Report: Relational Schema to MongoDB Schema Mapping

1. Overview

The goal was to migrate the university database from a relational model (PostgreSQL) to a non-relational model (MongoDB) while maintaining the core data relationships and optimizing performance for read-heavy operations. In the relational model, data is stored in normalized tables, while in MongoDB, the data is organized into collections with documents, often involving denormalization for performance gains.

The following relational tables were mapped to MongoDB collections:

- **Students** → `students` collection
- **Instructors** → `instructors` collection
- **Courses** → `courses` collection
- **Departments** → `departments` collection
- **Enrollments** (junction table) → `students` and `courses` collections (denormalized)

2. Mapping Details and Justifications

2.1. Students Table → Students Collection

- **Relational Schema**: In the relational schema, the `Students` table stores student information (e.g., `student_id`, `first_name`, `last_name`, `email`, `mobile`) and maintains a foreign key (`department_id`) to the `Departments` table.
- **MongoDB Schema**: In MongoDB, the `students` collection is structured similarly, but with some optimizations:
  - The `enrollments` of the student (from the junction table in the relational schema) are embedded as an array within each `student` document. Each enrollment contains references to the `course_id` and enrollment details (e.g., `enrollment_date`, `grade`).
  - This denormalization avoids the need for expensive joins when querying student data and their enrollments.

Justification for Design:

- Embedding the `enrollments` within the `students` collection improves read performance for common queries, such as retrieving student details and their enrolled courses. It reduces the need for multiple lookups or joins across collections.

2.2. Instructors Table → Instructors Collection

- **Relational Schema**: The `Instructors` table stores instructor details (e.g., `instructor_id`, `first_name`, `last_name`, `email`) and a foreign key (`department_id`) linking to the `Departments` table.
- **MongoDB Schema**: The `instructors` collection in MongoDB mirrors the structure of the relational schema, but with embedded data:
    - The `courses` taught by the instructor are embedded as an array within each `instructor` document. Each course includes details like `courseId`, `courseName`, and the department the course belongs to.

Justification for Design:

- Embedding the `courses` array within the `instructors` collection allows efficient querying for instructor profiles, including the courses they teach, without requiring a join to the `courses` collection.

2.3. Courses Table → Courses Collection

- **Relational Schema**: The `Courses` table stores course information (e.g., `course_id`, `course_name`, `department_id`, `instructor_id`). The course has foreign keys to the `Departments` and `Instructors` tables.
- **MongoDB Schema**: In MongoDB, the `courses` collection maintains:
    - References to the `departmentId` and `instructorId`.
    - An `enrollments` array, which contains `studentId`, `enrollmentDate`, and `grade` for all students enrolled in the course.

Justification for Design:

- Denormalizing enrollments within the `courses` collection reduces the need for multiple queries when retrieving course data with enrollments. This structure is optimized for use cases like getting all students in a particular course or finding which instructor is teaching a specific course.

2.4. Departments Table → Departments Collection

- **Relational Schema**: The `Departments` table holds the department details (e.g., `department_id`, `department_name`).

- **MongoDB Schema**: The `departments` collection in MongoDB is a direct mapping of the relational schema with no major changes.
    - Each document holds `departmentId` and `departmentName` as fields.

Justification for Design:

- Since departments are relatively static entities and are not frequently updated, keeping them in their own collection makes querying them directly or referencing them from other collections efficient.

2.5. Enrollments Table → Embedded in `students` and `courses`

- **Relational Schema**: The `Enrollments` table acts as a junction table that links students to courses. It stores `student_id`, `course_id`, `enrollment_date`, and `grade`.
- **MongoDB Schema**: The `Enrollments` table does not exist as a separate collection in MongoDB. Instead, enrollment information is embedded in both the `students` and `courses` collections:
    - In `students`, each document contains an `enrollments` array with the `courseId` and other details.
    - In `courses`, each document has an `enrollments` array with the `studentId` and enrollment details.

Justification for Denormalization:

- Denormalizing the `Enrollments` table into both the `students` and `courses` collections optimizes for read-heavy workloads where fetching student or course details along with their enrollments is common. This avoids complex joins and reduces the number of queries needed to fetch enrollment data.

3. Denormalization Decisions

Denormalization of Enrollments:

- The primary denormalization occurred with the `Enrollments` table. Instead of keeping it as a separate collection, enrollment data was embedded within both `students` and `courses`. This design is optimized for queries where you want to fetch either a student's enrollment details or the list of students enrolled in a course. The trade-off is some redundancy and additional storage space, but it greatly improves query efficiency in read-heavy applications.

Embedding Courses in Instructors:

- Courses taught by an instructor were embedded in the `instructors` collection. This allows querying an instructor's profile along with the courses they teach in a single query, making it faster to fetch relevant data.

4. Advantages of the MongoDB Schema

- **Optimized for Read-heavy Workloads**: Embedding related data reduces the need for complex joins, making it faster to retrieve information about students, instructors, courses, and enrollments in typical queries.
- **Denormalization for Performance**: By embedding data, we avoid expensive joins required in relational databases, resulting in more efficient data retrieval for common use cases.
- **Flexible Schema**: MongoDB allows for flexible document structures, which is helpful as new fields can be added without altering the existing documents, providing a more scalable and adaptable data model.

5. Potential Trade-offs

- **Data Redundancy**: Denormalization introduces redundancy, particularly with enrollment information stored in both `students` and `courses`. While this improves read performance, it increases the amount of storage used and requires careful consideration when updating data to ensure consistency.
- **Complexity of Updates**: Since the same data is stored in multiple places (e.g., enrollment details in both `students` and `courses`), updates must be propagated across all collections, which can increase the complexity of write operations.