ASSIGNMENT - 1
BDA (CSE 557)

NAME - PARAS DHIMAN

ROLL NO. - 2021482

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

Report - 2

Data Migration Report: PostgreSQL to MongoDB

Overview

This report outlines the steps taken to implement a data migration pipeline from a PostgreSQL database to a MongoDB database. The migration process involves extracting data from multiple tables in the relational database, transforming it to fit the document-based structure of MongoDB, and loading it into the destination database. This process ensures data consistency and integrity throughout the migration.

Data Migration Pipeline

The pipeline is structured into three main components:

1. **Extract**: Data is fetched from the PostgreSQL database.
2. **Transform**: The data is reshaped to fit MongoDB's document structure.
3. **Load**: Transformed data is inserted into MongoDB collections.

Step 1: Data Extraction

In this phase, data is extracted from the following PostgreSQL tables:

- `students`
- `courses`
- `instructors`
- `departments`
- `enrollments` (including course information)
- `course_instructors` (mapping between courses and instructors)

For each table, SQL queries are executed to fetch the data, and the results are stored in a Python dictionary for further processing. We handle the extraction of additional relationships, such as student enrollments in courses and instructor assignments, via JOIN queries in PostgreSQL. This ensures that relevant associations between entities are extracted for use in the transformation stage.

Example SQL Queries:

- `SELECT * FROM students`
- `SELECT instructor_id, course_id FROM course_instructors`

Key Considerations:

- Ensure that the extraction process captures all relevant data from the relational schema.
- Handle large datasets with efficient querying and pagination if needed.

Step 2: Data Transformation

The extracted data is transformed to match the MongoDB document-oriented structure. The relational data, previously normalized into multiple tables, is denormalized during the transformation process to fit MongoDB's nested document model. The transformation logic includes the following:

- **Students**: Each student record is represented as a document with nested enrollment information.
- **Courses**: Each course document contains instructor IDs and an enrollment count.
- **Instructors**: Instructor documents reference the courses they are teaching.
- **Departments**: Department documents contain lists of student and course IDs.

Data Cleaning and Transformation:

- Redundant or irrelevant fields are excluded.
- Nested fields, such as enrollments within student documents, are constructed based on the relational joins (e.g., course enrollments linked to student IDs).
- Relationships between entities (such as courses and instructors) are maintained by embedding relevant information directly within MongoDB documents.

Example Transformation for Students:

python:

```
{
    '_id': 1,
    'first_name': 'John',
    'last_name': 'Doe',
    'email': 'john.doe@example.com',
    'department_id': 101,
    'enrollments': [
        {'course_id': 1, 'grade': 'A'},
        {'course_id': 2, 'grade': 'B'}
    ]
}
```

Step 3: Data Loading

The final step is to load the transformed data into MongoDB. Data is inserted into the following MongoDB collections:

- `students`
- `courses`
- `instructors`
- `departments`

The `insert_many` method is used for bulk insertion of documents into MongoDB. This ensures efficient loading of large datasets.

Collections and Data Insertion:

- Each collection is populated with transformed documents.
- Data consistency is maintained by carefully mapping the relational data to the corresponding MongoDB collections and documents.

MongoDB Collections:

- `students`: Contains individual student records with nested enrollment data.
- `courses`: Stores course details, including enrollment count and assigned instructors.
- `instructors`: Represents instructors with references to courses they teach.
- `departments`: Contains department information with lists of associated students and courses.

Data Integrity and Consistency

Throughout the migration process, data integrity is ensured by:

- Correctly mapping relational data into the document-based model.
- Handling foreign key relationships through embedding and referencing (e.g., student enrollments and course assignments).
- Performing validation during extraction and transformation to detect and fix inconsistencies or missing data.

Challenges and Solutions

- **Handling Nested Relationships**: The relational model relies heavily on joins, while MongoDB supports embedding documents. To address this, relationships were carefully transformed to preserve the logical connections between entities while ensuring that data could be easily queried in MongoDB.
- **Data Consistency**: During transformation, care was taken to ensure that all data remained consistent and referential integrity was maintained between entities (e.g., students, courses, instructors).

Conclusion

The migration pipeline successfully transfers data from PostgreSQL to MongoDB, ensuring that the structure is optimized for the NoSQL database while preserving all essential relationships and data integrity. The process was completed in three main steps: extraction from PostgreSQL, transformation into MongoDB's schema, and loading into MongoDB collections.