

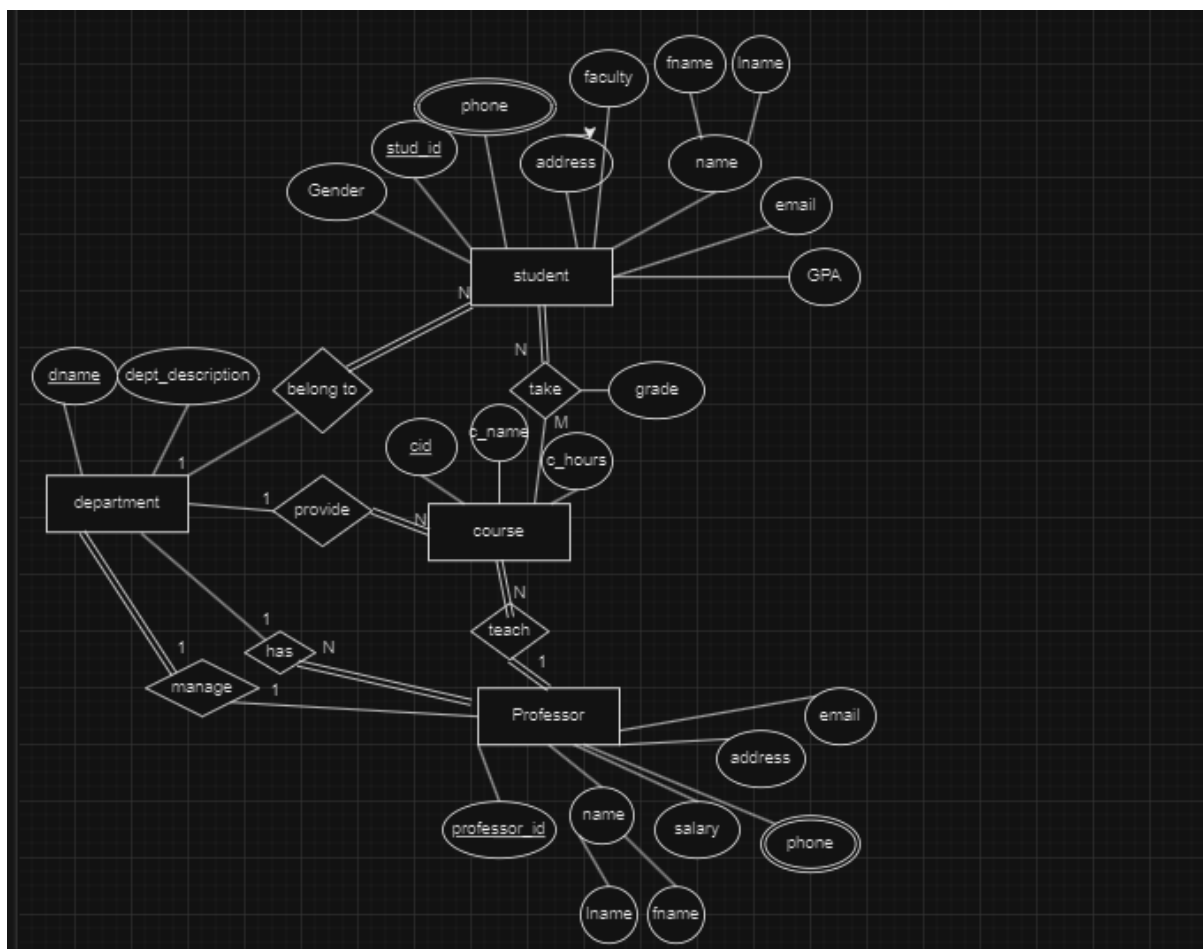
# Data Management System for a **University** Case Study

## Overview:

The objective of this Case Study is to design and implement a comprehensive data management system for a university, encompasses various aspects, including database design, SQL and PLSQL implementation, automation scripts and Java application development.

## Database Design

ERD:



- Students' data stored in table student.
- Departments data stored in table department.
- Courses data stored in table course.
- Professors' data stored in table Professor.

- Student must enrol in only 1 department and the department may has many students.
- Student must take course or courses and the course may has many students enrolled in it.
- Department must provide one or more courses and the course must be provided by a department.
- The Professor may teach one or more courses and the course must be taught by one professor.
- The department may have one or more professors and the professor must be in 1 department.
- The department must be managed by one professor and the professor may manage a department.

### Mapping:

Department (dname (PK), dept\_description, manager\_id (FK))

Student (stud\_id (PK), fname, lname, address, email, gender, faculty, dname (FK))

Stud\_pho (stud\_id (FK), phone)

Professor (professor\_id (PK), fname, lname, salary, address, email, dname (FK))

Prof\_pho (professor\_id (FK), phone)

Course (cid (PK), c\_name, c\_hours, dname (FK), professor\_id (FK))

Stud\_course (stud\_id (FK), cid (FK), grade\_)

## Database Schema:

### 1. Students Table:

- STUD\_ID (Primary Key)
- FNAME
- LNAME
- ADDRESS
- EMAIL
- GENDER
- FACULTY
- DNAME (Foreign Key: DEPARTMENT (DNAME))
- ACC\_GPA

### 2. Department table.

- DNAME(Primary Key)
- DEPT\_DESCRIPTION
- MANAGER\_ID(Foreign Key: PROFESSOR (PROFESSOR\_ID))



### 3. Professor table.

- PROFESSOR\_ID(Primary Key)
- FNAME
- LNAME
- SALARY
- EMAIL
- DNAME (Foreign Key: DEPARTMENT (DNAME))

### 4. Course table.

- CID (Primary Key)
- C\_NAME
- C\_HOURS
- DNAME (Foreign Key: DEPARTMENT (DNAME))
- YEAR
- PROFESSOR\_ID(Foreign Key: PROFESSOR (PROFESSOR\_ID))

### 5. Student\_course table.

- STUD\_ID(Primary Key)
- CID(Primary Key)
- GRADE\_

### 6. STUD\_PHO

- STUD\_ID(Primary Key)
- PHONE(Primary Key)

### 7. PROF\_PHO

- PROFESSOR\_ID(Primary Key)
- PHONE(Primary Key)

## SQL Implementation

```
ALTER TABLE UNIVERSITY.COURSE  
DROP CONSTRAINT FK_COURSE_DEPARTMENT;
```

```
ALTER TABLE UNIVERSITY.COURSE ADD (  
CONSTRAINT FK_COURSE_DEPARTMENT  
FOREIGN KEY (DNAME)  
REFERENCES UNIVERSITY.DEPARTMENT (DNAME));
```

```
-----  
ALTER TABLE UNIVERSITY.COURSE  
DROP CONSTRAINT FK_COURSE_PROFESSOR;
```

```
ALTER TABLE UNIVERSITY.COURSE ADD (  
    CONSTRAINT FK_COURSE_PROFESSOR  
    FOREIGN KEY (PROFESSOR_ID)  
    REFERENCES UNIVERSITY.PROFESSOR (PROFESSOR_ID));
```

```
-----  
ALTER TABLE UNIVERSITY.DEPARTMENT  
    DROP CONSTRAINT FK_MANAGER;
```

```
ALTER TABLE UNIVERSITY.DEPARTMENT ADD (  
    CONSTRAINT FK_MANAGER  
    FOREIGN KEY (MANAGER_ID)  
    REFERENCES UNIVERSITY.PROFESSOR (PROFESSOR_ID));
```

```
-----  
ALTER TABLE UNIVERSITY.PROF_PHO  
    DROP CONSTRAINT FK_PROF_PHO_PROFESSOR;
```

```
ALTER TABLE UNIVERSITY.PROF_PHO ADD (  
    CONSTRAINT FK_PROF_PHO_PROFESSOR  
    FOREIGN KEY (PROFESSOR_ID)  
    REFERENCES UNIVERSITY.PROFESSOR (PROFESSOR_ID));
```

```
-----  
ALTER TABLE UNIVERSITY.PROFESSOR  
    DROP CONSTRAINT FK_PROFESSOR_DEPARTMENT;
```

```
ALTER TABLE UNIVERSITY.PROFESSOR ADD (  
    CONSTRAINT FK_PROFESSOR_DEPARTMENT  
    FOREIGN KEY (DNAME)  
    REFERENCES UNIVERSITY.DEPARTMENT (DNAME));
```

```
-----  
ALTER TABLE UNIVERSITY.STUD_COURSE  
    DROP CONSTRAINT FK_STUD_COURSE_COURSE;
```

```
ALTER TABLE UNIVERSITY.STUD_COURSE ADD (  
    CONSTRAINT FK_STUD_COURSE_COURSE  
    FOREIGN KEY (CID)  
    REFERENCES UNIVERSITY.COURSE (CID));
```

```
-----  
ALTER TABLE UNIVERSITY.STUD_COURSE  
    DROP CONSTRAINT FK_STUD_COURSE_STUDENT;
```

```
ALTER TABLE UNIVERSITY.STUD_COURSE ADD (  
    CONSTRAINT FK_STUD_COURSE_STUDENT  
    FOREIGN KEY (STUD_ID)  
    REFERENCES UNIVERSITY.STUDENT (STUD_ID));
```

```
-----  
ALTER TABLE UNIVERSITY.STUD_PHO
```

```
DROP CONSTRAINT FK_STUD_PHO_STUDENT;
```

```
ALTER TABLE UNIVERSITY.STUD_PHO ADD (
  CONSTRAINT FK_STUD_PHO_STUDENT
  FOREIGN KEY (STUD_ID)
  REFERENCES UNIVERSITY.STUDENT (STUD_ID));
```

```
-----
ALTER TABLE UNIVERSITY.STUDENT
  DROP CONSTRAINT FK_STUDENT_DEPARTMENT;
```

```
ALTER TABLE UNIVERSITY.STUDENT ADD (
  CONSTRAINT FK_STUDENT_DEPARTMENT
  FOREIGN KEY (DNAME)
  REFERENCES UNIVERSITY.DEPARTMENT (DNAME));
```

```
ALTER TABLE Department
ADD CONSTRAINT fk_manager
FOREIGN KEY (manager_id) REFERENCES Professor(professor_id);
```

## Populating Sample Data

```
INSERT INTO department (dname, dept_description) VALUES ('bio', 'new department');
INSERT INTO department (dname, dept_description) VALUES ('business information
system', 'programming and commerce');
INSERT INTO department (dname, dept_description) VALUES ( 'electronics','new');
```

```
INSERT INTO course (cid, c_name, c_hours, dname) VALUES (1, 'principles of accounting',
60, 'business information system');
INSERT INTO course ( cid,c_name, c_hours, dname) VALUES (2,'chemistry', 50, 'bio');
INSERT INTO course ( cid,c_name, c_hours, dname) VALUES (3,'c#', 90, 'business
information system');
```

```
INSERT INTO student ( stud_id, fname, lname, address, email, gender, faculty,
dname)VALUES (1,'shorouk', 'mohamed', 'elsanta', 'shorouk@gmail.com', 'f', 'commerce',
'business information system');
INSERT INTO student ( stud_id, fname, lname, address, email, gender, faculty,
dname)VALUES (2, 'aya', 'ibrahim', 'elsanta', 'aya@gmail.com', 'f', 'engineering',
'electronics');
INSERT INTO student ( stud_id, fname, lname, address, email, gender, faculty,
dname)VALUES (3, 'hadeer', 'mohamed', 'kafr eldeep', 'hadeer@gmail.com', 'f', 'science',
'bio');
```

```
insert into stud_course(stud_id,CID,grade_)values(1,1,'A+');
insert into stud_course(stud_id,CID,grade_)values(2,1,'A');
insert into stud_course(stud_id,CID,grade_)values(2,2,'B');
```



```
insert into Professor( professor_id, fname ,lname ,salary,address , email ,dname
)values(1,'mohamed','handosa',5000,'mansora','mohamed@gmail.com','business information
system');
insert into Professor(professor_id, fname ,lname ,salary,address , email ,dname
)values(2,'sameh','matar',7000,'tanta','sameh@gmail.com','business information system');
insert into Professor(professor_id, fname ,lname ,salary,address , email ,dname
)values(3,'ali','ahmed',3000,'tanta','ali@gmail.com','bio');

insert into prof_pho(professor_id,phone) values(1,'01098758866');
insert into prof_pho(professor_id,phone) values(1,'01452879635');
insert into prof_pho(professor_id,phone) values(2,'01224109510');

insert into stud_pho(stud_id,phone) values(1,'010236985');
insert into stud_pho(stud_id,phone) values(1,'07788945');
insert into stud_pho(stud_id,phone) values(2,'0185263985');

select * from user_errors;
```

## PLSQL Implementation

### Procedures:

- procedure to update department table that takes input paramters(department name and new description)  
create or replace procedure update\_dept(v\_dname in varchar2,v\_new\_desc in varchar2)
- --procedure to insert into course table that takes prameters in (course name, number of hours,department name,professor id )  
**CREATE OR REPLACE procedure** UNIVERSITY.**insert\_course**(v\_c\_name in **varchar2**,v\_hours in **number**,v\_dname in **varchar2**,v\_prof **number**)
- --procedure in case of deleting a department ,it update the value of the foreign key in student table with null then it update the value of the foreign key in the professor table with null then it update the value of the foreign key in the course table with null  
**CREATE OR REPLACE PROCEDURE**  
UNIVERSITY.**DeleteDepartment**(department\_name **IN** **VARCHAR2**) **AS**



- -procedure in case of deleting a professor it updates the value of the foreign key in course table with null then it update the value of the foreign key in the professor \_phone table with null then it update the value of the foreign key in the department table with null.

**CREATE OR REPLACE PROCEDURE**

**UNIVERSITY.DeleteProfessor**(prof\_id **IN NUMBER**) **AS**

- procedure in case of deleting a student ,it deletes his record from the student\_pho table and deletes his record from the stud\_course table

**CREATE OR REPLACE PROCEDURE**

**UNIVERSITY.DeleteStudent**(student\_id **IN NUMBER**) **AS**

- **PROCEDURE** process\_updated\_students **IS**  
it updates the student table and set the value of the gpa for each student stored in the students\_to\_update array with the return value from the function (**FUNCTION** **UNIVERSITY.calculate\_acc\_gpa**(p\_stud\_id **IN NUMBER**)) and then clear the array.
- **PROCEDURE** add\_student\_to\_update(stud\_id **IN Student.stud\_id%TYPE**) **IS**  
It takes the id of the student and stores it in 'students\_to\_update' list

### Function:

- To calculate the average gpa of the course

**CREATE OR REPLACE FUNCTION**

**UNIVERSITY.CALCULATE\_AVG\_GPA**(course\_id **IN NUMBER**)

**RETURN NUMBER IS**

- To calculate the accumulated gpa of the student

**CREATE OR REPLACE FUNCTION**

**UNIVERSITY.calculate\_acc\_gpa**(p\_stud\_id **IN NUMBER**)

**RETURN NUMBER**

### Triggers:

- Before delete a course ,the trigger calls a procedure (**DeleteStudCourse**)  
Which delete from the stud\_course.

**CREATE OR REPLACE TRIGGER** **UNIVERSITY.Course\_BeforeDelete**

## BEFORE DELETE ON UNIVERSITY.COURSE FOR EACH ROW

- Before deleting a record from the department table it calls a procedure(**DeleteDepartment**)  
**CREATE OR REPLACE TRIGGER** UNIVERSITY.Department\_BeforeDelete  
**BEFORE DELETE ON UNIVERSITY.DEPARTMENT FOR EACH ROW**
- Before deleting a record from the professor table , the trigger calls a procedure(**DeleteProfessor**)  
**CREATE OR REPLACE TRIGGER** UNIVERSITY.Professor\_BeforeDelete  
**BEFORE DELETE ON UNIVERSITY.PROFESSOR FOR EACH ROW**
- Before deleting a record from the student table ,the trigger calls a Procedure (**DeleteStudent** )  
**CREATE OR REPLACE TRIGGER** UNIVERSITY.Student\_BeforeDelete  
**BEFORE DELETE ON UNIVERSITY.STUDENT FOR EACH ROW**
- In case of inserting or updating on the stud\_course table ,  
**CREATE OR REPLACE TRIGGER** UNIVERSITY.update\_GPA  
**FOR INSERT OR UPDATE ON UNIVERSITY.STUD\_COURSE**  
**COMPOUND TRIGGER**

the compound trigger

Calls a procedure

(**PROCEDURE** process\_updated\_students)

Before inserting or updating , it calls a procedure

**PROCEDURE** add\_student\_to\_update(stud\_id **IN** Student.stud\_id%**TYPE**) **IS**

## Bash Script

### 1. Bash script for database backup





### Steps:

Connect to the database:

store the username, password, and database name

defines the directory:

`direct="C:/Users/shoroukabelraouf/OneDrive/Desktop/bash":`

where the backup logs will be stored.

`d=$(date +"%Y%m%d"):`

to get the current date in the format YYYYMMDD.

`file_name="backup_${d}.dmp":`

defines the filename for the backup dump file using the current date .

`expdp ${db_owner}/${pass}@${db} DUMPFILE=${file_name}`

`DIRECTORY=dir ;`

the expdp command to perform the export operation.

`DIRECTORY=dir` specifies the directory where the dump file will be stored.

`if [ $? -eq 0 ]; then:`

To checks the status of the command (expdp).

If the status is 0 (means success), then print (echo " Make Backup

Successfully " >>) and redirect it to ( backup\_Logs.log file) located in the directory specified by the direct variable.

Else

print (echo " ERROR : Make Backup Failed " >>)

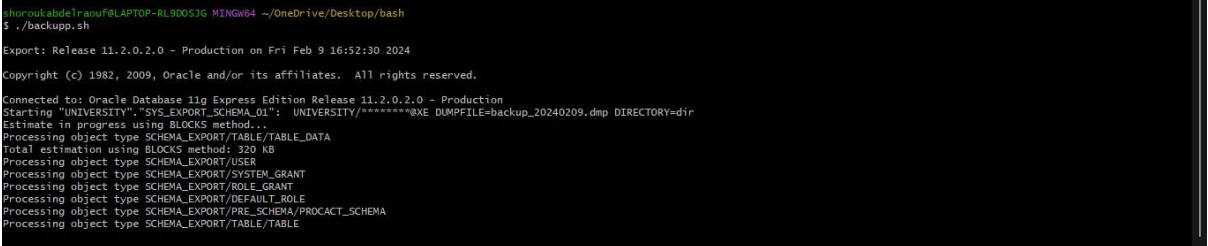


```
#!/bin/bash

db_owner=UNIVERSITY
pass=123
db=XE
direct="C:/Users/shoroukabelraouf/OneDrive/Desktop/bash"
d=$(date +%Y%m%d)
file_name="backup_${d}.dmp"

expdp ${db_owner}/${pass}@${db} DUMPFILE=${file_name} DIRECTORY=dir ;

if [ $? -eq 0 ]; then
    echo " Make Backup Successfully " >> "${direct}/backup_Logs.log"
else
    echo " ERROR : Make Backup Failed " >> "${direct}/backup_Logs.log"
fi
```



## 2. Bash script for monitoring disk space and sending alerts.

```
1  #!/bin/bash
2
3  log_file="C:/Users/shoroukabelraouf/OneDrive/Desktop/bash/show_message.log"
4  thresh=100
5  d=$(date +%Y-%m-%d)
6  used_space=$(df -h | awk 'NR==2 {print $6}' | sed 's/%//')
7
8  if [ "$used_space" -ge "$thresh" ]; then
9      echo "OOOPS!! The Disk Space limit is $thresh% | Date: ${d}" >> "${log_file}"
10 else
11     echo "The Disk Space Is Safe | Date: ${d}" >> "${log_file}"
12 fi
```

log\_file="C:/Users/shoroukabelraouf/OneDrive/Desktop/bash/show\_message.log":

the path to the log file where messages will appear.

thresh=100:

If the disk space usage exceeds this threshold, a warning message will appear in the file.

used\_space=\$(df -h | awk 'NR==2 {print \$6}' | sed 's/%//'):

to calculate the percentage of disk space used

IF

To check if the \$used\_space is greater than or equal to the thresh,

If it's greater then a warning message will be appended in the log file ,if it's not a message will appear also in the log file "the disk is safe".

```
000PS!! The Disk Space limit is 100% | Date: 2024-02-11  
The Disk Space Is Safe | Date: 2024-02-11  
|
```

### 3. **Schedule a script to check for anomalies and send notifications.**

The script defines thresholds for CPU usage, memory consumption, available disk space, network traffic, and database connection failures. These thresholds are expressed as percentages or specific values.

#### **Function Definitions:**

- **check\_cpu:** Checks the CPU usage using the wmic command on Windows systems. If the CPU usage exceeds the predefined threshold, it logs the anomaly.
- **check\_memory:** Checks memory consumption by calculating the percentage of used memory. If memory usage exceeds the threshold, it logs the anomaly.
- **check\_disk:** Checks available disk space using the df command. If disk usage exceeds the threshold, it logs the anomaly.
- **check\_network:** Checks network traffic using the netstat command. If network traffic exceeds the threshold, it logs the anomaly.
- **check\_db\_connections:** Checks database connection failures by counting occurrences of a specific message in a log file. If the count exceeds the threshold, it logs the anomaly.

## Java Application Development

The System allows the admin to perform operations such as adding, updating, deleting, and getting information about students, courses, grades, professors, and departments. Also enables you to know the average GPA for each course and enrolled students.

### Structure:

- Screens package:
  - Include the images used in the application.
- University package:
  1. Include: the controllers and FXML files used in the application.
    - DAO file (deals with the database whether get the data or perform operations on it.)

### Application scenes:

#### 1. Student scene

##### **(Add button):**

It enables you to register a student in the system.

##### **(Update button):**

To modify the student's information by changing the data in the fields.

##### **(Delete)**

to remove the student's record from the system by entering his ID.

##### **(Get info button)**

To get information about the student by entering his ID.

The full name of the student ,his faculty and the department.

**Also, there are some validations :**



- The email should be in the right format.
- In case of registering or updating existing student data ,all fields required to be filled to complete the operation.
- In case of registering a student with an existed ID ,a message will be displayed informing the user “this student is already enrolled in our university .”
- In case the user selected a department that doesn’t exist ,a message will appear "there is no department with this name.".
- After completing any operation , a message will appear and tell you whether the operation is done successfully or not .
- After pressing any button the fields will be automatically be cleared.

## **2. Course scene:**

for managing course information:

### **(Add button):**

It enables you to register a new course in the system.

### **(Update button):**

To modify the course information .

### **(Delete)**

to remove a course from the system by entering his ID.

### **(course info button)**

Display a table with information about all available courses in the system.  
Course ID, course name, the department in which the course is taught.

### **(departments info button)**

Display a table the available departments (name) in the university.



**(professor info button)**

To get information about our professors in the system .

Professor ID , full name.

**Also, there are some validations :**

- In case of registering course data ,all fields required to be filled to complete the operation.
- In case of registering a course with an existed code ,a message will be displayed informing the user "the course code is already exists."
- In case of updating or deleting a course , the course code and name is necessary to be filled"
- After completing any operation , a message will appear and tell you whether the operation is done successfully or not .
- After pressing any button the fields will be automatically cleared.

### **3. Department scene:**

for managing departments information:

**(Add button):**

It enables you to add a new department in the system.

**(Update button):**

To modify the department information .



**(Delete)**

to remove a department from the system .

**(our departments info button)**

Display a table with the available departments in the university.

Department name, description and its manager ID.

**(managers info button)**

Display a table with the available managers at the system , to assign a manager to the department.(Manager ID , full name )

**Also, there are some validations :**

- In case of adding a new department ,all fields required to be filled to complete the operation.
- In case of the manager with the given ID doesn't exists , a message will appear "Please check our available managers. Manager with ID "not found."
- In case of updating or deleting department data , the department name is required and a message will pop up "name required to be filled".
- In case of inserting a department with the given name that already exists , a message will pop up "we have this department at our university."
- After completing any operation , a message will appear and tell you whether the operation is done successfully or not .
- After pressing any button the fields will be automatically cleared.



#### **4. Professor scene:**

for managing professors information:

**(Add button):**

It enables you to register a new professor in the system.

**(Update button):**

To modify the to modify a professor data.

**(Delete)**

to remove a professor from the system by entering his ID .

**(our departments button)**

Display a table with the available departments(Department name) in the university.

**(our professors button)**

Display a table with the entire information of the professors.

**Also, there are some validations :**

- The email should be in the right format.
- In case of registering or updating existing professor data ,all fields required to be filled to complete the operation.
- In case of registering a professor with an existed ID ,a message will be displayed informing the user "this professor is already teaches at our university ."
- In case the user entered a department that doesn't exist ,a message will appear "there is no department with this name."
- After completing any operation , a message will appear and tell you whether the operation is done successfully or not .
- After pressing any button the fields will be automatically be cleared.





## 5. Report scene:

Display information about the GPA of the students and for course .

### (Students GPA button):

By pressing it ,you will see all the registered students in the system (their ID, full name and their accumulated GPA).

### (Course GPA button):

After inserting the course code pressing the button , a table will appear with (course code, course name ,enrolled students ,their grade in the course and average course GPA )

### (insert button):

It allows you to assign course to a student .

**Also, there are some validations :**

- The user must select a grade from the list , and if he doesn't ,a message will appear "Please choose a grade from the list."
- If the user entered a code for a course that doesn't exist , a pop up will appear" this course is not available"

## 6. Contact scene:

Allows you to add or deleted the contact information of the professors and students.

### (Get info button ):

When you press it , a table the students that have phone numbers will be displayed. And also the professors that have phone numbers.

**Also, there are some validations :**

- If the record already exists, a message will appear "the same ID and phone already exists."
- To check whether this student has a phone number or not in case of deleting .

