Assignment-3: Pig and Hive

- Keshav Chandak(IMT2021003)
- Sunny Kaushik(IMT2021007)
- Muteeb Sheikh(IMT2021008)
- Rishi Nelapati(IMT2021076)

Question-1

Overview

This part focuses on designing and implementing data pipelines using Hive to efficiently analyze and clean educational datasets. The datasets include:

- `Course_Attendance.csv`
- `Enrollment_Data.csv`
- `GradeRosterReport.csv`

The primary tasks include defining schemas, creating Hive tables, loading data, and performing data cleaning operations using HiveQL.

Folder Structure

- **Assignment_3_NoSQL_PiG_Hive.pdf**: The assignment document detailing the tasks and requirements.
- **Course_Attendance.csv**: Contains raw data on course attendance.
- **Enrollment_Data_v7.csv**: Cleaned and processed enrollment data.
- **GradeRosterReport_v4.csv**: Cleaned and processed grade roster data.
- **create_and_load_tables.hql**: HiveQL script to define schemas, create tables, and load raw data.
- **data_cleaning.hql**: HiveQL script to clean and transform data.
- **readme.md**: Part (a) documentation (this file).

Steps and Scripts

1. Define Schemas and Create Tables

The `create_and_load_tables.hql` script defines the schema and creates Hive tables for each dataset:

Course Attendance Table

Schema:

- Course (STRING)
- Instructor (STRING)
- Name (STRING)

- Email_Id (STRING)
- Member_Id (STRING)
- Number_of_classes_attended (INT)
- Number_of_classes_absent (INT)
- Average_Attendance_Percentage (FLOAT)

Enrollment Data Table

Schema:

- Course_Type (STRING)
- Student_ID (STRING)
- Student_Name (STRING)
- Program (STRING)
- Batch (STRING)
- Period (STRING)
- Enrollment_Date (DATE)
- Primary_Faculty (STRING)
- Subject_Code_Name (STRING)
- Section (STRING)

Grade Roster Report Table

Schema:

- Academy_Location (STRING)
- Student_ID (STRING)
- Student_Status (STRING)
- Admission_ID (STRING)
- Admission_Status (STRING)
- Student_Name (STRING)
- Program_Name (STRING)
- Batch (STRING)
- Period (STRING)
- Subject_Code_Name (STRING)
- Section (STRING)
- Faculty_Name (STRING)
- Course_Credit (INT)
- Obtained_Marks_Grade (STRING)
- Out_of_Marks_Grade (STRING)
- Exam_Result (STRING)

---

2. Load Data into Hive Tables

The data from the CSV files is loaded into the corresponding Hive tables using the `LOAD DATA` command in the `create_and_load_tables.hql` script.

---

3. Data Cleaning

The `data_cleaning.hql` script performs the following cleaning operations:

- **Fill Missing Faculty Names**: Uses a self-join to fill in missing faculty names in `GradeRosterReport.csv`.
- **Remove Unnecessary Columns**: Drops unnecessary columns like `Serial No.`, `Status`, and `Academia+LMS` from `Enrollment_Data.csv`.
- **Update Program Name**: Extracts and updates the `Program Name` field from `Program Code/Name` in `GradeRosterReport.csv`.
- **Handle Multiple Faculty Entries**: Extracts a single, primary entry from the `Primary Faculty` column in `Enrollment_Data.csv`.

---

4. Final Output

The cleaned data is saved in:

- `Enrollment_Data_v7.csv`
- `GradeRosterReport_v4.csv`

These are ready for further analysis and reporting.

Usage

1. **Set Up Hive Environment**
   Ensure Apache Hive is properly installed and configured in your environment.

2. **Run Table Creation and Load Script**
   Execute `create_and_load_tables.hql` to define schemas and load the raw data.

```
hive -f create_and_load_tables.hql
```

3. **Run Data Cleaning Script** Execute `data_cleaning.hql` to perform all cleaning operations.

```
hive -f data_cleaning.hql
```

Question-2

We have created schema for the tables in Q1 along with basic cleaning, and now we want to create dimensional and fact tables for the dimensional modelling. To do so, we have to define their schemas as done in Q1 along with the fact table.
Since, it is a dimensional modelling, we have to pre-process the data and perform joins to form the fact table. Thus, some additional pre-processing will be required, specially on the subject names as all the three files have different structures of defining them altogether. Typically, fact tables are a formed as a result of join on student id and course/subject name. Thus, the course/subject name needs to be pre-processed in the hiveql, so it has similar contents which essentially belong to the same course. The data is from various sources like erp, codetantra and/or LMS, thus creating different values for the same subjects.
The typical pre-processing steps that we had one are in pre-processing.hql. The details of these pre-processing along with reasoning are as follows:

1. **Standardization of Text Format**

- **Convert to Lowercase:**
    - Using functions like LOWER() to convert all subject/course names to lowercase.
    - Reason: Ensures that differences in capitalization (e.g., "Maths" vs. "maths") do not create duplicate keys.

- **Trim Whitespaces:**
    - Apply the TRIM() function to remove leading and trailing spaces from text fields, especially trimming around delimiters like **/** and **-**, as evident in enrollment and grade data course fields.
    - Reason: Removes accidental spaces that could lead to mismatches during joins.

- **Uniform Delimiter Replacement:**
    - Use REGEXP_REPLACE() to standardize delimiters (like replacing hyphens, slashes, or multiple spaces with a single delimiter /). This is done to separate course code with course name.
    - Reason: Multiple representations (e.g., "CSE-101", "CSE/101", "CSE 101") get unified to a single format.

2. **Issues with attendance data:**

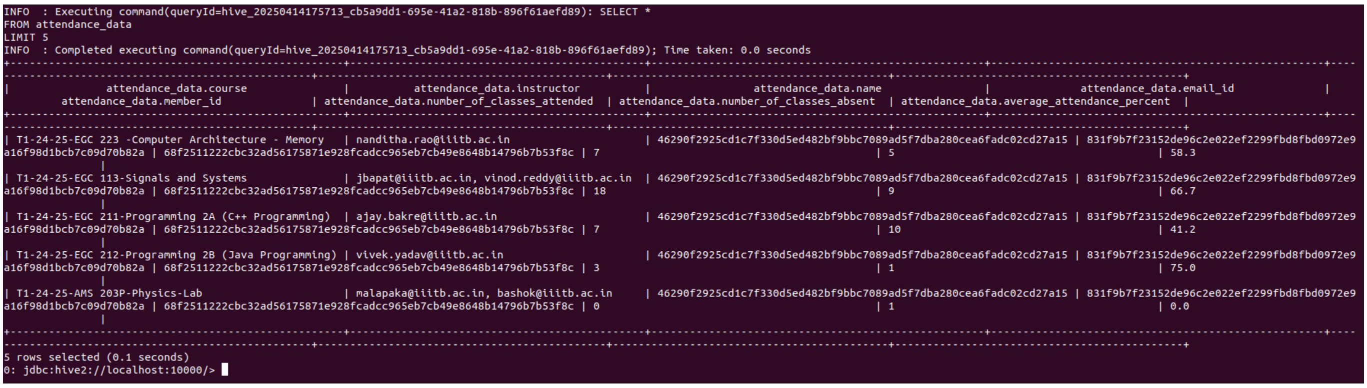
- **Uneven course_name in attendance data:**
    - Courses like "T1-24-25-AMS 211-Mathematics-3" are there in those fields, which should be ideally be "AMS 211-Mathematics-3" to maintain homogenity with other tables.
    - There are multiple rows which has email as **vishnu.raj@iiitb.org**. Those columns are essentially faculty meetings, and those rooms are removed and added to error_logs table, since they are erroneous values.
    - Some course names do not have any course code, and are essentially random staff/board meeting like **Audio testing Meeting by Prof Chandrashekar Ramanathan**. Those rows are removed from the table and added into error_logs table, since they are erroneous values.
- **Courses specifying batches:**
    - For courses with regard to first years, in some places they have mentioned batches they are teaching like **T1-24-25-GNL 101-English(BT1-IMT1-CSE)**. So, I have removed the contents of

the brackets except the ones which are programming courses like **T1-24-25-EGC 111-Programming 1A (C Programming)(BT1-IMT1)**.

3. **Final Course Details:**

- Now all the dimension tables have courses like **AMS 211/Mathematics-3**, meaning / is seperating the course code and course name.
    - Now, we could not merge directly with course codes since many rows are those of **programming and labs which have same course code**, but should have seperate grading and attendace records. Thus, standardisation of data across all the tables was required.

The hql queries for pre-processing is in **pre-processing.hql**.
Some images regarding sql queries done for pre-processing and data analytics are as follows:-



```
INFO  : Compiling command(queryId=hive_20250414181729_6604b463-7cff-4a92-b5af-62508cb5f1c5): CREATE TABLE merged_table (
serial_no INT,
course_type STRING,
student_id STRING,
student_name STRING,
program STRING,
batch STRING,
period STRING,
enrollment_date STRING,
primary_faculty STRING,
subject_code_name_enrollment STRING,
section_enrollment STRING,
academy_location STRING,
student_status STRING,
admission_id STRING,
admission_status STRING,
student_name_grade STRING,
program_name STRING,
batch_grade STRING,
period_grade STRING,
subject_code_name_grade STRING,
section_grade STRING,
faculty_name STRING,
course_credit INT,
obtained_marks_grade STRING,
out_of_marks_grade STRING,
exam_result STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
```

```
0: jdbc:hive2://localhost:10000/> INSERT OVERWRITE TABLE merged_table
. . . . . . . . . . . . . . . . > SELECT
. . . . . . . . . . . . . . . . >     e.serial_no,
. . . . . . . . . . . . . . . . >     e.course_type,
. . . . . . . . . . . . . . . . >     e.student_id,
. . . . . . . . . . . . . . . . >     e.student_name,
. . . . . . . . . . . . . . . . >     e.program,
. . . . . . . . . . . . . . . . >     e.batch,
. . . . . . . . . . . . . . . . >     e.period,
. . . . . . . . . . . . . . . . >     e.enrollment_date,
. . . . . . . . . . . . . . . . >     e.primary_faculty,
. . . . . . . . . . . . . . . . >     e.subject_code_name AS subject_code_name_enrollment,
. . . . . . . . . . . . . . . . >     e.section AS section_enrollment,
. . . . . . . . . . . . . . . . >     g.academy_location,
. . . . . . . . . . . . . . . . >     g.student_status,
. . . . . . . . . . . . . . . . >     g.admission_id,
. . . . . . . . . . . . . . . . >     g.admission_status,
. . . . . . . . . . . . . . . . >     g.student_name AS student_name_grade,
. . . . . . . . . . . . . . . . >     g.program_name,
. . . . . . . . . . . . . . . . >     g.batch AS batch_grade,
. . . . . . . . . . . . . . . . >     g.period AS period_grade,
. . . . . . . . . . . . . . . . >     g.subject_code_name AS subject_code_name_grade,
. . . . . . . . . . . . . . . . >     g.section AS section_grade,
. . . . . . . . . . . . . . . . >     g.faculty_name,
. . . . . . . . . . . . . . . . >     g.course_credit,
. . . . . . . . . . . . . . . . >     g.obtained_marks_grade,
. . . . . . . . . . . . . . . . >     g.out_of_marks_grade,
. . . . . . . . . . . . . . . . >     g.exam_result
. . . . . . . . . . . . . . . . > FROM
. . . . . . . . . . . . . . . . >     enrollment_data e
. . . . . . . . . . . . . . . . > LEFT JOIN
. . . . . . . . . . . . . . . . >     grade_roster g
. . . . . . . . . . . . . . . . >     ON e.student_id = g.student_id
. . . . . . . . . . . . . . . . >     AND e.student_name = g.student_name
. . . . . . . . . . . . . . . . >     AND e.subject_code_name = g.subject_code_name;
```

```
INFO  : Query ID = hive_20250414181736_4d9ba148-0f5f-49dd-9ab4-fa8f4229c41f
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-1:MAPRED] in serial mode
INFO  : Subscribed to counters: [] for queryId: hive_20250414181736_4d9ba148-0f5f-49dd-9ab4-fa8f4229c41f
INFO  : Session is already open
INFO  : Dag name: INSERT OVERWRITE TABL......subject_code_name (Stage-1)
INFO  : Setting tez.task.scale.memory.reserve-fraction to 0.30000001192092896
INFO  : HS2 Host: [ecb0cf9a7ce1], Query ID: [hive_20250414181736_4d9ba148-0f5f-49dd-9ab4-fa8f4229c41f], Dag ID: [dag_1744654494947_0001_2],
INFO  : Status: Running (Executing on YARN cluster with App id application_1744654494947_0001)

INFO  : Starting task [Stage-2:DEPENDENCY_COLLECTION] in serial mode
INFO  : Starting task [Stage-0:MOVE] in serial mode
INFO  : Loading data to table student_data.merged_table from file:/opt/hive/data/warehouse/student_data.db/merged_table/.hive-staging_hive_
INFO  : Starting task [Stage-3:STATS] in serial mode
INFO  : Executing stats task
----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 3 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 0.99 s
----------------------------------------------------------------------------------------------
INFO  : Table student_data.merged_table stats: [numFiles=1, numRows=3634, totalSize=1539904, rawDataSize=1536270, numFilesErasureCoded=0]
INFO  : Completed executing command(queryId=hive_20250414181736_4d9ba148-0f5f-49dd-9ab4-fa8f4229c41f); Time taken: 1.055 seconds
3,634 rows affected (1.342 seconds)
0: jdbc:hive2://localhost:10000/>
```

What we did was first write the python script for all the dimensional tables and pre-processed it such that on doing inner join, we will get maximum rows in the fact table. Now, the fact table has **2771 rows**, which would have been **less than 1000** without pre-processing. Then, we backtracked and form the hql queries and reported it in hql file.
The structure of fact tables is as follows:

```
CREATE TABLE IF NOT EXISTS fact_table (
    member_id STRING,
    course STRING,
    number_of_classes_attended INT,
    number_of_classes_absent INT,
    course_credit INT,
```

```
    average_attendance_percent FLOAT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count" = "1");
```

The structure of all the dimension tables as defined in Q1 are as follows:-

```
CREATE TABLE IF NOT EXISTS dim_enrollment_data (
  serial_no INT,
  course_type STRING,
  student_id STRING,
  student_name STRING,
  program STRING,
  batch STRING,
  period STRING,
  enrollment_date STRING,
  primary_faculty STRING,
  subject_code_name STRING,
  section STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");


CREATE TABLE IF NOT EXISTS dim_grade_roster (
    academy_location STRING,
    student_id STRING,
    student_status STRING,
    admission_id STRING,
    admission_status STRING,
    student_name STRING,
    program_name STRING,
    batch STRING,
    period STRING,
    section STRING,
    faculty_name STRING,
    course_credit INT,
    obtained_marks_grade STRING,
    out_of_marks_grade STRING,
    exam_result STRING,
    subject_code_name STRING
)
```

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");

CREATE TABLE IF NOT EXISTS dim_attendance_data (
    course STRING,
    instructor STRING,
    name STRING,
    email_id STRING,
    member_id STRING,
    number_of_classes_attended INT,
    number_of_classes_absent INT,
    average_attendance_percent FLOAT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

Firstly, we mount the csv files into the docker image folder, so as to use it for populating tables with the data.

```
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp attendance.csv hive4:/tmp/dim_attendance.csv
Successfully copied 2.36MB to hive4:/tmp/dim_attendance.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp enrollment.csv hive4:/tmp/dim_enrollment.csv
Successfully copied 868kB to hive4:/tmp/dim_enrollment.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp grade.csv hive4:/tmp/dim_grade.csv
Successfully copied 1.8MB to hive4:/tmp/dim_grade.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp fact_table_final1.csv hive4:/tmp/fact_table.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$
```

Then, we load the csv dataset into the above schema.
The code for loading it into hql table schemas is in load_queries.hql
The corresponding hql output after loading, and select statements are as follows:

```
8,495 rows selected (2.601 seconds)
```

```
3,101 rows selected (0.313 seconds)
```

```
4,477 rows selected (0.478 seconds)
```

Fater this is done, we try three HiveQl analytic queries. I have utilised these three queries since it covers the utility of all the numerical columns in the dimension and fact tables.

Before starting off, since we are utilising hive as a docker image due to various issues in the instllation as faced by many others, we are storing the tables everytime in our local system. So, first we load csv of dimensional tables and fact table onto the docker image: docker cp attendance.csv hive4:/tmp/dim_attendance.csv
docker cp enrollment.csv hive4:/tmp/dim_enrollment.csv docker cp grade.csv hive4:/tmp/dim_grade.csv

docker cp fact_table_final.csv hive4:/tmp/fact_table.csv

```
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp attendance.csv hive4:/tmp/dim_attendance.csv
Successfully copied 2.36MB to hive4:/tmp/dim_attendance.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp enrollment.csv hive4:/tmp/dim_enrollment.csv
Successfully copied 868kB to hive4:/tmp/dim_enrollment.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp grade.csv hive4:/tmp/dim_grade.csv
Successfully copied 1.8MB to hive4:/tmp/dim_grade.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$ docker cp fact_table_final1.csv hive4:/tmp/fact_table.csv
keshav-chandak@keshav-chandak-HP-Pavilion-Laptop-14-ec1xxx:~/Desktop/output Q2$
```

Query-1

Objective:
To compute the CGPA (Cumulative Grade Point Average) for each student based on the grade obtained and course credits.

Approach:

- Join `dim_grade_roster` and `fact_table` on `student_id` and `subject_code_name`.
- Use a weighted sum of grade points (based on institutional grading system) multiplied by `course_credit`.
- Divide total weighted grade points by total credits to derive CGPA.
- Order results by CGPA and then by total credits in descending order.

Query

```
SELECT
  g.student_id,
  SUM(g.course_credit) AS total_credits_completed,
  SUM(CASE
        WHEN g.obtained_marks_grade = 'A'  THEN 4.0 * g.course_credit
        WHEN g.obtained_marks_grade = 'A-' THEN 3.7 * g.course_credit
        WHEN g.obtained_marks_grade = 'B+' THEN 3.4 * g.course_credit
        WHEN g.obtained_marks_grade = 'B'  THEN 3.0 * g.course_credit
        WHEN g.obtained_marks_grade = 'B-' THEN 2.7 * g.course_credit
        WHEN g.obtained_marks_grade = 'C+' THEN 2.4 * g.course_credit
        WHEN g.obtained_marks_grade = 'C'  THEN 2.0 * g.course_credit
        WHEN g.obtained_marks_grade = 'D'  THEN 1.7 * g.course_credit
        ELSE 0.0
      END) / SUM(g.course_credit) AS cgpa
FROM dim_grade_roster g
JOIN fact_table f
  ON g.student_id = f.member_id
  AND g.subject_code_name = f.course
GROUP BY g.student_id
ORDER BY cgpa DESC, total_credits_completed DESC;
```

Use Case:
This query is essential for academic performance analysis, ranking students, and eligibility for honors or scholarships.

```
----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 3 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 2.34 s
----------------------------------------------------------------------------------------------
INFO  : Completed executing command(queryId=hive_20250414133532_77c2051d-3fbd-4fb1-92b1-c8a381e49198); Time taken: 5.485 seconds
+-------------------------------------------------------------------+--------------------------+----------------------+
|                          g.student_id                             | total_credits_completed  |         cgpa         |
+-------------------------------------------------------------------+--------------------------+----------------------+
| 006ebffbd115df9b6ef0e30a5cf33a86d6544a0bdb4b2e0c5f01addf199fbe8f | 28.0                     | 1.95                 |
| 01021eb63ad8ca36d35a6fd4ead1a931e4dc4b74999a5cf98c7900d8540c97ae | 8.0                      | 1.475                |
| 01104f71b9089725f8209bb949fb92555b90730dd4213561908386f1f0269a2b | 22.0                     | 3.0090909090909084   |
| 0133dbf630dcec089bb08ca3c4ec094ef4d383b985452330649c99a8acd5001a | 28.0                     | 2.2857142857142856   |
| 01e748f6f48344ff2bf1f20e5eb76b7411c8751af41798ff01d97fddae5d4234 | 12.0                     | 3.233333333333333    |
| 03c401666f88bd87df6663255493524ba394e8db25ba9af794c9febc0c03f12b | 26.0                     | 3.423076923076923    |
| 03e8af13a98d6f1287619ac0890c632fa203419b6f65a005c6c9d2f8478fe282 | 26.0                     | 2.8999999999999995   |
| 03f205b589909f0ea18950c4fac7e7d125a61a992e33556e8a3a8b0615ab0ab4 | 32.0                     | 2.21875              |
| 047236cffacc85ccec880c7b1b257e321af0ef1dd290899de7d6f9319decda76 | 32.0                     | 2.4                  |
| 075e7f21e42b4a5fb6e97df2bd17e65a0af0e5b11f547bfecc4ca690a2ece98e | 32.0                     | 3.1125               |
| 075f4288380a972f084731c23f3ae382165107e4c5a2a2cd85363d3a96046fed | 24.0                     | 1.4166666666666667   |
| 076449087afdae0e4172c37b1c10b693248751418392ad649ef57a52ad6e0e14 | 26.0                     | 2.2923076923076926   |
| 0821a962c2726e5df442dc86f74a371ce338c2436dd2e566f85f07883c5271c2 | 28.0                     | 3.5357142857142856   |
| 086ffcfc64ba1b317ff114d2d3dc632675ae75ee82788a8fa0b31e6be050394d | 28.0                     | 3.7642857142857147   |
| 08aa713e1d2c465191d99525020cf07f773e107a506a44229e7ffb500efd98dd | 26.0                     | 1.4615384615384615   |
```

```
| f7b37b09dd10930d9a0132e26d2830ca8677ad11d0f666db6fa0724fe57a1fff | 22.0                     | 2.0636363636363635   |
| f800cfdc8d739f2d384761d93f76d5f8d4d5c24f8b63f96556a754e6c1f86c8c | 24.0                     | 2.4166666666666665   |
| f853b03aaf270f8f8b6cab1ac5003975ffc2e14ce0f8d696e0f90d5c7e80421b | 32.0                     | 3.09375              |
| f9746d5926e1ef8be988f4a01b8897189476a4792deee63c7aa37e2d31b862a3 | 22.0                     | 2.9272727272727277   |
| f9a66b0efea2de779b86a5f40937feb83c080449e91f30eb7454b32d2d7295b6 | 22.0                     | 1.7363636363636366   |
| f9c3e40f66a95bff6864d2daff1a29d32b55d0034e5753ae9095585f0202314e | 36.0                     | 2.1055555555555556   |
| fa30950bc068d2bff9c983cb0853be94e0f15ba6fca5468c567db2ca275a7275 | 32.0                     | 2.09375              |
| fa97ea0f7b79d3347a03f5cdc5e96188d59f7e7098a0cec26b28d2f804fcf205 | 32.0                     | 2.03125              |
| fb82641a70b62444754aaca4126cf6d6566970fe04c5746b7f97312613a2f7fa | 32.0                     | 3.375                |
| fbd0443bf1e0d231601b6aff94a29877222aca65946506425863c35151df2084 | 22.0                     | 1.8636363636363635   |
| fc1e3958bf58979da2cd0fd53a5a62ba037f7eb11aebe44e08b2ea5f37cc2ffb | 36.0                     | 2.138888888888889    |
| fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce | 28.0                     | 3.1999999999999997   |
| fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c | 28.0                     | 2.9499999999999997   |
| fc5f93239ec1b27fd8bf7174a1f68e953d57e0b86e3c910135d02658a01a26ed | 26.0                     | 1.9                  |
| fcfa55660b5d441de2ef2e9b0b95b18c33a3f4853acdd231fea1eddd58dcc1ee | 22.0                     | 1.6363636363636365   |
| fcfbf656fb89ac105f2d0a8393c61f314a8449184a2f72349eef90b477c6c37b | 12.0                     | 1.9833333333333334   |
| fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5 | 28.0                     | 1.3642857142857143   |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | 28.0                     | 2.9285714285714284   |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | 28.0                     | 2.8928571428571423   |
| fedafcd150b9a17932760554a0ec9208266957a49da49214f4f9c7e1776f340d | 22.0                     | 2.8636363636363633   |
| ff6358e8fa8dce631d81990d463738796e3eb5cb545a29edad662cd92864cbfb | 8.0                      | 0.25                 |
| ffba274d8a68b64e86980a5d807a0057faa389d2c7a5857424d47dc960e8c434 | 12.0                     | 2.4166666666666665   |
| ffd48b5414c5c285193e34544de015ed643829e5bf39c79b107a5c41aaa612dd | 28.0                     | 2.857142857142857    |
| ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177 | 26.0                     | 2.4461538461538463   |
+-------------------------------------------------------------------+--------------------------+----------------------+
524 rows selected (6.54 seconds)
```

Time Elapsed: 6.54 seconds

Query-2

Objective:
To determine the number of students taught, average attendance, and maximum course credit for each faculty.

Approach:

- Join `dim_grade_roster` and `fact_table` on student and course.
- Filter for only those students who have passed (`exam_result = 'Pass'`).
- Aggregate data to:
  - Count distinct students per faculty.
  - Calculate average attendance using `average_attendance_percent`.
  - Determine the highest credit course taught by each faculty.

Use Case:
This helps analyze faculty engagement, workload distribution, and effectiveness in teaching based on student attendance and course difficulty.

Query:

```
SELECT
  g.faculty_name,
  COUNT(DISTINCT g.student_id) AS num_students,
  AVG(f.average_attendance_percent) AS avg_attendance,
  MAX(g.course_credit) AS max_course_credit
FROM fact_table f
JOIN dim_grade_roster g
  ON f.member_id = g.student_id
     AND f.course = g.subject_code_name
WHERE g.exam_result = 'Pass'
GROUP BY g.faculty_name;
```

g.faculty_name	num_students	avg_attendance	max_course_credit
Amit Chattopadhyay	159	84.39371069182388	4.0
Ashish Choudhury	6	80.73333333333333	4.0
Badrinath Ramamurthy	120	87.2225	2.0
G Srinivasa Raghavan	4	88.675	4.0
Jaya Sreevalsan Nair	1	70.8	4.0
Jyotsna Bapat	2	97.2	4.0
Karthikeyan Vaidyanathan	1	85.7	4.0
Kurian Polachan	91	86.91978021978026	4.0
Manisha Kulkarni	119	76.56722689075629	4.0
Meenakshi D Souza	3	86.26666666666667	4.0
Nanditha Rao	42	66.87857142857142	4.0
Pillalamarri Sridhar	160	80.71	4.0
Preeti Mudliar	33	80.2	4.0
Priyanka Das	6	77.18333333333335	4.0
Priyanka Sharma	280	66.44857142857144	2.0
Prof. Amrita Mishra	120	79.95333333333339	4.0
S Malapaka	166	80.00903614457827	4.0
Sachit Rao	150	74.7174311926605 7	4.0
Sakshi Arora	30	73.76666666666667	4.0
Srinath Srinivasa	3	88.90000000000002	4.0
Srinivas Vivek	198	77.55353535353527	4.0
Sujit Kumar Chakrabrati	160	86.43624999999997	2.0
Sushree Behera	4	81.825	4.0
Thangaraju B	149	92.32364864864864	4.0
Tulika Saha	120	73.93666666666667	2.0
Uttam Kumar	2	28.0	4.0
V Sridhar	313	83.2861271676299	4.0
Vinod Reddy	5	67.03999999999999	4.0
Vinu E V	59	87.05762711864405	4.0
Viswanath G	145	85.38620689655166	4.0

30 rows selected (0.932 seconds)

Time Elapsed:0.912 seconds

Query-3

Objective:
To identify students who have an attendance percentage below 75% in any course.

Approach:

- Join `dim_grade_roster` and `fact_table` on `student_id` and `subject_code_name`.
- Calculate overall attendance percentage as (classes_attended / (attended + absent)) * 100:
- Filter (`HAVING`) to return only those records with less than 75% attendance.

Query:

```
SELECT
    g.student_id,
    g.subject_code_name AS course,
    SUM(f.number_of_classes_attended) AS total_classes_attended,
    SUM(f.number_of_classes_absent)  AS total_classes_absent,
    (SUM(f.number_of_classes_attended) * 100.0) /
(SUM(f.number_of_classes_attended) + SUM(f.number_of_classes_absent)) AS
overall_attendance_percentage
FROM fact_table f
INNER JOIN dim_grade_roster g
  ON f.member_id = g.student_id
  AND f.course = g.subject_code_name
GROUP BY
    g.student_id,
    g.subject_code_name
HAVING
    (SUM(f.number_of_classes_attended) * 100.0) /
(SUM(f.number_of_classes_attended) + SUM(f.number_of_classes_absent)) < 75;
```

Use Case:
Used for academic warnings, eligibility checks for exams, and enforcing minimum attendance policies.

```
44444                |
| fcfbf656fb89ac105f2d0a8393c61f314a8449184a2f72349eef90b477c6c37b | VLS 864/Embedded Systems Design                | 16.0   | 8.0    | 66.666666666
66667                |
| fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5 | EGC 112/Programming 1B (Python Programming)    | 7.0    | 4.0    | 63.636363636
36363                |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | AMS 101/Probability & Statistics               | 68.0   | 32.0   | 68.0
                     |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | AMS 103/Calculus                               | 92.0   | 40.0   | 69.696969696
9697                 |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | EGC 102/Digital Design                         | 25.0   | 9.0    | 73.529411764
70588                |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | EGC 112/Programming 1B (Python Programming)    | 7.0    | 4.0    | 63.636363636
36363                |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | GNL 101/English                                | 7.0    | 4.0    | 63.636363636
36363                |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | AMS 101/Probability & Statistics               | 40.0   | 28.0   | 58.823529411
7647                 |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | AMS 103/Calculus                               | 48.0   | 56.0   | 46.153846153
84615                |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | EGC 102/Digital Design                         | 17.0   | 10.0   | 62.962962962
96296                |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | GNL 101/English                                | 0.0    | 1.0    | 0.0
                     |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | HSS 111/Economics-1                            | 2.0    | 10.0   | 16.666666666
666668               |
| fedafcd150b9a17932760554a0ec9208266957a49da49214f4f9c7e1776f340d | GNL 101/English                                | 7.0    | 3.0    | 70.0
                     |
| ff6358e8fa8dce631d81990d463738796e3eb5cb545a29edad662cd92864cbfb | VLS 505/System design with FPGA                | 6.0    | 3.0    | 66.666666666
66667                |
| ffba274d8a68b64e86980a5d807a0057faa389d2c7a5857424d47dc960e8c434 | AIM 511/Machine Learning                       | 0.0    | 4.0    | 0.0
                     |
| ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177 | EGC 111/Programming 1A (C Programming)         | 32.0   | 28.0   | 53.333333333
333336               |
| ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177 | GNL 101/English                                | 7.0    | 3.0    | 70.0
                     |
+------------------------------------------------------------------+------------------------------------------------+--------+--------+-------------
-----+
850 rows selected (1.23 seconds)
```

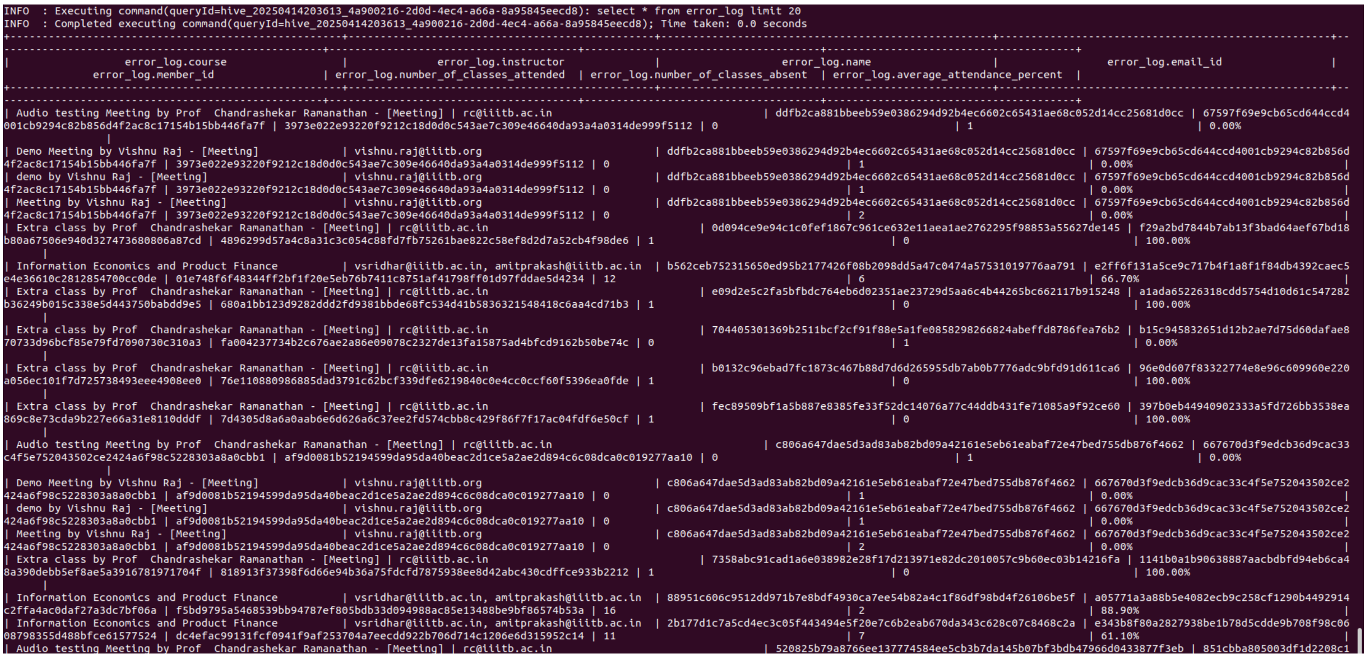
Time Elapsed:1.23 seconds

Note: You might be seeing that I am using only two tables in the queries, but since the fact table contains all the numerical data regarding attendance, thus **dim_attendance** table is not used. Similarly enrollment data had no numerical values, thus it is not part of join, as there cannot be any analytical query possible.

Error logs

The error_log.csv in the output folder of Q2 contains the inconsistent and erroneous data that we found out earlier. Since, the rest of the data was pre-processed and retained in the table, only erroneous values in the attendance table has been copied to the error_logs table.

```
INSERT INTO TABLE error_log
SELECT *
FROM a_data
WHERE (course NOT REGEXP '[0-9]' OR email_id = 'vishnu.raj@iiitb.org');

INSERT OVERWRITE DIRECTORY '/tmp/error_log_csv'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
SELECT * FROM error_log;

docker cp hive4:/tmp/error_log_csv/000000_0 ./error_log.csv
```



Question-3

Now on running the hiveql, we set Hive properties for dynamic bucketing:

1. SET hive.enforce.bucketing=true;
2. SET hive.enforce.sorting=true;

```
0: jdbc:hive2://localhost:10000/> SET hive.enforce.bucketing=true;
No rows affected (0.018 seconds)
0: jdbc:hive2://localhost:10000/> SET hive.enforce.sorting=true;
No rows affected (0.009 seconds)
0: jdbc:hive2://localhost:10000/>
```

Now, the schema for the optimised dimensional and fact tables using data modelling concepts, and the effective concept of how the partitioning and bucketing will increase query performance. The schema for the tables are as follows:-

```
CREATE TABLE IF NOT EXISTS dim_grade_roster_optimised (
    academy_location STRING,
    student_id STRING,
    student_status STRING,
    admission_id STRING,
    admission_status STRING,
    program_name STRING,
    batch STRING,
    period STRING,
    faculty_name STRING,
    course_credit INT,
    obtained_marks_grade STRING,
    out_of_marks_grade STRING,
    exam_result STRING,
    subject_code_name STRING
)
PARTITIONED BY (section STRING)
CLUSTERED BY (student_id) INTO 8 BUCKETS
STORED AS ORC;



CREATE TABLE IF NOT EXISTS dim_attendance_data_optimised (
    instructor STRING,
    name STRING,
    member_id STRING,
    number_of_classes_attended INT,
    number_of_classes_absent INT,
    average_attendance_percent FLOAT
)
PARTITIONED BY (course STRING)
CLUSTERED BY (member_id) INTO 8 BUCKETS
STORED AS ORC;
```

```
CREATE TABLE IF NOT EXISTS fact_table_optimised (
    member_id STRING,
    course STRING,
    number_of_classes_attended INT,
    number_of_classes_absent INT,
    course_credit INT,
    average_attendance_percent FLOAT
)
CLUSTERED BY (member_id) INTO 8 BUCKETS
STORED AS ORC;


CREATE TABLE IF NOT EXISTS dim_enrollment_data_optimised (
  serial_no INT,
  course_type STRING,
  student_id STRING,
  program STRING,
  batch STRING,
  period STRING,
  enrollment_date STRING,
  primary_faculty STRING,
  section STRING
)
PARTITIONED BY (subject_code_name STRING)
CLUSTERED BY (student_id) INTO 8 BUCKETS
STORED AS ORC;
```

You can see that,we have partitioned and bucketed it into optimised tables.
The justification for that is as follows:-

1. `dim_grade_roster_optimised`

- **Partitioned by `section`:**
  - Improves query performance for section-specific queries.
  - Ideal for filtering when analyzing grades per class or section.
- **Clustered by `student_id`:**
  - Boosts performance for student-wise joins (e.g., with attendance or fact table).
  - Enables efficient aggregation operations like CGPA computation.

2. `dim_attendance_data_optimised`

- **Partitioned by `course`:**
  - Optimizes queries analyzing attendance by course.
  - Reduces scan overhead for course-level reports.
- **Clustered by `member_id`:**
  - Enhances performance for per-student attendance analysis.
  - Useful in joins and aggregations involving individual students.

3. **`fact_table_optimised`**

- **No Partitioning:**
  - Acts as a central fact table joined with all dimensions.
  - Uniform query access across multiple attributes, so partitioning might not help.
- **Clustered by `member_id`:**
  - Speeds up joins with `dim_grade_roster_optimised` and `dim_attendance_data_optimised`.
  - Supports student-wise performance analysis (e.g., attendance + credit aggregation).

4. **`dim_enrollment_data_optimised`**

- **Partitioned by `subject_code_name`:**
  - Reduces data scanned for subject-specific queries or filters.
  - Common in analytics related to specific courses.
- **Clustered by `student_id`:**
  - Improves query performance for student-based tracking.
  - Useful for joins with grade and fact tables on `student_id`.

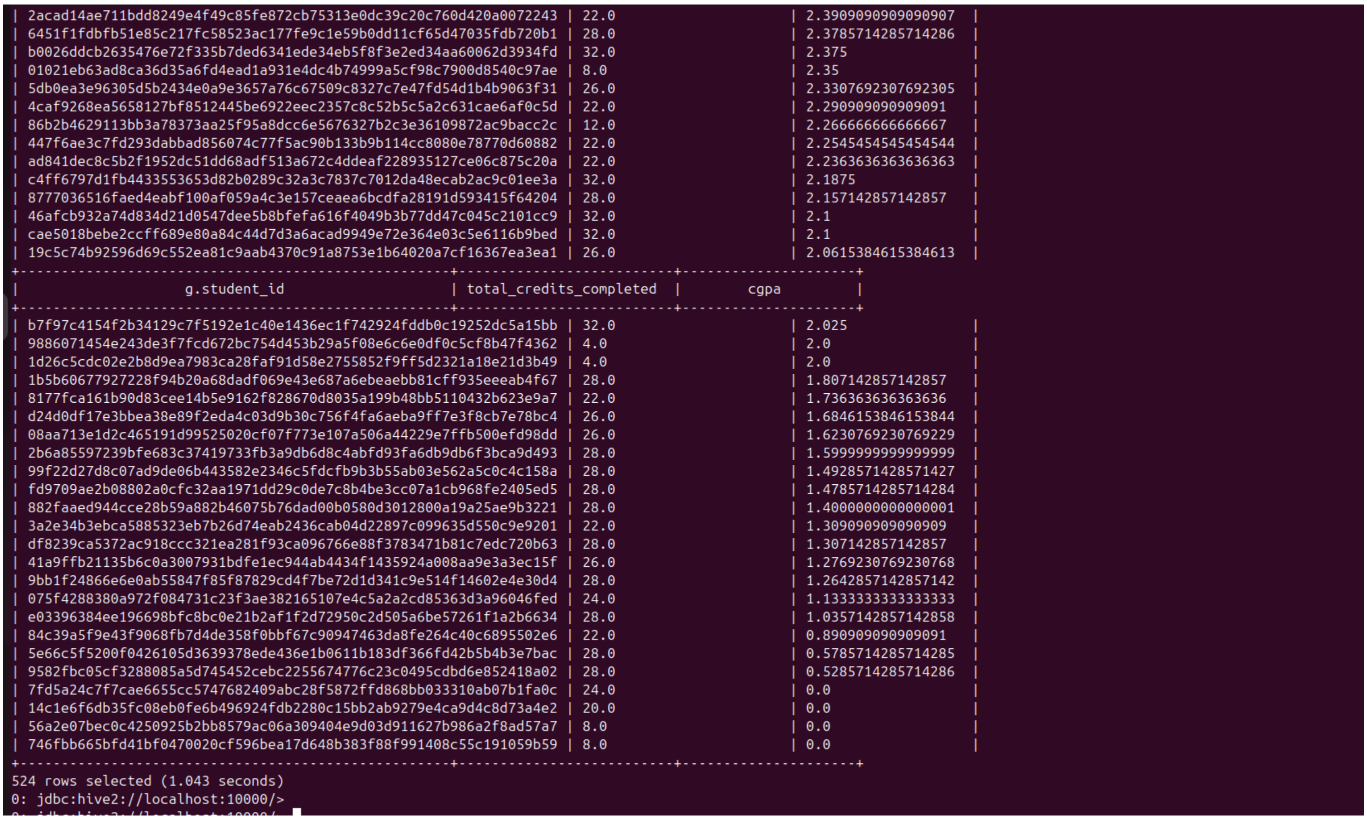
Query-1

Objective:
To compute the CGPA (Cumulative Grade Point Average) for each student based on the grade obtained and course credits.

Approach:

- Join `dim_grade_roster_optimised` and `fact_table_optimised` on `student_id` and `subject_code_name`.
- Use a weighted sum of grade points (based on institutional grading system) multiplied by `course_credit`.
- Divide total weighted grade points by total credits to derive CGPA.
- Order results by CGPA and then by total credits in descending order.
- Since, we have done effective clustering and partitioning, we get the results in much lesser time, greater query performance.

Use Case:
This query is essential for academic performance analysis, ranking students, and eligibility for honors or scholarships.



```
| 2acad14ae711bdd8249e4f49c85fe872cb75313e0dc39c20c760d420a0072243 | 22.0                    | 2.3909090909090907  |
| 6451f1fdbfb51e85c217fc58523ac177fe9c1e59b0dd11cf65d47035fdb720b1 | 28.0                    | 2.3785714285714286  |
| b0026ddcb2635476e72f335b7ded6341ede34eb5f8f3e2ed34aa60062d3934fd | 32.0                    | 2.375               |
| 01021eb63ad8ca36d35a6fd4ead1a931e4dc4b74999a5cf98c7900d8540c97ae | 8.0                     | 2.35                |
| 5db0ea3e96305d5b2434e0a9e3657a76c67509c8327c7e47fd54d1b4b9063f31 | 26.0                    | 2.3307692307692305  |
| 4caf9268ea5658127bf8512445be6922eec2357c8c52b5c5a2c631cae6af0c5d | 22.0                    | 2.290909090909091   |
| 86b2b4629113bb3a78373aa25f95a8dcc6e5676327b2c3e36109872ac9bacc2c | 12.0                    | 2.266666666666667   |
| 447f6ae3c7fd293dabbad856074c77f5ac90b133b9b114cc8080e78770d60882 | 22.0                    | 2.2545454545454544  |
| ad841dec8c5b2f1952dc51dd68adf513a672c4ddeaf228935127ce06c875c20a | 22.0                    | 2.2363636363636363  |
| c4ff6797d1fb4433553653d82b0289c32a3c7837c7012da48ecab2ac9c01ee3a | 32.0                    | 2.1875              |
| 8777036516faed4eabf100af059a4c3e157ceaea6bcdfa28191d593415f64204 | 28.0                    | 2.157142857142857   |
| 46afcb932a74d834d21d0547dee5b8bfefa616f4049b3b77dd47c045c2101cc9 | 32.0                    | 2.1                 |
| cae5018bebe2ccff689e80a84c44d7d3a6acad9949e72e364e03c5e6116b9bed | 32.0                    | 2.1                 |
| 19c5c74b92596d69c552ea81c9aab4370c91a8753e1b64020a7cf16367ea3ea1 | 26.0                    | 2.0615384615384613  |
+-------------------------------------------------+-------------------------+--------------------+
|                   g.student_id                  | total_credits_completed |        cgpa        |
+-------------------------------------------------+-------------------------+--------------------+
| b7f97c4154f2b34129c7f5192e1c40e1436ec1f742924fddb0c19252dc5a15bb | 32.0                    | 2.025               |
| 9886071454e243de3f7fcd672bc754d453b29a5f08e6c6e0df0c5cf8b47f4362 | 4.0                     | 2.0                 |
| 1d26c5cdc02e2b8d9ea7983ca28faf91d58e2755852f9ff5d2321a18e21d3b49 | 4.0                     | 2.0                 |
| 1b5b60677927228f94b20a68dadf069e43e687a6ebeaebb81cff935eeeab4f67 | 28.0                    | 1.807142857142857   |
| 8177fca161b90d83cee14b5e9162f828670d8035a199b48bb5110432b623e9a7 | 22.0                    | 1.736363636363636   |
| d24d0df17e3bbea38e89f2eda4c03d9b30c756f4fa6aeba9ff7e3f8cb7e78bc4 | 26.0                    | 1.6846153846153844  |
| 08aa713e1d2c465191d99525020cf07f773e107a506a44229e7ffb500efd98dd | 26.0                    | 1.6230769230769229  |
| 2b6a85597239bfe683c37419733fb3a9db6d8c4abfd93fa6db9db6f3bca9d493 | 28.0                    | 1.5999999999999999  |
| 99f22d27d8c07ad9de06b443582e2346c5fdcfb9b3b55ab03e562a5c0c4c158a | 28.0                    | 1.4928571428571427  |
| fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5 | 28.0                    | 1.4785714285714284  |
| 882faaed944cce28b59a882b46075b76dad00b0580d3012800a19a25ae9b3221 | 28.0                    | 1.4000000000000001  |
| 3a2e34b3ebca5885323eb7b26d74eab2436cab04d22897c099635d550c9e9201 | 22.0                    | 1.309090909090909   |
| df8239ca5372ac918ccc321ea281f93ca096766e88f3783471b81c7edc720b63 | 28.0                    | 1.307142857142857   |
| 41a9ffb21135b6c0a3007931bdfe1ec944ab4434f1435924a008aa9e3a3ec15f | 26.0                    | 1.2769230769230768  |
| 9bb1f24866e6e0ab55847f85f87829cd4f7be72d1d341c9e514f14602e4e30d4 | 28.0                    | 1.2642857142857142  |
| 075f4288380a972f084731c23f3ae382165107e4c5a2a2cd85363d3a96046fed | 24.0                    | 1.1333333333333333  |
| e03396384ee196698bfc8bc0e21b2af1f2d72950c2d505a6be57261f1a2b6634 | 28.0                    | 1.0357142857142858  |
| 84c39a5f9e43f9068fb7d4de358f0bbf67c90947463da8fe264c40c6895502e6 | 22.0                    | 0.890909090909091   |
| 5e66c5f5200f0426105d3639378ede436e1b0611b183df366fd42b5b4b3e7bac | 28.0                    | 0.5785714285714285  |
| 9582fbc05cf3288085a5d745452cebc2255674776c23c0495cdbd6e852418a02 | 28.0                    | 0.5285714285714286  |
| 7fd5a24c7f7cae6655cc5747682409abc28f5872ffd868bb033310ab07b1fa0c | 24.0                    | 0.0                 |
| 14c1e6f6db35fc08eb0fe6b496924fdb2280c15bb2ab9279e4ca9d4c8d73a4e2 | 20.0                    | 0.0                 |
| 56a2e07bec0c4250925b2bb8579ac06a309404e9d03d911627b986a2f8ad57a7 | 8.0                     | 0.0                 |
| 746fbb665bfd41bf0470020cf596bea17d648b383f88f991408c55c191059b59 | 8.0                     | 0.0                 |
+-------------------------------------------------+-------------------------+--------------------+
524 rows selected (1.043 seconds)
0: jdbc:hive2://localhost:10000/>
```

Time Elapsed:1.643 seconds

Query-2

Objective:
To determine the number of students taught, average attendance, and maximum course credit for each faculty.

Approach:

- Join `dim_grade_roster_optimised` and `fact_table_optimised` on student and course.
- Filter for only those students who have passed (`exam_result = 'Pass'`).
- Aggregate data to:
  - Count distinct students per faculty.
  - Calculate average attendance using `average_attendance_percent`.
  - Determine the highest credit course taught by each faculty.
- Since, we have done effective clustering and partitioning, we get the results in much lesser time, greater query performance.

Use Case:
This helps analyze faculty engagement, workload distribution, and effectiveness in teaching based on student attendance and course difficulty.

```
| Nanditha Rao              | 42            | 66.87857142857142  | 4.0                |
| Pillalamarri Sridhar      | 160           | 80.71              | 4.0                |
| Preeti Mudliar            | 33            | 80.2               | 4.0                |
| Priyanka Das              | 6             | 77.18333333333335  | 4.0                |
| Priyanka Sharma           | 280           | 66.44857142857144  | 2.0                |
| Prof. Amrita Mishra       | 120           | 79.95333333333339  | 4.0                |
| S Malapaka                | 166           | 80.00903614457827  | 4.0                |
| Sachit Rao                | 150           | 74.7174311926605 7 | 4.0                |
| Sakshi Arora              | 30            | 73.76666666666667  | 4.0                |
| Srinath Srinivasa         | 3             | 88.90000000000002  | 4.0                |
| Srinivas Vivek            | 198           | 77.55353535353527  | 4.0                |
| Sujit Kumar Chakrabrati   | 160           | 86.43624999999997  | 2.0                |
| Sushree Behera            | 4             | 81.825             | 4.0                |
| Thangaraju B              | 149           | 92.32364864864864  | 4.0                |
| Tulika Saha               | 120           | 73.93666666666667  | 2.0                |
| Uttam Kumar               | 2             | 28.0               | 4.0                |
| V Sridhar                 | 313           | 83.2861271676299   | 4.0                |
| Vinod Reddy               | 5             | 67.03999999999999  | 4.0                |
| Vinu E V                  | 59            | 87.05762711864405  | 4.0                |
| Viswanath G               | 145           | 85.38620689655166  | 4.0                |
+---------------------------+---------------+--------------------+--------------------+
30 rows selected (0.793 seconds)
```

Time Elapsed:0.793 seconds

Query-3

Objective:

To identify students who have an attendance percentage below 75% in any course.

Approach:

- Join `dim_grade_roster_optimised` and `fact_table_optimised` on `student_id` and `subject_code_name`.
- Calculate overall attendance percentage as (classes_attended / (attended + absent)) * 100:
- Filter (`HAVING`) to return only those records with less than 75% attendance.
- Since, we have done effective clustering and partitioning, we get the results in much lesser time, greater query performance.

Use Case:

Used for academic warnings, eligibility checks for exams, and enforcing minimum attendance policies.

```
| fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5 | EGC 112/Programming 1B (Python Programming)        | 7.0                | 4.0                | 63.636363636
36363              |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | AMS 101/Probability & Statistics                   | 68.0               | 32.0               | 68.0
                   |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | AMS 103/Calculus                                   | 92.0               | 40.0               | 69.696969696
9697               |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | EGC 102/Digital Design                             | 25.0               | 9.0                | 73.529411764
70588              |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | EGC 112/Programming 1B (Python Programming)        | 7.0                | 4.0                | 63.636363636
36363              |
| fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292 | GNL 101/English                                    | 7.0                | 4.0                | 63.636363636
36363              |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | AMS 101/Probability & Statistics                   | 40.0               | 28.0               | 58.823529411
7647               |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | AMS 103/Calculus                                   | 48.0               | 56.0               | 46.153846153
84615              |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | EGC 102/Digital Design                             | 17.0               | 10.0               | 62.962962962
96296              |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | GNL 101/English                                    | 0.0                | 1.0                | 0.0
                   |
| fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033 | HSS 111/Economics-1                                | 2.0                | 10.0               | 16.666666666
666668             |
| fedafcd150b9a17932760554a0ec9208266957a49da49214f4f9c7e1776f340d | GNL 101/English                                    | 7.0                | 3.0                | 70.0
                   |
| ff6358e8fa8dce631d81990d463738796e3eb5cb545a29edad662cd92864cbfb | VLS 505/System design with FPGA                    | 6.0                | 3.0                | 66.666666666
66667              |
| ffba274d8a68b64e86980a5d807a0057faa389d2c7a5857424d47dc960e8c434 | AIM 511/Machine Learning                           | 0.0                | 4.0                | 0.0
                   |
| ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177 | EGC 111/Programming 1A (C Programming)             | 32.0               | 28.0               | 53.333333333
333336             |
| ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177 | GNL 101/English                                    | 7.0                | 3.0                | 70.0
                   |
+-------------------------------------------------------------------+----------------------------------------------------+--------------------+--------------------+------------------------
-----+
850 rows selected (1.018 seconds)
0: jdbc:hive2://localhost:10000/>
```

Time Elapsed:1.018 seconds

Comparsion With and Without Bucketing

: Comparison of HiveQL Query Execution Time With and Without Bucketing and Partitioning

Query	Without (seconds)	With (seconds)
Query 1	6.54	1.643
Query 2	0.912	0.793
Query 3	1.23	1.018

Here, we can clearly see an increase in the query performance where the elapsed time has decreased by almost 4 times in some cases. But, the create and loading data into the optimised tables takes ample time, due to their internal pre-processing.

Question-4: Pig Query

Note: To run a pig query and make a time comparison, we can simply put:

```
time pig -x local sample.pig
```

The scripts that were used for querying are their in the **Q4 directory**.
The output from the pig query is there in the output directory

Query 1: CGPA Calculation per Student

Objective:
Calculate the CGPA for each student along with their total credits completed using the institutional grading system.

Steps and Logic:

1. **Loading Data:**
   - Load `enrollment.csv` into `enrollment_data` and drop the header row.
   - Load `grade.csv` into `grade_roster` and filter out its header row.
   - Load `attendance.csv` and `fact_table_final1.csv` similarly, ensuring that header rows are filtered out.
2. **Joining Datasets:**
   - Join `grade_roster` and `fact_table` on matching student and course identifiers (i.e. `student_id` with `member_id` and `subject_code_name` with `course`).
3. **Calculating Weighted Points:**
   - Use a `CASE` expression within a `FOREACH` to calculate the weighted points for each course based on the letter grade (e.g., `'A'` as 4.0, `'A-'` as 3.7, etc.) multiplied by the course credit.
4. **Grouping and Aggregation:**
   - Group the resulting data by `student_id` to aggregate values.
   - Compute the total credits by summing `course_credit` and the total weighted points.
   - Calculate the CGPA as the ratio of total weighted points to total credits.
5. **Ordering and Storing:**
   - Order the results by CGPA (in descending order) and by total credits completed.
   - Dump and store the output using PigStorage into the `/output/Query-1` directory.

Pig Script Excerpt:

```
-- Join grade_roster and fact_table
joined_data = JOIN grade_roster BY (student_id, subject_code_name),
```

```
fact_table BY (member_id, course);

-- Calculate weighted points
cgpa_data = FOREACH joined_data GENERATE
    grade_roster::student_id AS student_id,
    grade_roster::course_credit AS course_credit,
    (CASE grade_roster::obtained_marks_grade
        WHEN 'A'  THEN 4.0 * grade_roster::course_credit
        WHEN 'A-' THEN 3.7 * grade_roster::course_credit
        WHEN 'B+' THEN 3.4 * grade_roster::course_credit
        WHEN 'B'  THEN 3.0 * grade_roster::course_credit
        WHEN 'B-' THEN 2.7 * grade_roster::course_credit
        WHEN 'C+' THEN 2.4 * grade_roster::course_credit
        WHEN 'C'  THEN 2.0 * grade_roster::course_credit
        WHEN 'D'  THEN 1.7 * grade_roster::course_credit
        ELSE 0.0
    END) AS weighted_points;

-- Group and compute totals and CGPA
grouped_data = GROUP cgpa_data BY student_id;
result = FOREACH grouped_data {
    total_credits = SUM(cgpa_data.course_credit);
    total_weighted_points = SUM(cgpa_data.weighted_points);
    cgpa = total_weighted_points / total_credits;
    GENERATE group AS student_id, total_credits AS total_credits_completed,
cgpa AS cgpa;
}
ordered_result = ORDER result BY cgpa DESC, total_credits_completed DESC;

DUMP ordered_result;
STORE ordered_result INTO '/output/Query-1' USING PigStorage(',');
```

Image

```
(6451f1fdbfb51e85c217fc58523ac177fe9c1e59b0dd11cf65d47035fdb720b1,28,2.3785714285714286)
(b0026ddcb2635476e72f335b7ded6341ede34eb5f8f3e2ed34aa60062d3934fd,32,2.375)
(01021eb63ad8ca36d35a6fd4ead1a931e4dc4b74999a5cf98c7900d8540c97ae,8,2.35)
(5db0ea3e96305d5b2434e0a9e3657a76c67509c8327c7e47fd54d1b4b9063f31,26,2.3307692307692305)
(4caf9268ea5658127bf8512445be6922eec2357c8c52b5c5a2c631cae6af0c5d,22,2.2909090909090906)
(86b2b4629113bb3a78373aa25f95a8dcc6e5676327b2c3e36109872ac9bacc2c,12,2.2666666666666667)
(447f6ae3c7fd293dabbad856074c77f5ac90b133b9b114cc8080e78770d60882,22,2.2545454545454544)
(ad841dec8c5b2f1952dc51dd68adf513a672c4ddeaf228935127ce06c875c20a,22,2.2363636363636363)
(c4ff6797d1fb4433553653d82b0289c32a3c7837c7012da48ecab2ac9c01ee3a,32,2.1875)
(8777036516faed4eabf100af059a4c3e157ceaea6bcdfa28191d593415f64204,28,2.157142857142857)
(46afcb932a74d834d21d0547dee5b8bfefa616f4049b3b77dd47c045c2101cc9,32,2.1)
(cae5018bebe2ccff689e80a84c44d7d3a6acad9949e72e364e03c5e6116b9bed,32,2.1)
(19c5c74b92596d69c552ea81c9aab4370c91a8753e1b64020a7cf16367ea3ea1,26,2.0615384615384613)
(b7f97c4154f2b34129c7f5192e1c40e1436ec1f742924fddb0c19252dc5a15bb,32,2.025)
(9886071454e243de3f7fcd672bc754d453b29a5f08e6c6e0df0c5cf8b47f4362,4,2.0)
(1d26c5cdc02e2b8d9ea7983ca28faf91d58e2755852f9ff5d2321a18e21d3b49,4,2.0)
(1b5b60677927228f94b20a68dadf069e43e687a6ebeaebb81cff935eeeab4f67,28,1.807142857142857)
(8177fca161b90d83cee14b5e9162f828670d8035a199b48bb5110432b623e9a7,22,1.7363636363636366)
(d24d0df17e3bbea38e89f2eda4c03d9b30c756f4fa6aeba9ff7e3f8cb7e78bc4,26,1.6846153846153844)
(08aa713e1d2c465191d99525020cf07f773e107a506a44229e7ffb500efd98dd,26,1.623076923076923)
(2b6a85597239bfe683c37419733fb3a9db6d8c4abfd93fa6db9db6f3bca9d493,28,1.5999999999999999)
(99f22d27d8c07ad9de06b443582e2346c5fdcfb9b3b55ab03e562a5c0c4c158a,28,1.4928571428571427)
(fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5,28,1.4785714285714284)
(882faaed944cce28b59a882b46075b76dad00b0580d3012800a19a25ae9b3221,28,1.4000000000000001)
(3a2e34b3ebca5885323eb7b26d74eab2436cab04d22897c099635d550c9e9201,22,1.3090909090909089)
(df8239ca5372ac918ccc321ea281f93ca096766e88f3783471b81c7edc720b63,28,1.3071428571428572)
(41a9ffb21135b6c0a3007931bdfe1ec944ab4434f1435924a008aa9e3a3ec15f,26,1.2769230769230768)
(9bb1f24866e6e0ab55847f85f87829cd4f7be72d1d341c9e514f14602e4e30d4,28,1.2642857142857142)
(075f4288380a972f084731c23f3ae382165107e4c5a2a2cd85363d3a96046fed,24,1.1333333333333333)
(e03396384ee196698bfc8bc0e21b2af1f2d72950c2d505a6be57261f1a2b6634,28,1.0357142857142858)
(84c39a5f9e43f9068fb7d4de358f0bbf67c90947463da8fe264c40c6895502e6,22,0.890909090909091)
(5e66c5f5200f0426105d3639378ede436e1b0611b183df366fd42b5b4b3e7bac,28,0.5785714285714285)
(9582fbc05cf3288085a5d745452cebc2255674776c23c0495cdbd6e852418a02,28,0.5285714285714286)
(7fd5a24c7f7cae6655cc5747682409abc28f5872ffd868bb033310ab07b1fa0c,24,0.0)
(14c1e6f6db35fc08eb0fe6b496924fdb2280c15bb2ab9279e4ca9d4c8d73a4e2,20,0.0)
(746fbb665bfd41bf0470020cf596bea17d648b383f88f991408c55c191059b59,8,0.0)
(56a2e07bec0c4250925b2bb8579ac06a309404e9d03d911627b986a2f8ad57a7,8,0.0)
2025-04-14 23:43:35,602 [main] INFO  org.apache.pig.Main - Pig script completed in 5 seconds and 5 milliseconds (5005 ms)

real	0m6.705s
user	0m17.949s
sys	0m1.493s
```

Elapsed time:5.005 seconds

Query 2: Faculty-wise Summary of Attendance and Course Credit

Objective:
Calculate the number of students, average attendance percentage, and maximum course credit for each faculty by filtering for passed students.

Steps and Logic:

1. **Joining Datasets:**
    - Join the `fact_table` and `grade_roster` on student and course identifiers.
2. **Filtering:**
    - Filter the joined dataset for records where the exam result is `'Pass'` to focus on successful outcomes.
3. **Grouping by Faculty:**
    - Group the filtered records by `faculty_name`.
4. **Aggregation:**
    - Count distinct students per faculty.

- Compute the average attendance using the `average_attendance_percent` from the fact table.
- Determine the maximum course credit awarded for courses taught by each faculty.

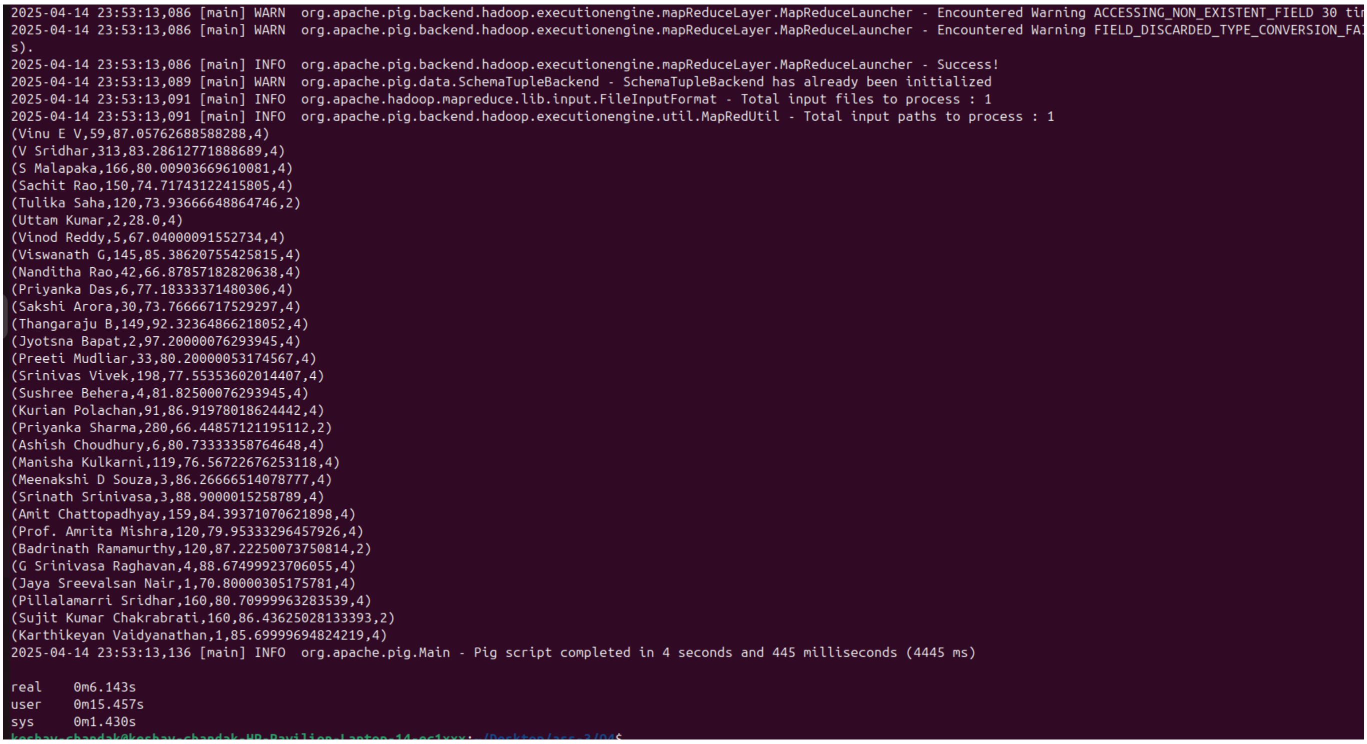
5. **Output:**

   - Dump the results and store them into `/output/Query-2`.

Pig Script Excerpt:

```
-- Join fact_table and grade_roster
joined_data = JOIN fact_table BY (member_id, course), grade_roster BY
(student_id, subject_code_name);

-- Filter for students who passed
filtered_data = FILTER joined_data BY grade_roster::exam_result == 'Pass';

-- Group by faculty_name and compute aggregates
grouped_data = GROUP filtered_data BY grade_roster::faculty_name;
result = FOREACH grouped_data {
    unique_students = DISTINCT filtered_data.grade_roster::student_id;
    GENERATE group AS faculty_name,
             COUNT(unique_students) AS num_students,
             AVG(filtered_data.fact_table::average_attendance_percent) AS
avg_attendance,
             MAX(filtered_data.grade_roster::course_credit) AS
max_course_credit;
}

DUMP result;
STORE result INTO '/output/Query-2' USING PigStorage(',');
```

Image



```
2025-04-14 23:53:13,086 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning ACCESSING_NON_EXISTENT_FIELD 30 ti
2025-04-14 23:53:13,086 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning FIELD_DISCARDED_TYPE_CONVERSION_FA
s).
2025-04-14 23:53:13,086 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2025-04-14 23:53:13,089 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2025-04-14 23:53:13,091 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2025-04-14 23:53:13,091 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Vinu E V,59,87.05762688588288,4)
(V Sridhar,313,83.28612771888689,4)
(S Malapaka,166,80.00903669610081,4)
(Sachit Rao,150,74.71743122415805,4)
(Tulika Saha,120,73.93666648864746,2)
(Uttam Kumar,2,28.0,4)
(Vinod Reddy,5,67.04000091552734,4)
(Viswanath G,145,85.38620755425815,4)
(Nanditha Rao,42,66.87857182820638,4)
(Priyanka Das,6,77.18333371480306,4)
(Sakshi Arora,30,73.76666717529297,4)
(Thangaraju B,149,92.32364866218052,4)
(Jyotsna Bapat,2,97.20000076293945,4)
(Preeti Mudliar,33,80.20000053174567,4)
(Srinivas Vivek,198,77.55353602014407,4)
(Sushree Behera,4,81.82500076293945,4)
(Kurian Polachan,91,86.91978018624442,4)
(Priyanka Sharma,280,66.44857121195112,2)
(Ashish Choudhury,6,80.73333358764648,4)
(Manisha Kulkarni,119,76.56722676253118,4)
(Meenakshi D Souza,3,86.26666514078777,4)
(Srinath Srinivasa,3,88.9000015258789,4)
(Amit Chattopadhyay,159,84.39371070621898,4)
(Prof. Amrita Mishra,120,79.95333296457926,4)
(Badrinath Ramamurthy,120,87.22250073750814,2)
(G Srinivasa Raghavan,4,88.67499923706055,4)
(Jaya Sreevalsan Nair,1,70.80000305175781,4)
(Pillalamarri Sridhar,160,80.70999963283539,4)
(Sujit Kumar Chakrabrati,160,86.43625028133393,2)
(Karthikeyan Vaidyanathan,1,85.69999694824219,4)
2025-04-14 23:53:13,136 [main] INFO  org.apache.pig.Main - Pig script completed in 4 seconds and 445 milliseconds (4445 ms)

real    0m6.143s
user    0m15.457s
sys     0m1.430s
```

Elapsed time:4.445 seconds

Query 3: Identify Low Attendance (Below 75%) per Student-Course

Objective:
Determine the attendance percentage for each student in each course and identify those records where attendance is below 75%.

Steps and Logic:

1. **Joining Datasets:**

   - Join `fact_table` with `grade_roster` on matching student and course identifiers.

2. **Grouping Data:**

   - Group the joined data by both `student_id` and `subject_code_name` to work at the granularity of each student's course.

3. **Attendance Calculation:**

   - Compute total classes attended and absent for each group.
   - Calculate the overall attendance percentage using the formula: (\text{attendance_percentage} = \frac{\text{total attended} \times 100}{\text{total attended} + \text{total absent}})

4. **Filtering:**

   - Filter out groups where the attendance percentage is less than 75%.

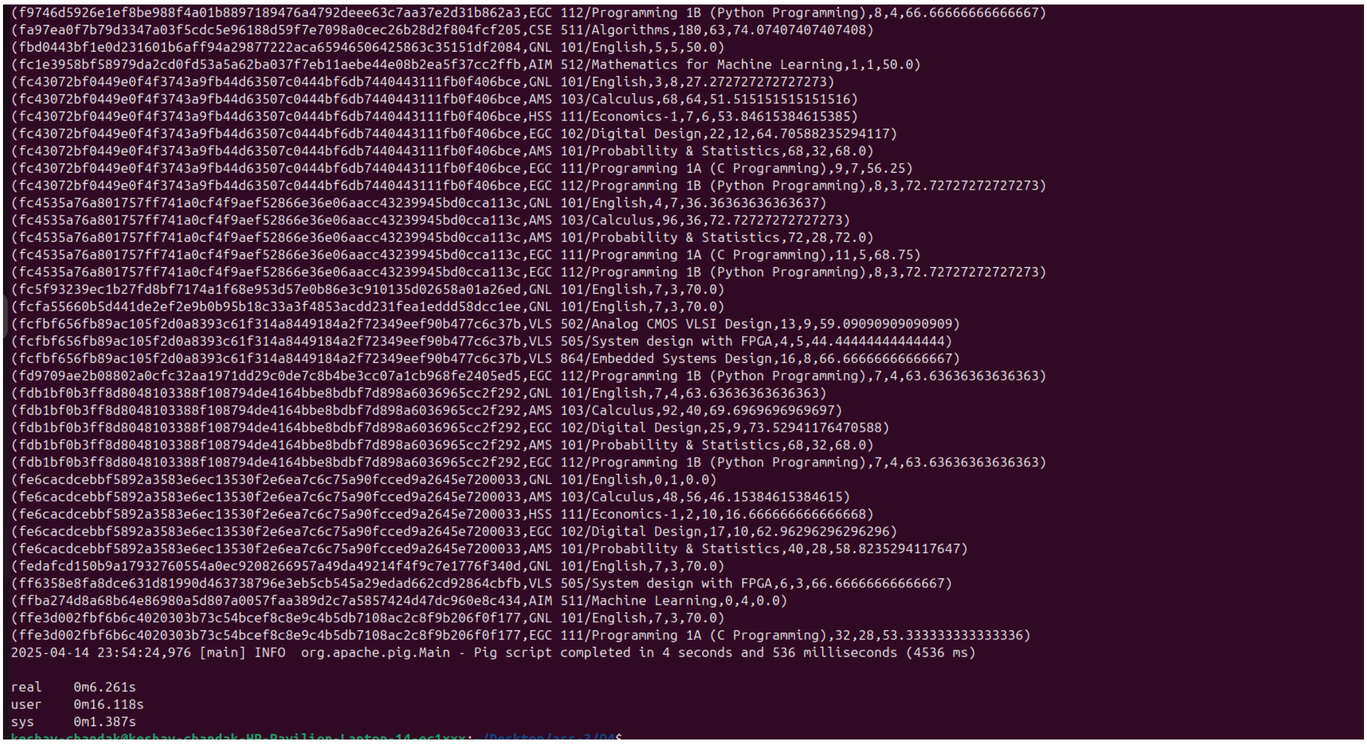
5. **Output:**

   - Dump and store the final filtered output into `/output/Query-3`.

Pig Script Excerpt:

```
-- Join fact_table and grade_roster
joined_data = JOIN fact_table BY (member_id, course), grade_roster BY
(student_id, subject_code_name);

-- Group by student_id and subject_code_name
grouped_data = GROUP joined_data BY (grade_roster::student_id,
grade_roster::subject_code_name);

-- Calculate attendance metrics per group
attendance_data = FOREACH grouped_data {
    total_attended =
SUM(joined_data.fact_table::number_of_classes_attended);
    total_absent = SUM(joined_data.fact_table::number_of_classes_absent);
    attendance_percentage = (total_attended * 100.0) / (total_attended +
total_absent);
    GENERATE FLATTEN(group) AS (student_id, course),
             total_attended AS total_classes_attended,
             total_absent AS total_classes_absent,
             attendance_percentage AS overall_attendance_percentage;
}

-- Filter groups with attendance below 75%
filtered_attendance = FILTER attendance_data BY
overall_attendance_percentage < 75;

DUMP filtered_attendance;
STORE filtered_attendance INTO '/output/Query-3' USING PigStorage(',');
```

Image



```
(f9746d5926e1ef8be988f4a01b8897189476a4792deee63c7aa37e2d31b862a3,EGC 112/Programming 1B (Python Programming),8,4,66.66666666666667)
(fa97ea0f7b79d3347a03f5cdc5e96188d59f7e7098a0cec26b28d2f804fcf205,CSE 511/Algorithms,180,63,74.07407407407408)
(fbd0443bf1e0d231601b6aff94a29877222aca65946506425863c35151df2084,GNL 101/English,5,5,50.0)
(fc1e3958bf58979da2cd0fd53a5a62ba037f7eb11aebe44e08b2ea5f37cc2ffb,AIM 512/Mathematics for Machine Learning,1,1,50.0)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,GNL 101/English,3,8,27.272727272727273)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,AMS 103/Calculus,68,64,51.515151515151516)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,HSS 111/Economics-1,7,6,53.84615384615385)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,EGC 102/Digital Design,22,12,64.70588235294117)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,AMS 101/Probability & Statistics,68,32,68.0)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,EGC 111/Programming 1A (C Programming),9,7,56.25)
(fc43072bf0449e0f4f3743a9fb44d63507c0444bf6db7440443111fb0f406bce,EGC 112/Programming 1B (Python Programming),8,3,72.72727272727273)
(fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c,GNL 101/English,4,7,36.36363636363637)
(fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c,AMS 103/Calculus,96,36,72.72727272727273)
(fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c,AMS 101/Probability & Statistics,72,28,72.0)
(fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c,EGC 111/Programming 1A (C Programming),11,5,68.75)
(fc4535a76a801757ff741a0cf4f9aef52866e36e06aacc43239945bd0cca113c,EGC 112/Programming 1B (Python Programming),8,3,72.72727272727273)
(fc5f93239ec1b27fd8bf7174a1f68e953d57e0b86e3c910135d02658a01a26ed,GNL 101/English,7,3,70.0)
(fcfa55660b5d441de2ef2e9b0b95b18c33a3f4853acdd231fea1eddd58dcc1ee,GNL 101/English,7,3,70.0)
(fcfbf656fb89ac105f2d0a8393c61f314a8449184a2f72349eef90b477c6c37b,VLS 502/Analog CMOS VLSI Design,13,9,59.09090909090909)
(fcfbf656fb89ac105f2d0a8393c61f314a8449184a2f72349eef90b477c6c37b,VLS 505/System design with FPGA,4,5,44.44444444444444)
(fcfbf656fb89ac105f2d0a8393c61f314a8449184a2f72349eef90b477c6c37b,VLS 864/Embedded Systems Design,16,8,66.66666666666667)
(fd9709ae2b08802a0cfc32aa1971dd29c0de7c8b4be3cc07a1cb968fe2405ed5,EGC 112/Programming 1B (Python Programming),7,4,63.63636363636363)
(fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292,GNL 101/English,7,4,63.63636363636363)
(fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292,AMS 103/Calculus,92,40,69.6969696969697)
(fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292,EGC 102/Digital Design,25,9,73.52941176470588)
(fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292,AMS 101/Probability & Statistics,68,32,68.0)
(fdb1bf0b3ff8d8048103388f108794de4164bbe8bdbf7d898a6036965cc2f292,EGC 112/Programming 1B (Python Programming),7,4,63.63636363636363)
(fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033,GNL 101/English,0,1,0.0)
(fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033,AMS 103/Calculus,48,56,46.15384615384615)
(fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033,HSS 111/Economics-1,2,10,16.666666666666668)
(fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033,EGC 102/Digital Design,17,10,62.96296296296296)
(fe6cacdcebbf5892a3583e6ec13530f2e6ea7c6c75a90fcced9a2645e7200033,AMS 101/Probability & Statistics,40,28,58.8235294117647)
(fedafcd150b9a17932760554a0ec9208266957a49da49214f4f9c7e1776f340d,GNL 101/English,7,3,70.0)
(ff6358e8fa8dce631d81990d463738796e3eb5cb545a29edad662cd92864cbfb,VLS 505/System design with FPGA,6,3,66.66666666666667)
(ffba274d8a68b64e86980a5d807a0057faa389d2c7a5857424d47dc960e8c434,AIM 511/Machine Learning,0,4,0.0)
(ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177,GNL 101/English,7,3,70.0)
(ffe3d002fbf6b6c4020303b73c54bcef8c8e9c4b5db7108ac2c8f9b206f0f177,EGC 111/Programming 1A (C Programming),32,28,53.333333333333336)
2025-04-14 23:54:24,976 [main] INFO  org.apache.pig.Main - Pig script completed in 4 seconds and 536 milliseconds (4536 ms)

real	0m6.261s
user	0m16.118s
sys	0m1.387s
```

Elapsed time:4.536 seconds

Comparison of Hive vs. Pig

Table 1: Comparison of HiveQL Query Execution and Pig Query Execution

Query	Hive Without Bucketing(seconds)	Hive With Bucketing (seconds)	Pig performance (seconds
Query 1	6.54	1.643	5.005
Query 2	0.912	0.793	4.445
Query 3	1.23	1.018	4.536

1. **Installation & Setup**
   - **Hive:**
     - Typically involves setting up a Hive metastore along with Hadoop.
     - More components (HiveServer2, Metastore, etc.) need to be configured.
     - Can be complex to install and manage, especially in a production environment.
   - **Pig:**
     - Generally easier to install and lightweight.
     - Runs as a single script without the need for a separate metastore.
     - Quick to set up on local mode or within a Hadoop cluster.
2. **Query Language & Ease of Writing**
   - **Hive:**
     - Uses a SQL-like language (HiveQL) that is familiar to users with a relational database background.
     - Declarative queries make it easier for those accustomed to SQL.
     - Built-in functions and windowing can make complex queries simpler.
   - **Pig:**
     - Uses a scripting language called Pig Latin, which is procedural.
     - Offers more flexibility and control when writing data transformation logic.
     - Can be easier for iterative data processing tasks, but may require more lines of code for similar SQL operations.
3. **Query Performance & Optimization**
   - **Hive:**
     - Optimized for complex, long-running queries over large datasets.
     - Supports indexing, partitioning, and bucketing, which can significantly improve query performance when properly tuned.
     - More suitable for batch processing analytical queries.
   - **Pig:**
     - Also handles large datasets but can be more efficient for ETL tasks and transformations.
     - Performance can be comparable to Hive for many transformation operations; however, highly optimized **Hive queries may outperform Pig on complex aggregations.**
     - Less emphasis on indexing and more on user-defined optimizations via scripting logic.
4. **Suitability & Use Cases**
   - **Hive:**

        - Best suited for analysts comfortable with SQL.
        - Ideal for ad hoc queries and reporting where the data schema is well-defined.
        - Strong integration with BI tools and reporting systems.
    - **Pig:**
        - Excellent for ETL workflows and data processing pipelines.
        - Preferred when you need fine-grained control over data transformations.
        - Often used in scenarios where rapid prototyping of data flows is required.

5. **Community & Ecosystem**

    - **Hive:**
        - Widely adopted in enterprises, with robust community support and integration with many Hadoop components.
        - Part of the broader SQL-on-Hadoop ecosystem.
    - **Pig:**
        - Once very popular for data processing tasks, but usage has decreased in favor of Spark and other processing frameworks.
        - Still a viable option for specific transformation-heavy workflows.

In the table as well, we can see that Hive outperforms Pig on almost all queries in terms of time, and when Hive is optimised with partitioning and bucketing, it always outperforms Pig.
Also, it was a little difficult to write pig scripts since it is easier to write SQL statements, which is similar to Hive queries.