**Exercise 5: Advanced Data Transformations using Pig**

**Aim:**

To learn how to perform advanced data transformations using Pig, including joins and grouping operations.

**Procedure:**

* Start the Pig CLI.
* Load the dataset into Pig with the appropriate schema.
* Filter the dataset to include only relevant records.
* Group the data by a specified attribute.
* Perform joins between two datasets based on a common key.
* Execute aggregate functions to analyze the grouped data.
* Store the results in a new dataset.

**Program:**

**-- Load the product data**

product\_data = LOAD 'product\_data.csv' USING PigStorage(',')

AS (product\_id:chararray, product\_name:chararray, price:float, category:chararray);

**-- Load the sales data**

sales\_data = LOAD 'sales\_data.csv' USING PigStorage(',')

AS (order\_id:int, product\_id:chararray, quantity:int, order\_date:chararray);

**-- Filter products in the 'Electronics' category**

filtered\_products = FILTER product\_data BY category == 'Electronics';

**-- Group sales data by product\_id**

grouped\_sales = GROUP sales\_data BY product\_id;

**-- Join filtered products with grouped sales**

joined\_data = JOIN filtered\_products BY product\_id, grouped\_sales BY product\_id;

**-- Calculate total sales per product**

total\_sales = FOREACH joined\_data GENERATE

filtered\_products::product\_name,

SUM(sales\_data.quantity \* filtered\_products.price) AS total\_sales;

**-- Store the result in a new dataset**

STORE total\_sales INTO 'total\_sales\_data' USING PigStorage(',');

**Sample Dataset:**

**product\_data.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| product\_id | product\_name | price | Category |
| P001 | Laptop | 500.0 | Electronics |
| P002 | Smartphone | 300.0 | Electronics |
| P003 | Refrigerator | 700.0 | Appliances |
| P004 | Headphones | 100.0 | Electronics |
| P005 | Microwave | 150.0 | Appliances |

**sales\_data.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| order\_id | product\_id | quantity | order\_date |
| 1 | P001 | 2 | 2024-01-15 |
| 2 | P002 | 1 | 2024-02-05 |
| 3 | P003 | 1 | 2024-01-25 |
| 4 | P004 | 5 | 2024-03-12 |
| 5 | P005 | 3 | 2024-01-30 |

**Output:**

* **Filtered Products in Electronics Category:**

|  |  |  |  |
| --- | --- | --- | --- |
| product\_id | product\_name | price | Category |
| P001 | Laptop | 500.0 | Electronics |
| P002 | Smartphone | 300.0 | Electronics |
| P004 | Headphones | 100.0 | Electronics |

* **Total Sales per Product:**

|  |  |
| --- | --- |
| product\_name | total\_sales |
| Laptop | 1000.0 |
| Smartphone | 300.0 |
| Headphones | 500.0 |

**Result:**

Successfully executed advanced data transformations in Pig, including filtering, grouping, joining datasets, and calculating total sales for products in the Electronics category. The results were stored in a new dataset for further analysis.

**Exercise 5.1: Advanced Data Transformations using Pig**

This exercise focuses on advanced data transformations using Pig, where you will perform tasks such as loading datasets, calculating revenue, filtering data, grouping by categories, and identifying top products based on revenue.

**Dataset:**

* **products.csv**: { product\_id, product\_name, category, price }
* **sales.csv**: { sale\_id, product\_id, customer\_id, quantity, sale\_date }

**Tasks & Pig Script:**

1. **Load the Datasets:**
   * Load the products.csv and sales.csv files into Pig using the appropriate schema.

pig

Copy code

-- Load products dataset

products = LOAD 'products.csv' USING PigStorage(',')

AS (product\_id:chararray, product\_name:chararray, category:chararray, price:float);

-- Load sales dataset

sales = LOAD 'sales.csv' USING PigStorage(',')

AS (sale\_id:int, product\_id:chararray, customer\_id:chararray, quantity:int, sale\_date:chararray);

1. **Calculate Total Revenue Per Sale:**
   * Calculate the total revenue for each sale by multiplying the quantity sold by the product price.

pig

Copy code

-- Join products and sales datasets to calculate total revenue per sale

joined\_data = JOIN sales BY product\_id, products BY product\_id;

-- Calculate total revenue per sale

revenue\_per\_sale = FOREACH joined\_data GENERATE

sales::sale\_id,

sales::product\_id,

(sales::quantity \* products::price) AS total\_revenue;

1. **Filter Sales for High-Value Products:**
   * Filter out sales where the product price is greater than or equal to $500.

pig

Copy code

-- Filter sales where product price is >= 500

high\_value\_sales = FILTER joined\_data BY products::price >= 500;

1. **Group Sales by Category:**
   * Group the sales by product category and calculate the total quantity sold and total revenue for each category.

pig

Copy code

-- Group sales by product category

grouped\_by\_category = GROUP joined\_data BY products::category;

-- Calculate total quantity sold and total revenue per category

category\_totals = FOREACH grouped\_by\_category GENERATE

group AS category,

SUM(sales::quantity) AS total\_quantity,

SUM(sales::quantity \* products::price) AS total\_revenue;

1. **Join Products and Sales Data:**
   * Perform a join between the products and sales datasets based on product\_id to enrich the sales data with product details.

pig

Copy code

-- Already joined in step 2

-- "joined\_data" contains the result of joining sales and products datasets

1. **Calculate Top 3 Products by Revenue:**
   * For each product, calculate the total revenue and sort products based on the revenue. Extract the top 3 products.

pig

Copy code

-- Group sales by product to calculate total revenue per product

grouped\_by\_product = GROUP joined\_data BY products::product\_id;

-- Calculate total revenue per product

total\_revenue\_per\_product = FOREACH grouped\_by\_product GENERATE

group AS product\_id,

MAX(products::product\_name) AS product\_name,

SUM(sales::quantity \* products::price) AS total\_revenue;

-- Sort products by total revenue in descending order and take top 3

sorted\_products = ORDER total\_revenue\_per\_product BY total\_revenue DESC;

top\_3\_products = LIMIT sorted\_products 3;

1. **Identify Product Categories with Sales Above a Threshold:**
   * Filter out categories with total sales revenue greater than $1000.

pig

Copy code

-- Filter categories where total revenue is greater than $1000

high\_revenue\_categories = FILTER category\_totals BY total\_revenue > 1000;

**Sample Output:**

1. **Total Revenue Per Sale:**

yaml

Copy code

sale\_id product\_id total\_revenue

1 P001 1000.0

2 P002 600.0

3 P003 700.0

4 P004 500.0

1. **Filtered High-Value Sales (Products with price >= $500):**

Copy code

sale\_id product\_id product\_name price

1 P001 Laptop 500.0

3 P003 Refrigerator 700.0

1. **Grouped Sales by Category (with total quantity and revenue):**

yaml

Copy code

category total\_quantity total\_revenue

Electronics 10 2100.0

Appliances 4 850.0

1. **Top 3 Products by Revenue:**

yaml

Copy code

product\_name total\_revenue

Laptop 1000.0

Refrigerator 700.0

Smartphone 600.0

1. **Categories with Revenue Above $1000:**

yaml

Copy code

category total\_revenue

Electronics 2100.0

**Conclusion:**

In this exercise, you successfully performed advanced data transformations using Pig, including loading datasets, calculating total revenue, filtering, grouping by categories, joining data, and identifying top-performing products and categories based on revenue thresholds. This showcases the flexibility and power of Pig for processing large datasets in a streamlined and efficient manner.

**Exercise 6: Creating and Querying Tables in Hive with SQL**

**Aim:**

To learn how to create tables in Hive, load data into tables, and perform basic SQL queries such as SELECT, WHERE, and GROUP BY.

**Procedure:**

* Start the Hive CLI.
* Create a new database to organize your tables and switch to it.
* Create a new table to store sales data with appropriate columns and data types.
* Load the data into the sales\_data table from an HDFS location.
* Run SQL queries to analyze the data, including displaying all rows, calculating total sales per product, and filtering data by category.

**Program:**

CREATE DATABASE IF NOT EXISTS sales\_db;

USE sales\_db;

CREATE TABLE sales\_data (

order\_id INT,

product\_name STRING,

category STRING,

price FLOAT,

quantity INT,

order\_date STRING

) ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

LOAD DATA INPATH 'sales\_data.csv' INTO TABLE sales\_data;

**-- Display all data**

SELECT \* FROM sales\_data;

**-- Calculate total sales per product**

SELECT product\_name, SUM(price \* quantity) AS total\_salesFROM sales\_dataGROUP BY product\_name;

**-- Filter data for 'Electronics' category**

SELECT \* FROM sales\_dataWHERE category = 'Electronics';

**Sample Dataset:**

* **sales\_data.csv**{order\_id,product\_name,category,price,quantity,order\_date}

1,Laptop,Electronics,500.0,2,2024-01-15

2,Smartphone,Electronics,300.0,1,2024-02-05

3,Refrigerator,Appliances,700.0,1,2024-01-25

4,Headphones,Electronics,100.0,5,2024-03-12

5,Microwave,Appliances,150.0,3,2024-01-30

**Output:**

* **Displaying all rows from the table:**

| **order\_id** | **product\_name** | **category** | **price** | **quantity** | **order\_date** |
| --- | --- | --- | --- | --- | --- |
| 1 | Laptop | Electronics | 500.0 | 2 | 2024-01-15 |
| 2 | Smartphone | Electronics | 300.0 | 1 | 2024-02-05 |
| 3 | Refrigerator | Appliances | 700.0 | 1 | 2024-01-25 |
| 4 | Headphones | Electronics | 100.0 | 5 | 2024-03-12 |
| 5 | Microwave | Appliances | 150.0 | 3 | 2024-01-30 |

* **Total sales per product:**

| **product\_name** | **total\_sales** |
| --- | --- |
| Laptop | 1000.0 |
| Smartphone | 300.0 |
| Refrigerator | 700.0 |
| Headphones | 500.0 |
| Microwave | 450.0 |

* **Filtered data for the 'Electronics' category:**

| **order\_id** | **product\_name** | **category** | **price** | **quantity** | **order\_date** |
| --- | --- | --- | --- | --- | --- |
| 1 | Laptop | Electronics | 500.0 | 2 | 2024-01-15 |
| 2 | Smartphone | Electronics | 300.0 | 1 | 2024-02-05 |
| 4 | Headphones | Electronics | 100.0 | 5 | 2024-03-12 |

**Result:**

Successfully created a table in Hive, loaded sales data, and performed various SQL queries to analyze the data.

**Exercise 6.1: Creating and Querying Tables in Hive with SQL**

**DataSet**

* **products.csv**:{product\_id, product\_name, category, price} : 5Records
* **sales.csv**:{sale\_id, product\_id, customer\_id, quantity, sale\_date} :10Records

**1. Start Hive CLI**

To start the Hive CLI, you can open the terminal and type:

bash

Copy code

hive

**2. Create a Database**

Once the Hive CLI is running, create a new database:

sql

Copy code

CREATE DATABASE retail\_db;

**3. Use the Created Database**

Switch to the newly created database:

sql

Copy code

USE retail\_db;

**4. Create a Table for Products**

Now, create a table for products:

sql

Copy code

CREATE TABLE products (

product\_id INT,

product\_name STRING,

category STRING,

price FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**5. Load Data into Products Table**

Assume the products.csv file is in HDFS at the path /user/hive/warehouse/retail\_db/products.csv. You can load data into the table like this:

sql

Copy code

LOAD DATA INPATH '/user/hive/warehouse/retail\_db/products.csv'

INTO TABLE products;

**6. Create a Table for Sales**

Now, create a table for sales:

sql

Copy code

CREATE TABLE sales (

sale\_id INT,

product\_id INT,

customer\_id INT,

quantity INT,

sale\_date STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**7. Load Data into Sales Table**

Assume the sales.csv file is in HDFS at the path /user/hive/warehouse/retail\_db/sales.csv. Load data into the sales table:

sql

Copy code

LOAD DATA INPATH '/user/hive/warehouse/retail\_db/sales.csv'

INTO TABLE sales;

**8. Query to Select All Products**

To select all products:

sql

Copy code

SELECT \* FROM products;

**9. Query to Get Products with Price Greater than $500**

To get products with a price greater than $500:

sql

Copy code

SELECT \* FROM products

WHERE price > 500;

**10. Query to Count Total Sales by Product**

To count the total sales by each product:

sql

Copy code

SELECT p.product\_name, COUNT(s.sale\_id) AS total\_sales

FROM products p

JOIN sales s

ON p.product\_id = s.product\_id

GROUP BY p.product\_name;

**11. Query to Get Total Revenue by Product**

To get the total revenue by product (quantity sold \* product price):

sql

Copy code

SELECT p.product\_name, SUM(s.quantity \* p.price) AS total\_revenue

FROM products p

JOIN sales s

ON p.product\_id = s.product\_id

GROUP BY p.product\_name;

**Expected Output:**

1. **Select All Products**:

diff

Copy code

+------------+----------------+-----------+-------+

| product\_id | product\_name | category | price |

+------------+----------------+-----------+-------+

| 1 | Laptop | Electronics| 800 |

| 2 | Smartphone | Electronics| 600 |

| 3 | Headphones | Accessories| 150 |

| 4 | TV | Electronics| 1000 |

| 5 | Microwave | Home | 250 |

+------------+----------------+-----------+-------+

1. **Products with Price Greater than $500**:

diff

Copy code

+------------+----------------+-----------+-------+

| product\_id | product\_name | category | price |

+------------+----------------+-----------+-------+

| 1 | Laptop | Electronics| 800 |

| 2 | Smartphone | Electronics| 600 |

| 4 | TV | Electronics| 1000 |

+------------+----------------+-----------+-------+

1. **Count Total Sales by Product**:

diff

Copy code

+----------------+-------------+

| product\_name | total\_sales |

+----------------+-------------+

| Laptop | 3 |

| Smartphone | 2 |

| Headphones | 1 |

| TV | 2 |

| Microwave | 2 |

+----------------+-------------+

1. **Total Revenue by Product**:

diff

Copy code

+----------------+--------------+

| product\_name | total\_revenue|

+----------------+--------------+

| Laptop | 2400 |

| Smartphone | 1200 |

| Headphones | 150 |

| TV | 2000 |

| Microwave | 500 |

**Exercise 7: Implementing Partitioning in Hive and Querying Partitioned Data**

#### **Aim:**

To understand and apply partitioning in Hive to improve query performance and efficiently manage large datasets by partitioning based on a specific column, such as course.

#### **Procedure:**

* Start the Hive CLI.
* Create a new database to organize your tables and switch to it.
* Create a partitioned table to store student data, where the data is partitioned by the course column.
* Load data into the partitioned table, specifying the partition values for each course.
* Run queries on the partitioned table to verify that partitioning improves query performance and to check if partitioned data can be accessed correctly.

#### **Program:**

**-- Create a database**

CREATE DATABASE IF NOTEXISTSstudent\_db;

USE student\_db;

**-- Create a partitioned table**

CREATETABLEstudent\_data (

student\_idINT,

student\_name STRING,

ageINT,

grade STRING

)

PARTITIONED BY (course STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

STORED AS TEXTFILE;

**-- Load data into the partitioned table**

**-- You need to specify the partition value when loading data**

ALTER TABLE student\_data ADD PARTITION (course='Mathematics');

LOAD DATA INPATH 'student\_data\_mathematics.csv' INTO TABLE student\_data PARTITION (course='Mathematics');

ALTER TABLE student\_data ADD PARTITION (course='Science');

LOAD DATA INPATH '/user/hive/student\_data\_science.csv' INTO TABLE student\_data PARTITION (course='Science');

**--Display available partitions**

Show partitions student\_data

**-- Run queries on the partitioned table**

**-- Display all data from a specific partition (e.g., Mathematics)**

SELECT\*FROMstudent\_dataWHERE course ='Mathematics';

**-- Query across all partitions to get the count of students per course**

SELECT course, COUNT(student\_id) ASstudent\_countFROMstudent\_dataGROUPBY course;

**-- Filter data for students above a certain age across all partitions**

SELECT\*FROMstudent\_dataWHERE age >20;

#### **Sample Dataset:**

* **student\_data\_mathematics.csv**:{student\_id,student\_name,age,grade}

101,John Doe,22,A

102,Jane Smith,21,B

* **student\_data\_science.csv**:{student\_id,student\_name,age,grade}

201,Emily Davis,23,A

202,Michael Brown,24,B

#### **Output:**

* **Available Partitions**

**Course=Mathematics**

**Course=Science**

* **Displaying all data from a specific partition (e.g., Mathematics):**

| **student\_id** | **student\_name** | **age** | **grade** |
| --- | --- | --- | --- |
| 101 | John Doe | 22 | A |
| 102 | Jane Smith | 21 | B |

* **Count of students per course across all partitions:**

| **course** | **student\_count** |
| --- | --- |
| Mathematics | 2 |
| Science | 2 |

* **Filtered data for students above age 20 across all partitions:**

| **student\_id** | **student\_name** | **age** | **grade** | **course** |
| --- | --- | --- | --- | --- |
| 101 | John Doe | 22 | A | Mathematics |
| 102 | Jane Smith | 21 | B | Mathematics |
| 201 | Emily Davis | 23 | A | Science |
| 202 | Michael Brown | 24 | B | Science |

#### **Result:**

Successfully created a partitioned table in Hive, loaded student data into specific partitions based on the course column, and performed queries to verify that partitioning improves query performance and data management.

**Exercise 7.1: Implementing Partitioning in Hive and Querying Partitioned Data**

**-- Start the Hive CLI and create a database**

CREATE DATABASE IF NOT EXISTS products\_db;

USE products\_db;

**-- Create a partitioned table for products**

CREATE TABLE products (

product\_id INT,

product\_name STRING,

price FLOAT

)

PARTITIONED BY (category STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**-- Add and load data into the partitioned table for the Electronics partition**

ALTER TABLE products ADD PARTITION (category='Electronics');

LOAD DATA INPATH 'products-electronics.csv' INTO TABLE products PARTITION (category='Electronics');

**-- Add and load data into the partitioned table for the Wearables partition**

ALTER TABLE products ADD PARTITION (category='Wearables');

LOAD DATA INPATH 'products-wearables.csv' INTO TABLE products PARTITION (category='Wearables');

**-- Query to select all products**

SELECT \* FROM products;

**-- Query to filter products by category (e.g., Electronics)**

SELECT \* FROM products WHERE category = 'Electronics';

**-- Query to get the average price by category**

SELECT category, AVG(price) AS avg\_price FROM products GROUP BY category;

**Sample Dataset:**

1. products-electronics.csv

1,Laptop,1000

2,Smartphone,700

3,Tablet,400

4,Monitor,300

5,TV,1200

1. products-wearables.csv

6,Smartwatch,200

7,Fitness Tracker,250

8,Wireless Earbuds,150

9,VR Headset,400

10,Smart Ring,300

**Output:**

1. **Select all products:**

product\_id product\_name price category

1 Laptop 1000 Electronics

2 Smartphone 700 Electronics

3 Tablet 400 Electronics

4 Monitor 300 Electronics

5 TV 1200 Electronics

6 Smartwatch 200 Wearables

7 Fitness Tracker 250 Wearables

8 Wireless Earbuds 150 Wearables

9 VR Headset 400 Wearables

10 Smart Ring 300 Wearables

1. **Filter products by Electronics category:**

product\_id product\_name price category

1 Laptop 1000 Electronics

2 Smartphone 700 Electronics

3 Tablet 400 Electronics

4 Monitor 300 Electronics

5 TV 1200 Electronics

1. **Average price by category:**

category avg\_price

Electronics 920.00

Wearables 260.00

### Exercise 8: Implementing Indexing on a Hive Table and Querying Indexed Data

**Aim:**  
To demonstrate the process of creating an indexed table in Hive and querying data from it, understanding how indexing improves query optimization.

#### Procedure:

1. **Start Hive CLI**

bash

Copy code

hive

1. **Create a new database:**

sql

Copy code

CREATE DATABASE IF NOT EXISTS demo\_db;

USE demo\_db;

1. **Create a student\_data table:**

sql

Copy code

CREATE TABLE student\_data (

student\_id INT,

student\_name STRING,

course STRING,

grade STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

1. **Load data into the student\_data table:** Assuming the CSV file student\_data.csv is available locally:

sql

Copy code

LOAD DATA LOCAL INPATH 'student\_data.csv' INTO TABLE student\_data;

1. **Create an index on the course column:**

sql

Copy code

CREATE INDEX idx\_course ON TABLE student\_data (course) AS 'COMPACT';

1. **Rebuild the index:**

sql

Copy code

ALTER INDEX idx\_course ON student\_data REBUILD;

1. **Query the indexed data:**

sql

Copy code

SELECT \* FROM student\_data WHERE course = 'Mathematics';

1. **Display query execution plan:**

sql

Copy code

EXPLAIN SELECT \* FROM student\_data WHERE course = 'Mathematics';

1. **Drop the index and the table (cleanup):**

sql

Copy code

DROP INDEX IF EXISTS idx\_course ON student\_data;

DROP TABLE IF EXISTS student\_data;

#### Dataset: student\_data.csv

css

Copy code

student\_id,student\_name,course,grade

1,John Doe,Mathematics,A

2,Jane Smith,Science,B

3,Emily Davis,Mathematics,C

4,Michael Brown,History,A

5,Lucas White,Mathematics,B

6,Anna Johnson,Science,A

7,Paul Walker,History,B

8,Emma Wilson,Mathematics,A

9,Olivia Brown,Science,C

10,James Smith,History,A

#### Expected Output:

* **Query Results for Indexed Table:**  
  Students enrolled in "Mathematics":

css

Copy code

student\_id | student\_name | course | grade

-----------------------------------------------

1 | John Doe | Mathematics | A

3 | Emily Davis | Mathematics | C

5 | Lucas White | Mathematics | B

8 | Emma Wilson | Mathematics | A

* **Query Execution Plan:**  
  The EXPLAIN output shows the usage of the index:

sql

Copy code

+---------------------------------------+

| Table Scan |

+---------------------------------------+

| Table: student\_data (course) |

| Index: idx\_course |

| Filter: course = 'Mathematics' |

+---------------------------------------+

### Exercise 8.1: Implementing Indexing on Employees Table and Querying Indexed Data

#### Procedure:

1. **Start Hive CLI**

bash

Copy code

hive

1. **Create a new database:**

sql

Copy code

CREATE DATABASE IF NOT EXISTS company\_db;

USE company\_db;

1. **Create employees table:**

sql

Copy code

CREATE TABLE employees (

employee\_id INT,

employee\_name STRING,

department STRING,

salary FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

1. **Load data into the employees table:** Assuming the CSV file employees.csv is available locally:

sql

Copy code

LOAD DATA LOCAL INPATH 'employees.csv' INTO TABLE employees;

1. **Create an index on the department column:**

sql

Copy code

CREATE INDEX idx\_department ON TABLE employees (department) AS 'COMPACT';

1. **Rebuild the index:**

sql

Copy code

ALTER INDEX idx\_department ON employees REBUILD;

1. **Query to select all employees:**

sql

Copy code

SELECT \* FROM employees;

1. **Query to filter employees by department:**

sql

Copy code

SELECT \* FROM employees WHERE department = 'HR';

1. **Display query execution plan:**

sql

Copy code

EXPLAIN SELECT \* FROM employees WHERE department = 'HR';

1. **Drop the index and table (cleanup):**

sql

Copy code

DROP INDEX IF EXISTS idx\_department ON employees;

DROP TABLE IF EXISTS employees;

#### Dataset: employees.csv

employee\_id,employee\_name,department,salary

1,John Doe,HR,50000

2,Jane Smith,Finance,60000

3,Emily Davis,HR,55000

4,Michael Brown,IT,70000

5,Lucas White,Finance,58000

6,Anna Johnson,HR,52000

7,Paul Walker,IT,72000

8,Emma Wilson,Finance,59000

9,Olivia Brown,IT,68000

10,James Smith,HR,51000

#### Expected Output:

* **Query Results:**

markdown

Copy code

employee\_id | employee\_name | department | salary

----------------------------------------------

1 | John Doe | HR | 50000

3 | Emily Davis | HR | 55000

6 | Anna Johnson | HR | 52000

10 | James Smith | HR | 51000

* **Query Execution Plan:**  
  The EXPLAIN output would show how the index is used for optimizing the query:

sql

Copy code

+----------------------------------------+

| Table Scan |

+----------------------------------------+

| Table: employees (department) |

| Index: idx\_department |

| Filter: department = 'HR' |

+----------------------------------------+

**Exercise 9: Performing Joins and Aggregations on Large Datasets in Hive**

#### **Aim:**

To perform various types of joins and aggregations on large datasets in Hive, demonstrating how to combine and analyze data from multiple tables.

#### **Procedure:**

* Start the Hive CLI.
* Create a new database to organize your tables and switch to it.
* Create and load data into two tables: one for student information and one for course information.
* Perform different types of joins (e.g., inner join, left join) between the tables.
* Execute aggregation queries to summarize the data, such as calculating average grades or total enrollments per course.

#### **Program:**

**-- Create a database**

CREATE DATABASE IF NOTEXISTS school\_db;

USE school\_db;

**-- Create tables and load data**

CREATE TABLE students (

student\_idINT,

student\_name STRING,

course STRING,

grade STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

STORED AS TEXTFILE;

CREATE TABLE courses (

course\_name STRING,

instructor STRING,

creditsINT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

STORED AS TEXTFILE;

**-- Load data into the tables**

LOAD DATA INPATH 'students.csv'INTOTABLE students;

LOAD DATA INPATH ‘courses.csv'INTOTABLE courses;

**-- Inner Join: Get student names and course details for students enrolled in each course**

SELECTs.student\_name, c.course\_name, c.instructorFROM students s JOIN courses c ONs.course=c.course\_name;

**-- Left Join: Get all students and their corresponding course details, including students not enrolled in any course**

SELECTs.student\_name, c.course\_name, c.instructorFROMstudents sLEFTJOIN courses c

ONs.course=c.course\_name;

**-- Aggregation: Calculate the average grade per course**

SELECT course, AVG(

CASE grade

WHEN'A'THEN4

WHEN'B'THEN3

WHEN'C'THEN2

WHEN'D'THEN1

ELSE0

END

) ASaverage\_grade

FROM students

GROUPBY course;

**-- Aggregation: Count the number of students enrolled in each course**

SELECT course, COUNT(student\_id) ASstudent\_countFROM students GROUPBY course;

#### **Sample Dataset:**

* **students.csv**: {student\_id,student\_name,course,grade}

1,JohnDoe,Mathematics,A

2,JaneSmith,Science,B

3,EmilyDavis,Mathematics,C

4,MichaelBrown,History,A

* **courses.csv**: {course\_name,instructor,credits}

Mathematics,Dr. Smith,3

Science,Dr. Johnson,4

History,Dr. Lee,3

#### **Output:**

* **Inner Join Result:**

| **student\_name** | **course\_name** | **instructor** |
| --- | --- | --- |
| John Doe | Mathematics | Dr. Smith |
| Jane Smith | Science | Dr. Johnson |
| Emily Davis | Mathematics | Dr. Smith |
| Michael Brown | History | Dr. Lee |

* **Left Join Result:**

| **student\_name** | **course\_name** | **instructor** |
| --- | --- | --- |
| John Doe | Mathematics | Dr. Smith |
| Jane Smith | Science | Dr. Johnson |
| Emily Davis | Mathematics | Dr. Smith |
| Michael Brown | History | Dr. Lee |

* **Average Grade Per Course:**

| **course** | **average\_grade** |
| --- | --- |
| Mathematics | 3.0 |
| Science | 3.0 |
| History | 4.0 |

* **Student Count Per Course:**

| **course** | **student\_count** |
| --- | --- |
| Mathematics | 2 |
| Science | 1 |
| History | 1 |

#### **Result:**

Successfully performed various types of joins and aggregations on large datasets in Hive. Demonstrated how to combine data from multiple tables and summarize information effectively.

**Exercise 9.1: Performing Joins and Aggregations on Large Datasets in Hive**, follow the steps below to create tables for employees and departments, load data, perform joins, and execute aggregation queries.

**Steps:**

1. **Start the Hive CLI**: Open the Hive CLI by running:

bash

Copy code

hive

1. **Create a Database (if needed)**: If you don’t have a specific database created, you can create a new one using:

sql

Copy code

CREATE DATABASE IF NOT EXISTS company\_db;

USE company\_db;

1. **Create the Employees Table**: Create the employees table with columns for employee details:

sql

Copy code

CREATE TABLE employees (

employee\_id INT,

employee\_name STRING,

department\_id INT,

salary FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

1. **Load Data into the Employees Table**: Load data from the employees.csv file into the employees table:

sql

Copy code

LOAD DATA INPATH 'path/to/employees.csv' INTO TABLE employees;

1. **Create the Departments Table**: Create the departments table for department details:

sql

Copy code

CREATE TABLE departments (

department\_id INT,

department\_name STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

1. **Load Data into the Departments Table**: Load data from the departments.csv file into the departments table:

sql

Copy code

LOAD DATA INPATH 'path/to/departments.csv' INTO TABLE departments;

1. **Perform a Join between Employees and Departments**: Perform an **inner join** to retrieve the employee name and department name for each employee:

sql

Copy code

SELECT e.employee\_name, d.department\_name

FROM employees e

JOIN departments d

ON e.department\_id = d.department\_id;

1. **Calculate Average Salary by Department**: To calculate the average salary for each department:

sql

Copy code

SELECT d.department\_name, AVG(e.salary) AS avg\_salary

FROM employees e

JOIN departments d

ON e.department\_id = d.department\_id

GROUP BY d.department\_name;

1. **Get Total Employees per Department**: To get the count of employees in each department:

sql

Copy code

SELECT d.department\_name, COUNT(e.employee\_id) AS employee\_count

FROM employees e

JOIN departments d

ON e.department\_id = d.department\_id

GROUP BY d.department\_name;

**Sample Data:**

* **employees.csv**:

mathematica

Copy code

1,John Doe,101,60000

2,Jane Smith,102,72000

3,Emily Davis,101,50000

4,Michael Brown,103,80000

* **departments.csv**:

Copy code

101,Engineering

102,Marketing

103,Sales

**Output:**

1. **Inner Join Result**:

| **employee\_name** | **department\_name** |
| --- | --- |
| John Doe | Engineering |
| Jane Smith | Marketing |
| Emily Davis | Engineering |
| Michael Brown | Sales |

1. **Average Salary by Department**:

| **department\_name** | **avg\_salary** |
| --- | --- |
| Engineering | 55000 |
| Marketing | 72000 |
| Sales | 80000 |

1. **Total Employees per Department**:

| **department\_name** | **employee\_count** |
| --- | --- |
| Engineering | 2 |
| Marketing | 1 |
| Sales | 1 |

**Conclusion:**

This exercise demonstrated how to perform joins and aggregations in Hive using multiple tables (employees and departments), calculating average salary per department and the total number of employees per department.

**Exercise 10: Demonstrating the Use of Internal and External Tables in Hive**

#### **Aim:**

To understand and demonstrate the use of internal and external tables in Hive, including how to create, manage, and query both types of tables. This exercise also covers accessing the data of an external table after it has been dropped.

#### **Procedure:**

* Start the Hive CLI.
* (Optional) Create a new database to organize your tables and switch to it.
* Create and load an internal table with data.
* Create and load an external table with data from an external location.
* Run queries to compare the behavior of internal and external tables.
* Drop the internal and external tables.
* Attempt to access the dropped tables and their data to observe the differences in behavior.

#### **Sample Dataset:**

1. **students.csv**: {student\_id,student\_name,course,grade}

1,JohnDoe,Mathematics,A

2,JaneSmith,Science,B

3,EmilyDavis,Mathematics,C

4,MichaelBrown,History,A

#### **Program:**

**-- Create a database**

CREATE DATABASE IF NOTEXISTSdemo\_db;

USE demo\_db;

**-- Create an internal table**

CREATETABLEinternal\_table (

student\_idINT,

student\_name STRING,

course STRING,

grade STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

STORED AS TEXTFILE;

**-- Load data into the internal table**

LOAD DATA LOCAL INPATH 'students.csv'INTOTABLEinternal\_table;

**-- Create an external table**

CREATE**EXTERNAL**TABLEexternal\_table (

student\_idINT,

student\_name STRING,

course STRING,

grade STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

LOCATION **'/user/hive/external\_students**';

**-- Load data into the external table**

LOAD DATA LOCAL INPATH 'students.csv'INTOTABLEexternal\_table;

**-- Query internal table**

SELECT\*FROMinternal\_table;

**-- Query external table**

SELECT\*FROMexternal\_table;

**-- Drop the internal table (this will remove the data)**

DROPTABLEinternal\_table;

**-- Drop the external table (this will not remove the data)**

DROPTABLEexternal\_table;

**-- Accessing external table data directly in HDFS after dropping the table**

**-- Verify that the data still exists in HDFS**

**-- Use Hadoop commands to check the data location**

dfs-ls/user/hive/external\_students;

**-- Read data directly from HDFS to confirm it exists**

dfs-cat /user/hive/external\_students/student\_data.csv;

**-- To reuse the data, Create an external table that refers the existing data file path**

CREATE**EXTERNAL**TABLEexternal\_table (

student\_idINT,

student\_name STRING,

course STRING,

grade STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY','

LOCATION **'/user/hive/external\_students**';

**-- Query external table**

SELECT\*FROMexternal\_table;

#### **Output:**

* **Query Results for Internal Table:**

| **student\_id** | | **student\_name** | | **course** | | | **grade** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | John Doe | | Mathematics | | A |
| 2 | | Jane Smith | | Science | | B |
| 3 | | Emily Davis | | Mathematics | | C |
| 4 | | Michael Brown | | History | | A |

* **Query Results for External Table:**

| **student\_id** | **student\_name** | **course** | **grade** |
| --- | --- | --- | --- |
| 1 | John Doe | Mathematics | A |
| 2 | Jane Smith | Science | B |
| 3 | Emily Davis | Mathematics | C |
| 4 | Michael Brown | History | A |

* **Read data directly from HDFS to confirm it exists**

1,JohnDoe,Mathematics,A

2,JaneSmith,Science,B

3,EmilyDavis,Mathematics,C

4,MichaelBrown,History,A

#### **Result:**

Successfully demonstrated the creation, data loading, and querying of both internal and external tables in Hive. Illustrated the key differences in data management and persistence between internal and external tables, and showed how to access data from an external table even after it has been dropped.

**Exercise 10.1: Demonstrating the Use of Internal and External Tables in Hive**, follow the steps below to create internal and external tables for products, load data, and observe the differences in behavior between internal and external tables when they are dropped.

**Steps:**

1. **Start the Hive CLI**: Open the Hive CLI by running:

bash

Copy code

hive

1. **Create a Database (if needed)**: If you don’t already have a specific database, create a new one:

sql

Copy code

CREATE DATABASE IF NOT EXISTS product\_db;

USE product\_db;

1. **Create the Internal Table**: Create the pdt\_internal table for storing product data internally:

sql

Copy code

CREATE TABLE pdt\_internal (

product\_id INT,

product\_name STRING,

category STRING,

price FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

1. **Load Data into the Internal Table**: Load data from the Products.csv file into the pdt\_internal table:

sql

Copy code

LOAD DATA LOCAL INPATH 'path/to/Products.csv' INTO TABLE pdt\_internal;

1. **Query the Internal Table**: Run a query to verify that the data is loaded correctly:

sql

Copy code

SELECT \* FROM pdt\_internal;

1. **Create the External Table**: Create the pdt\_external table for external data storage:

sql

Copy code

CREATE EXTERNAL TABLE pdt\_external (

product\_id INT,

product\_name STRING,

category STRING,

price FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/user/hive/external\_products';

1. **Load Data into the External Table**: Load data from Products.csv into the pdt\_external table:

sql

Copy code

LOAD DATA LOCAL INPATH 'path/to/Products.csv' INTO TABLE pdt\_external;

1. **Query the External Table**: Run a query to check the data in the external table:

sql

Copy code

SELECT \* FROM pdt\_external;

1. **Drop the Tables (Cleanup)**:
   * Drop the **internal table** (this will remove the data along with the table):

sql

Copy code

DROP TABLE pdt\_internal;

* + Drop the **external table** (this will only remove the table definition, not the data):

sql

Copy code

DROP TABLE pdt\_external;

1. **Access Data from the External Table after Dropping**: Even though the external table is dropped, the data still exists in the external location (/user/hive/external\_products), which you can verify using HDFS commands.
   * List the contents of the external location:

bash

Copy code

dfs -ls /user/hive/external\_products;

* + Read the file directly to confirm the data exists:

bash

Copy code

dfs -cat /user/hive/external\_products/Products.csv;

1. **Recreate the External Table to Reuse the Data**: If you want to reuse the data, you can recreate the external table:

sql

Copy code

CREATE EXTERNAL TABLE pdt\_external (

product\_id INT,

product\_name STRING,

category STRING,

price FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/user/hive/external\_products';

1. **Query the Recreated External Table**: After recreating the external table, query it again to verify the data:

sql

Copy code

SELECT \* FROM pdt\_external;

**Sample Data:**

* **Products.csv**:

Copy code

1,Apple,Electronics,1000.50

2,Chair,Furniture,150.75

3,Shirt,Apparel,35.99

4,Laptop,Electronics,850.25

**Output:**

1. **Internal Table Query Result**:

| **product\_id** | **product\_name** | **category** | **price** |
| --- | --- | --- | --- |
| 1 | Apple | Electronics | 1000.50 |
| 2 | Chair | Furniture | 150.75 |
| 3 | Shirt | Apparel | 35.99 |
| 4 | Laptop | Electronics | 850.25 |

1. **External Table Query Result**:

| **product\_id** | **product\_name** | **category** | **price** |
| --- | --- | --- | --- |
| 1 | Apple | Electronics | 1000.50 |
| 2 | Chair | Furniture | 150.75 |
| 3 | Shirt | Apparel | 35.99 |
| 4 | Laptop | Electronics | 850.25 |

1. **HDFS Data** (after dropping external table):

Copy code

1,Apple,Electronics,1000.50

2,Chair,Furniture,150.75

3,Shirt,Apparel,35.99

4,Laptop,Electronics,850.25

**Conclusion:**

Successfully demonstrated the creation, data loading, and querying of both internal and external tables in Hive. Showed the key differences in how internal tables delete data upon dropping while external tables retain data in the external storage location, allowing for reuse even after the table is dropped.

4o