Big Data Analytics

Assignment-1

Submitted By

Arnav Singh
2021019

Part 1. Data Modeling and Schema Design

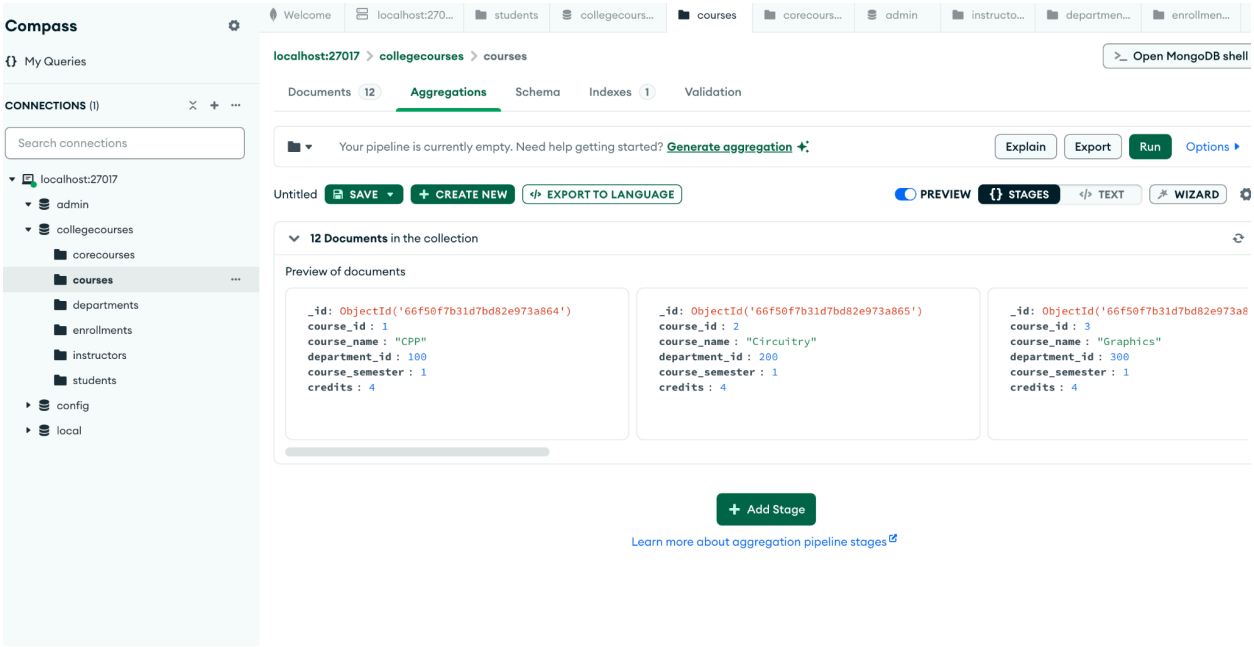
- Created a Relational Database using PostgreSQL, and further defined 6 tables.

    1. students (s_id, d_id,yoe,fname,lname,email)
    2. courses (c_id,c_name,credits,semester,credits)
    3. corecourses (c_id,d_id)
    4. departments (d_id,d_name)
    5. enrollments (e_id,s_id,c_id,i_id)
    6. instructors (i_id,fname,lname,email,did)

- Populated the tables
- Created a MongoDB database and created corresponding collections
- Added validations corresponding to the fields in the RDBMS
- The Table in its current stage doesn't need any further denormalisations, however, it is a denormalised form of a previous table that had multiple boolean fields in the course table to check if a course was a core subject for a particular department, this was rectified by creating a new table.
- I have formed logical relationships b/w different tables through fkeys, like using dept_id in core_courses to prevent unnecessary joins and comparisons in queries which leads to smooth-er processing.
- The enrollment table helps create many to many relationships b/w students and courses, allowing a student to be linked to multiple courses and vice versa
- And obviously I have enabled unique, pks and other descriptors, to ensure correct insertions throughout

Part 2. Data Migration

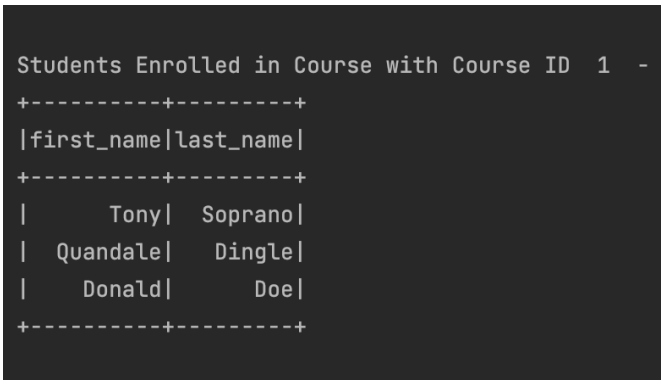
- Connected the relational database using psycopg2, and the Mongo DB using pymongo to the Python Environment
- Read the postgres data using Pandas' read_sql_query function
- Used to_dict(orient='records') to convert the postgres records to dictionaries, fitting the mongo format.
- Used the insert_many function for each pair of table-collection to load data in the mongo client.
- Verified the process by opening the Compass App, noting the increased storage taken by the database, and documents in each collection.



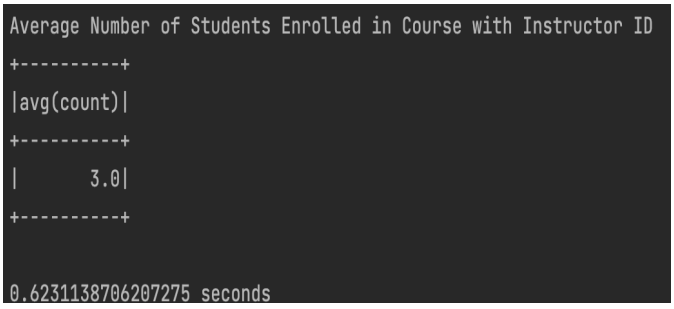
Part 4. Query Implementation using Apache Spark

- Imported pyspark and created a spark session to the local host using a mongo spark connector (which was accomplished with great difficulty after switching to Java 11)
- Loaded the data from MongoDB and performed queries on the loaded data, after performing a rather unnecessary student_df.show() command, to ensure that the effects of the cold start didn't affect the performances of the actual query.
- The results for the actual queries matched the outputs in the pgadmin server, but did take greater time to compute (obviously).

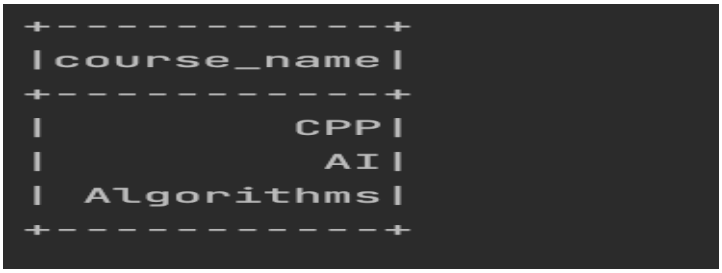
  The Results for each query are as follows:-
  - Joined enrollments and students



  - Joined enrollments and courses followed by avg

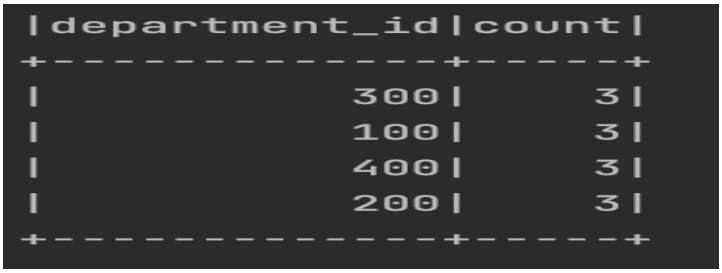


- Filter courses by department_id



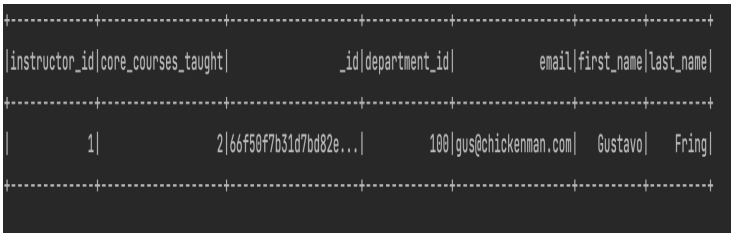
```
+-------------+
|course_name|
+-------------+
|          CPP|
|           AI|
|  Algorithms|
+-------------+
```

- Grouped students by department_id



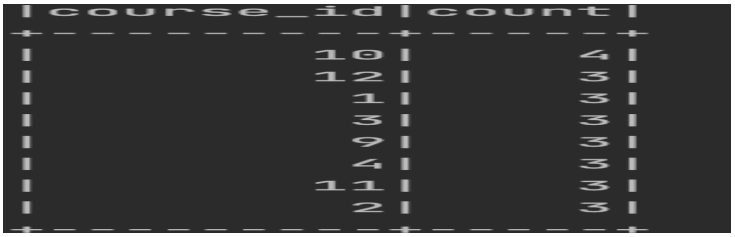
```
|department_id|count|
+---------------+-----+
|              300|    3|
|              100|    3|
|              400|    3|
|              200|    3|
+---------------+-----+
```

- Filters and joins among students, instruct., enroll.



```
+------------+-----------------+--------------------+-------------+--------------------+----------+---------+
|instructor_id|core_courses_taught|                 _id|department_id|               email|first_name|last_name|
+------------+-----------------+--------------------+-------------+--------------------+----------+---------+
|           1|                2|66f58f7b31d7bd82e...|          100|gus@chickenman.com|    Gustavo|    Fring|
+------------+-----------------+--------------------+-------------+--------------------+----------+---------+
```

- Sorted enrollments by course id counts



```
|course_id|count|
+---------+-----+
|       10|    4|
|       12|    3|
|        1|    3|
|        3|    3|
|        9|    3|
|        4|    3|
|       11|    3|
|        2|    3|
+---------+-----+
```

Processing time (in sec) for each query using Python's Time library:-

[0.6514639854431152, 0.6963191032409668, 0.15226292610168457, 0.313531160354614 26, 0.7763099670410156, 0.194403171539306 64]

Part 4. Performance Analysis and Optimisation

- The Processing times for each query were discussed in the last part, they were proportional to the complexity (number of joins, sorting ops etc.) associated with the query. The data transfer was successful and maintained integrity.
- As for the strategies, In my experiments, Indexing in MongoDB (single field index) and query optimization in spark proved to be the most successful. Data transformation didn't yield decent results throughout owing to a smaller dataset, where maintaining transformations was costlier than its benefits, as opposed to query optimization, which improves performance regardless of the dataset's size, while single indexes outperformed compound indexes in tables of similar importance.