******

***BD Project***

*University Management System*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

João Ferreira, Guilherme Moreira & Rodrigo Manão

2023217920 | 2023222537 | 2023207589

# ***Index***

[**Index**](#_f7yhdy5k9g14) **1**

[**Introduction**](#_8etb3z8086q)**2**

[**Transactions**](#_srvk6icfryk2) **2**

**Potencial Concurrency Conflicts****3**

**Entity-Relation Model Description 4**

**Development plan 4**

# 

# ***Introduction***

* In the context of higher education, the efficient management of academic and administrative information is essential for the proper functioning of universities, making it crucial to adopt technological solutions that allow organizing, storing, and processing large volumes of data in a structured and secure manner. In this project, our goal is to develop a University Management System. This system handles the course administration, student enrollment, class scheduling, financial tracking, and evaluation processes.

# ***Transactions***

* Transactions are blocks of operations that **must be executed entirely or rolled back in case of failure** to ensure the consistency of the database. In this case, some critical scenarios require transactions.
* ***Course enrollment***
  + In this transaction, a student enrolls in a course, requiring multiple operations to be completed successfully. First, the system verifies whether the student meets the course prerequisites and checks for available slots in the course edition. Second, it verifies if the classes he wishes to attend still have available slots and if the student has enough balance in his account. If everything is in order, the enrollment is recorded, the student's financial account and course edition capacity are updated, and a transaction entry is created. However, if anything goes wrong at any step, the entire process is rolled back, ensuring that no incomplete or incorrect data is saved.
* ***Activity registration***
  + In this transaction, a student registers in an activity and the associated fee is deducted from the student’s account. If the student withdraws from an activity, the charges will be recalculated. If anything goes wrong at any step, the entire process is rolled back, ensuring that no incomplete or incorrect data is saved.
* ***Grade Submission***
  + In this transaction, when a grade is submitted, the system first checks if the student is enrolled in the course, if so, the grade is submitted to the respective table together with its student and its period. The academic averages of the approved students are also updated, within their degree programs.

***Potencial concurrency conflicts***

* ***Financial account update***
  + If a student tries to pay a fee while his balance is being updated, there can be an incorrect calculation of the student’s balance. To avoid this we can set an access exclusive lock, so that the balance cant be read until it is updated.
* ***Student enrollment***
  + If two students try to enroll at the same time there could be a breach of the maximum capacity of an edition or class. To prevent that we can use an exclusive lock, so that only one student can be updated at a time.
* ***Activity register***
  + Here, the same concurrency problem as the student enrollment one could happen (two students trying to register at the same time), so to fix that, we can also use an exclusive lock, only allowing one update at a time.

# ***Entity-Relation Model Description***

* In the relationship between the entities *edition* and *class*, we considered *class* as a weak entity since classes only exist when associated with a specific course edition. We also reflected on the possibility of not treating it as a weak entity to accommodate scenarios such as standalone or open classes, but we choose not to explore that case further in this model.
* We consider *enrollment* a weak entity in relation to *student*, as enrollments only exist if there is a student to enroll, and a student can have multiple enrollments.
* *Evaluation* is weak in relation to *evaluation\_period* because the evaluation belongs to a period, therefore, it can’t be an evaluation without a period.
* *Evaluation\_period* is weak in relation to edition because it belongs to an edition of a course and therefore there can’t be an evaluation period without an edition. With this evaluation\_period can be identified through its name and its corresponding edition.
* In this project, we consider that only students have a financial account, as it is not explicit in the statement that employees must have a financial account.
* We decided to create the *assistant\_instructor* and *coordinator\_instructor* entities to better understand the exercise. However, following the project statement, it would not be necessary, as they could be included in the instructor entity, manipulating this through relationships.
* *Payment* entity is used to activate triggers in *financial\_acount* entity, meaning it will be responsible for adding/taking money from student accounts.
* The *grade\_log* entity is used to save student grade logs among every degree program and is associated with a specific edition.
* During the ER elaboration, we thought of creating a University entity, which would be directly assigned to a department, where the classroom was located. However, we thought that it was not needed, so we kept the entity *building* only.
* We decided to relate the *staff* entity with student, person (maintaining heritage relation) and enrollment, so we can keep track of the staff who added that registries.

***Development Plan***

**Phase 1: Planning & Database Design (Week 1-3)**

* Task 1: Understanding Requirements (All Members)
  + Read project documentation
  + Identify key functionalities
* Task 2: Design ER Diagram & Relational Model (All Members)
  + Use ONDA for data modeling
  + Define entities, attributes, relationships
* Task 3: Define Transactions & Concurrency Strategies (All members)
  + Writing the introduction and observations
  + Identify operations requiring transactions
  + Define strategies for conflict handling

### **Phase 2: Database Implementation & Basic API Setup (Week 4-6)**

* Task 4: Implement Database Schema in PostgreSQL (Guilherme)
  + Create tables, constraints, sequences
  + Write SQL scripts for database initialization
* Task 5: Develop REST API (Basic Functions) (João, Rodrigo)
  + Set up Python backend with Flask/FastAPI
  + Implement authentication (JWT)
  + Create a simple retrieval endpoint

### **Phase 3: Core Features Implementation (Week 7-10)**

* Task 6: Implement queries (Guilherme, Rodrigo)
  + User registration & authentication
  + Transaction operations implementations
* Task 7: Implement Business Logic & Triggers (João)
  + Write PL/pgSQL procedures and triggers
  + Ensure automatic updates for financial records, grades, etc.

### **Phase 4: Testing, Debugging & Finalizing (Week 11-12)**

* Task 8: Complete REST API & Test Endpoints (Rodrigo, João)
  + Use Postman to validate API
  + Ensure security & transaction management
* Task 9: Final Documentation & Video Demo (All members)
  + Write installation & user manuals
  + Record a 5-minute demo video