Program 6:

AIM: Develop a Pig Latin Scripts to sort, group, join, Project and filter the data

Source Code:

1. Sample Data

Let's assume we have two datasets:

- employees.txt (Employee ID, Name, Age, Department ID, Salary)
- departments.txt (Department ID, Department Name)

employees.txt (stored in HDFS at /user/cloudera/employees.txt)

```
101,John,30,1,50000
102,Sam,28,2,60000
103,Anna,32,1,75000
104,David,29,3,62000
105,Lily,27,2,58000
```

departments.txt (stored in HDFS at /user/cloudera/departments.txt)

```
1,HR
2,Finance
3,IT
```

Pig Latin Script

Save the script as employee_analysis.pig and execute it in Cloudera.

```
-- Load the employees dataset
employees = LOAD 'hdfs://localhost:9000/user/cloudera/employees.txt'
USING PigStorage(',')
AS (emp_id:int, name:chararray, age:int, dept_id:int, salary:int);


-- Load the departments dataset
departments = LOAD 'hdfs://localhost:9000/user/cloudera/departments.txt'
USING PigStorage(',')
AS (dept_id:int, dept_name:chararray);

-- 1. FILTER: Select employees with age greater than 28
filtered_employees = FILTER employees BY age > 28;

-- 2. PROJECT: Select only emp_id, name, and salary columns
projected_employees = FOREACH filtered_employees GENERATE emp_id, name, salary;

-- 3. SORT: Order employees by salary in descending order
sorted_employees = ORDER projected_employees BY salary DESC;
```

```
-- 4. GROUP: Group employees by department ID
grouped_by_department = GROUP employees BY dept_id;

-- 5. JOIN: Join employees with department names using dept_id
joined_data = JOIN employees BY dept_id, departments BY dept_id;

-- STORE results in HDFS
STORE sorted_employees INTO 'hdfs://localhost:9000/user/cloudera/output/sorted_employees' USING PigStorage(',');

STORE grouped_by_department INTO 'hdfs://localhost:9000/user/cloudera/output/grouped_by_department' USING PigStorage(',');
STORE joined_data INTO 'hdfs://localhost:9000/user/cloudera/output/joined_data' USING PigStorage(',');

-- DISPLAY the results on the screen
DUMP sorted_employees;
DUMP grouped_by_department;
DUMP joined_data;
```

Upload the data in HDFS

```
hdfs dfs -mkdir -p /user/cloudera
hdfs dfs -put employees.txt /user/cloudera/
hdfs dfs -put departments.txt /user/cloudera/
```

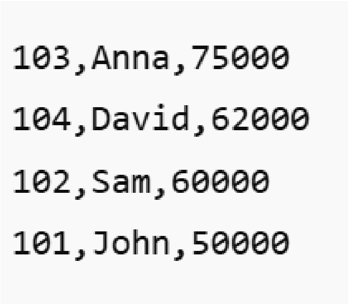
Run the Script

```
pig -x mapreduce employee_analysis.pig
```

ouput commands

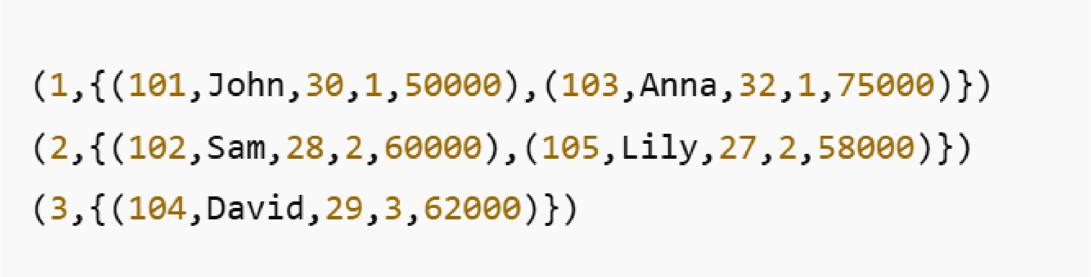
```
hdfs dfs -cat /user/cloudera/output/sorted_employees/part-r-00000
hdfs dfs -cat /user/cloudera/output/grouped_by_department/part-r-00000
hdfs dfs -cat /user/cloudera/output/joined_data/part-r-00000
```

OUTPUT :Sorted Employee by Salary



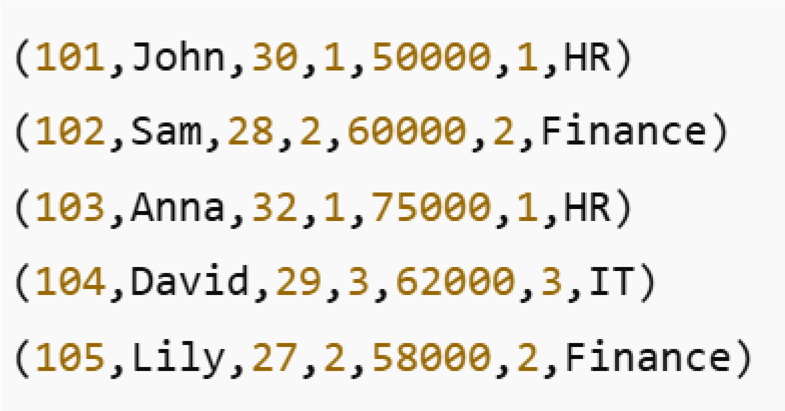
```
103,Anna,75000
104,David,62000
102,Sam,60000
101,John,50000
```

Grouped Employees by Departments



```
(1,{(101,John,30,1,50000),(103,Anna,32,1,75000)})
(2,{(102,Sam,28,2,60000),(105,Lily,27,2,58000)})
(3,{(104,David,29,3,62000)})
```

Joined Employees with Departments



```
(101,John,30,1,50000,1,HR)
(102,Sam,28,2,60000,2,Finance)
(103,Anna,32,1,75000,1,HR)
(104,David,29,3,62000,3,IT)
(105,Lily,27,2,58000,2,Finance)
```

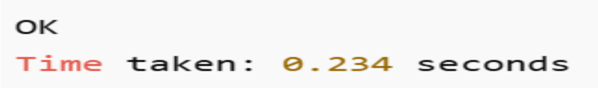
Program 7:

AIM: Use HIVE to create, alter, and drop databases, tables, views, functions and indexes

Create a Database:

CREATE DATABASE employee_db;

Output:

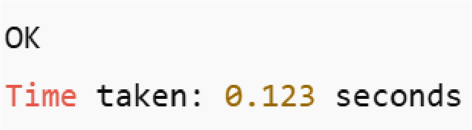


```
OK
Time taken: 0.234 seconds
```

Use the database:

USE employee_db;

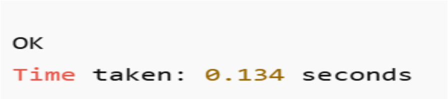
Output:



```
OK
Time taken: 0.123 seconds
```

Alter a Database:

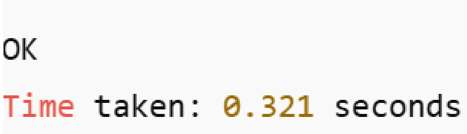
ALTER DATABASE employee_db SET DBPROPERTIES ('Owner'='Admin');



```
OK
Time taken: 0.134 seconds
```

Drop a Data Base

DROP DATABASE employee_db CASCADE;



```
OK
Time taken: 0.321 seconds
```

Create a Table

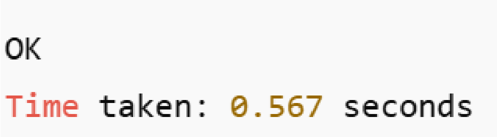
```
CREATE TABLE employees (
   emp_id INT,
   name STRING,
   age INT,
   dept_id INT,
   salary FLOAT
)
```

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

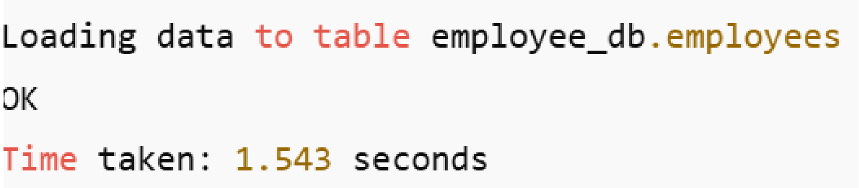
Output:



```
OK
Time taken: 0.567 seconds
```

LOAD DATA INPATH '/user/cloudera/employees.txt' INTO TABLE employees;

Output:



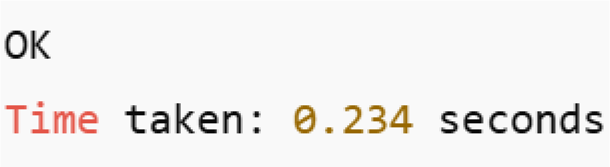
```
Loading data to table employee_db.employees
OK
Time taken: 1.543 seconds
```

Alter a Table

Add a New Column

ALTER TABLE employees ADD COLUMNS (email STRING);

Output:



```
OK
Time taken: 0.234 seconds
```

Rename:

ALTER TABLE employees RENAME TO employees_new;

Drop Table:

DROP TABLE employees_new;

Create a View

CREATE VIEW high_salary_employees AS

SELECT emp_id, name, salary

FROM employees

WHERE salary > 50000;

Alter a View:

ALTER VIEW high_salary_employees AS

SELECT emp_id, name, age, salary

FROM employees

WHERE salary > 60000;

Drop View:

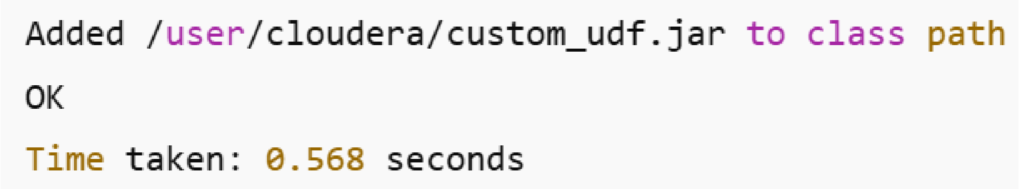
DROP VIEW high_salary_employees;

Create a Function

Add a JAR file containing a Java-based UDF:

ADD JAR /user/cloudera/custom_udf.jar;

CREATE FUNCTION to_upper AS 'com.example.hiveudf.ToUpperUDF';

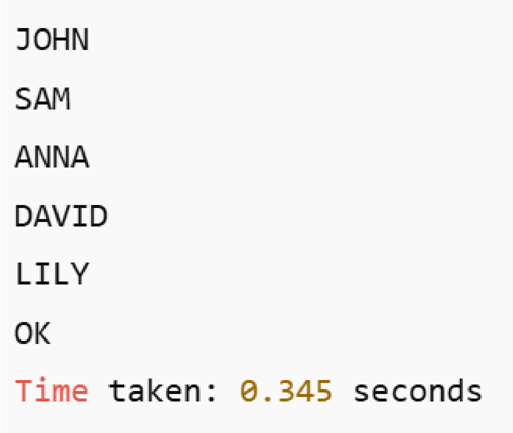


```
Added /user/cloudera/custom_udf.jar to class path
OK
Time taken: 0.568 seconds
```

Use the Function

SELECT to_upper(name) FROM employees;

Output:



```
JOHN
SAM
ANNA
DAVID
LILY
OK
Time taken: 0.345 seconds
```

Drop Function

DROP FUNCTION to_upper;

Create an Index

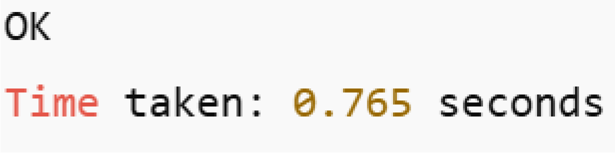
CREATE INDEX emp_dept_idx

ON TABLE employees (dept_id)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'

WITH DEFERRED REBUILD;

Output:



```
OK
Time taken: 0.765 seconds
```

Rebuild the Index:

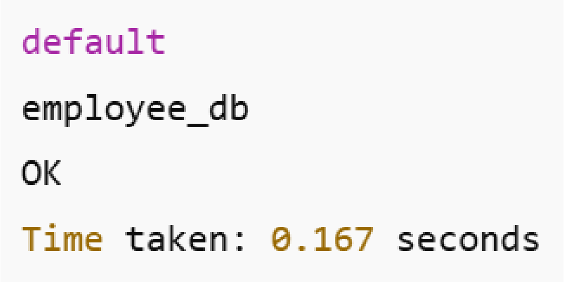
ALTER INDEX emp_dept_idx ON employees REBUILD;

Drop Index

DROP INDEX emp_dept_idx ON employees;

Check all tables in the current database:

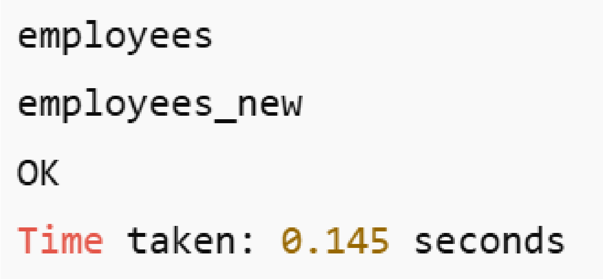
SHOW TABLES;



```
default
employee_db
OK
Time taken: 0.167 seconds
```

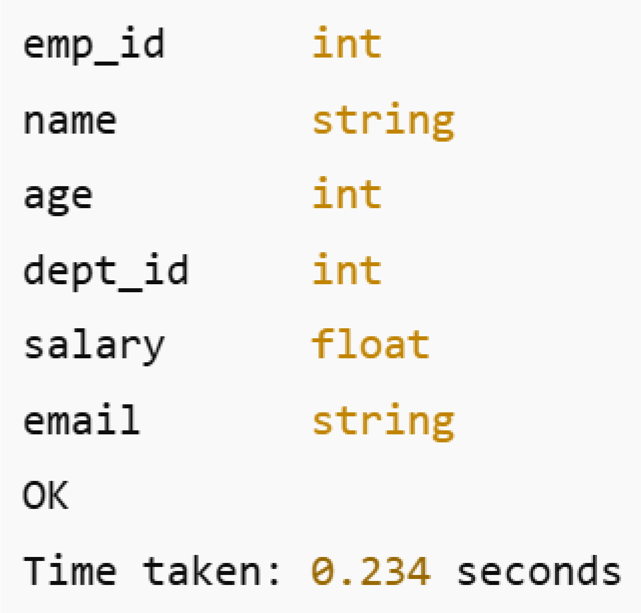
Check all tables in the current database:

SHOW TABLES;



```
employees
employees_new
OK
Time taken: 0.145 seconds
```

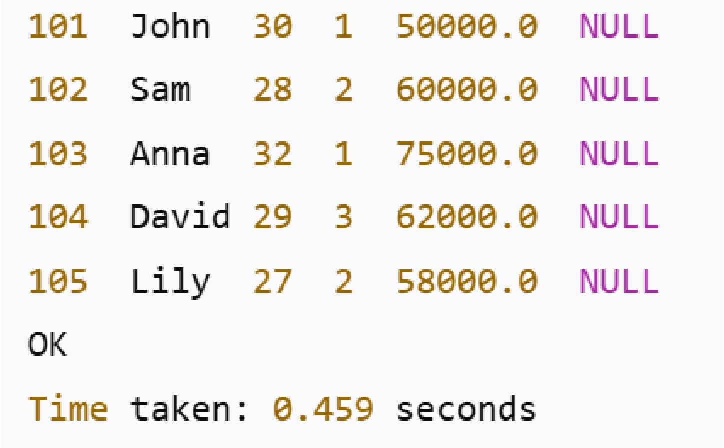
DESCRIBE employees;



```
emp_id      int
name        string
age         int
dept_id     int
salary      float
email       string
OK
Time taken: 0.234 seconds
```

Display the table data

SELECT * FROM employees LIMIT 5;



```
101  John  30  1  50000.0  NULL
102  Sam   28  2  60000.0  NULL
103  Anna  32  1  75000.0  NULL
104  David 29  3  62000.0  NULL
105  Lily  27  2  58000.0  NULL
OK
Time taken: 0.459 seconds
```

Program 8:

AIM: Implement word count program in Hadoop and spark

Source Code;

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.StringTokenizer;

public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Compile and Package the Java Code

```
javac -classpath `hadoop classpath` -d . WordCount.java
jar cf wc.jar WordCount*.class
```

Run the Hadoop Word Count Job

Upload the input file to HDFS:

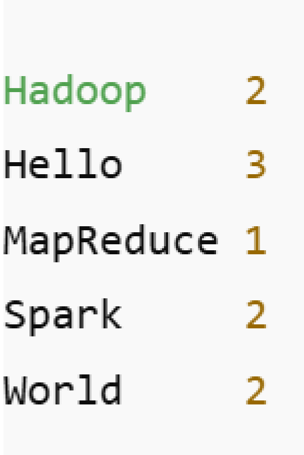
```
hdfs dfs -mkdir -p /user/cloudera/input
hdfs dfs -put sample.txt /user/cloudera/input/
```

Run the Map Reduce job

```
hadoop jar wc.jar WordCount /user/cloudera/input /user/cloudera/output
```

View the Output:

```
hdfs dfs -cat /user/cloudera/output/part-r-00000
```



```
Hadoop    2
Hello     3
MapReduce 1
Spark     2
World     2
```

Word Count in Apache Spark (PySpark)

1.Prepare the Input File

Upload the Sample.txt

hdfs dfs -mkdir -p /user/cloudera/input

hdfs dfs -put sample.txt /user/cloudera/input/

sample input File:

Hello Hadoop

Hello Spark

Hello World

Spark is fast

Hadoop is slow

2. Create a new script file using nano or vi:

```
nano wordcount.py
```

3.code

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("WordCount").getOrCreate()
```

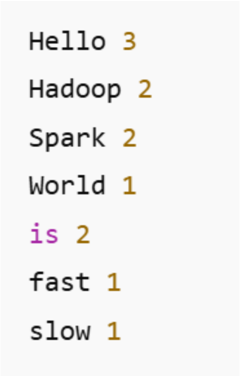
```
# Read text file from HDFS
text_file = spark.sparkContext.textFile("hdfs://localhost:9000/user/cloudera/input/sample.txt")

# Process data
word_counts = (text_file
               .flatMap(lambda line: line.split(" "))   # Split lines into words
               .map(lambda word: (word, 1))             # Map each word to (word, 1)
               .reduceByKey(lambda a, b: a + b))        # Reduce by key (word) and sum counts

# Save the output to HDFS
word_counts.saveAsTextFile("hdfs://localhost:9000/user/cloudera/output_spark")

# Print results
for word, count in word_counts.collect():
    print(f"{word}: {count}")

# Stop Spark Session
spark.stop()
```

3.Execute the Script

```
spark-submit wordcount.py
```

4.View the output:

```
hdfs dfs -ls /user/cloudera/output_spark

hdfs dfs -cat /user/cloudera/output_spark/part-00000
```



```
Hello 3
Hadoop 2
Spark 2
World 1
is 2
fast 1
slow 1
```

Program 9:

AIM: Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze the data and generate reports for sample data sets

Steps and Source Code:

1.Start Cloudera services:

```
sudo service cloudera-scm-server start
sudo service cloudera-scm-agent star
```

Check the status:

```
sudo service --status-all | grep cloudera
```

2. Access HUE Web Interface

Open your browser and navigate to:

http://localhost:8888

Login using HUE credentials:

Username: cloudera

Password: cloudera

3. Upload Sample Dataset to HDFS

Example Dataset: Employee Data (employees.csv)

```
id,name,department,salary
1,John,IT,70000
2,Alice,HR,60000
3,Bob,IT,75000
4,Charlie,Finance,80000
5,David,HR,62000
6,Eva,IT,72000
7,Frank,Finance,81000
8,Grace,HR,65000
```

Upload employees.csv to HDFS using HUE

1. **Navigate to HUE → File Browser → HDFS**
2. **Create a new directory** /user/cloudera/data
3. Click **Upload** → Select employees.csv → Upload it

Or through terminal

```
hdfs dfs -mkdir -p /user/cloudera/data
 hdfs dfs -put employees.csv /user/cloudera/data/
```

Create a Hive Table in HUE

Open HUE → Click Query Editors → Select Hive

4. Create a Hive table for the dataset:

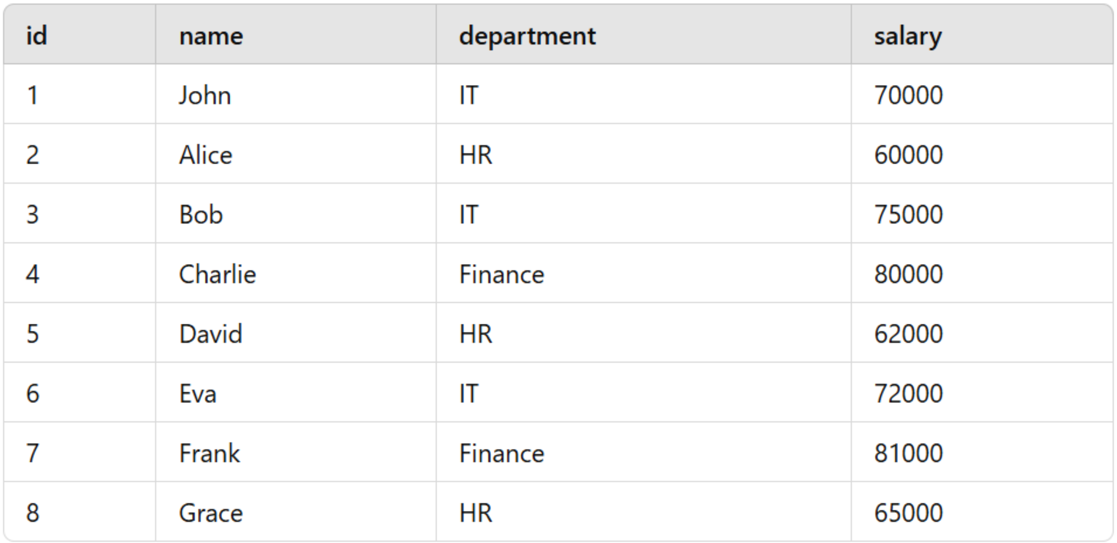
```
CREATE DATABASE IF NOT EXISTS company;
USE company;

CREATE TABLE employees (
    id INT,
    name STRING,
    department STRING,
    salary INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Load data into the Hive table:

```
LOAD DATA INPATH '/user/cloudera/data/employees.csv' INTO TABLE employees;
```

Verify the data:

```
SELECT * FROM employees;
```

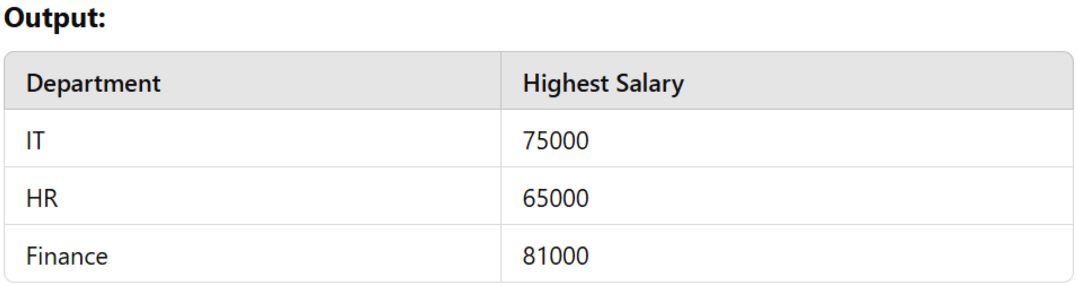


id	name	department	salary
1	John	IT	70000
2	Alice	HR	60000
3	Bob	IT	75000
4	Charlie	Finance	80000
5	David	HR	62000
6	Eva	IT	72000
7	Frank	Finance	81000
8	Grace	HR	65000

Data Analysis Using Hive Queries in HUE

Find the Highest Salary in Each Department

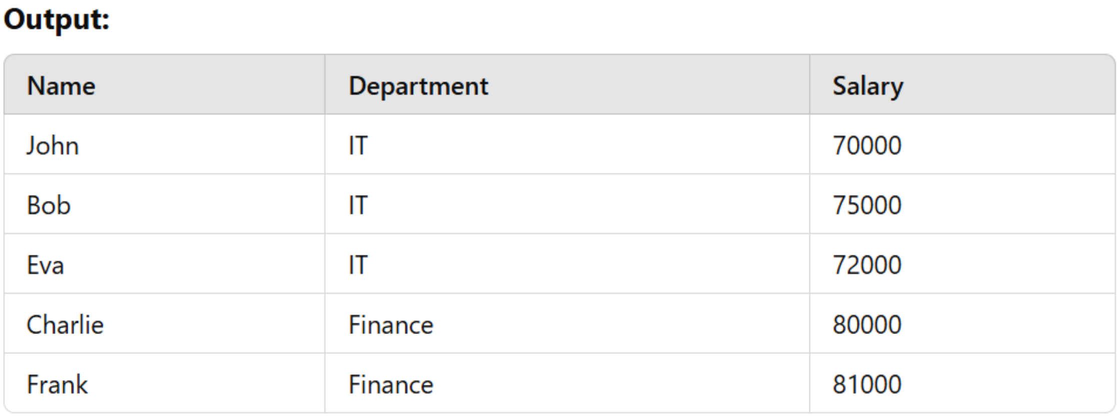
```
SELECT department, MAX(salary) AS highest_salary
FROM employees
GROUP BY department;
```



Output:

Department	Highest Salary
IT	75000
HR	65000
Finance	81000

Get Employees with Salary Greater Than 65000;



Output:

Name	Department	Salary
John	IT	70000
Bob	IT	75000
Eva	IT	72000
Charlie	Finance	80000
Frank	Finance	81000

Generate Reports in HUE

Step 1: Export Query Results

1. Run any of the above SQL queries in HUE Query Editor
2. Click Export → Choose format (CSV, Excel, JSON)
3. Download the report

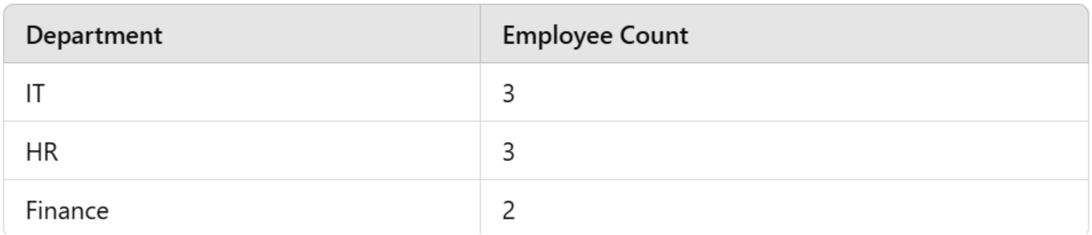
Step 2: Create HUE Dashboard for Visualization

1. Open HUE → Click Dashboard
2. Click Create New Dashboard
3. Click Add Widget → Select Chart Type (Bar Chart, Pie Chart, etc.)

Enter Query → Example for Employee Count:

SELECT department, COUNT(*) AS employee_count FROM employees GROUP BY department;

Click "Run Query" → The visualization will be generated



Department	Employee Count
IT	3
HR	3
Finance	2