

zip:INT

>,

date\_joined TIMESTAMP

)

PARTITIONED BY (country STRING, state STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

COLLECTION ITEMS TERMINATED BY '|'

STORED AS TEXTFILE;

**Explanation:**

| **Part** | **Meaning** |
| --- | --- |
| STRUCT<...> | Stores multiple fields under address — like a mini table inside a column |
| PARTITIONED BY | Splits data physically by country and state for faster access |
| `COLLECTION ITEMS TERMINATED BY ' | '` |
| FIELDS TERMINATED BY ',' | Normal fields separated by commas |

Next : Create .csv files to populate the above created 3 tables.

After that, upload the .csv files into your Cloudera VM using WinSCP

**Steps to Upload Files to Cloudera VM Using WinSCP**

**Step 1: Start the Cloudera VM**

* Open **Oracle VirtualBox**
* Start the **cloudera-quickstart-vm**
* Wait for the CentOS desktop to fully load

**Step 2: Find the IP Address of the Cloudera VM**

1. Open the **Terminal** inside the Cloudera VM.
2. Type this command:

ifconfig

1. Look for a section like eth0 or enp0s3 and find the line:

inet 192.168.x.x

This is your VM's **IP address** (e.g., 192.168.56.101).  
Note it down.

**Step 3: Open WinSCP on Your Windows Machine**

1. Launch **WinSCP**
2. You will see a login screen with the following fields:

| **Field** | **Value to Enter** |
| --- | --- |
| **File Protocol** | SCP |
| **Host Name** | IP address of your VM (e.g., 192.168.56.101) |
| **Port Number** | 22 |
| **Username** | cloudera |
| **Password** | cloudera |

1. Click **Login**

If everything is correct, you will see a two-panel window:

* **Left Panel** = Your Windows files
* **Right Panel** = Cloudera VM's Linux file system

### ****Step 4: Upload the**** .csv ****Files****

1. On the **left panel**, browse to the folder where you saved:
   * sales\_order.csv
   * product.csv
   * customer.csv
2. On the **right panel**, navigate to:

/home/cloudera

1. **Drag and drop** all three .csv files from left to right.

You’ve now uploaded the files to:

/home/cloudera/sales\_order.csv

/home/cloudera/product.csv

/home/cloudera/customer.csv

Your files are now available inside the Cloudera VM.

## Next Step: Load These Files into Hive Tables

We’ll now load the data into our Hive tables using the LOAD DATA command.

We already created these three Hive tables:

* sales\_order
* product
* customer

## Step 1: Open Hive Shell

In the Cloudera **Terminal**, type:

hive

You should see the prompt:

hive>

Then switch to your database (if not already in it):

USE bdaplab;

**Step 2: Load sales\_order.csv into Hive Table**

LOAD DATA LOCAL INPATH '/home/cloudera/sales\_order.csv'

OVERWRITE INTO TABLE sales\_order;

**Explanation:**

| **Part** | **Meaning** |
| --- | --- |
| LOAD DATA LOCAL INPATH | Load file from the **local file system (not HDFS)** |
| '/home/cloudera/sales\_order.csv' | Full path of the uploaded file |
| OVERWRITE INTO TABLE sales\_order | Replace all previous rows with this data (safe for fresh loads) |

**Step 3: Load product.csv into product Table**

LOAD DATA LOCAL INPATH '/home/cloudera/product.csv'

OVERWRITE INTO TABLE product;

**Step 4: Load customer.csv into customer Table**

**Note : Important:**

Since your customer table is **partitioned by country and state**, you must **specify the partition** when loading.

sql

CopyEdit

ALTER TABLE customer ADD PARTITION (country='India', state='Telangana');

LOAD DATA LOCAL INPATH '/home/cloudera/customer.csv'

OVERWRITE INTO TABLE customer

PARTITION (country='India', state='Telangana');

**Final Check**

To confirm your data is loaded correctly:

**For sales\_order:**

SELECT \* FROM sales\_order;

**For product:**

SELECT \* FROM product;

**For customer:**

SELECT \* FROM customer WHERE state='Telangana';

**Run 6 Hive Queries for Analysis**

**a) Find Total Sales Amount**

SELECT SUM(total\_amount) AS total\_sales

FROM sales\_order;

This adds up all total\_amount values and returns a single total.

**b) Find Top N Products by Sales**

Let's say **Top 2 products** based on total amount sold:

SELECT product\_id, SUM(total\_amount) AS total\_sales

FROM sales\_order

GROUP BY product\_id

ORDER BY total\_sales DESC

LIMIT 2;

This:

* Groups sales by product\_id
* Sums total amount
* Orders in descending
* Limits result to top N products

**c) Find Customer-Wise Sales**

SELECT customer\_id, SUM(total\_amount) AS customer\_total

FROM sales\_order

GROUP BY customer\_id;

This shows **how much each customer has spent in total.**

**d) Find Monthly Sales Trends**

SELECT MONTH(order\_date) AS month, SUM(total\_amount) AS monthly\_sales

FROM sales\_order

GROUP BY MONTH(order\_date)

ORDER BY month;

This:

* Extracts the month from the order\_date
* Sums the sales for each month
* Shows how sales vary across months

**e) Find Product-Wise Quantity Sold**

SELECT product\_id, SUM(quantity) AS total\_quantity\_sold

FROM sales\_order

GROUP BY product\_id;

This tells you how many units were sold for each product.

**f) Find the Highest-Priced Product in Each Category**

SELECT category, MAX(price) AS max\_price

FROM product

GROUP BY category;

This:

* Groups by category
* Returns the **most expensive product** in each category