University System Design

By : yousef saber abd-elkarim
ITI Data Management Intake (44)

Table of Contents:

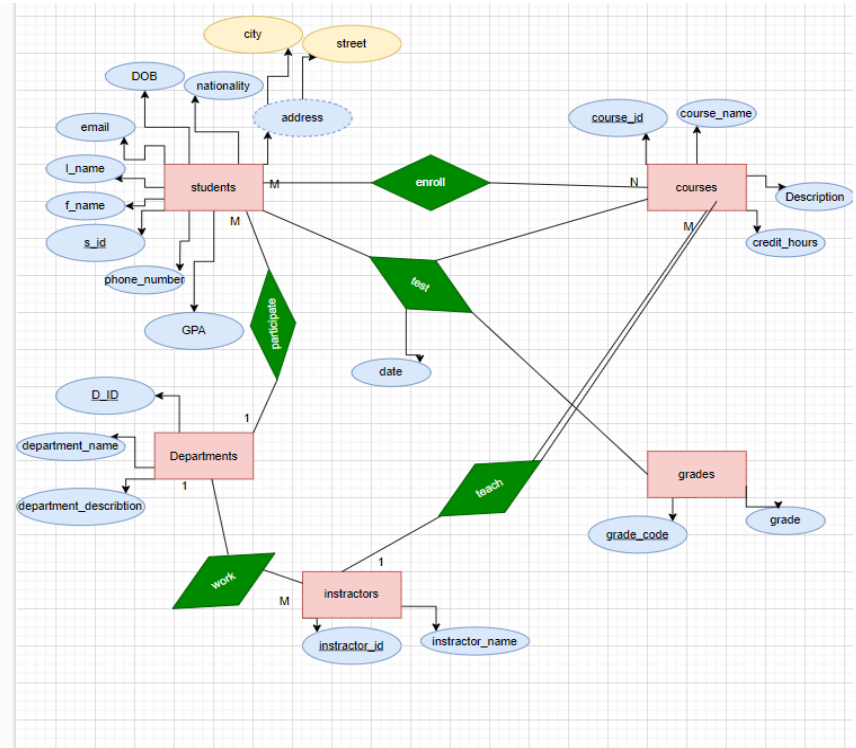
1. Introduction:

This detailed guide provides a thorough overview of a University System, including its design, execution, and features. It encompasses database structuring, SQL incorporation, PL/SQL procedures, an automated script, and a Java program. Each segment is intricately explained to ensure a comprehensive grasp of the system's framework, aiding in its smooth development, upkeep, and issue resolution.

2. Database design.

The document offers a detailed insight into the relational database schema designed to manage student, instructors ,course, department, and grade data efficiently. It follows normalization principles meticulously to maintain data integrity. This schema forms the basis for further actions, such as creating SQL scripts, populating data, and crafting PL/SQL procedures and Java applications.

- ERD



Our ERD show that:

1. Every student attends many courses and courses are offered to many students.
2. Every Department has many students, but the students join one department.
3. The course explained by one instructor only and instructor have many courses to explain.
4. Every Department has many instructors, but the instructor work on one department.
5. there is ternary relationship between 3 tables to give us the test result.

So, Mapping & Normalization will be:

Mapping:

1. Students(student_id(pk), f_name, l_name, DOB, email, nationality, phone_number,city, street, GPA, department_id(fk))
2. Courses(course_id(pk), course_name, description, credit_hours, instractor_id(fk),)
3. Departments(department_id(pk), department_name, department_describtion)
4. Grades(grade_code(pk), grade)
5. Instractors(instractor_id(pk), instractor_name, department_id(fk))
6. Coursrs_enrolled(student_id(fk), course_id(fk))
7. Test_result(student_id(fk), course_id(fk), date, grade_code(fk))

After Mapping & Normalization we will have 7 tables which every table have primary key or composite primary key and foreign key from other tables that will help us to build our system.

3. SQL implementation:

We will insert code in database (mysql or oracle) to crate all tables with details like we will explain bellow and we will give an example

1. students Table:

- Purpose: Stores information about university students.
- Columns & constrains:

student_id: Unique identifier for each student. constrain:(Primary Key, Auto Increment)

f_name: First name of the student constrain: (Not Null)

l_name: Last name of the student constrain: (Not Null)

Email: email of student constrain: (unique)

phone_number constrain: (unique)

DOB: date of birth of student nationality: nationality of the student

City & street: city and street which students live in. GPA: student total grade

department_id(fk) : Foreign key referencing the Departments table constrain: (Not Null)

2. courses Table:

- Purpose: Stores information about university courses.
- Columns & constrains:

course_id: Unique identifier for each course. constrain:(Primary Key, Auto Increment)

course_name: name of the scourse constrain: (Not Null)

course_describtion: information about the course

credit_hours: time of lecs eaplain the course

instractor_id(fk) : Foreign key referencing the instractors table

3. courses Table:

- Purpose: Stores information about university departments.
- Columns & constrains:

department_id: Unique identifier for each department. constrain:(Primary Key, Auto Increment)

department_name: name of the department

department_describtion: information about the department

4. instractors Table:

- Purpose: Stores information about university instractors.
- Columns & constrains:

instractor_id: Unique identifier for each instractor. constrain:(Primary Key, Auto Increment)

instractor_name: name of the instractor constrain: (Not Null)

departmentt_id(fk): Foreign key referencing the departments table constrain: (Not Null)

5. Grades Table:

- Purpose: Stores information about university grades.
- Columns & constrains:

grade_code: the symbols that describe the grade constrain:(Primary Key)

grade: the actual values to this symbol

6. Enroll Table:

- Purpose: Stores information about university students who enroll courses.
- Columns & constrains:

student_id(fk): Foreign key referencing the student table constrain: (Not Null)

course_id(fk): Foreign key referencing the coursetable constrain: (Not Null)

constrain: (the two columns are composite primary key)

7. test Table:

• Purpose: Stores information about university students who enroll courses and the result of that examination

• Columns & constrains:

student_id(fk): Foreign key referencing the student table constrain: (Not Null)

course_id(fk): Foreign key referencing the coursetable constrain: (Not Null)

date: date of the exam constrain: (Not Null)

constrain: (the three columns are composite primary key)

grade_code : grade of that exam for that sudent

example on crate students table with that details:

```
CREATE TABLE `students` (
  'student_id' int NOT NULL AUTO_INCREMENT,
  'f_name' varchar(45) NOT NULL,
  'l_name' varchar(45) NOT NULL,
  'email' varchar(45) NOT NULL,
  'DOB' varchar(30) DEFAULT NULL,
  'city' varchar(45) DEFAULT NULL,
  'street' varchar(45) DEFAULT NULL,
  'nationality' varchar(45) DEFAULT NULL,
  'GPA' double DEFAULT NULL,
  'department_id' int NOT NULL,
  'phone_number' varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin DEFAULT NULL,
  PRIMARY KEY ('student_id'),
  UNIQUE KEY 'email_UNIQUE' ('email'),
  UNIQUE KEY 'ohone_number_UNIQUE' ('phone_number'),
  KEY 'department_id_idx' ('department_id'),
  CONSTRAINT 'department_id' FOREIGN KEY ('department_id') REFERENCES 'departments' ('departments_id'));
```

4. PL\SQL

- Create functions & procedures.

I created 2 procdure which I can update details through them in students & courses tables

```
DELIMITER //

CREATE PROCEDURE update_course(
    IN p_course_id INT,
    IN p_course_name VARCHAR(255))
BEGIN
    UPDATE courses
    SET course_name = p_course_name
    WHERE course_id = p_course_id;
END;
//

DELIMITER ;

CALL update_course(1, 'it');
```

```
DELIMITER //

CREATE PROCEDURE update_students(
    IN p_student_id INT,
    IN p_f_name VARCHAR(255),
    IN p_l_name VARCHAR(255),
    IN p_email VARCHAR(255),
    IN p_city VARCHAR(255),
    IN p_phone_number VARCHAR(15)
)
BEGIN
    UPDATE students
    SET f_name = p_f_name,
        l_name = p_l_name,
        email = p_email,
        city = p_city,
        phone_number = p_phone_number
    WHERE student_id = p_student_id;
END;
//
```

- Create a trigger to calculate GPA of the sudents
  so when student have I grade in any course he enrolled in his gpa will be updated automatically

And it's the code:

```
DELIMITER //

CREATE TRIGGER calculate_gpa

AFTER INSERT ON Test

FOR EACH ROW

BEGIN

  DECLARE v_total_points DECIMAL(8, 2);

  DECLARE v_total_hours INT;

  DECLARE v_gpa DECIMAL(8, 2);
```

```
    -- Calculate total points for all courses taken by the student
    SELECT COALESCE(SUM(Grades.grade * Courses.credit_hours), 0)
    INTO v_total_points
    FROM Test
    INNER JOIN Grades ON Test.grade_code = Grades.grade_code
    INNER JOIN Courses ON Test.course_id = Courses.course_id
    WHERE Test.student_id = NEW.student_id;

    -- Calculate total hours for all courses taken by the student
    SELECT COALESCE(SUM(Courses.credit_hours), 0)
    INTO v_total_hours
    FROM Test
    INNER JOIN Courses ON Test.course_id = Courses.course_id
    WHERE Test.student_id = NEW.student_id;

    -- Calculate GPA
    IF v_total_hours > 0 THEN
        SET v_gpa = v_total_points / v_total_hours;
    ELSE
        SET v_gpa = 0;
    END IF;

    -- Update GPA in the students table for the same student
    UPDATE students
    SET GPA = v_gpa
    WHERE student_id = NEW.student_id;
END;
//
DELIMITER ;
```

5. Automation Script

- Bash script for database backup.

this script automates the process of creating backups for a MySQL database and includes basic error handling to ensure the backup process runs smoothly.

```bash
#!/bin/bash

# Database connection details
DB_HOST="localhost"
DB_PORT="3306"
DB_USER="root"
DB_PASS="yousefsaber_1999"
DB_NAME="university_case_study"

# Backup directory
BACKUP_DIR="D:/New Folder/backup"

# Date format for backup file
DATE=$(date +"%Y-%m-%d_%H-%M-%S")

# Backup file name
BACKUP_FILE="$BACKUP_DIR/$DB_NAME-$DATE.sql"

# Check if the backup directory exists, otherwise create it
if [ ! -d "$BACKUP_DIR" ]; then
    mkdir -p "$BACKUP_DIR"
    if [ $? -ne 0 ]; then
        echo "Failed to create backup directory: $BACKUP_DIR"
        exit 1
    fi
fi

# Create a backup using mysqldump
mysqldump -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" > "$BACKUP_FILE"

# Check if the backup was successful
if [ $? -eq 0 ]; then
    echo "Backup completed successfully: $BACKUP_FILE"
else
    echo "Backup failed"
fi
```

Script Functionality:

- **Database Connection Details**: The script starts by defining variables for the database connection details such as host, port, username, password, and database name.

- **Backup Directory**: Another variable BACKUP_DIR is defined to specify the directory where the backup files will be stored.

- **Date Format for Backup File**:The DATE variable is set using the date command to get the current date and time in the format specified ("%Y-%m-%d_%H-%M-%S").

- **Backup File Name:** The BACKUP_FILE variable is defined using the backup directory and the current date to create a unique name for the backup file.

- **Check Backup Directory:** The script checks if the backup directory exists. If it doesn't, the script creates it using mkdir -p. If the directory creation fails, an error message is displayed, and the script exits with an error status.
- **Backup Creation**:The script creates a backup of the MySQL database using the mysqldump command. It connects to the database using the provided connection details and writes the dump to the specified backup file.

- **Backup Success/Failure Check:**After the backup command (mysqldump) is executed, the script checks the exit status ($?) to determine if the backup was successful. If the exit status is 0, it displays a success message along with the path to the backup file. Otherwise, it displays a failure message.

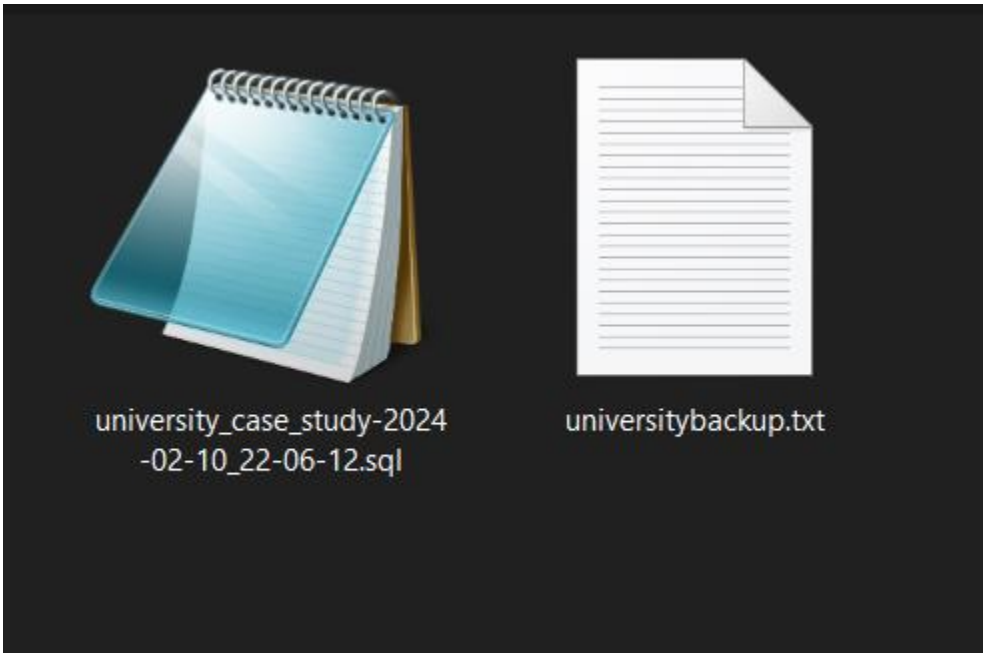
- **Considerations:**I provided some considerations for enhancing the script's security, error handling, backup rotation, and testing.

To create the back up you will need to:

1. give X permissions to the file (chmod +x 'filename')
2. run the file (./'filename')

Then the backup file will be created.

>>



- Bash script for monitoring disk space and sending alerts.
  This script can be scheduled to run periodically using a cron job or similar task scheduling mechanism to continuously monitor disk usage and send alerts when necessary.

```bash
#!/bin/bash

# Email address to receive alerts
recipient_email="yosefsaber390@gmail.com"
log_file="D:/New Folder/backup.log"
# Function to monitor disk usage
monitor_disk_usage() {
    echo "Disk Usage:"
    df -h | awk 'NR==2 {print "Used: " $5}'
    disk_usage=$(df -h | awk 'NR==2 {print $5}' | sed 's/%//')
    if [ $disk_usage -gt 70 ]; then
        echo "Disk usage is above 70%!"
        # Send email alert
        echo "Disk usage is above 70% on $(hostname)" | mail -s "Disk Alert" "$recipient_email"
        echo "Disk usage is above 70% on $(hostname)" >> "${log_file}"

    fi
}

# Main function to run all monitoring functions
main() {
    monitor_disk_usage
}
```

Script Functionality:

Variables:

- recipient_email: Specifies the email address where alerts will be sent.
- log_file: Defines the path to the log file where alerts will be logged.

Monitoring Disk Usage:

- The monitor_disk_usage function displays the current disk usage using the df -h command.
- It extracts the percentage of disk usage and checks if it's greater than 70%.
- If the disk usage exceeds 70%, it sends an email alert to the specified recipient using the mail command.
- Additionally, it appends a message to the log file indicating the disk usage alert.

Main Function:

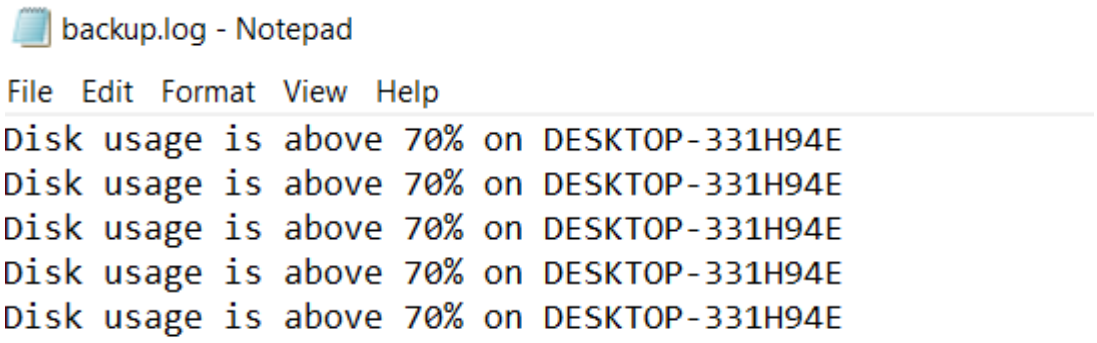
- The main function is responsible for executing the monitoring functions. Currently, it only calls monitor_disk_usage.

To create the function you will need to:

1. give X permissions to the file (chmod +x ‘filename’)
2. give X permissions to the file (chmod +w ‘filename’)
3. run the file (./’filename’)

Then the alert will be sent to the log file if the condition happened

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>



---

6. Java Application

In my project I provided the Java Source Code:

This folder contains the source code for the Java application, enabling you to use any code editor for building and running the application.

• Client: Contains classes shared across applications (DTOs).

• database: Houses the Singleton class managing the Database Connection.

• gui: Encompasses code for all front-end (GUI) classes, including the source code for the application's backend.

• Images: Stores assets (pictures) used by the application.

Code Structure:

The code is structured into various methods and functions, each responsible for specific tasks. The main sections include:

• Scenes: Manages all scenes related functionalities.

• Main Controller: Controls the main functionality and GUI switching.

• Database Access Layer: Provides database connectivity and SQL operations.

• Data Transfer: Object (DTO) For Each Entity Showed In the app.

The Application contain 7 main scenes:

- **Login scene.**
- **Students' scene.**
- **Courses scene.**
- **Departments scene.**
- **Instructor scene.**
- **Reporting (GPA) scene.**
- **Enroll scene.**
- **Test scene.**

Scenario:

1- **Run the from file ((file name). java)**

```
 */
package university_case_study;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 *
 * @author yousef saber
 */
public class University_case_study extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));

        Scene scene = new Scene(root);

        stage.setScene(scene);
        stage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
```

2- It will access the login controller and send you to the login fxml
Login controller some details:

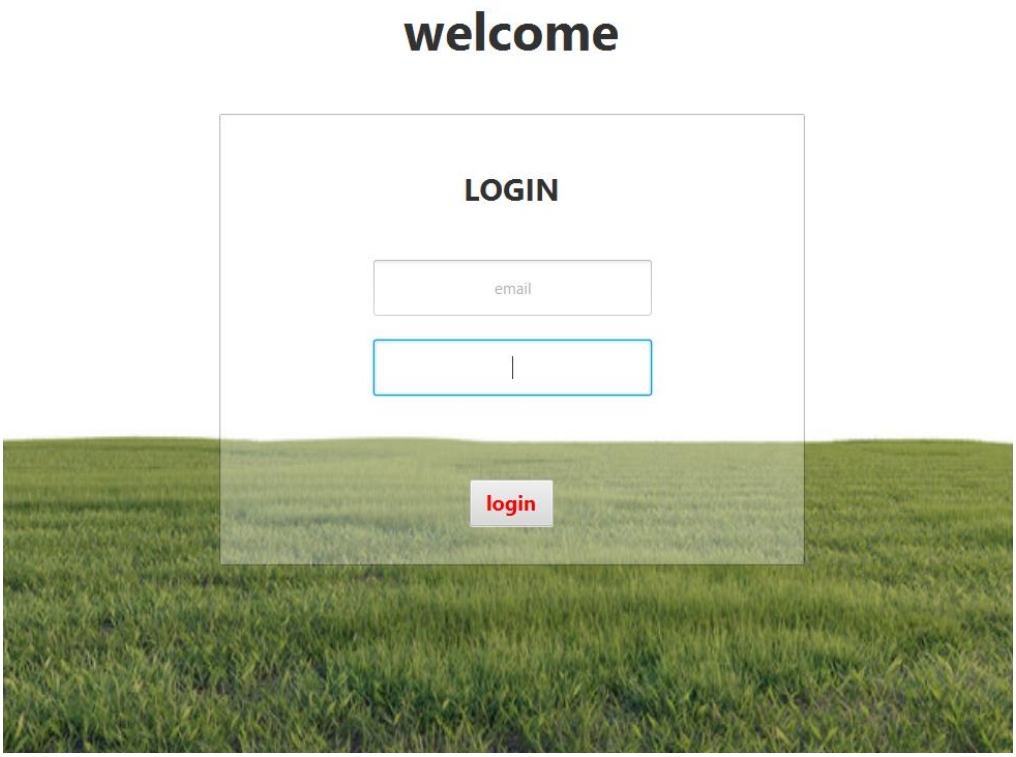
```
private Parent root;
private Stage stage;
private Scene scene;

@Override
public void initialize(URL url, ResourceBundle rb) {

}

@FXML
private void login(ActionEvent event) throws IOException {
    if ("admin".equals(email.getText().toLowerCase()) && "123456789".equals(password.getText()))
        // Load the server_pane.fxml scene for the root user
        root = FXMLLoader.load(getClass().getResource("student.fxml"));
        stage = (Stage) login_btn.getScene().getWindow();
        scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    } else {
        wronglogin.setText("Wrong Email or Password");
    }
}
```

Login scene:



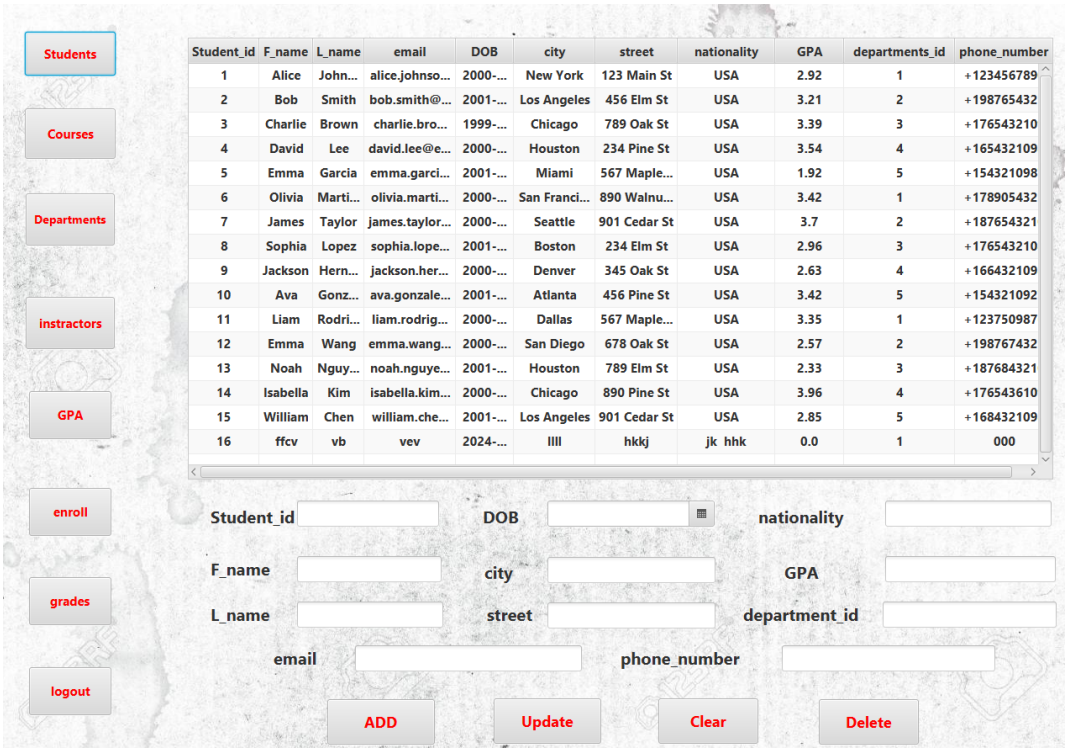
3- From login controller if the email and password is write will transport you to the student fxml scene which have the seven scene we mentioned up:

And when we click in any button from this buttons it will make us see the details about the scene that describe

4- We will describe that button
A.(students-courses-departments-instructors-enroll- grades) have the same thing

1-tables which show the data
2-text filed which we can through it (add-update-delete-clear)
The data and putted in the table.

Student:



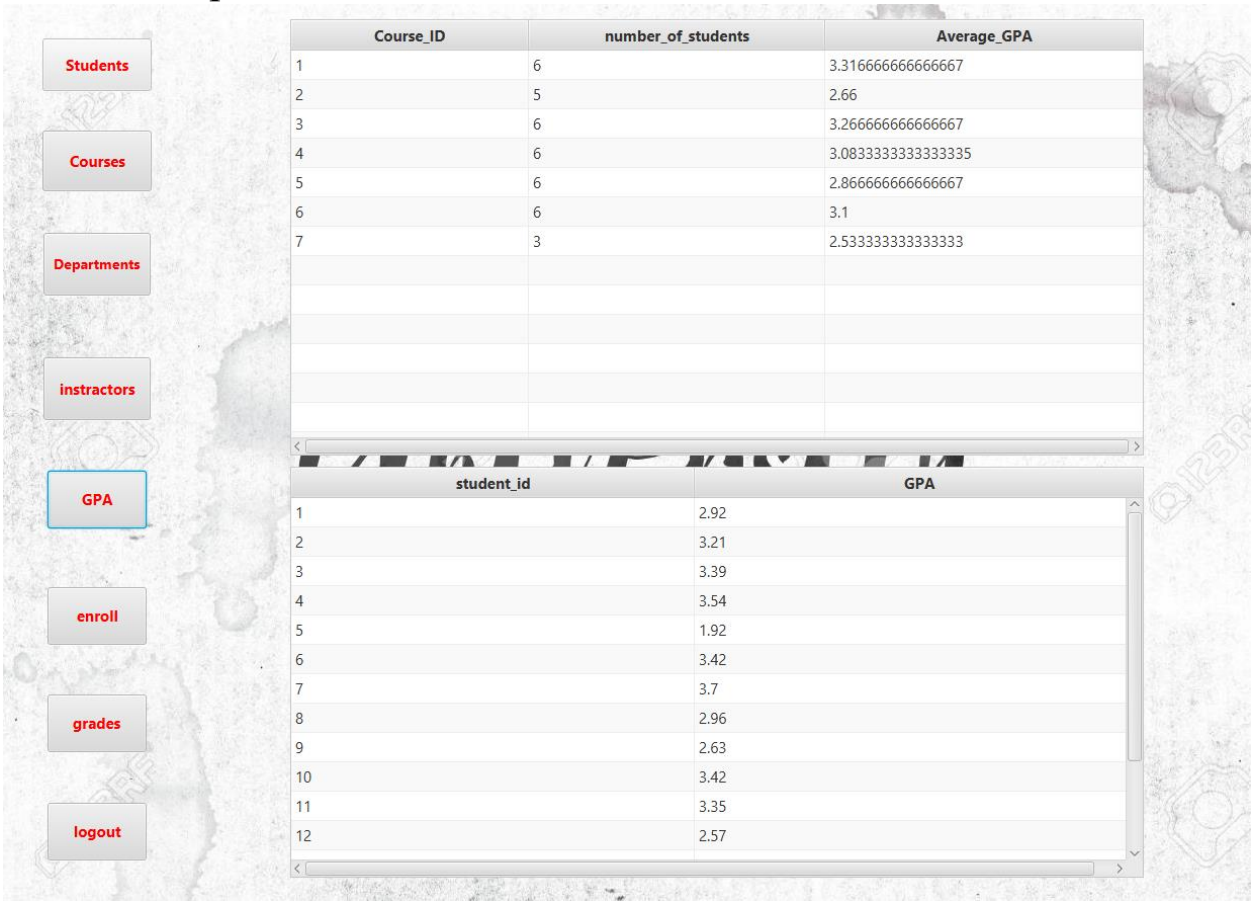
Student_id	F_name	L_name	email	DOB	city	street	nationality	GPA	departments_id	phone_number
1	Alice	John...	alice.johnso...	2000-...	New York	123 Main St	USA	2.92	1	+123456789
2	Bob	Smith	bob.smith@...	2001-...	Los Angeles	456 Elm St	USA	3.21	2	+198765432
3	Charlie	Brown	charlie.bro...	1999-...	Chicago	789 Oak St	USA	3.39	3	+176543210
4	David	Lee	david.lee@e...	2000-...	Houston	234 Pine St	USA	3.54	4	+165432109
5	Emma	Garcia	emma.garci...	2001-...	Miami	567 Maple...	USA	1.92	5	+154321098
6	Olivia	Marti...	olivia.marti...	2000-...	San Franci...	890 Walnu...	USA	3.42	1	+178905432
7	James	Taylor	james.taylor...	2000-...	Seattle	901 Cedar St	USA	3.7	2	+187654321
8	Sophia	Lopez	sophia.lope...	2001-...	Boston	234 Elm St	USA	2.96	3	+176543210
9	Jackson	Hern...	jackson.her...	2000-...	Denver	345 Oak St	USA	2.63	4	+166432109
10	Ava	Gonz...	ava.gonzale...	2001-...	Atlanta	456 Pine St	USA	3.42	5	+154321092
11	Liam	Rodri...	liam.rodrig...	2000-...	Dallas	567 Maple...	USA	3.35	1	+123750987
12	Emma	Wang	emma.wang...	2000-...	San Diego	678 Oak St	USA	2.57	2	+198767432
13	Noah	Nguy...	noah.nguye...	2001-...	Houston	789 Elm St	USA	2.33	3	+187684321
14	Isabella	Kim	isabella.kim...	2000-...	Chicago	890 Pine St	USA	3.96	4	+176543610
15	William	Chen	william.che...	2001-...	Los Angeles	901 Cedar St	USA	2.85	5	+168432109
16	ffcv	vb	vev	2024-...	llll	hkkj	jk hhk	0.0	1	000

Test:

student_id	course_id	grade_code	date
1	1	A	2023-01-15
2	3	A	2023-03-15
3	4	A	2023-09-10
4	5	A	2023-05-01
14	6	A	2023-05-01
5	1	A-	2023-03-05
6	2	A-	2023-09-05
9	6	A-	2023-03-10
11	3	A-	2023-06-20
13	4	A-	2023-03-10
6	7	A+	2023-09-05
7	3	A+	2023-05-05
10	1	A+	2023-05-12
14	5	A+	2023-09-05
3	5	B	2023-04-05
10	6	B	2023-04-05
11	2	B	2023-01-25
12	3	B	2023-07-15

And so on in the other 4 buttons

B .GPA button (have two tables) describe what is in the photo and this is the report that we do.



Course_ID	number_of_students	Average_GPA
1	6	3.316666666666667
2	5	2.66
3	6	3.266666666666667
4	6	3.0833333333333335
5	6	2.866666666666667
6	6	3.1
7	3	2.533333333333333

student_id	GPA
1	2.92
2	3.21
3	3.39
4	3.54
5	1.92
6	3.42
7	3.7
8	2.96
9	2.63
10	3.42
11	3.35
12	2.57

c. Logout button : make you back again to login scene

and all that handle in the (student controller) which have around 1500 lines of code to have all things(action , alert, exceptions) in the scene and (DTOS) that I created.