**Practical File**

**Big Data - 2 (BCA-DS-311)**



| Submitted To | Submitted By |
| --- | --- |
| Mr. Saransh | **Asad** |
| Assistant Professor | **22-BCADS-08** |

**Department of Computer Science and Engineering**

****



# **INDEX**

| **Sno** | **Program** | **Date** | **Signature** |
| --- | --- | --- | --- |
| 1 | Installation of Cloud era to work with Pig. |  |  |
| 2 | Execute various commands and queries. |  |  |
| 3. | Loading Data in Pig |  |  |
| 4 | Producing a Histogram in Pig |  |  |
| 5 | Sum Word Counts for Each Word Length Using SUM Function in Pig |  |  |
| 6 | Copy the Data File into HDFS |  |  |
| 7 | Importing CSV Files and Creating Tables in Pig |  |  |
| 8 | Facebook Data Analysis using Graph API |  |  |
| 9 | Installation of Hive |  |  |
| 10 | Introduction to Hive Shell |  |  |
| 11 | Making a Script File in Hive |  |  |
| 12 | Loading Dataset and Loading Tables in Hive |  |  |
| 13 | Creating Tables, Creating Databases, and Editing Tables in Hive |  |  |
| 14 | Applying Various Queries in Hive |  |  |
| 15 | Different Ways of Querying in Hive |  |  |

## 

## **Experiment 1: Installation of Cloud era to work with Pig.**

### Aim:

To install and set up Cloudera so we can use Pig for processing big data.

### Theory:

Apache Pig is a platform used to analyze large data sets. It uses a language called *Pig Latin*, which is easy to write and understand. Pig scripts are converted into MapReduce jobs and run on a Hadoop cluster.

Cloudera provides a virtual machine (VM) called **Cloudera QuickStart VM**, which already includes Pig, Hadoop, Hive, and many other big data tools. This makes it easy for students to learn and practice without setting up everything from scratch.

To run this virtual machine, we need software like **VirtualBox** or **VMware**. These are virtualization tools that let us run another operating system inside our existing one, like a computer inside a computer.

After we download and open the Cloudera VM in VirtualBox, we get a Linux-based environment with all tools pre-installed. We don’t have to manually install Hadoop, Pig, or Hive. We can simply open the terminal in the VM and start writing Pig commands.

There are two modes in which Pig can be used:

* **Local mode**: Used for testing with small files. It runs Pig on a single machine without needing Hadoop.
* **MapReduce mode**: Used to process large files. It runs on the Hadoop system built into the VM.

To check if Pig is installed and working, we can open the terminal in the Cloudera VM and type:

| pig -version |
| --- |

This will show the version of Pig installed.

To start Pig in local mode:

| pig -x local |
| --- |

To start Pig in MapReduce mode (default mode):

| pig |
| --- |

After starting Pig, you will see the grunt> prompt, where you can type and run Pig Latin commands.

**Requirements:**

* A computer with at least 8 GB of RAM
* VirtualBox installed
* Cloudera QuickStart VM file (usually .ova format)

### Result:

We successfully installed Cloudera and verified that Pig is working. Now we can write and run Pig scripts to process data.

## **Experiment 2: Execute various commands and queries.**

### Aim:

To learn how to run different Pig commands and queries for analyzing data.

### Theory:

Pig is a tool used in Big Data to work with large files. It uses its own language called **Pig Latin**. Pig Latin is similar to SQL but is more flexible for handling unstructured data.

We can write Pig commands to do many things like:

* Loading data from a file
* Filtering rows based on a condition
* Grouping data
* Finding average, max, min, and sum
* Joining two datasets
* Sorting data

Pig can work with structured and semi-structured data. It runs these commands as MapReduce jobs in the background. But Pig scripts are much easier to write than Java MapReduce programs.

Let’s try some basic commands and queries step-by-step using a sample dataset.

**Sample Dataset (students.txt):**

| John,Math,**85** Alice,Science,**90** Bob,Math,**70** John,Science,**78** Alice,Math,**88** |
| --- |

This file has three columns: Name, Subject, and Marks.

**Pig Commands and Queries:**

1. **Load the Data**

| student\_data = LOAD 'students.txt' USING PigStorage(',') AS (name:chararray, subject:chararray, marks:int); |
| --- |

This command loads the data from the file and gives names to each column.

1. **See the Loaded Data**

| DUMP student\_data; |
| --- |

This shows the content of the file.

1. **Filter Students with Marks > 80**

| high\_scores = FILTER student\_data BY marks > 80; DUMP high\_scores; |
| --- |

This shows only those students who scored more than 80.

1. **Group Data by Name**

| grouped\_data = GROUP student\_data BY name; |
| --- |

This groups all records by student name.

1. **Calculate Average Marks per Student**

| avg\_marks = FOREACH grouped\_data GENERATE group AS name, AVG(student\_data.marks) AS average; DUMP avg\_marks; |
| --- |

This gives the average marks for each student.

1. **Sort Students by Marks**

| sorted\_data = ORDER student\_data BY marks DESC; DUMP sorted\_data; |
| --- |

This arranges the records from highest to lowest marks.

1. **Count the Number of Records**

| record\_count = FOREACH (**GROUP** student\_data ALL) GENERATE COUNT(**student\_data**); DUMP record\_count; |
| --- |

### Result:

We successfully executed different Pig commands such as LOAD, FILTER, GROUP, FOREACH, AVG, ORDER, and COUNT. These helped us process and analyze the data easily.

## **Experiment 3: Loading Data in Pig**

### Aim:

To understand how to load data into Pig from a file using Pig Latin.

### Theory:

Before we can process any data in Pig, we must first load it. Pig provides a simple command called LOAD to bring data from a file into Pig so we can work on it.

When we load data, we also tell Pig:

* Where the file is located.
* What separator is used (like comma , or tab \t).
* What are the names and data types of each column.

Pig uses a function called PigStorage() to read text files. For example, if our file has comma-separated values, we use PigStorage(',').

We can give names to each field and also mention the data type such as:

* chararray for text
* int for whole numbers
* float for decimal numbers

Once the data is loaded, we can use other commands to process it.

**Sample Data File (students.txt):**

| John,Math,**85** Alice,Science,**90** Bob,Math,**70** |
| --- |

This file contains student name, subject, and marks.

**Pig Command to Load Data:**

| student\_data = LOAD 'students.txt' USING PigStorage(',')   AS (name:chararray, subject:chararray, marks:int); |
| --- |

* 'students.txt' is the name of the file.
* PigStorage(',') tells Pig to use comma as the separator.
* AS (...) defines column names and their data types.

**To View the Data:**

| DUMP student\_data; |
| --- |

This command shows all rows from the file.

**Output:**

| (**John**,Math,**85**) (**Alice**,Science,**90**) (**Bob**,Math,**70**) |
| --- |

### Result:

We successfully loaded data from a text file into Pig using the LOAD command and displayed it using DUMP.

## **Experiment 4: Producing a Histogram in Pig**

### Aim:

To produce a histogram using Pig by grouping values and counting their frequency.

### Theory:

A **histogram** is a type of chart that shows how often each value or range of values appears in a dataset. In Pig, we can create a histogram by **grouping** data based on a column and then **counting** how many times each value appears.

For example, if we have student marks, we can create a histogram that shows how many students scored in each mark range like:

* 0–50
* 51–70
* 71–90
* 91–100

Pig doesn’t draw a chart directly, but it gives us the data needed to build a histogram — the ranges and the count of records in each.

We use:

* FILTER to separate values into ranges
* GROUP and COUNT to count how many values fall in each range

**Sample Data File (marks.txt):**

| John,**85** Alice,**90** Bob,**70** Max,**45** Neha,**76** Tom,**52** Sara,**66** |
| --- |

**Step-by-Step Pig Commands to Create a Histogram:**

1. **Load the data:**

| data = LOAD 'marks.txt' USING PigStorage(',') AS (name:chararray, marks:int); |
| --- |

1. **Create bins (ranges) using FILTER:**

| range1 = FILTER data BY marks <= 50; range2 = FILTER data BY marks > 50 AND marks <= 70; range3 = FILTER data BY marks > 70 AND marks <= 90; range4 = FILTER data BY marks > 90; |
| --- |

1. **Count how many students fall into each range:**

| group1 = GROUP range1 ALL; count1 = FOREACH group1 GENERATE '0-50' AS range, COUNT(range1) AS count;  group2 = GROUP range2 ALL; count2 = FOREACH group2 GENERATE '51-70' AS range, COUNT(range2) AS count;  group3 = GROUP range3 ALL; count3 = FOREACH group3 GENERATE '71-90' AS range, COUNT(range3) AS count;  group4 = GROUP range4 ALL; count4 = FOREACH group4 GENERATE '91-100' AS range, COUNT(range4) AS count; |
| --- |

1. **Combine all counts together:**

| final\_histogram = UNION count1, count2, count3, count4; DUMP final\_histogram; |
| --- |

**Expected Output:**

| (**0-50**,**1**) (**51-70**,**3**) (**71-90**,**3**) (**91-100**,**0**) |
| --- |

This output shows how many students fall in each range of marks, which is our histogram data.

### Result:

We successfully created histogram data using Pig by grouping student marks into different ranges and counting how many students fall in each.

## **Experiment 5: Sum Word Counts for Each Word Length Using SUM Function in Pig**

### Aim:

To calculate the total word count for each word length using the SUM function and FOREACH GENERATE in Pig.

### Theory:

In Pig, we often work with large text files and analyze word patterns. One common task is to count how many words have a certain length (e.g., 3-letter words, 4-letter words, etc.).

We do this in steps:

1. **Split lines into words**
2. **Calculate the length of each word**
3. **Group words by length**
4. **Use SUM and FOREACH GENERATE to get total word counts for each length**

The SUM() function is an **aggregate function** that adds all values in a group. FOREACH ... GENERATE is used to apply functions and produce the final output.

**Sample Input File (lines.txt):**

| hello world big data is fun pig is powerful |
| --- |

**Step-by-Step Pig Commands:**

1. **Load the file:**

| lines = LOAD 'lines.txt' AS (line:chararray); |
| --- |

1. **Split each line into words:**

| words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word; |
| --- |

1. **Get the length of each word:**

| word\_lengths = FOREACH words GENERATE word, SIZE(word) AS length; |
| --- |

1. **Group by word length:**

| grouped = GROUP word\_lengths BY length; |
| --- |

1. **Count how many words are there in each length:**

| counted = FOREACH grouped GENERATE group AS word\_length, COUNT(word\_lengths) AS total\_words; |
| --- |

1. **(Optional) If each word had a frequency and you wanted to SUM that:** If your input had word and count (like from a previous map step), you would write:

| -- Example relation: word\_counts = (word:chararray, length:int, count:int) grouped = GROUP word\_counts BY length; summed = FOREACH grouped GENERATE group AS word\_length, SUM(word\_counts.count) AS total\_count; |
| --- |

**Expected Output:**

| (2,2) -- **2**-letter words: 'is' (twice) (3,1) -- **3**-letter word: 'big' (4,1) -- **4**-letter word: 'data' (5,3) -- **5**-letter words: 'hello', 'world', 'power' (6,1) -- **6**-letter word: 'fun' |
| --- |

### Result:

We used SUM and FOREACH GENERATE in Pig to compute the total number of words grouped by their length.

## **Experiment 6: Copy the Data File into HDFS**

### Aim:

To copy a data file from the local file system into HDFS.

### Theory:

HDFS stands for **Hadoop Distributed File System**. It is the storage system used in Hadoop. Before we can use data in tools like Pig or Hive, we must first copy it into HDFS.

Copying a file into HDFS means moving it from our local computer (or local virtual machine) to Hadoop’s distributed storage.

This is done using Hadoop commands in the terminal.

Some important HDFS commands:

* hadoop fs -put → to copy files from local to HDFS
* hadoop fs -ls → to list files in HDFS
* hadoop fs -mkdir → to make a directory in HDFS

**Step-by-Step Commands:**

1. **Open your terminal and go to the folder where your file is located:**

| cd /home/cloudera/Desktop |
| --- |

1. **Create a directory in HDFS (optional but good practice):**

| hadoop fs -mkdir /input |
| --- |

1. **Copy your file into HDFS:** Let’s say your file is students.txt

| hadoop fs -put students.txt /input |
| --- |

1. **Check if the file was successfully copied:**

| hadoop fs -ls /input |
| --- |

You should see something like:

| Found **1** items -rw-r--r--  **1** cloudera supergroup  **102** 2025-05-01 10:00 /input/students.txt |
| --- |

### Result:

The data file students.txt was successfully copied into the HDFS directory /input.

## **Experiment 7: Importing CSV Files and Creating Tables in Pig**

### Aim:

To import a CSV file and create a structured relation (like a table) using Pig.

### Theory:

CSV stands for **Comma Separated Values**. It is a common format for storing tabular data, where each row represents a record and columns are separated by commas.

In Pig, we don’t use the term “table” like in databases. Instead, we work with **relations** — a set of records that looks like a table. Each relation has rows and columns.

To import a CSV file into Pig:

* We use the LOAD command.
* We use PigStorage(',') to read comma-separated values.
* We can assign names and types to each column using AS.

This gives structure to the data, similar to creating a table.

**Sample CSV File (employees.csv):**

| **101**,John,Manager,**60000** **102**,Alice,Developer,**50000** **103**,Bob,Tester,**45000** |
| --- |

Each row has: employee ID, name, role, and salary.

**Step-by-Step Commands in Pig:**

1. **Place the file into HDFS (if not done already):**

| hadoop fs -put employees.csv /input |
| --- |

1. **Load the CSV file in Pig and define structure (like a table):**

| employee\_data = LOAD '/input/employees.csv'   USING PigStorage(',')   AS (emp\_id:int, name:chararray, role:chararray, salary:int); |
| --- |

Here:

* /input/employees.csv is the path in HDFS.
* PigStorage(',') reads comma-separated fields.
* AS (...) defines the columns (like a table header) and their data types.

1. **View the data:**

| DUMP employee\_data; |
| --- |

**Expected Output:**

| (**101**,John,Manager,**60000**) (**102**,Alice,Developer,**50000**) (**103**,Bob,Tester,**45000**) |
| --- |

### 

### Result:

We successfully imported a CSV file into Pig and created a structured relation similar to a table with column names and data types.

## **Experiment 8: Facebook Data Analysis using Graph API**

### Aim:

To analyze Facebook data, including user interactions like liked pages, friend activities, and other metrics, using the **Facebook Graph API**.

### Theory:

The **Facebook Graph API** is a powerful tool that allows developers to interact with Facebook’s data. You can retrieve a variety of information, such as:

* **User’s likes (pages, posts, etc.)**
* **Friends list**
* **Posts and comments**
* **Profile data**

You can access the Graph API by sending HTTP requests. These requests return data in JSON format, which you can process using any programming language or tool (like Python, R, or a Big Data framework).

To analyze data like **liked pages of friends**:

* You first need access to **user permission** to retrieve such information.
* Then, use the API to extract the data and analyze it.

**Step-by-Step Guide to Access Facebook Data:**

#### **1. Set Up the Facebook Developer Account:**

* Visit [Facebook Developers](https://developers.facebook.com/).
* Create an app (you’ll need to sign in with your Facebook account).
* Get your **App ID** and **App Secret**.

#### **2. Request Access to the Graph API:**

* You’ll need a **user access token** to fetch personal data. This is done by logging in to Facebook and requesting specific permissions.

| https://www.facebook.com/v12.0/dialog/oauth?client\_id=<your-app-id>&redirect\_uri=<your-redirect-uri> |
| --- |

After successful authentication, you’ll get a token that allows you to access data.

#### **3. Make API Calls:**

Once you have the access token, you can start making requests to the Graph API. Here are some example calls:

* **Get user’s liked pages:**

| http**s:**//graph.facebook.com/me/likes?access\_token=<access-token> |
| --- |

* **Get a list of friends:**

| http**s:**//graph.facebook.com/me/friends?access\_token=<access-token> |
| --- |

* **Get posts by a user:**

| http**s:**//graph.facebook.com/me/posts?access\_token=<access-token> |
| --- |

* **Get information about specific pages:**

| http**s:**//graph.facebook.com/<page-id>?access\_token=<access-token> |
| --- |

#### **4. Process the JSON Data:**

The data returned from these API calls is in JSON format. Here’s an example of what the output might look like when retrieving a user’s liked pages:

| {  "data": [  { "name": "Tech News", "id": "123456789" },  { "name": "Music Lovers", "id": "987654321" },  { "name": "Travel Blogs", "id": "112233445" }  ] } |
| --- |

You can write a **Python script** or use **Pig/Hive** to process this JSON data and analyze user activity.

#### **5. Analyze the Data:**

You could:

* **Count the number of liked pages** for each user.
* **Compare the liked pages** of users and friends.
* **Categorize** liked pages (e.g., technology, music, travel).
* Use **Pig** or **Hive** for further analysis if the data becomes large.

For example, in **Pig**:

* Load the JSON data using LOAD with JSONLoader (if you store it in HDFS).
* Parse and analyze the liked pages.

### **Example Pig Script to Analyze Liked Pages Data:**

Let’s assume you have a JSON file facebook\_data.json stored in HDFS containing Facebook data about liked pages.

1. **Load JSON Data:**

| facebook\_data = LOAD '/input/facebook\_data.json' USING JsonLoader('data:bag{tup(name:chararray, id:chararray)}'); |
| --- |

1. **Filter and Group the Data:**

| tech\_pages = FILTER facebook\_data BY name == 'Tech News'; grouped\_data = GROUP tech\_pages BY name; |
| --- |

1. **Count the Number of Liked Pages:**

| count\_pages = FOREACH grouped\_data GENERATE group AS page\_name, COUNT(tech\_pages) AS count; DUMP count\_pages; |
| --- |

### Result:

You will get the count of liked pages like "Tech News" based on the JSON data processed in Pig or Hive.

## **Experiment 9: Installation of Hive**

### Aim:

To install **Apache Hive** on a system, configure it, and ensure it's ready for use

### Theory:

**Apache Hive** is a data warehouse system built on top of **Hadoop**. It is used for querying and analyzing large datasets stored in Hadoop's HDFS. Hive provides a SQL-like interface (HiveQL) for users to interact with data in Hadoop without needing to write complex MapReduce code.

There are different ways to install Hive, but a basic installation involves the following steps:

1. **Download Hive**
2. **Install Hive on your system**
3. **Configure Hive with Hadoop**
4. **Start Hive service**

**Step-by-Step Installation of Hive:**

#### **Step 1: Install Hadoop (If not already installed)**

Hive works on top of Hadoop, so you need a working Hadoop setup. If Hadoop is not installed, follow these steps:

1. Download Hadoop from the [Apache website](https://hadoop.apache.org/releases.html).
2. Extract it and configure the environment variables (HADOOP\_HOME, PATH, etc.).
3. Start Hadoop services (namenode, datanode, resource manager, etc.).

#### **Step 2: Download Hive**

1. Go to the official [Apache Hive Downloads page](https://hive.apache.org/downloads.html).
2. Download the latest stable version of Hive (e.g., apache-hive-3.1.2-bin.tar.gz).

#### **Step 3: Extract the Hive Tarball**

1. Move the downloaded file to the desired directory:

| mv apache-hive-**3.1**.**2**-bin.tar.gz /home/cloudera/ |
| --- |

1. Extract the tarball:

| **tar** **-xzvf** **apache-hive-3**.1.2-bin.tar.gz |
| --- |

#### **Step 4: Configure Hive**

1. Navigate to the Hive directory:

| **cd** **apache-hive-3**.1.2-bin |
| --- |

1. Configure Hive to work with Hadoop. First, create the Hive configuration directory:

| mkdir -p /home/cloudera/hive/conf |
| --- |

1. Copy the sample configuration files to the conf directory:

| cp conf/hive-default.xml.template conf/hive-site.xml |
| --- |

1. Edit hive-site.xml to specify the configuration for your Hive setup. Common properties you may need to modify are:  
   * **hive.metastore.uri**: The URI to your metastore database.
   * **hive.exec.scratchdir**: Location of the scratch directory for Hive queries.
   * **hive.metastore.warehouse.dir**: Directory where data is stored in HDFS (e.g., /user/hive/warehouse).
2. Set the **Hadoop home directory** in the environment variable file (hive-env.sh):

| nano conf/hive-env.sh |
| --- |

Set the Hadoop home directory path:

| export HADOOP\_HOME=/home/cloudera/hadoop-3.2.1 |
| --- |

#### **Step 5: Install and Configure the Hive Metastore**

1. Hive requires a relational database to store metadata, usually **MySQL** or **PostgreSQL**.
2. Install MySQL (if not installed):

| sudo apt-get install mysql-server |
| --- |

1. Create a database for Hive:

| CREATE DATABASE metastore; |
| --- |

1. Initialize the Hive schema in MySQL:

| schematool -initSchema -dbType mysql |
| --- |

#### **Step 6: Set Hive Environment Variables**

1. Open the .bashrc file:

| nano ~/.bashrc |
| --- |

1. Add these lines to the end of the file:

| export HIVE\_HOME=/home/cloudera/apache-hive-3.1.2-bin export PATH=**$PATH**:$HIVE\_HOME/bin |
| --- |

1. Save and close the file. Then, refresh the environment:

| source ~/.bashrc |
| --- |

#### **Step 7: Start Hive Service**

To start Hive in **command-line interface** mode, use:

| hive |
| --- |

This command will start the Hive shell, and you should see a prompt like:

| **hive>** |
| --- |

### Result:

1. Hive will be installed successfully and accessible via the command line.
2. You will be able to execute Hive queries and interact with your Hadoop data warehouse.

## **Experiment 10: Introduction to Hive Shell**

### Aim:

To understand the basic functionality and usage of the **Hive Shell**.

### Theory:

The **Hive Shell** is the command-line interface (CLI) for interacting with Apache Hive. It allows users to run HiveQL (Hive Query Language) commands to manage and query data stored in **HDFS** (Hadoop Distributed File System). The Hive Shell is similar to SQL-based relational databases but tailored for large-scale data processing in the Hadoop ecosystem.

HiveQL is a SQL-like language that is easy to learn and is used to perform queries on data stored in HDFS. It can also be extended for more complex operations using **UDFs** (User Defined Functions).

Once you install Hive and set up the necessary configurations, you can start using the Hive Shell.

### **Step-by-Step Guide to Using Hive Shell:**

#### **Step 1: Starting the Hive Shell**

After installing and configuring Hive (as we did in the previous experiment), open your terminal and type the following command:

| hive |
| --- |

This command starts the Hive Shell, and you should see the following prompt:

| **hive>** |
| --- |

This indicates that you are inside the Hive Shell, and you can begin typing HiveQL commands.

#### **Step 2: Basic Hive Shell Commands**

Here are some basic commands you can run in the Hive Shell:

1. **Display the version of Hive:**

| hive --version |
| --- |

This will display the current version of Hive installed.

1. **Show available databases:**

| SHOW DATABASES; |
| --- |

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

1. **Create a new database:**

| CREATE DATABASE my\_database; |
| --- |

This creates a new database named my\_database in Hive.

1. **Switch to a specific database:**

| USE **my\_database**; |
| --- |

After running this command, any table or query you run will be applied to the my\_database database.

1. **Show tables in the current database:**

| SHOW TABLES; |
| --- |

This lists all tables within the currently active database.

1. **Create a table:**

| CREATE TABLE employees (  emp\_id INT,  name STRING,  role STRING,  salary INT ); |
| --- |

This command creates a table called employees with the columns emp\_id, name, role, and salary.

1. **Load data into a table from HDFS:**

| LOAD DATA INPATH '/input/employees.csv' INTO TABLE employees; |
| --- |

This loads data from the HDFS path /input/employees.csv into the employees table.

1. **Run a query (SELECT statement):**

| SELECT \* FROM employees; |
| --- |

This command retrieves all rows from the employees table and displays them in the Hive Shell.

1. **Exit the Hive Shell:**

| EXIT; |
| --- |

This command exits the Hive Shell and returns you to the regular terminal.

### **Examples of Basic HiveQL Queries:**

1. **Create a Table and Insert Data:**

| CREATE TABLE employees (  emp\_id INT,  name STRING,  role STRING,  salary INT );  INSERT INTO TABLE employees VALUES (**101**, 'John', 'Manager', **60000**); INSERT INTO TABLE employees VALUES (**102**, 'Alice', 'Developer', **50000**); |
| --- |

1. **Select Data from a Table:**

| SELECT name, role FROM employees WHERE salary > **50000**; |
| --- |

This retrieves the names and roles of employees with a salary greater than 50,000.

1. **Aggregate Functions:**

| SELECT AVG(salary) FROM employees; |
| --- |

This calculates the average salary of employees in the employees table.

1. **Group Data:**

| SELECT role, COUNT(\*) FROM employees GROUP BY role; |
| --- |

This groups the data by employee role and counts the number of employees in each role.

1. **Drop a Table:**

| DROP TABLE employees; |
| --- |

This deletes the employees table from the current database.

### Result:

* You should be able to start the Hive Shell using the hive command.
* You should be able to run basic HiveQL commands like creating tables, loading data, querying data, and performing simple data manipulations.

## **Experiment 11: Making a Script File in Hive**

### Aim:

To create and execute a **Hive script file** that contains HiveQL commands.

### Theory:

In Hive, a **script file** is a plain text file containing one or more HiveQL commands (such as creating tables, inserting data, running queries, etc.). Instead of typing each command manually in the Hive Shell, you can store all the commands in a file and execute them at once. This is useful for automation and batch processing.

The file usually has a .hql extension (similar to .sql files in traditional databases).

### **Why Use Script Files in Hive?**

* Makes work easier by automating repetitive tasks.
* Helps in organizing complex sequences of HiveQL operations.
* Useful for saving, sharing, and reusing queries.
* Useful in data pipelines or scheduled jobs.

### **Steps to Create and Run a Hive Script File**

#### **Step 1: Open a Terminal**

Navigate to the directory where you want to save the script file.  
 For example:

| cd /home/cloudera/hive-scripts |
| --- |

#### **Step 2: Create the Script File**

Use a text editor like nano or vi to create a Hive script:

| nano employee\_script.hql |
| --- |

Now, write the following HiveQL commands inside the file:

| -- Create database CREATE DATABASE IF NOT EXISTS company;  -- Use the created database USE company;  -- Create a table CREATE TABLE IF NOT EXISTS employees (  id INT,  name STRING,  role STRING,  salary FLOAT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;  -- Load data into the table LOAD DATA LOCAL INPATH '/home/cloudera/data/employees.csv' INTO TABLE employees;  -- Display all records SELECT \* FROM employees;  -- Count number of employees SELECT COUNT(\*) FROM employees;  -- Find average salary SELECT AVG(salary) FROM employees; |
| --- |

Save and exit the file (in nano: press Ctrl + O, then Enter, then Ctrl + X).

#### **Step 3: Execute the Hive Script**

Run the Hive script from the terminal using:

| hive -f employee\_script.hql |
| --- |

This will execute all the HiveQL commands in the file, one by one.

### Result:

* A new database and table will be created.
* Data from the employees.csv file will be loaded into the table.
* Basic queries like displaying all records, counting employees, and average salary will be executed.

## **Experiment 12: Loading Dataset and Loading Tables in Hive**

### Aim:

To load a dataset (e.g., a .csv file) into a Hive table and query the data.

### Theory:

In Hive, once a table is created, you can load data into it either from:

* **Local file system** (using LOCAL)
* **HDFS (Hadoop Distributed File System)**

This process is important because Hive doesn’t store data directly—it works on data stored in HDFS. After loading, you can run queries to analyze or transform the data.

### **Key Commands to Know:**

1. **CREATE TABLE** – Defines the schema of the table.
2. **LOAD DATA [LOCAL] INPATH** – Loads the dataset file into the Hive table.
3. **SELECT** – Used to view or process the loaded data.

### **Step-by-Step: Loading a Dataset into a Hive Table**

#### **Step 1: Prepare Your Dataset**

Let’s say you have a CSV file named students.csv with the following content:

| **101**,John,Math,**78** **102**,Alice,Science,**88** **103**,Bob,English,**92** |
| --- |

Save this file at:  
 /home/cloudera/data/students.csv

#### **Step 2: Start Hive Shell**

| hive |
| --- |

#### **Step 3: Create a Database (optional)**

| CREATE DATABASE IF NOT EXISTS school; USE school; |
| --- |

#### **Step 4: Create a Table for the Dataset**

| CREATE TABLE IF NOT EXISTS students (  id INT,  name STRING,  subject STRING,  marks INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE; |
| --- |

This command defines a table of students with four columns. The data is expected to be comma-separated and stored as a plain text file.

#### **Step 5: Load Data into the Table**

If the file is in the **local file system**, use:

| LOAD DATA LOCAL INPATH '/home/cloudera/data/students.csv' INTO TABLE students; |
| --- |

If the file is in **HDFS**, use:

| LOAD DATA INPATH '/user/cloudera/students.csv' INTO TABLE students; |
| --- |

##### Note: LOCAL keyword means the file is from the local file system, not HDFS.

#### **Step 6: Verify the Data is Loaded**

| SELECT \* FROM students; |
| --- |

You should see:

| **101** John Math **78** **102** Alice Science **88** **103** Bob English **92** |
| --- |

### Result:

* Hive table students will be created.
* Dataset from students.csv will be loaded into the table.
* You will be able to view the data using SELECT \*.

## **Experiment 13: Creating Tables, Creating Databases, and Editing Tables in Hive**

### Aim:

To learn how to create databases, create tables, and modify table structures in Hive.

### Theory:

Hive organizes data in a structure similar to traditional databases. The structure follows:

Database → Tables → Rows and Columns (Data)

* **Databases** in Hive group related tables together.
* **Tables** hold the actual data, with a defined schema (columns and types).
* **Editing tables** means altering their structure — like adding columns or renaming them.

Hive uses **HiveQL**, which is similar to SQL, to manage these operations.

### **1. Creating a Database**

To create a new database:

| CREATE DATABASE IF NOT EXISTS college; |
| --- |

To switch to that database:

| USE **college**; |
| --- |

To view all databases:

| SHOW DATABASES; |
| --- |

To view tables in a specific database:

| USE college; SHOW TABLES; |
| --- |

### **2. Creating a Table**

Example: Creating a table to store student data.

| CREATE TABLE IF NOT EXISTS students (  id INT,  name STRING,  course STRING,  marks INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE; |
| --- |

This defines a table with 4 columns: id, name, course, and marks.

### **3. Editing (Altering) Tables**

Once a table is created, you may need to change its structure. Hive allows altering tables using the ALTER TABLE command.

#### **Add a Column:**

| ALTER TABLE students ADD COLUMNS (email STRING); |
| --- |

#### **Rename a Column:**

| ALTER TABLE students CHANGE name full\_name STRING; |
| --- |

#### **Rename a Table:**

| ALTER TABLE students RENAME TO student\_records; |
| --- |

#### **Replace All Columns:**

| ALTER TABLE student\_records REPLACE COLUMNS (  id INT,  full\_name STRING,  course STRING,  marks INT,  grade STRING ); |
| --- |

### **4. Viewing Table Structure**

To see how the table is defined (its schema):

| DESCRIBE student\_records; |
| --- |

Or to see more details including file format and location:

| DESCRIBE FORMATTED student\_records; |
| --- |

### Result:

* You will create a new Hive database and a table.
* You will modify the table by adding a column and renaming it.
* You can view the updated structure using DESCRIBE.

## **Experiment 14: Applying Various Queries in Hive**

### Aim:

To learn how to apply basic and advanced queries in Hive to retrieve and analyze data from tables.

### Theory:

Hive uses **HiveQL**, which is very similar to SQL, to run queries on structured data stored in tables. After creating and loading a table, we can use different types of queries to filter, sort, group, and summarize the data.

### **Assume we have this table:**

| CREATE TABLE IF NOT EXISTS students (  id INT,  name STRING,  course STRING,  marks INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE; |
| --- |

Let’s say the table contains the following records:

| **id** | **name** | **course** | **marks** |
| --- | --- | --- | --- |
| 1 | John | Math | 75 |
| 2 | Alice | English | 88 |
| 3 | Bob | Science | 66 |
| 4 | Raj | Math | 90 |
| 5 | Neha | English | 72 |

### **Various Hive Queries**

#### **1. Select all records**

| SELECT \* FROM students; |
| --- |

#### **2. Select specific columns**

| SELECT name, course FROM students; |
| --- |

#### **3. Apply WHERE condition**

| SELECT \* FROM students WHERE marks > **70**; |
| --- |

#### **4. Use AND/OR operators**

| SELECT \* FROM students WHERE course = 'English' AND marks > **75**; |
| --- |

#### **5. Use ORDER BY (ascending/descending)**

| SELECT \* FROM students ORDER BY marks DESC; |
| --- |

#### **6. Group By (for aggregation)**

| SELECT course, COUNT(\*) FROM students GROUP BY course; |
| --- |

This counts how many students are in each course.

#### **7. Find Average Marks per Course**

| SELECT course, AVG(marks) AS avg\_marks FROM students GROUP BY course; |
| --- |

#### **8. Find Maximum and Minimum Marks**

| SELECT MAX(marks) AS max\_marks, MIN(marks) AS min\_marks FROM students; |
| --- |

#### **9. Rename Output Columns (using alias)**

| SELECT name AS student\_name, marks AS score FROM students; |
| --- |

#### **10. Use LIMIT to restrict output**

| SELECT \* FROM students LIMIT **3**; |
| --- |

### **Expected Output Examples**

For:

| SELECT course, COUNT(\*) FROM students GROUP BY course; |
| --- |

You may see:

| course count Math **2** English **2** Science **1** |
| --- |

## **Experiment 15: Different Ways of Querying in Hive**

### Aim:

To understand and perform queries in Hive using two different approaches:

1. Using **Hive Interactive Shell**
2. Using a **Hive Script (.hql) File**

### Theory:

Hive allows you to run HiveQL queries in multiple ways depending on your need. The two most common and practical methods are:

**1. Interactive Hive Shell**

This is the command-line environment where you type HiveQL queries **manually**, one at a time.

#### **Steps to Use Interactive Shell:**

1. Open terminal.
2. Type: hive
3. Then inside the Hive prompt (hive>), enter your queries:

| USE college;  SELECT \* FROM students;  SELECT course, AVG(marks) FROM students GROUP BY course;  SHOW TABLES; |
| --- |

#### **Advantages:**

* Good for quick testing and learning.
* You can see immediate results and correct errors on the spot.

#### **Limitations:**

* Manual and repetitive.
* Not good for batch processing or long queries.

**2. Hive Script File (.hql)**

A Hive script is a plain text file that contains a list of HiveQL commands. Instead of typing queries one by one, you write them in a file and execute them all at once.

#### **Steps to Use Hive Script:**

1. Open a text editor and create a script file, e.g., student\_queries.hql  
    Example content:

| USE college;  SELECT \* FROM students;  SELECT name, marks FROM students WHERE marks > **70**;  SELECT course, COUNT(\*) FROM students GROUP BY course; |
| --- |

1. Save the file.
2. In the terminal, run the script:

| hive -f student\_queries.hql |
| --- |

#### **Advantages:**

* Useful for automating queries.
* Easy to debug, reuse, and share.
* Saves time for long or repeated tasks.

#### **Limitations:**

* Less interactive; you can’t change queries on the fly.
* You need to edit the script file and rerun it for any change.

### **Summary Table:**

| **Feature** | **Interactive Shell** | **Hive Script File** |
| --- | --- | --- |
| Input Type | Manual | Pre-written in .hql file |
| Execution Style | One query at a time | Multiple queries in one go |
| Use Case | Testing, quick tasks | Automation, batch processing |
| Example Command | SELECT \* FROM students; | hive -f student\_queries.hql |
| Flexibility | Very flexible (interactive) | Less flexible during execution |