Project

Polyglot Persistence

Urban Transportation system

The project is carried out during practical classes and outside class hours, in groups of 3/4 students. Each student must present and discuss the project individually, either orally or in writing. All group members must be present during the presentation; otherwise, absent members will receive a zero grade. The project must be submitted within the specified deadline, with penalties or restrictions applied for late submissions, as described in Moodle.

Objectives

The goal of the project is to help students of the "Advanced Topics in Databases" course develop skills in using different database paradigms and apply them to real-world systems design and implementation. The project involves the development of an application that simulates an intelligent urban vehicle transport system, using various database management systems (DBMS), both relational and non-relational. Development will be done in the Python programming language.

The application should have a user-friendly interface, allowing users to consult lines and itineraries, view available routes, select their preferred routes, and receive real-time updates on schedules and service changes. After using the system, users should provide feedback on the usability of the application and evaluate their satisfaction with the overall experience. The platform should be designed with a focus on scalability, performance, and flexibility in data management.

By integrating Python with databases, the project aims to provide a comprehensive solution for an intelligent urban transportation system that meets the needs of both end users and administrators.

Requirements

With the growth of cities and the increased demand for urban mobility solutions, a more intelligent, dynamic, and user-centered public transportation system has become essential.
In this context, we propose the development of an intelligent urban transportation monitoring and management application capable of integrating both operational data and relevant passenger information. The application will include with features aimed at both end users and system administrators.
The application must use:

- ⇒ a relational database, responsible for storing structured data with strong relationships, such as information about users, drivers, vehicles, transport lines, and schedules;
- ⇒ three NoSQL databases of different models, such as document model, key-value, graphs, column-oriented, etc.

The types of data to be stored include structured data, geospatial data (stop locations) and flexible data (settings and preferences), historical data (trips taken, usage by line, etc.), and complex relationships (e.g., route suggestions and connections between transport lines).
This polyglot architecture is designed to provide a richer user experience by ensuring better performance and more appropriate data handling, tailored to the specific needs of each data type.

Application Functionalities

The application should include at least the following functionalities:

- **Public transport route information**, such as route name, associated vehicle, operating hours, origin, and destination;
- **Itineraries and stops details**, showing geographical data, estimated stop times, and location on the map, among others;
- **Travel history**, allowing users to view past trips, including routes taken and start/end times;
- **User interaction options**, like marking favorite lines, getting personalized notifications (e.g., route changes or estimated arrival times), and viewing recent trip history;

- **Real-time vehicle tracking simulation**, with regular updates of their locations and dynamic visualization on the map, allowing the user to track the approximate position of vehicles in circulation;
- **Automatic suggestion of optimized routes** with automatic recommendations for the best way to travel between two points in the city;
- **User authentication system**, supporting for secure login, account creation and management and automatic logout after a period of inactivity.

Tasks

Part I - Definition of Schemas and Architecture

This part involves defining the databases schemas and application architecture, focusing on theoretical foundations, design decisions, and suitability for the system's needs

A. Database Schemas (Relational and NoSQL)

- Representation Models:
  - Relational Model (RM) for the relational database.
  - Data structure for the chosen NoSQL databases
- Design Patterns:
  - Normalization and relational modeling for the relational database.
  - Modeling based on design patterns for NoSQL databases (documents, graphs, key-value, etc)
- Justification of Choices:
  - Summary of the decisions made for each database model, focusing on performance, flexibility and scalability.

B. Architecture

- Detailed description of the application architecture.
- Detailed explanation of why the architecture was chosen, taking into account the application's technical and functional requirements.

Part II – Application Implementation

A. The application must be implemented with the following characteristics:

- **Backend**
  - The backend must be capable of integrating the three databases (1 relational and 3 NoSQL), performing read and write operations on all databases, depending on the data.
- **Frontend**
  - The frontend must be implemented to display the system's features in an interactive and intuitive way for the end user.
- **Data Integration and Optimization:**

  Evaluate how well the application integrates data from different databases (SQL and NoSQL) and applies performance optimization techniques. The goal is to demonstrate an understanding of polyglot persistence and data management efficiency, not to implement complex real-time cross-queries.
  - The system shall include at least two features that involve data from multiple databases, demonstrating the ability to integrate or combine information from different persistence models;
  - Queries and data access must be optimized using indexes and best practices to improve performance.

B. Select two different scenarios that show when indexes help improve query performance and when they don't. Explain your choices.

Requirements:

⇒ one query using the relational database;

⇒ one query using one of the NoSQL databases used in the project.

For each scenario

⇒ Describe the original query (without index);

⇒ Create and apply the index(es);

⇒ Show results that compare performance before and after (like execution time, query plan, logs);

⇒ Explain why the indexes were effective or not.

Queries must be related to the application's features and show real use cases.

Deadlines and Deliverables

Interim Delivery - Definition of Schemas and Architecture:

Delivery: By 23:59 p.m. on **November 1**, 2025.

Objective: Submit a detailed report presenting the database schemas, a description of the application architecture, and justification for all decisions made.

Final Delivery - Application Implementation and Updates to Part I (if needed)

Delivery: By 23:59 p.m. on **December 7**, 2025.

Objective: Submit a detailed report covering all changes made to Part I after feedback, as well as all tasks developed in Part II**.**

Oral Presentation:

Date: During weeks 13 and 14.

Objective: Each group must present their work, discussing the implementation, technical choices, and evaluation of optimization scenarios with indices.

Note: Reports must be submitted in ZIP format. The ZIP file should be named following format: ClassQ_Branch_xxxxxxx_yyyyyyy_zzzzzzz.ZIP where:

⇒ ClassQ - class name

⇒ Branch – MEI branch name

⇒ xxxxxx_yyyyyyy_zzzzzzz - student number.

Penalties

⇒ Late submissions will be penalized 10% for each day of delay past the deadline

⇒ Any group member who is absent during the scheduled presentation will receive a ZERO grade.