Final Project of Data Management

Alessandro Mecchia

December 2024

1 Introduction

1.1 Objective

In this report, I compare SQL and NoSQL solutions for managing prescriptions in a medical database. The goal is to evaluate functionality, complexity, robustness, and performance of both approaches based on practical implementation and testing.

2 SQL Solution: Relational Database Implementation

2.1 Schema Design and Setup

I created three new entities: `Prescription`, `RepeatablePrescription`, and `Drug`. Then I designed relational connections between them:
- 1:N between `InternalDoctor` and `Prescription`
- 1:N between `Patient` and `Prescription`
- 1:N between `Drug` and `Prescription`
- 1:1 between `Prescription` and `RepeatablePrescription`

2.2 Steps for Implementation

I configured the database connection using an `.env` file for Oracle DB credentials, then I created a `db.py` file to manage the database connection and define the `Model` base class. Then I developed a `models.py` file defining all entities and relationships and finally I populated the database and wrote queries using `population_and_SQL_queries.py`.

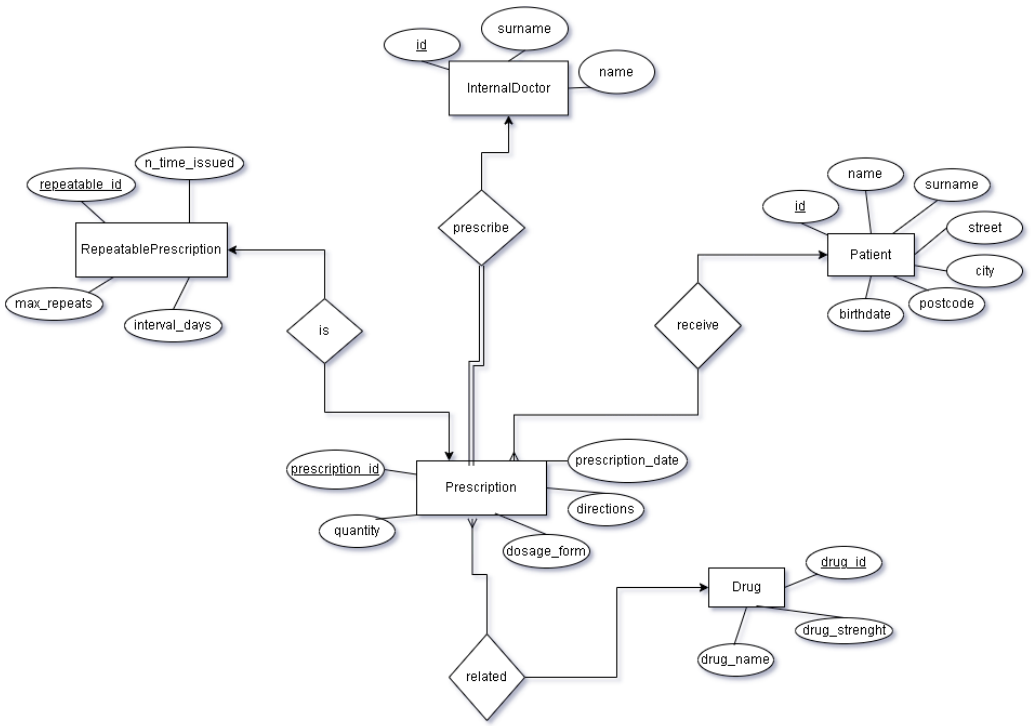
2.3 SQL Queries Implemented

I tested the new tables with three queries:
- Query 1: Retrieve all prescriptions along with doctor and patient details.
- Query 2: Retrieve all patients who have repeatable prescriptions.
- Query 3: Find the most prescribed drug based on the number of prescriptions.

3 NoSQL Solution: Migration to MongoDB

3.1 Data Migration

I developed a script `migration_to_no_sql.py` to transform relational data into MongoDB-compatible documents, it means that I had to transform the data into a JSON-type format in order to be able to create documents for MongoDB correctly. You can see that unlike the SQL case where I had to create a separate entity for the repetition of the prescription, in the case of NoSQL I simply added a repeatable key to which the necessary values are assigned only if the prescription is repeatable, otherwise the field is not created.



3.2 Queries on MongoDB

After the migration, I implemented again the same queries of the SQL solution to test if the data are stored in a correct way and also to see in what format they are returned.

4 Comparison: SQL vs NoSQL Solutions

4.1 Functionality

SQL: I had to define clear relationships between tables, such as creating a separate RepeatablePrescription table linked to Prescription. This approach was very structured and made it easier to apply the rules, but at the same time I had to manage multiple relationships, which increased the complexity of the database.
NoSQL: In MongoDB, I added the details of the repeatable prescription directly into the Prescription document as a field. This made the schema simpler and more flexible.

4.2 Complexity

SQL: Since I have multiple entities, writing queries often required multiple joins between tables, which made them complex to manage. The ER model was also difficult to create.
NoSQL: Queries were much simpler since all related data are stored in a single document. Fetching a prescription with its details required just one query.

4.3 Robustness

SQL: Data integrity was guaranteed by foreign keys and constraints. In this case I was quite clear about the constraints but this is not always the case.
NoSQL: Without hard constraints, I had more flexibility but I had to ensure data consistency manually. I had to pay special attention to the possible repetition of the prescription, which created a new key to which all fields of the entity had to be assigned.

4.4 Performance

SQL: In this case, the number of joins affects performance. The more joins, the slower the queries.
NoSQL: Retrieving a prescription with its doctor, patient, and drug details was fast because all this data was stored together in one MongoDB document. but When data is denormalized, it can be duplicated across multiple documents. If the same data (e.g., a doctor's details) needs to be updated, all documents containing that data must be updated as well.

5 Advantages and Drawbacks of Each Solution

5.1 Advantages of SQL

It is well suited to structured data with strict relationships: SQL was ideal for maintaining well-defined relationships between prescriptions, repeats, doctors, patients and drugs. For example, foreign key constraints ensured that a prescription could not exist without a valid doctor and patient.

5.2 Drawbacks of SQL

As the number of entities and relationships increases, so does the complexity of both the model and the queries, in fact as I explained the increase in joins also affects the efficiency of the query.

5.3 Advantages of NoSQL

In MongoDB, I incorporated related data (such as doctor, patient and drug details) directly within the `Prescription` document. This eliminated the need for a separate entity, simplifying the schema.
- **Faster retrieval of nested data**:
Retrieving a prescription with its doctor, patient and repeatable data details was much faster, as everything was stored in one document. There was no need for joins, as required by SQL.

5.4 Drawbacks of NoSQL

MongoDB lacks built-in rules to ensure data consistency, so I had to rely on application logic to validate connections between data, like ensuring a prescription refers to a valid doctor or patient. This made the system more flexible but introduced the risk of errors. Additionally, MongoDB struggles with managing complex relationships since it doesn't have strict rules like SQL does.

6 Conclusion

Based on this exercise, SQL databases are more suitable for applications requiring strict relationships and data integrity, while NoSQL databases offer flexibility and scalability for unstructured or semi-structured data. For this specific case, I preferred to use MongoDB because it was easier to implement the concept of repeatable prescriptions. In addition, querying the data was faster and more immediate, allowing me to easily retrieve the necessary information with minimal complexity, as the structure is very similar to a dictionary with keys and values assigned to them, making MongoDB a more practical choice for this use case.