Welcome to the CS306 Project- Phase 2

In this file, I aim to explain the tools and technologies I used to demonstrate triggers and procedures in a relational database. I will also share the SQL code written for each trigger and procedure, along with their purposes. Additionally, I've developed a simple web application to showcase how the database changes when we insert related rows into the tables.

💻 How the Web Application Works

Normally, this app would require users to register. Upon registration, a UserID would be automatically assigned. When writing a review (currently the review description is not stored in the database, but I plan to add that feature later), users would select a restaurant, and we would retrieve the corresponding RestaurantID. After submitting a review, a unique ReviewID would also be generated automatically.

However, since the goal of this phase is to demonstrate triggers and procedures, I've chosen to manually enter all the data into the form, simulating a user uploading a review from their account.

I used PHP for the backend, HTML and CSS for the frontend, and MySQL for database management. To connect all components and enable real-time database updates upon new insertions, I used the XAMPP Control Panel, which provides a local server environment for testing and development.

🔁 Triggers

Trigger 1 — Update Average Rating
As shown in my Entity-Relation diagram, in order to maintain the average rating of each restaurant, every new review must be included in the calculation. Therefore, I created a trigger that updates a restaurant's average rating *after* each new review is inserted.

Trigger 2 — Flag Low-Rated Restaurants
Since this is a touristic app, it should help users identify which restaurants are not worth visiting. I added a flag column to the restaurants table. If the restaurant's average rating drops below 4, this flag is set to 1; otherwise, it remains 0. This is handled through another trigger executed after each review is inserted.

Triggers 3, 4, and 5 — Update User Reputation
A user's reputation is influenced by:

- Number of reviews they write
- Total likes and follow-ups they receive
- Number of restaurants they mark

To handle this, I created three separate triggers that call the same stored procedure for reputation updates whenever a relevant event (e.g., marking a restaurant, submitting a review, receiving a like or follow-up) occurs.

⚙️ Procedure

Procedure 1 — UpdateUserReputation
This stored procedure calculates a user's reputation using a formula that considers:

- Number of reviews submitted
- Total likes and follow-ups received
- Number of restaurants marked

🔒 Constraints

I also implemented the following constraints in the schema:

- Unique email constraint to ensure no duplicate registrations
- Age check to validate age boundaries

📌 Additional Notes

I removed the ISA hierarchy from the original entity sets and instead added a type column to the restaurants table to distinguish between different restaurant types (e.g., cafes, patisseries). This design choice simplifies the implementation since I'm working solo on this project. The distinct properties of these restaurant types are stored directly in the restaurants table.

I will also be sharing the SQL code for each trigger and procedure in this file so that you can review them if you're interested.