DAS 839: NoSQL Systems April 29, 2025

EndTerm Project

Instructor: Dr. Vinu E. Venugopal

Team Members

- **IMT2022034** - Tarun Kondapalli
- **IMT2022065** - Amruth Gadepalli
- **IMT2022100** - Tahir Khadarabad

Problem 1: Schema Definition and Hive Table Creation

This section outlines the process of defining appropriate schemas for the given datasets and creating corresponding Hive tables. The three CSV files provided are `Course_Attendance.csv`, `Enrollment_Data.csv`, and `GradeRosterReport.csv`. The goal is to define Hive-compatible schemas that accurately represent the data structure and relationships and to load the data correctly into Hive tables. Additionally, a separate schema is created to handle and log any erroneous or missing value tuples for further inspection and data quality assurance.

Course Attendance Schema and Table

The schema for the `Course_Attendance.csv` file captures basic details about students, their course enrollment, attendance statistics, and calculated average attendance. A staging table is also created to facilitate data validation and transformation before inserting the data into the final table with cleaned and validated types.

```
CREATE TABLE IF NOT EXISTS course_attendance_staging (
    course STRING,
    instructor STRING,
    name STRING,
    email STRING,
    member_id STRING,
    classes_attended INT,
    classes_absent INT,
    avg_attendance STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

The main table includes a change in datatype for the `avg_attendance` field to `FLOAT` for accurate numerical representation.

```
CREATE TABLE IF NOT EXISTS course_attendance (
    course STRING,
    instructor STRING,
    name STRING,
    email STRING,
    member_id STRING,
    classes_attended INT,
    classes_absent INT,
    avg_attendance FLOAT
```

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

Enrollment Data Schema and Table

The Enrollment_Data.csv file records detailed information about course enrollment and student demographics. The schema is designed to cover both academic and administrative data such as student IDs, course types, enrollment status, and program details.

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

Grade Roster Report Schema and Table

The GradeRosterReport.csv file is structured to include student performance data, including grades, course credits, and course-specific metadata. This table captures details across various academic periods and sections.

```
CREATE TABLE IF NOT EXISTS grade_roster (
    academy_location STRING,
    student_id STRING,
    student_status STRING,
    admission_id STRING,
    admission_status STRING,
    student_name STRING,
    program STRING,
    batch STRING,
    period STRING,
    subject_code STRING,
    course_type STRING,
    section STRING,
    faculty_name STRING,
```

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

Error Logging Schema

To ensure robust handling of incorrect or malformed data, an `error_log` table is defined. This table captures metadata about errors, including the source table, affected column, the erroneous row, and the nature of the issue. This facilitates systematic data quality checks.

```
CREATE TABLE IF NOT EXISTS error_log (
    source_table STRING,
    column_name STRING,
    row_data STRING,
    error_type STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

Data Loading

The data from the CSV files is loaded into the respective Hive tables using the `LOAD DATA LOCAL INPATH` command, which supports local file paths. This operation assumes that the input files are properly formatted and located in the specified directory.

```
LOAD DATA LOCAL INPATH '/home/narayana/Desktop/sem6/nosqlTahir/dataCSV/Course_Attendance.csv'
INTO TABLE course_attendance_staging;

LOAD DATA LOCAL INPATH '/home/narayana/Desktop/sem6/nosqlTahir/dataCSV/Enrollment_Data.csv'
INTO TABLE enrollment_data;

LOAD DATA LOCAL INPATH '/home/narayana/Desktop/sem6/nosqlTahir/dataCSV/GradeRosterReport.csv'
INTO TABLE grade_roster;
```

These steps collectively ensure the creation of a well-structured data warehouse environment using Hive, with appropriate mechanisms for data validation, type safety, and error handling.

Problem 2: Data Warehouse Schema, Transformation Pipeline, and Error Handling

This section explains the integration of the three source datasets into a unified data warehouse schema using Hive. The workflow includes data transformation, error handling, and data consolidation, resulting in a centralized warehouse table that facilitates efficient analytical queries.

Data Transformation and Cleaning

Initially, raw data from the staging table `course_attendance_staging` is inserted into the cleaned version `course_attendance`, while converting the average attendance values from strings (with %) to float:

```
INSERT INTO TABLE course_attendance
SELECT
    course,
    instructor,
    name,
    email,
    member_id,
    classes_attended,
    classes_absent,
    CAST(REGEXP_REPLACE(avg_attendance, '%', '') AS FLOAT)
FROM course_attendance_staging;
```

Next, for each of the three source tables, cleaned versions are created to ensure only well-formed course codes are retained. These use regular expressions to extract valid course identifiers of the format `ABCD 1234`, and discard entries that are null, empty, or incorrectly formatted:

```
CREATE TABLE cleaned_course_attendance AS
SELECT
    *,
    REGEXP_EXTRACT(course, '([A-Z]{2,4}\\s?\\d{3,4})', 1) AS course_code
FROM course_attendance
WHERE course IS NOT NULL
   AND TRIM(course) != ''
   AND course RLIKE '.*[^A-Z0-9]?([A-Z]{2,4} [0-9]{3,4})[^A-Z0-9]?.*' = TRUE;
```

Similar cleaning steps are applied to `enrollment_data` and `grade_roster`:

```
CREATE TABLE cleaned_enrollment_data AS
SELECT
    *,
    REGEXP_EXTRACT(course, '([A-Z]{2,4}\\s?\\d{3,4})', 1) AS course_code
FROM enrollment_data
WHERE course IS NOT NULL
   AND TRIM(course) != ''
   AND course RLIKE '.*[^A-Z0-9]?([A-Z]{2,4} [0-9]{3,4})[^A-Z0-9]?.*' = TRUE;

CREATE TABLE cleaned_grade_roster AS
SELECT
    *,
    REGEXP_EXTRACT(subject_code, '([A-Z]{2,4}\\s?\\d{3,4})', 1) AS course_code
FROM grade_roster
WHERE subject_code IS NOT NULL
   AND TRIM(subject_code) != ''
   AND subject_code RLIKE '.*[^A-Z0-9]?([A-Z]{2,4} [0-9]{3,4})[^A-Z0-9]?.*' = TRUE;
```

Error Handling and Logging

Records that fail validation checks (due to missing or malformed course codes) are redirected into the `error_log` table. This ensures data integrity in the data warehouse by isolating problematic rows:

```
INSERT INTO TABLE error_log
SELECT
    'course_attendance',
    'course',
    CONCAT_WS(',', course, instructor, name, email, member_id, classes_attended, classes_absent),
    CASE
        WHEN course IS NULL OR TRIM(course) = '' THEN 'missing_value'
        ELSE 'invalid_format'
    END
FROM course_attendance
WHERE course IS NULL
   OR TRIM(course) = ''
   OR course RLIKE '.*[^A-Z0-9]?([A-Z]{2,4} [0-9]{3,4})[^A-Z0-9]?.*' = FALSE;
```

This error-handling logic is similarly applied to the `enrollment_data` and `grade_roster` tables. All error rows are logged with metadata including the table name, column in question, and type of issue.

Data Warehouse Schema and Integration Pipeline

A consolidated schema is created in the form of the `student_data_warehouse` table. This table integrates selected fields from the cleaned datasets and captures a holistic view of student participation, enrollment, and academic performance.

```
CREATE TABLE student_data_warehouse (
    record_id INT,
    student_full_name STRING,
    student_roll_no STRING,
    subject_name STRING,
    clean_course_code STRING,
    subject_type STRING,
    subject_variant STRING,
    enrollment_status STRING,
    academic_program STRING,
    batch_year STRING,
    semester_period STRING,
    lead_faculty STRING,
    instructor_names STRING,
    total_attended INT,
    attendance_percentage FLOAT,
    course_credit INT,
    final_grade STRING,
    result_status STRING
) STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

The pipeline for populating this table involves joining the cleaned tables based on both student ID and course code. This ensures that only entries present across all three datasets are considered for final analysis:

```
INSERT INTO student_data_warehouse
SELECT
  ROW_NUMBER() OVER () AS record_id,
  e.student_name AS student_full_name,
```

```
  e.student_id AS student_roll_no,
  e.course AS subject_name,
  e.course_code AS clean_course_code,
  e.course_type AS subject_type,
  e.course_variant AS subject_variant,
  e.status AS enrollment_status,
  e.program AS academic_program,
  e.batch AS batch_year,
  e.period AS semester_period,
  e.primary_faculty AS lead_faculty,
  c.instructor AS instructor_names,
  c.classes_attended AS total_attended,
  c.avg_attendance AS attendance_percentage,
  g.course_credit AS course_credit,
  g.obtained_grade AS final_grade,
  g.exam_result AS result_status
FROM cleaned_enrollment_data e
JOIN cleaned_course_attendance c
  ON e.student_id = c.member_id AND e.course_code = c.course_code
JOIN cleaned_grade_roster g
  ON e.student_id = g.student_id AND e.course_code = g.course_code;
```

This structured integration ensures a consistent, analyzable format suitable for complex HiveQL queries. It also provides a single source of truth for downstream data analytics tasks, such as academic performance analysis, attendance impact studies, and enrollment trends.

Analytical Queries in HiveQL

This section presents three advanced analytical queries executed on the unified ‘student$_{d}ata_{w}arehouse$‘$table.Eachqueryprovid$

Query 1: Calculating CGPA for All Students

This query calculates the Cumulative Grade Point Average (CGPA) for each student based on weighted averages of grade points across all courses taken.

```
SELECT
  student_full_name,
  COUNT(*) AS num_courses,
  ROUND(SUM(
    CASE final_grade
      WHEN 'A'  THEN 4.0
      WHEN 'A-' THEN 3.7
      WHEN 'B+' THEN 3.4
      WHEN 'B'  THEN 3.0
      WHEN 'B-' THEN 2.7
      WHEN 'C+' THEN 2.4
      WHEN 'C'  THEN 2.0
      WHEN 'D'  THEN 1.0
      WHEN 'F'  THEN 0.0
      ELSE 0.0
    END * CAST(course_credit AS FLOAT)
  ) / SUM(CAST(course_credit AS FLOAT)), 2) AS gpa
FROM student_data_warehouse
GROUP BY student_full_name
ORDER BY gpa DESC;
```

Explanation:

- Each grade is mapped to a corresponding grade point.
- Credits are used as weights in the GPA calculation.
- Students are ranked in descending order by GPA.

Output Screenshot for Query 1: CGPA Calculation

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.95 sec   HDFS Read: 2299837 HDFS Write: 47871 HDFS EC Read: 0 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.59 sec   HDFS Read: 57336 HDFS Write: 44484 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 540 msec
+----------------------------------------------------+-------------+-------+
|                 student_full_name                  | num_courses |  gpa  |
+----------------------------------------------------+-------------+-------+
| ac60764563bc5fb8d8152ed76c7ec9916b84eadb72b8d04ef7fe7baa29e02fa1 | 2             | 4.0   |
| cadd19f32b7adf28605e0d68368bc4566bcac90f155412e381d45333d2db3074 | 4             | 4.0   |
| 08d5b026019e5408ab6773d721c9017c54cc18d984ef7e668cfe0fe5949291ec | 2             | 4.0   |
| df3a2c2e821deb3110d845ee2c9ecb795fef8fe3fe3f539e5049084c3ff38b04 | 14            | 4.0   |
| 3813eb5d4e73958de7085fc777544a145031017447ad49c28913057d40fab1a3 | 14            | 3.97  |
| f3abe82ff502da1d906eea56afa7f34f0ed5b02cd2117f50b31ad6fb4faab47d | 14            | 3.94  |
| 34e835be31e1b05907e762e4c37461c25df045faaae96ee9b1cdf9a540fe7625 | 14            | 3.9   |
| 6132fbefbde297de7ae23317a53106023b24ffde98781efc61e2ac0a57296de2 | 12            | 3.9   |
| d71c36e6c5e2aa3c65c0d87df02a24aa9c8ef172248511712500e14cf860c9f9 | 14            | 3.88  |
| cf29f5f681d1df5d430a92a9afb82be43092bfbeb113706270a1f6815972ba17 | 14            | 3.87  |
| 83fba4fa869cc6e9a3b100aed2dc53c65f2402b8e93d2488962a224ac49c8734 | 14            | 3.87  |
| 0dfe8f1929b054d6f3fe74b5bc63e439ef93d1bd40f357193033994468cf841d | 4             | 3.85  |
| b73a68e8474c2c0282340a0b947b77b3a968353393e4e4c86a4f29775e413b98 | 14            | 3.85  |
| 51f38e3bf905a987ac0be1b55b2224d094c51566a09bd427b3c37b698c1139b9 | 12            | 3.85  |
| 8dd39f3d9b9333248a67a7989f9ddfdd685f179669b1fe0837ce492ef95ebed0 | 8             | 3.85  |
| 520508f61c9b1c82bbd9513c8ba6b93d152efc828187eb4b5b8c22f03af42ded | 12            | 3.85  |
| bb5ffc2c63a0a9fccadeaff634e261720b756032748dbe212b39015006635cbd | 12            | 3.85  |
| d0e18168d6fb5510abc27979ab5f55b11142cb024a6683f44be1de59370d615e | 14            | 3.82  |
| dcddd1880d82208da46992fc57dc6254809c0b7689f7a365c9c75065099bb3b4 | 6             | 3.8   |
| 1d696614de08dcb41c69f661dd6bfb7c3b2a8a117a10e384be7433f54cd1563d | 12            | 3.8   |
| 12324eb9355064d6e3971dd3c7521373336104842eb4787045d9a563be2ee45a | 6             | 3.8   |
| 840b9377845c8e84d516c1c35f42b107e240bff84e3c633e949fed3d9a138cf1 | 14            | 3.8   |
| 1bd2d2f2ad206f2c06fb4f449bfa5a4c5b4da017faf7a1579144b26621f18b35 | 10            | 3.8   |
| fd65b4b2ed8cee8ad12e12533f8d073eb29749bb080f3ebbb5f3ef8572271fbf | 12            | 3.8   |
| bba6fb231f88f60905e0d9e0bf2b8b33d9e5294bb98a1061cce47cca7353029d | 12            | 3.8   |
| 1992bec2c35a74a802d6260fe2a2b9694793debf94490ac0ac8dcaedff7b6df1 | 12            | 3.8   |
| 4345c4d7276be92f5184fb99478aebd4b3530d71389e1a7edba88977aea22304 | 14            | 3.79  |
| 67599981fd0c8e8f9f8bd79326fcfce9740dc523ee359621665d7125c15da79f | 14            | 3.77  |
| e3805ff6ffb3c4ac9fb6d24f86ac35e050c9fa73a513e297b61aa8690cf2a269 | 14            | 3.76  |
| effafbaab649624cb186038ed4f88937df1a94bda6254fd897ad04c05403c136 | 14            | 3.76  |
| 734b8c6a964c2643fd860f7dadb6218f5f23754638c8aa443d04821ab98a68b4 | 14            | 3.76  |
| b687e6bb95f2cd6173d5cb3283ceffa5b5f0cd25af8641349efd05572a967f27 | 10            | 3.76  |
| c571e2db3221dd8fe643cdfd5cace13ca7b00d7147203ea6345ce5502b03d547 | 12            | 3.75  |
```

Figure 1: Sample output of student CGPAs and number of courses taken

```
| 87bf91494cd4ab6e563e5a3abcfa5c6a4ee4cb00c6734927961f49eabf571094 | 14            | 1.61  |
| 8998d340aed38cb9308d89656e608f35f66666f3f85bbef70884f29579686898 | 14            | 1.59  |
| 224b3df51bb44e65b08fd1f14e15af139ab85e0559473f830723d983490afdb7 | 12            | 1.57  |
| b4d76ac4f388d03c1ca8da5af88c44b007853726d5235b29f22f63004c4c445e | 14            | 1.54  |
| 51b821021e95baa20fbe973aa7fccf667a038e44164fa638c0c5da3149786a0b | 14            | 1.45  |
| 031303634a2974359db15193d50b3a5713d3eeddf0689ad084456e978464c5d3 | 14            | 1.44  |
| ab52c9f1300299928dd71f2ccf55dca437d16ddecf7d692bc059111c53e499bc | 14            | 1.27  |
| 1311c281d984327d56608dcf4db535ed2894d0a3663555d7de22b6075b634e3a | 14            | 1.2   |
| befcab76f7ca4aba6f5e642ce30000c07c7628d35a2266486cbaad4cf951c169 | 8             | 1.0   |
| 03cb2b3cf2e49b0feff2c04fd76cd0cffba518cf34a1a5cc127580c34104c784 | 14            | 0.97  |
| 72cb5d920092cb8827decd0cc6c6e315c02c1eef147e0b23e7596303145f3424 | 14            | 0.93  |
| 70f84e9ca614d5a561f2869e43279e3c48538e2df29ab0f302378df67f596c4c | 14            | 0.86  |
| e3e98a16e5cf9f7d5c8705736a99f3dd8447c91e663633e84dc560a8c5930f65 | 14            | 0.74  |
| f9eba7c4a063d26e1cee26ce6577850b3f143dbe402dee29439b62c0d2a215ed | 14            | 0.6   |
| 228865ac67d934df4ee5c956822c0d8907c7307af153d1d179684caa3085f758 | 4             | 0.0   |
| 9c778d61386925c05a39cc21bef883d7e71e63461dfca447c16f1fe69f9c0ce4 | 6             | 0.0   |
| f9797763274c87385d9c67f8fb7304df8cca2af4de6e012f2539c767074542f5 | 8             | 0.0   |
| cce7b4e40ce1e816b3ff30d74d32df5e8994f13338db463d4ff1ac16cd1753f2 | 4             | 0.0   |
+----------------------------------------------------+-------------+-------+
525 rows selected (37.157 seconds)
```

Figure 2: Runtime visible for Query 1 at the bottom

Query 2: Failure Percentage in a Course and Relation to Average Attendance

This query identifies courses with the highest failure rates and analyzes how these correlate with average student attendance.

```
SELECT
    subject_name,
    lead_faculty,
    COUNT(*) AS total_students,
    SUM(CASE WHEN final_grade = 'F' THEN 1 ELSE 0 END) AS failing_students,
    ROUND(100.0 * SUM(CASE WHEN final_grade = 'F' THEN 1 ELSE 0 END) / COUNT(*), 2)
    AS fail_percentage,
    ROUND(AVG(attendance_percentage), 2) AS avg_attendance
FROM student_data_warehouse
WHERE result_status IS NOT NULL
GROUP BY subject_name, lead_faculty
ORDER BY fail_percentage DESC;
```

Explanation:

- For each course and faculty, it computes:
  - Total students enrolled.
  - Count and percentage of students who received an 'F'.
  - Average attendance across all students.
- The results are sorted to highlight the courses with the highest failure rates.

Output Screenshot for Query 2: Failure Percentage vs Attendance

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.28 sec   HDFS Read: 2297835 HDFS Write: 2794 HDFS EC Read: 0 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.91 sec   HDFS Read: 13889 HDFS Write: 2683 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 190 msec
```

subject_name	lead_faculty	total_students	failing_students	fail_percentage	avg_attendance
VLS 864 / Embedded Systems Design	Kurian Polachan	62	4	6.45	88.7
AMS 102 / Probability & Random Process	Prof. Amrita Mishra	240	14	5.83	79.95
VLS 505 / System design with FPGA	Nanditha Rao	74	4	5.41	67.58
AMS 103 / Calculus	Manisha Kulkarni	236	8	3.39	76.56
EGC 102 / Digital Design	Pillalamarri Sridhar	314	8	2.55	80.47
AMS 101 / Probability & Statistics	Amit Chattopadhyay	628	16	2.55	84.25
CSE 511 / Algorithms	Muralidhara V N	822	12	1.46	78.63
AIM 512 / Mathematics for Machine Learning	Sachit Rao	268	2	0.75	62.72
DHS 301 / Data Analysis and Visualization	Jaya Sreevalsan Nair,V Sridhar	132	0	0.00	72.3
VLS 503 / Digital CMOS VLSI Design	Madhav Rao	70	0	0.00	91.0
VLS 502 / Analog CMOS VLSI Design	Sakshi Arora	60	0	0.00	73.77
NWC 882 / Special Topics - Network-Based Computing for HPC	Karthikeyan Vaidyanathan	2	0	0.00	85.7
HSS 111 / Economics-1	V Sridhar	560	0	0.00	85.88
GNL 101 / English	Priyanka Sharma	560	0	0.00	66.45
EGC 223 / Computer Architecture - Memory	Nanditha Rao	10	0	0.00	61.66
EGC 112 / Programming 1B (Python Programming)	Tulika Saha	242	0	0.00	74.41
EGC 112 / Programming 1B (Python Programming)	Sujit Kumar Chakrabrati	318	0	0.00	86.15
EGC 111 / Programming 1A (C Programming)	Srinivas Vivek	322	0	0.00	79.0
EGC 111 / Programming 1A (C Programming)	Badrinath Ramamurthy	478	0	0.00	86.96
EGC 102 / Digital Design	Kurian Polachan	6	0	0.00	93.13
DHS 304 / User Research	Preeti Mudliar	66	0	0.00	80.2
DHS 303 / Software Product Management	Laxmi Gunupudi	120	0	0.00	79.06
DAS 703 / Geographic Information Systems	Uttam Kumar	4	0	0.00	28.0
CSE 857 / Secure Computation	Ashish Choudhury	2	0	0.00	66.7
CSE 816 / Software Production Engineering	Thangaraju B	4	0	0.00	71.4
CSE 731 / Software Testing	Meenakshi D Souza	6	0	0.00	86.27
CSE 514 / Concrete Mathematics	Srinivas Vivek	74	0	0.00	72.97
COM 827 / Internet of Things	Jyotsna Bapat	4	0	0.00	97.2
AMS 103 / Calculus	Jaya Sreevalsan Nair,S Malapaka	6	0	0.00	83.87
AIM 841 / Medical Image Analysis with AI	Sushree Behera	8	0	0.00	81.83
AIM 840 / Self-Supervised Learning	V Ramasubramanian	2	0	0.00	86.4
AIM 608 / Networks and Semantics	Srinath Srinivasa	6	0	0.00	88.9
AIM 511 / Machine Learning	Raghuram Bharadwaj	580	0	0.00	85.39

33 rows selected (39.038 seconds)

Figure 3: Output showing relationship between failure percentage and average attendance

Query 3: Course Difficulty Analysis Based on Average Grade Points

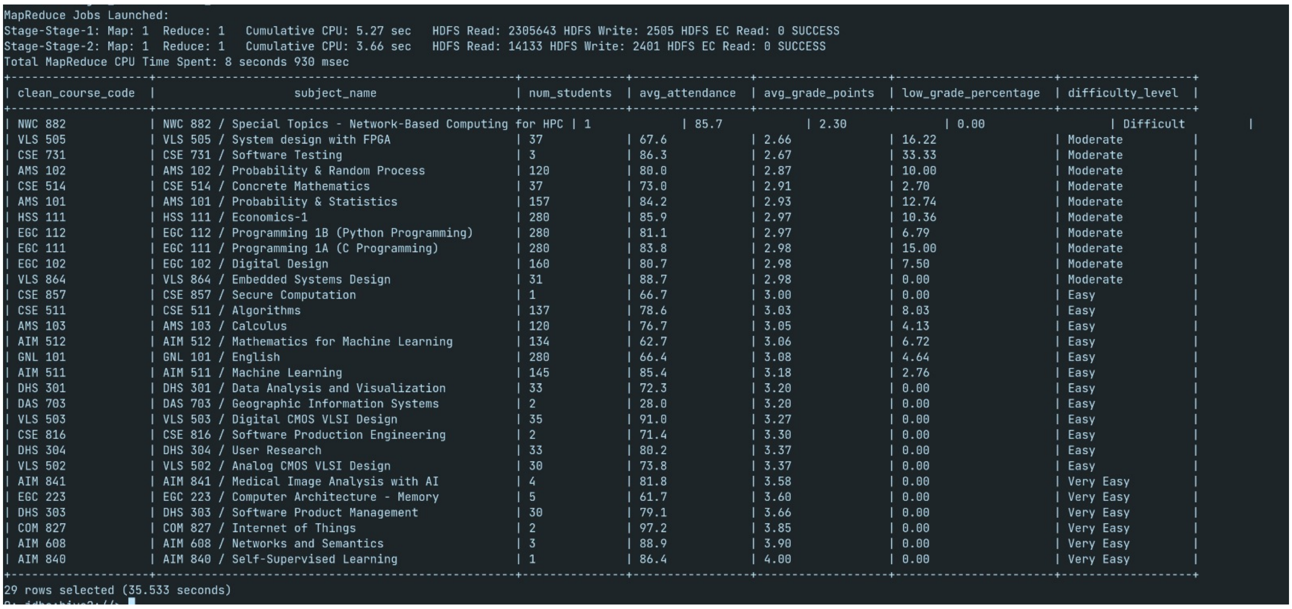
This query classifies courses by difficulty level based on students' grade performance and participation.

```
WITH subject_stats AS (
    SELECT
        clean_course_code,
        subject_name,
        COUNT(DISTINCT student_roll_no) AS num_students,
        AVG(attendance_percentage) AS avg_attendance,
        AVG(CASE
            WHEN final_grade = 'A' THEN 4.0
            WHEN final_grade = 'A-' THEN 3.7
            WHEN final_grade = 'B+' THEN 3.3
            WHEN final_grade = 'B' THEN 3.0
            WHEN final_grade = 'B-' THEN 2.7
            WHEN final_grade = 'C+' THEN 2.3
            WHEN final_grade = 'C' THEN 2.0
            WHEN final_grade = 'D' THEN 1.0
            ELSE 0.0
        END) AS avg_grade_points,
        SUM(CASE WHEN final_grade IN ('D', 'C') THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
        AS low_grade_percentage
    FROM student_data_warehouse
    GROUP BY clean_course_code, subject_name
)

SELECT
    clean_course_code,
    subject_name,
    num_students,
    ROUND(avg_attendance, 1) AS avg_attendance,
    ROUND(avg_grade_points, 2) AS avg_grade_points,
    ROUND(low_grade_percentage, 2)
    AS low_grade_percentage,
    CASE
        WHEN avg_grade_points < 2.0 THEN 'Very Difficult'
        WHEN avg_grade_points < 2.5 THEN 'Difficult'
        WHEN avg_grade_points < 3.0 THEN 'Moderate'
        WHEN avg_grade_points < 3.5 THEN 'Easy'
        ELSE 'Very Easy'
    END AS difficulty_level
FROM subject_stats
ORDER BY avg_grade_points, low_grade_percentage DESC;
```

Explanation:

- Courses are analyzed for:
  - Number of enrolled students.
  - Average attendance.
  - Average grade points.
  - Percentage of students receiving low grades ('C' or 'D').
- A difficulty label is assigned based on average grade points.
- Results are sorted by average grade points and low grade percentage.

Output Screenshot for Query 3: Course Difficulty Classification


MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.27 sec HDFS Read: 2305643 HDFS Write: 2505 HDFS EC Read: 0 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 3.66 sec HDFS Read: 14133 HDFS Write: 2401 HDFS EC Read: 0 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 930 msec

clean_course_code	subject_name	num_students	avg_attendance	avg_grade_points	low_grade_percentage	difficulty_level
NWC 882	NWC 882 / Special Topics - Network-Based Computing for HPC	1	85.7	2.30	0.00	Difficult
VLS 505	VLS 505 / System design with FPGA	37	67.6	2.66	16.22	Moderate
CSE 731	CSE 731 / Software Testing	3	86.3	2.67	33.33	Moderate
AMS 102	AMS 102 / Probability & Random Process	120	80.0	2.87	10.00	Moderate
CSE 514	CSE 514 / Concrete Mathematics	37	73.0	2.91	2.70	Moderate
AMS 101	AMS 101 / Probability & Statistics	157	84.2	2.93	12.74	Moderate
HSS 111	HSS 111 / Economics-1	280	85.9	2.97	10.36	Moderate
EGC 112	EGC 112 / Programming 1B (Python Programming)	280	81.1	2.97	6.79	Moderate
EGC 111	EGC 111 / Programming 1A (C Programming)	280	83.8	2.98	15.00	Moderate
EGC 102	EGC 102 / Digital Design	160	80.7	2.98	7.50	Moderate
VLS 864	VLS 864 / Embedded Systems Design	31	88.7	2.98	0.00	Moderate
CSE 857	CSE 857 / Secure Computation	1	66.7	3.00	0.00	Easy
CSE 511	CSE 511 / Algorithms	137	78.6	3.03	8.03	Easy
AMS 103	AMS 103 / Calculus	120	76.7	3.05	4.13	Easy
AIM 512	AIM 512 / Mathematics for Machine Learning	134	62.7	3.06	6.72	Easy
GNL 101	GNL 101 / English	280	66.4	3.08	4.64	Easy
AIM 511	AIM 511 / Machine Learning	145	85.4	3.18	2.76	Easy
DHS 301	DHS 301 / Data Analysis and Visualization	33	72.3	3.20	0.00	Easy
DAS 703	DAS 703 / Geographic Information Systems	2	28.0	3.20	0.00	Easy
VLS 503	VLS 503 / Digital CMOS VLSI Design	35	91.0	3.27	0.00	Easy
CSE 816	CSE 816 / Software Production Engineering	2	71.4	3.30	0.00	Easy
DHS 304	DHS 304 / User Research	33	80.2	3.37	0.00	Easy
VLS 502	VLS 502 / Analog CMOS VLSI Design	30	73.8	3.37	0.00	Easy
AIM 841	AIM 841 / Medical Image Analysis with AI	4	81.8	3.58	0.00	Very Easy
EGC 223	EGC 223 / Computer Architecture - Memory	5	61.7	3.60	0.00	Very Easy
DHS 303	DHS 303 / Software Product Management	30	79.1	3.66	0.00	Very Easy
COM 827	COM 827 / Internet of Things	2	97.2	3.85	0.00	Very Easy
AIM 608	AIM 608 / Networks and Semantics	3	88.9	3.90	0.00	Very Easy
AIM 840	AIM 840 / Self-Supervised Learning	1	86.4	4.00	0.00	Very Easy

29 rows selected (35.533 seconds)

Figure 4: Output showing course difficulty levels based on student grades and attendance

Problem 3: Enhancing Query Performance with Partitioning and Bucketing

This section of the assignment focuses on improving Hive query performance by applying partitioning and bucketing techniques. Partitioning reduces the amount of data scanned by restricting query execution to relevant partitions, while bucketing distributes the data into manageable files, optimizing joins and aggregations. We apply these strategies to create three optimized Hive tables tailored for specific analytical queries. The results are compared with the non-optimized warehouse queries in terms of performance and execution time.

Query 1: Calculating CGPA per Student (Optimized)

To improve the efficiency of the GPA calculation query, a new table `student_gpa_optimized` was created. It uses partitioning on `subject_type` for better filter performance, and bucketing on `student_roll_no` to evenly distribute student data across 64 buckets. The data is stored in ORC format with Snappy compression.

```
CREATE TABLE student_gpa_optimized (
    student_full_name STRING,
    student_roll_no STRING,
    course_credit INT,
    final_grade STRING,
    academic_program STRING
)
PARTITIONED BY (subject_type STRING)
CLUSTERED BY (student_roll_no) INTO 64 BUCKETS
STORED AS ORC
TBLPROPERTIES ("orc.compress"="SNAPPY", "orc.create.index"="true");
```

Dynamic partitioning was enabled to populate the table:

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE student_gpa_optimized PARTITION (subject_type)
SELECT
    student_full_name,
    student_roll_no,
    course_credit,
    final_grade,
    academic_program,
    subject_type
FROM student_data_warehouse
WHERE final_grade IS NOT NULL AND course_credit > 0;
```

The optimized query benefits from partition pruning and efficient aggregation due to bucketing:

```
SELECT
    student_full_name,
    COUNT(*) AS num_courses,
    ROUND(SUM(
      CASE final_grade
        WHEN 'A'  THEN 4.0
        WHEN 'A-' THEN 3.7
        WHEN 'B+' THEN 3.4
        WHEN 'B'  THEN 3.0
        WHEN 'B-' THEN 2.7
        WHEN 'C+' THEN 2.4
        WHEN 'C'  THEN 2.0
        WHEN 'D'  THEN 1.0
```

```
        WHEN 'F'  THEN 0.0
        ELSE 0.0
      END * course_credit
    ) / SUM(course_credit), 2) AS gpa
FROM student_gpa_optimized
WHERE subject_type = 'Core'
GROUP BY student_full_name
ORDER BY gpa DESC;
```

```
+------------------------------------------------------------------+--------------+-------+
| a9a9db49f7ae07c99a2d593bfa36ea5a8719f201cef12719de36915e9e371e25 | 12           | 2.0   |
| 564df3798f6fc51c0267b3a607fdc9b4faa61e161ec22ca6c96fd78d66e2b026 | 12           | 2.0   |
| b777c8714e10dc414c6db4ce856dc85034f144cbb877e6d393f74713b631eba0 | 14           | 1.86  |
| 8ec4fa73aef70188f014b22e3f66883200bfa5a94c08d50eede174c0ec67514f | 14           | 1.81  |
| 4daba09bc692a3e812d2fc60a58636b3863e447cdfbc56faccec0f5b677ce819 | 14           | 1.79  |
| e988e9c8d039cf9258d1556b8f905a2bf016757181f8332fa52003d0db3a3826 | 14           | 1.78  |
| 2d137273b124677023222aa4d0fc060f53c15b75862a69adddca8b496482fd76 | 14           | 1.7   |
| 87bf91494cd4ab6e563e5a3abcfa5c6a4ee4cb00c6734927961f49eabf571094 | 14           | 1.61  |
| 8998d340aed38cb9308d89656e608f35f66666f3f85bbef70884f29579686898 | 14           | 1.59  |
| 224b3df51bb44e65b08fd1f14e15af139ab85e0559473f830723d983490afdb7 | 12           | 1.57  |
| b4d76ac4f388d03c1ca8da5af88c44b007853726d5235b29f22f63004c4c445e | 14           | 1.54  |
| 51b821021e95baa20fbe973aa7fccf667a038e44164fa638c0c5da3149786a0b | 14           | 1.45  |
| 031303634a2974359db15193d50b3a5713d3eeddf0689ad084456e978464c5d3 | 14           | 1.44  |
| ab52c9f1300299928dd71f2ccf55dca437d16ddecf7d692bc059111c53e499bc | 14           | 1.27  |
| 1311c281d984327d56608dcf4db535ed2894d0a3663555d7de22b6075b634e3a | 14           | 1.2   |
| befcab76f7ca4aba6f5e642ce30000c07c7628d35a2266486cbaad4cf951c169 | 8            | 1.0   |
| 03cb2b3cf2e49b0feff2c04fd76cd0cffba518cf34a1a5cc127580c34104c784 | 14           | 0.97  |
| 72cb5d920092cb8827decd0cc6c6e315c02c1eef147e0b23e7596303145f3424 | 14           | 0.93  |
| 70f84e9ca614d5a561f2869e43279e3c48538e2df29ab0f302378df67f596c4c | 14           | 0.86  |
| e3e98a16e5cf9f7d5c8705736a99f3dd8447c91e663633e84dc560a8c5930f65 | 14           | 0.74  |
| f9eba7c4a063d26e1cee26ce6577850b3f143dbe402dee29439b62c0d2a215ed | 14           | 0.6   |
| 228865ac67d934df4ee5c956822c0d8907c7307af153d1d179684caa3085f758 | 4            | 0.0   |
| 9c778d61386925c05a39cc21bef883d7e71e63461dfca447c16f1fe69f9c0ce4 | 6            | 0.0   |
| f9797763274c87385d9c67f8fb7304df8cca2af4de6e012f2539c767074542f5 | 8            | 0.0   |
| cce7b4e40ce1e816b3ff30d74d32df5e8994f13338db463d4ff1ac16cd1753f2 | 4            | 0.0   |
+------------------------------------------------------------------+--------------+-------+
525 rows selected (32.293 seconds)
```

Figure 5: Screenshot of final CGPA output

Query 2: Faculty-wise Performance Analysis (Optimized)

The `faculty_performance_optimized` table uses partitioning on `subject_name` and bucketing on `lead_faculty`, helping in grouped aggregations at the faculty-subject level.

```
CREATE TABLE faculty_performance_optimized (
    lead_faculty STRING,
    final_grade STRING,
    attendance_percentage FLOAT,
    result_status STRING
)
PARTITIONED BY (subject_name STRING)
CLUSTERED BY (lead_faculty) INTO 32 BUCKETS
STORED AS ORC
TBLPROPERTIES (
    "orc.compress"="SNAPPY",
    "orc.bloom.filter.columns"="lead_faculty,final_grade",
    "orc.create.index"="true"
);
```

```sql
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE faculty_performance_optimized PARTITION (subject_name)
SELECT
    lead_faculty,
    final_grade,
    attendance_percentage,
    result_status,
    subject_name
FROM student_data_warehouse
WHERE result_status IS NOT NULL;

SELECT
    subject_name,
    lead_faculty,
    COUNT(*) AS total_students,
    SUM(CASE WHEN final_grade = 'F' THEN 1 ELSE 0 END) AS failing_students,
    ROUND(100.0 * SUM(CASE WHEN final_grade = 'F' THEN 1 ELSE 0 END) / COUNT(*), 2) AS fail_percentage,
    ROUND(AVG(attendance_percentage), 2) AS avg_attendance
FROM faculty_performance_optimized
GROUP BY subject_name, lead_faculty
ORDER BY fail_percentage DESC;
```

```
 0.00           | 80.2          |
| DHS 303 / Software Product Management            | Laxmi Gunupudi                  | 120            | 0                 |
 0.00           | 79.06         |
| DAS 703 / Geographic Information Systems         | Uttam Kumar                     | 4              | 0                 |
 0.00           | 28.0          |
| CSE 857 / Secure Computation                     | Ashish Choudhury                | 2              | 0                 |
 0.00           | 66.7          |
| CSE 816 / Software Production Engineering        | Thangaraju B                    | 4              | 0                 |
 0.00           | 71.4          |
| CSE 731 / Software Testing                       | Meenakshi D Souza               | 6              | 0                 |
 0.00           | 86.27         |
| CSE 514 / Concrete Mathematics                   | Srinivas Vivek                  | 74             | 0                 |
 0.00           | 72.97         |
| COM 827 / Internet of Things                     | Jyotsna Bapat                   | 4              | 0                 |
 0.00           | 97.2          |
| AMS 103 / Calculus                               | Jaya Sreevalsan Nair,S Malapaka | 6              | 0                 |
 0.00           | 83.87         |
| AIM 841 / Medical Image Analysis with AI         | Sushree Behera                  | 8              | 0                 |
 0.00           | 81.83         |
| AIM 840 / Self-Supervised Learning               | V Ramasubramanian               | 2              | 0                 |
 0.00           | 86.4          |
| AIM 608 / Networks and Semantics                 | Srinath Srinivasa               | 6              | 0                 |
 0.00           | 88.9          |
| AIM 511 / Machine Learning                       | Raghuram Bharadwaj              | 580            | 0                 |
 0.00           | 85.39         |
+--------------------------------------------------+---------------------------------+----------------+-------------------+
----------------+----------------+
33 rows selected (33.153 seconds)
```

Figure 6: Screenshot of faculty performance analysis

Query 3: Subject Difficulty Analysis (Optimized)

In this query, partitioning by `subject_name` and bucketing by `final_grade` in the `subject_difficulty_optimized` table allows for faster analysis of subject-level trends and grade distributions.

```sql
CREATE TABLE subject_difficulty_optimized (
    clean_course_code STRING,
    student_roll_no STRING,
    attendance_percentage FLOAT,
    final_grade STRING
)
```

```
PARTITIONED BY (subject_name STRING)
CLUSTERED BY (final_grade) INTO 10 BUCKETS
STORED AS ORC
TBLPROPERTIES (
    "orc.compress"="SNAPPY",
    "orc.bloom.filter.columns"="final_grade",
    "orc.create.index"="true"
);

SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;

INSERT OVERWRITE TABLE subject_difficulty_optimized PARTITION (subject_name)
SELECT
    clean_course_code,
    student_roll_no,
    attendance_percentage,
    final_grade,
    subject_name
FROM student_data_warehouse
WHERE final_grade IS NOT NULL;

WITH subject_stats AS (
    SELECT
        clean_course_code,
        subject_name,
        COUNT(DISTINCT student_roll_no) AS num_students,
        AVG(attendance_percentage) AS avg_attendance,
        AVG(CASE
            WHEN final_grade = 'A' THEN 4.0
            WHEN final_grade = 'A-' THEN 3.7
            WHEN final_grade = 'B+' THEN 3.3
            WHEN final_grade = 'B' THEN 3.0
            WHEN final_grade = 'B-' THEN 2.7
            WHEN final_grade = 'C+' THEN 2.3
            WHEN final_grade = 'C' THEN 2.0
            WHEN final_grade = 'D' THEN 1.0
            ELSE 0.0
        END) AS avg_grade_points,
        SUM(CASE WHEN final_grade IN ('D', 'C') THEN 1 ELSE 0 END) * 100.0 / COUNT(*) AS low_grade_perc
    FROM subject_difficulty_optimized
    GROUP BY clean_course_code, subject_name
)

SELECT
    clean_course_code,
    subject_name,
    num_students,
    ROUND(avg_attendance, 1) AS avg_attendance,
    ROUND(avg_grade_points, 2) AS avg_grade_points,
    ROUND(low_grade_percentage, 2) AS low_grade_percentage,
    CASE
        WHEN avg_grade_points < 2.0 THEN 'Very Difficult'
        WHEN avg_grade_points < 2.5 THEN 'Difficult'
        WHEN avg_grade_points < 3.0 THEN 'Moderate'
        WHEN avg_grade_points < 3.5 THEN 'Easy'
        ELSE 'Very Easy'
    END AS difficulty_level
```

```
FROM subject_stats
ORDER BY avg_grade_points, low_grade_percentage DESC;
```

```
 | 2.76               | Easy              |
| DHS 301           | DHS 301 / Data Analysis and Visualization        | 33           | 72.3            | 3.20
 | 0.00               | Easy              |
| DAS 703           | DAS 703 / Geographic Information Systems         | 2            | 28.0            | 3.20
 | 0.00               | Easy              |
| VLS 503           | VLS 503 / Digital CMOS VLSI Design               | 35           | 91.0            | 3.27
 | 0.00               | Easy              |
| CSE 816           | CSE 816 / Software Production Engineering        | 2            | 71.4            | 3.30
 | 0.00               | Easy              |
| DHS 304           | DHS 304 / User Research                          | 33           | 80.2            | 3.37
 | 0.00               | Easy              |
| VLS 502           | VLS 502 / Analog CMOS VLSI Design                | 30           | 73.8            | 3.37
 | 0.00               | Easy              |
| AIM 841           | AIM 841 / Medical Image Analysis with AI         | 4            | 81.8            | 3.58
 | 0.00               | Very Easy         |
| EGC 223           | EGC 223 / Computer Architecture - Memory         | 5            | 61.7            | 3.60
 | 0.00               | Very Easy         |
| DHS 303           | DHS 303 / Software Product Management            | 30           | 79.1            | 3.66
 | 0.00               | Very Easy         |
| COM 827           | COM 827 / Internet of Things                     | 2            | 97.2            | 3.85
 | 0.00               | Very Easy         |
| AIM 608           | AIM 608 / Networks and Semantics                 | 3            | 88.9            | 3.90
 | 0.00               | Very Easy         |
| AIM 840           | AIM 840 / Self-Supervised Learning               | 1            | 86.4            | 4.00
 | 0.00               | Very Easy         |
+-------------------+--------------------------------------------------+--------------+-----------------+------------------
-+----------------------+-------------------+
29 rows selected (31.007 seconds)
```

Figure 7: Screenshot of subject difficulty classification

Problem 4: Export, Load, Query in Pig and Performance Comparison with Hive

This problem focuses on exporting data from the data warehouse into a CSV format, loading it into Apache Pig, executing analytical queries, and comparing their performance to equivalent HiveQL queries.

Export and Load Data into Pig

The data is first exported from the data warehouse into a CSV file format. The following Pig script loads this file while preserving the schema:

```
-- load_student_dw.pig

student_dw = LOAD '/user/student_dw.csv'
USING PigStorage(',')
AS (
    record_id: int,
    student_full_name: chararray,
    student_roll_no: chararray,
    subject_name: chararray,
    clean_course_code: chararray,
    subject_type: chararray,
    subject_variant: chararray,
    enrollment_status: chararray,
    academic_program: chararray,
    batch_year: chararray,
    semester_period: chararray,
    lead_faculty: chararray,
    instructor_names: chararray,
    total_attended: int,
    attendance_percentage: float,
    course_credit: int,
    final_grade: chararray,
    result_status: chararray
);

-- Remove header if needed (assuming first line is header)
student_dw_clean = FILTER student_dw BY record_id != 0;
```

Query 1: Student GPA Analysis in Pig

This script processes the dataset to compute GPA scores for each student. It loads the data with correct CSV handling using the PiggyBank library, filters and cleans the records, maps grades to points, and computes weighted GPA.

```
-- student_gpa_analysis_working.pig

REGISTER /home/narayana/pig-0.17.0/contrib/piggybank/java/piggybank.jar;

raw_data = LOAD '/user/student_dw.csv'
    USING org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'NO_MULTILINE', 'UNIX', 'SKIP_INPUT_HEA
    AS (
        record_id:chararray,
        student_full_name:chararray,
        student_roll_no:chararray,
        subject_name:chararray,
        clean_course_code:chararray,
```

```
        subject_type:chararray,
        subject_variant:chararray,
        enrollment_status:chararray,
        academic_program:chararray,
        batch_year:chararray,
        semester_period:chararray,
        lead_faculty:chararray,
        instructor_names:chararray,
        total_attended:chararray,
        attendance_percentage:chararray,
        course_credit:chararray,
        final_grade:chararray,
        result_status:chararray
    );

student_data = FOREACH raw_data GENERATE
    student_full_name,
    (course_credit MATCHES '\\d+' ? (float)course_credit : NULL) AS course_credit,
    final_grade;

valid_student_data = FILTER student_data BY
    course_credit IS NOT NULL AND
    final_grade IS NOT NULL;

record_count = FOREACH (GROUP valid_student_data ALL) GENERATE COUNT(valid_student_data);
DUMP record_count;

grades_mapped = FOREACH valid_student_data GENERATE
    student_full_name,
    course_credit,
    CASE final_grade
        WHEN 'A' THEN 4.0f
        WHEN 'A-' THEN 3.7f
        WHEN 'B+' THEN 3.3f
        WHEN 'B' THEN 3.0f
        WHEN 'B-' THEN 2.7f
        WHEN 'C+' THEN 2.3f
        WHEN 'C' THEN 2.0f
        WHEN 'D' THEN 1.0f
        WHEN 'F' THEN 0.0f
        ELSE NULL
    END AS grade_point;

valid_grades = FILTER grades_mapped BY grade_point IS NOT NULL;

weighted = FOREACH valid_grades GENERATE
    student_full_name,
    course_credit,
    grade_point,
    (course_credit * grade_point) AS weighted_points;

grouped = GROUP weighted BY student_full_name;

gpa_calc = FOREACH grouped {
    total_credits = SUM(weighted.course_credit);
    total_weighted = SUM(weighted.weighted_points);
    gpa = (float)(total_weighted / total_credits);
    GENERATE
```

```
        group AS student_full_name,
        COUNT(weighted) AS num_courses,
        total_credits AS total_credits,
        ROUND_TO(gpa, 2) AS gpa;
}

STORE gpa_calc INTO '/output/student_gpa_results_debug';

final_results = LIMIT gpa_calc 20;
DUMP final_results;
```

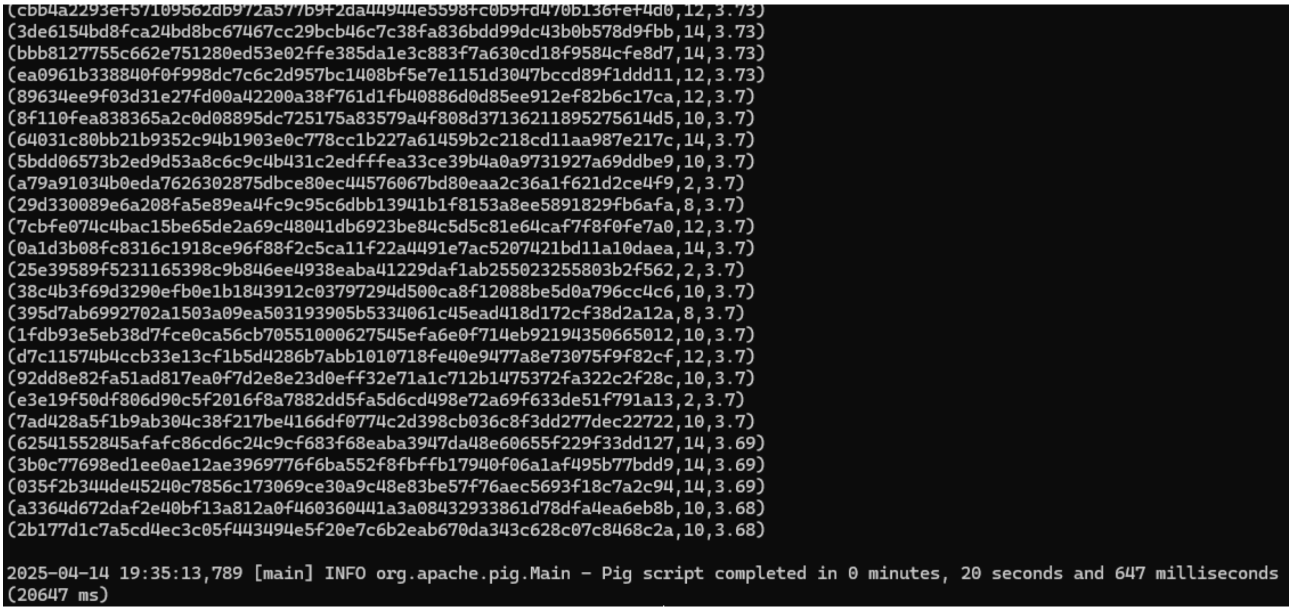


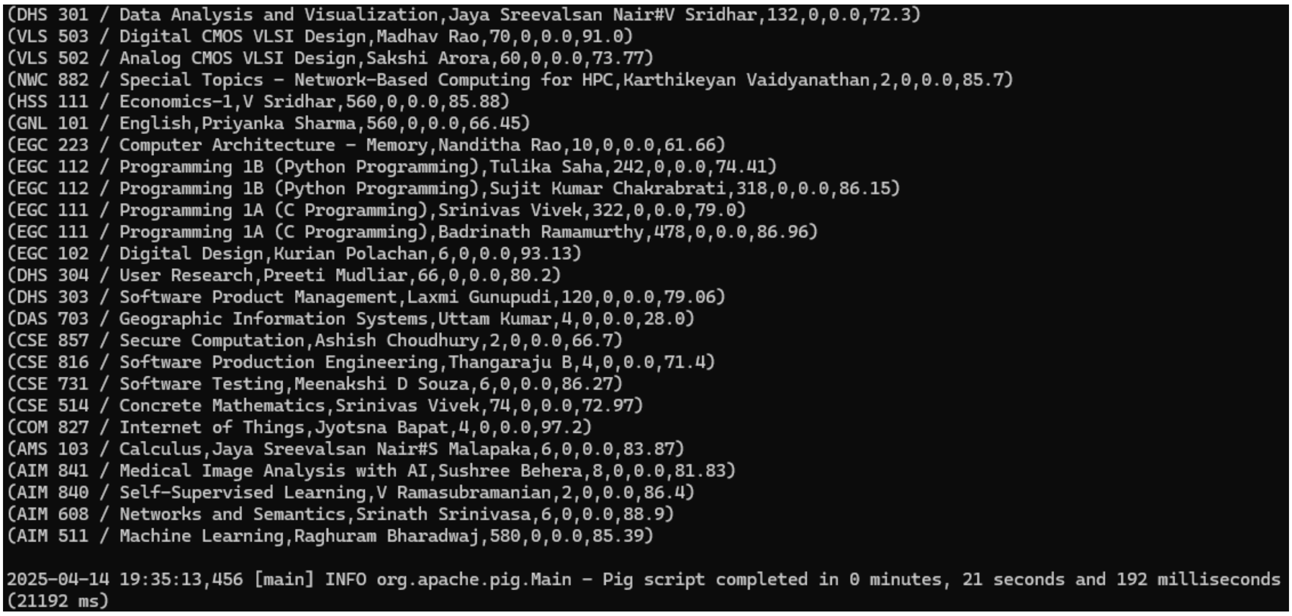
Figure 8: Screenshot of final GPA results

Query 2: Subject Failure and Attendance Analysis

This query identifies the number of students per subject-faculty combination, how many failed, and the average attendance.

```
filtered_data = FILTER student_data_warehouse BY result_status IS NOT NULL;

grouped_data = GROUP filtered_data BY (subject_name, lead_faculty);

subject_analysis = FOREACH grouped_data {
    total = COUNT(filtered_data);
    failing = FILTER filtered_data BY final_grade == 'F';
    failing_count = COUNT(failing);
    attendance_data = FOREACH filtered_data GENERATE (float)attendance_percentage;
    avg_attendance = AVG(attendance_data);
    fail_percent = (100.0f * (float)failing_count / (float)total);
    GENERATE
        group.subject_name AS subject_name,
        group.lead_faculty AS lead_faculty,
```

```
        total AS total_students,
        failing_count AS failing_students,
        ROUND_TO(fail_percent, 2) AS fail_percentage,
        ROUND_TO(avg_attendance, 2) AS avg_attendance;
};

ordered_results = ORDER subject_analysis BY fail_percentage DESC;

STORE ordered_results INTO 'subject_failure_analysis';
```



```
(DHS 301 / Data Analysis and Visualization,Jaya Sreevalsan Nair#V Sridhar,132,0,0.0,72.3)
(VLS 503 / Digital CMOS VLSI Design,Madhav Rao,70,0,0.0,91.0)
(VLS 502 / Analog CMOS VLSI Design,Sakshi Arora,60,0,0.0,73.77)
(NWC 882 / Special Topics - Network-Based Computing for HPC,Karthikeyan Vaidyanathan,2,0,0.0,85.7)
(HSS 111 / Economics-1,V Sridhar,560,0,0.0,85.88)
(GNL 101 / English,Priyanka Sharma,560,0,0.0,66.45)
(EGC 223 / Computer Architecture - Memory,Nanditha Rao,10,0,0.0,61.66)
(EGC 112 / Programming 1B (Python Programming),Tulika Saha,242,0,0.0,74.41)
(EGC 112 / Programming 1B (Python Programming),Sujit Kumar Chakrabrati,318,0,0.0,86.15)
(EGC 111 / Programming 1A (C Programming),Srinivas Vivek,322,0,0.0,79.0)
(EGC 111 / Programming 1A (C Programming),Badrinath Ramamurthy,478,0,0.0,86.96)
(EGC 102 / Digital Design,Kurian Polachan,6,0,0.0,93.13)
(DHS 304 / User Research,Preeti Mudliar,66,0,0.0,80.2)
(DHS 303 / Software Product Management,Laxmi Gunupudi,120,0,0.0,79.06)
(DAS 703 / Geographic Information Systems,Uttam Kumar,4,0,0.0,28.0)
(CSE 857 / Secure Computation,Ashish Choudhury,2,0,0.0,66.7)
(CSE 816 / Software Production Engineering,Thangaraju B,4,0,0.0,71.4)
(CSE 731 / Software Testing,Meenakshi D Souza,6,0,0.0,86.27)
(CSE 514 / Concrete Mathematics,Srinivas Vivek,74,0,0.0,72.97)
(COM 827 / Internet of Things,Jyotsna Bapat,4,0,0.0,97.2)
(AMS 103 / Calculus,Jaya Sreevalsan Nair#S Malapaka,6,0,0.0,83.87)
(AIM 841 / Medical Image Analysis with AI,Sushree Behera,8,0,0.0,81.83)
(AIM 840 / Self-Supervised Learning,V Ramasubramanian,2,0,0.0,86.4)
(AIM 608 / Networks and Semantics,Srinath Srinivasa,6,0,0.0,88.9)
(AIM 511 / Machine Learning,Raghuram Bharadwaj,580,0,0.0,85.39)

2025-04-14 19:35:13,456 [main] INFO org.apache.pig.Main - Pig script completed in 0 minutes, 21 seconds and 192 milliseconds
(21192 ms)
```

Figure 9: Screenshot of final subject failure analysis output

Query 3: Subject Difficulty Classification

This final query classifies subjects based on grade point averages and low grade percentages, identifying subjects as 'Very Easy' to 'Very Difficult'.

```
student_data = LOAD 'student_data_warehouse' USING org.apache.hive.hcatalog.pig.HCatLoader();

grouped_subjects = GROUP student_data BY (clean_course_code, subject_name);

subject_stats = FOREACH grouped_subjects {
    student_list = FOREACH student_data GENERATE student_roll_no;
    unique_students = DISTINCT student_list;
    num_students = COUNT(unique_students);

    attendance_data = FOREACH student_data GENERATE (float)attendance_percentage;
    avg_attendance = AVG(attendance_data);

    grade_points = FOREACH student_data GENERATE
        CASE final_grade
            WHEN 'A' THEN 4.0f
```

```
            WHEN 'A-' THEN 3.7f
            WHEN 'B+' THEN 3.3f
            WHEN 'B' THEN 3.0f
            WHEN 'B-' THEN 2.7f
            WHEN 'C+' THEN 2.3f
            WHEN 'C' THEN 2.0f
            WHEN 'D' THEN 1.0f
            ELSE 0.0f
        END AS grade_point;
    avg_grade_points = AVG(grade_points);

    all_grades = FOREACH student_data GENERATE final_grade;
    low_grades = FILTER all_grades BY final_grade == 'D' OR final_grade == 'C';
    total_count = COUNT(all_grades);
    low_count = COUNT(low_grades);
    low_grade_percentage = (100.0f * (float)low_count / (float)total_count;

    GENERATE
        group.clean_course_code AS clean_course_code,
        group.subject_name AS subject_name,
        num_students AS num_students,
        avg_attendance AS avg_attendance,
        avg_grade_points AS avg_grade_points,
        low_grade_percentage AS low_grade_percentage;
};

final_results = FOREACH subject_stats GENERATE
    clean_course_code,
    subject_name,
    num_students,
    ROUND_TO(avg_attendance, 1) AS avg_attendance,
    ROUND_TO(avg_grade_points, 2) AS avg_grade_points,
    ROUND_TO(low_grade_percentage, 2) AS low_grade_percentage,
    (
        CASE
            WHEN avg_grade_points < 2.0f THEN 'Very Difficult'
            WHEN avg_grade_points < 2.5f THEN 'Difficult'
            WHEN avg_grade_points < 3.0f THEN 'Moderate'
            WHEN avg_grade_points < 3.5f THEN 'Easy'
            ELSE 'Very Easy'
        END
    ) AS difficulty_level;

ordered_results = ORDER final_results BY avg_grade_points ASC, low_grade_percentage DESC;

STORE ordered_results INTO 'subject_difficulty_analysis';
```

```
(CSE 514,CSE 514 / Concrete Mathematics,37,73.0,2.91,2.7,Moderate)
(AMS 101,AMS 101 / Probability & Statistics,157,84.2,2.93,12.74,Moderate)
(HSS 111,HSS 111 / Economics-1,280,85.9,2.97,10.36,Moderate)
(EGC 112,EGC 112 / Programming 1B (Python Programming),280,81.1,2.97,6.79,Moderate)
(EGC 111,EGC 111 / Programming 1A (C Programming),280,83.8,2.98,15.0,Moderate)
(EGC 102,EGC 102 / Digital Design,160,80.7,2.98,7.5,Moderate)
(VLS 864,VLS 864 / Embedded Systems Design,31,88.7,2.98,0.0,Moderate)
(CSE 857,CSE 857 / Secure Computation,1,66.7,3.0,0.0,Easy)
(CSE 511,CSE 511 / Algorithms,137,78.6,3.03,8.03,Easy)
(AMS 103,AMS 103 / Calculus,120,76.7,3.05,4.13,Easy)
(AIM 512,AIM 512 / Mathematics for Machine Learning,134,62.7,3.06,6.72,Easy)
(GNL 101,GNL 101 / English,280,66.4,3.08,4.64,Easy)
(AIM 511,AIM 511 / Machine Learning,145,85.4,3.18,2.76,Easy)
(DHS 301,DHS 301 / Data Analysis and Visualization,33,72.3,3.2,0.0,Easy)
(DAS 703,DAS 703 / Geographic Information Systems,2,28.0,3.2,0.0,Easy)
(VLS 503,VLS 503 / Digital CMOS VLSI Design,35,91.0,3.27,0.0,Easy)
(CSE 816,CSE 816 / Software Production Engineering,2,71.4,3.3,0.0,Easy)
(DHS 304,DHS 304 / User Research,33,80.2,3.37,0.0,Easy)
(VLS 502,VLS 502 / Analog CMOS VLSI Design,30,73.8,3.37,0.0,Easy)
(AIM 841,AIM 841 / Medical Image Analysis with AI,4,81.8,3.58,0.0,Very Easy)
(EGC 223,EGC 223 / Computer Architecture - Memory,5,61.7,3.6,0.0,Very Easy)
(DHS 303,DHS 303 / Software Product Management,30,79.1,3.66,0.0,Very Easy)
(COM 827,COM 827 / Internet of Things,2,97.2,3.85,0.0,Very Easy)
(AIM 608,AIM 608 / Networks and Semantics,3,88.9,3.9,0.0,Very Easy)
(AIM 840,AIM 840 / Self-Supervised Learning,1,86.4,4.0,0.0,Very Easy)

2025-04-14 19:35:13,567 [main] INFO org.apache.pig.Main - Pig script completed in 0 minutes, 20 seconds and 454 milliseconds
(20454 ms)
```

Figure 10: Screenshot of final subject difficulty output

Performance Comparison and Analysis

Pig Latin scripts offer high flexibility in transforming and filtering unstructured data, particularly with the use of UDFs and PiggyBank libraries. In many cases, Pig jobs can run faster than equivalent HiveQL queries due to:

- More direct control over data flow and transformation logic.
- Efficient execution for pipeline-style processing with fewer intermediate stages.
- Lightweight scripting without the overhead of query compilation and optimization layers.

While HiveQL provides a more SQL-like syntax and is well-suited for complex joins and analytical queries, Pig often outperforms Hive in scenarios involving procedural logic, streaming transformations, and rapid data prototyping. Its concise and script-driven design makes it ideal for data engineers looking to fine-tune performance at a lower level.