**School Management System**

**DATA BASE**

****

Submitted by:

Muhammad Yasir

Submitted to:

Syed Shayan Ali Shah

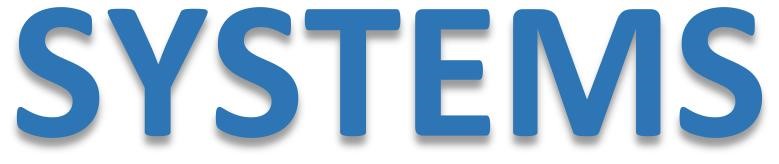
Semester: 4th BSDS(Group-B)

Session 2023 –2027

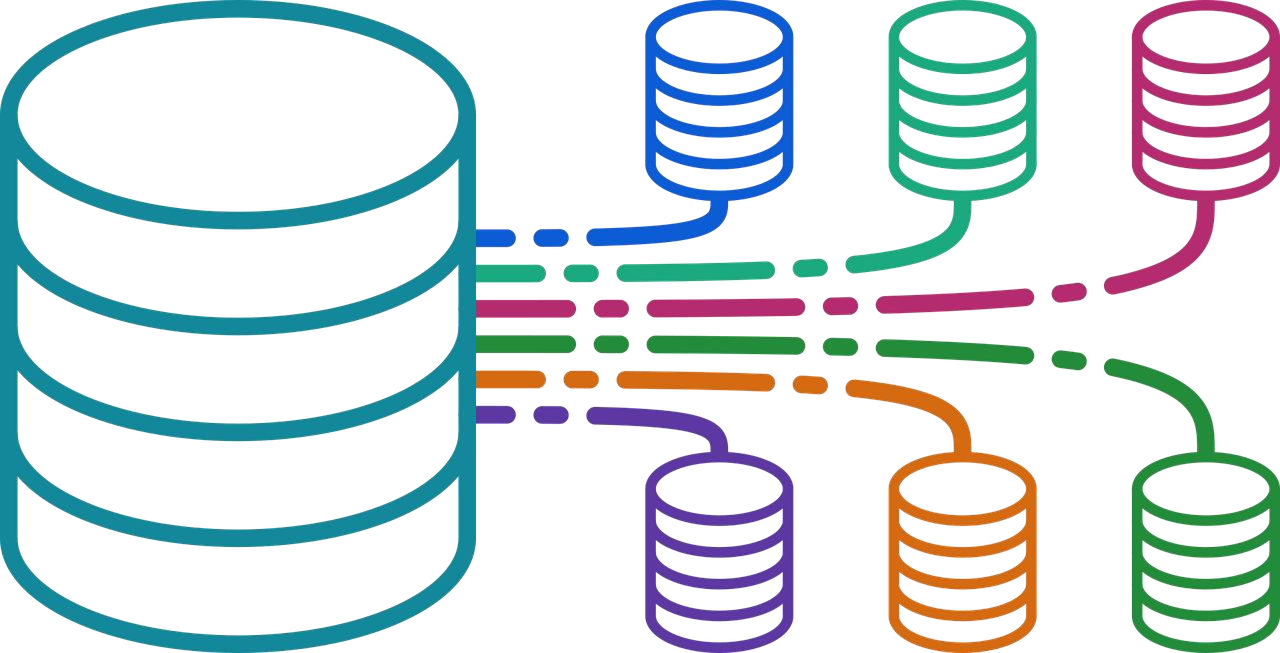
**INSTITUTE OF MANAGEMENT SCIENCES**

**HAYATABAD, PESHAWAR (2025)**

School Management System



**PROJECT REPORT**



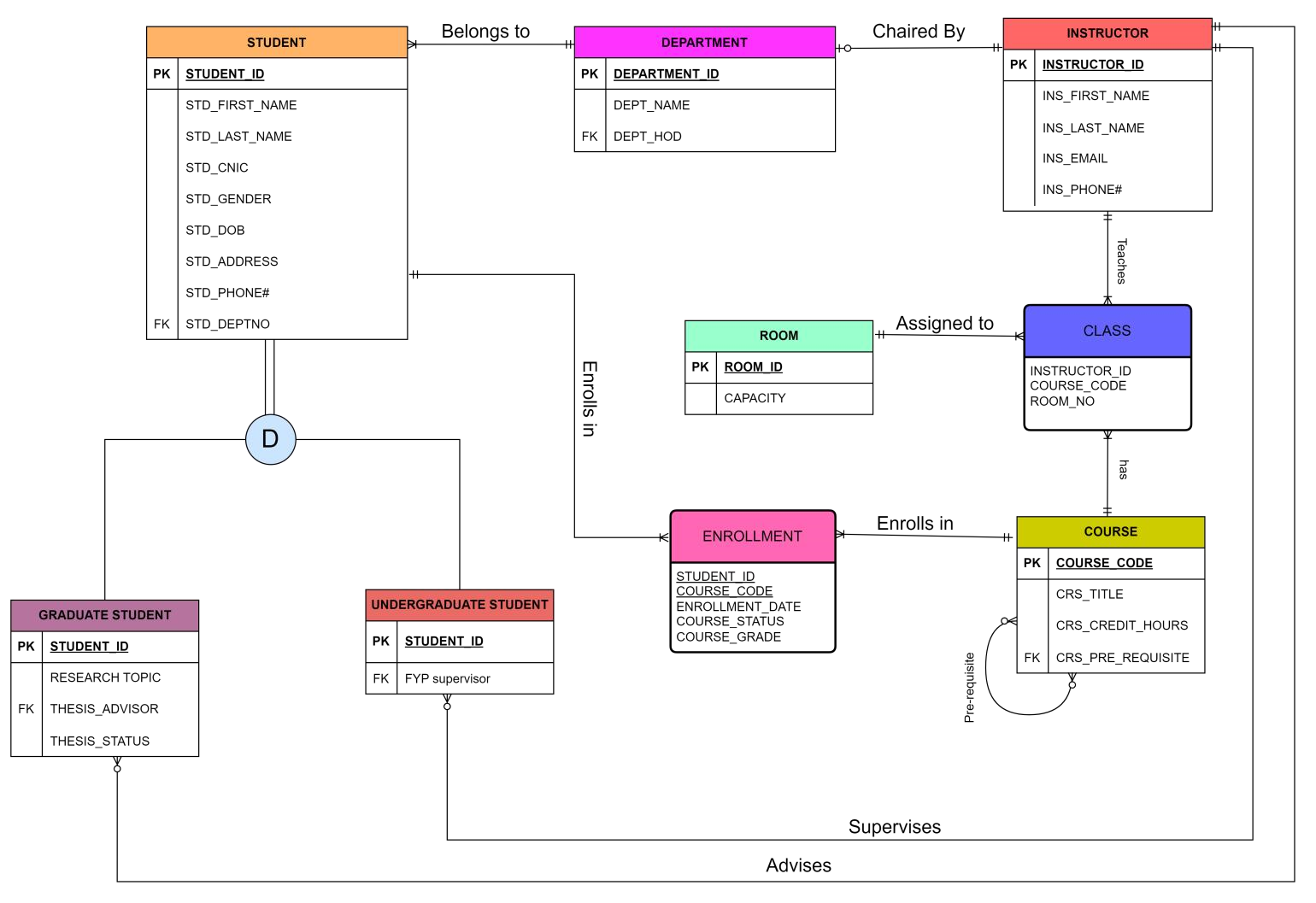
**1. Introduction to the working of the system:**

A database model has been developed at **IMSciences** to efficiently manage student records, including information about students, departments, instructors, and courses. This system streamlines operations, supports queries, and ensures future scalability.

1. **Problems in the existing system:**

A database model has been developed at **IMSciences** to efficiently manage academic records, encompassing information about **students, departments, instructors, and courses**. The system is designed to streamline administrative operations, facilitate complex queries, maintain data consistency, and support future scalability for institutional growth.

1. **Entity - Relation Diagram:**

c

# 4. ERD Transformation

## Student

A strong entity

Student (Student\_ID, First\_Name, Last\_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo)

## Graduate Student

Student’s Subtype

Graduate Student (Student\_ID, Thesis\_Advisor\_ID, Thesis\_Status)

## Undergraduate Student

Student’s Subtype

Undergraduate Student (Student\_ID, FYP\_Advisor\_ID)

## Department

Strong Entity

Department (DeptNo , Dept\_Name , Dept\_HOD)

## Instructor

A regular entity

Instructor (Instructor\_ID, First\_Name , Last\_Name, Email, Phone#)

## Room

A regular entity

Room (RoomNo , Capacity)

## Course

Recursive relation

Course (Course\_Code, Course\_Title, Credit\_Hours, Pre\_Requisite\_ID)

## Class

An associative entity

Class (Instructor\_ID, Course\_Code, RoomNo)

## Enrollment

An associative entity

Enrollment (Student\_ID, Course\_Code, Enrollment\_Date, Course\_Grade, Status)

**NOTE:**

\_\_\_\_\_\_\_\_\_\_\_ Indicates primary key

Indicates foreign key

# 5. CONSTRUCTION OF RELATIONAL SCHEMA

* TOP DOWN APPROACH
* BOTTOM UP APPROACH

## TOPDOWNAPPROACH

**Identified entities:**

* Student
  + Graduate students
  + Undergraduate students
* Instructor
* Department
* Course
* Room
* Class  Enrollment

**Relations:**

**Student** (Student\_ID, First\_Name, Last\_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo)

**Graduate Student** (Student\_ID, Thesis\_Advisor\_ID, Thesis\_Status)

**Undergraduate Student** (Student\_ID, FYP\_Advisor\_ID)

**Department** (DeptNo , Dept\_Name , Dept\_HOD)

**Instructor** (Instructor\_ID, First\_Name , Last\_Name, Email, Phone#)

**Room** (RoomNo , Capacity)

**Course** (Course\_Code, Course\_Title, Credit\_Hours, Pre\_Requisite\_ID)

**Class** (Instructor\_ID, Course\_Code, RoomNo)

**Enrollment** (Student\_ID, Course\_Code, Enrollment\_Date, Course\_Grade, Status)

# Normalization

**Student** (Student\_ID, First\_Name, Last\_Name, CNIC, Gender, DOB, Address, Phone#, DeptNo) *1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Graduate Student** (Student\_ID, Thesis\_Advisor\_ID, Thesis\_Status)

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Undergraduate Student** (Student\_ID, FYP\_Advisor\_ID)

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency.

*3NF* : Already in 3NF as there is no Transitive Dependency.

**Department** (DeptNo , Dept\_Name , Dept\_HOD) *1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Instructor** (Instructor\_ID, First\_Name , Last\_Name, Email, Phone#)

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Room** (RoomNo , Capacity)

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Course** (Course\_Code, Course\_Title, Credit\_Hours, Pre\_Requisite\_ID)

*1NF :* A single course can have Multiple pre-requisites. So, there is a repeating group.

Shift the pre-requisite of course to another relation.

**After removing repeating group:**

**Course** (Course\_Code, Course\_Title, Credit\_Hours)

**Pre\_Requisite\_Course** (Course\_Code, Pre\_Requisite\_ID)

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Class** (Instructor\_ID, Course\_Code, RoomNo) Normalization:

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency. *3NF* : Already in 3NF as there is no Transitive Dependency.

**Enrollment** (Student\_ID, Course\_Code, Enrollment\_Date, Course\_Grade, Status)

*1NF :* Already in 1NF as there is no repeating group.

*2NF :* Already in 2NF as there is no Partial Functional Dependency.

*3NF* : Already in 3NF as there is no Transitive Dependency.

**Result:**

Now all relations are in 3NF, there's no further normalization needed. This indicates that the relations are structured well, with minimal redundancy and efficient data organization.

## BOTTOMUPAPPROACH

**Bulky relation comprising all attributes:**

**Relation → {**Student\_ID, Std\_First\_Name, Std\_Last\_Name, Std\_CNIC, Std\_Gender, Std\_DOB,

Std\_Address, Std\_Phone#, Std\_DeptNo, DeptNo, Dept\_Name ,Dept\_HOD, Thesis\_Advisor\_ID,

Thesis\_Status, FYP\_Advisor\_ID, Instructor\_ID, Ins\_First\_Name , Ins\_Last\_Name, Ins\_Email,

Ins\_Phone#, RoomNo , Capacity, Course\_Code, Course\_Title, Credit\_Hours, Pre\_Requisite\_ID,

Enrollment\_ID, Enrollment\_Date, Course\_Grade, Status**}**

The relation consists of all the attributes in our present ERD. Now, we will construct a sub-relation from above and perform normalization.

### Relation

**Student(Student\_ID,STD\_First\_name,Last\_name,Std\_CNIC,Gender,DOB,Address,Phone#,Deptno)**

1NF : No repeating Group

2NF : No Partial Functional Dependency

3NF : No Transitive Dependency

***Subtypes:***

**Graduate Student (Student\_ID, Thesis\_Advisor\_ID, Thesis\_Status)**

*1*NF *:* There is no repeating group

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency. *3*NF : Already in 3NF as there is no Transitive Dependency.

**Undergraduate Student (Student\_ID, FYP\_Advisor\_ID)**

*1*NF *:* Already in 1NF as there is no repeating group.

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency. *3*NF : Already in 3NF as there is no Transitive Dependency.

**DEPARTMENT (Deptno,Dept\_Name,Dept\_HOD)**

*1*NF *:* Already in 1NF as there are no multivalued attributes.

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency. *3*NF : Already in 3NF as there is no Transitive Dependency.

**Instructor (Instructor\_ID, First\_Name , Last\_Name, Email, Phone#)** *1*NF *:* Already in 1NF as no duplicating values.

*2*NF *:* Already in 2NF as atomic primary key.

*3*NF : Already in 3NF as no non-key attributes determine other attributes.

**Room (RoomNo , Capacity)**

*1*NF *:* No duplicate data as there is only one Primary key *2*NF *:* Already in 2NF as there is no Partial Functional Dependency.

*3*NF : Already in 3NF as there is no Transitive Dependency.

**Course (Course\_Code, Course\_Title, Credit\_Hours, Pre\_Requisite\_ID)** There may be multiple pre-requisite for one course.

So,It does not hold 1NF requirements

***After 1NF***

Course(Course\_Code, Course\_Title, Credit\_Hours)

Course\_Pre\_Req (Course\_Code, Pre\_Requisite\_ID)

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency. *3*NF : Already in 3NF as there is no Transitive Dependency.

**Class (Instructor\_ID, Course\_Code, RoomNo)**

*1*NF *:* Already in 1NF as there is no repeating group.

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency. *3*NF : Already in 3NF as there is no Transitive Dependency.

**Enrollment (Student\_ID, Course\_Code, Enrollment\_Date, Course\_Grade, Status)** *1*NF *:* Already in 1NF as there is no repeating group.

*2*NF *:* Already in 2NF as there is no Partial Functional Dependency.

*3*NF : Already in 3NF as there is no Transitive Dependency.

**Connectivity Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity** | **Relationship** | **Connectivity** | **Entity** |
| Student | Is a | 1:1 | Graduate Student |
| Student | Is a | 1:1 | Under Graduate Student |
| Department | Has | 1:M | Student |
| Instructor | Has | 1:1 | department |
| Instructor | Has | 1:M | Graduate Student |
| Instructor | Has | 1:M | Under graduate student |
| Student | Has | 1:M | Course |
| Course | Has | 1:M | Course |
| Course | Has | 1:M | Student |
| Room | Has | 1:M | Class |
| Instructor | Has | 1:M | Class |

**6. Description of relations**

**Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| STUDENT\_ID | CHAR | 10 | PRIMARY KEY |
| FIRST\_NAME | VARCHAR2 | 20 | NOT NULL |
| LAST\_NAME | VARCHAR2 | 20 | NOT NULL |
| CNIC | CHAR | 13 | UNIQUE |
| GENDER | CHAR | 1 | M OR F |
| DOB | DATE |  |  |
| ADDRESS | VARCHAR2 | 50 |  |
| PHONE# | CHAR | 13 | NOT NULL , UNIQUE |
| DEPTNO | NUMBER |  | REFERENCE TO DEPARTMENT |

**Graduate Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| STUDENT\_ID | CHAR | 10 | PRIMARY KEY, REFERENCE TO STUDENT |
| THESIS\_ADVISOR | CHAR | 10 | REFERENCE TO INSTRUCTOR |
| THESIS\_STATUS | VARCHAR2 | 20 | ‘COMPLETE’ OR ‘IN PROGRESS’ |

**Under Graduate Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| STUDENT\_ID | CHAR | 10 | PRIMARY KEY, REFERENCE TO STUDENT |
| FYP\_ADVISOR | CHAR | 10 | REFERENCE TO INSTRUCTOR |

**Department**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| DEPTNO | NUMBER |  | PRIMARY KEY |
| DEPTNAME | VARCHAR2 | 50 | NOT NULL |
| HOD | CHAR | 10 | REFERENCE TO INSTRUCTOR |

**Instructor**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| INSTRUCTOR\_ID | CHAR | 10 | PRIMARY KEY |
| FIRST\_NAME | VARCHAR2 | 20 | NOT NULL |
| LAST\_NAME | VARCHAR2 | 20 | NOT NULL |
| EMAIL | VARCHAR2 | 30 | UNIQUE |
| PHONE# | CHAR | 13 | NOT NULL , UNIQUE |

**Room**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| ROOMNO | NUMBER |  | PRIMARY KEY |
| CAPACITY | NUMBER |  | POSITIVE INTEGER |

**Course**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| COURSE\_CODE | VARCHAR2 | 10 | PRIMARY KEY |
| COURSE\_TITLE | VARCHAR2 | 50 | NOT NULL |
| CREDIT\_HOURS | NUMBER |  | BETWEEN 0.5 AND 3 |

**Pre\_Requisite\_Course**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| COURSE\_CODE | VARCHAR2 | 10 | PRIMARY KEY, REFERENCE TO COURSE |
| PRE\_REQ | VARCHAR2 | 10 | PRIMARY KEY, REFERENCE TO COURSE |

**Class**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| INSTRUCTOR\_ID | CHAR | 10 | PRIMARY KEY, REFERENCE TO INSTRUCTOR |
| COURSE\_CODE | VARCHAR2 | 10 | PRIMARY KEY, REFERENCE TO COURSE |
| ROOM\_NO | NUMBER |  | REFERENCE TO ROOM |

**Enrollment**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Type** | **Size** | **Constraints** |
| STUDENT\_ID | CHAR | 10 | PRIMARY KEY, REFERENCE TO STUDENT |
| COURSE\_CODE | VARCHAR2 | 10 | PRIMARY KEY,REFERENCE TO COURSE |
| ENROLLMENT\_DATE | DATE |  |  |
| GRADE | CHAR | 1 | ‘A’,’B’,’C’,’D’ OR F |
| STATUS | CHAR | 4 | ‘PASS’ OR ‘FAIL’ |

## 7. CREATE TABLE statements for all relations

**Student**

CREATE TABLE Student

(

STUDENT\_ID CHAR(10) CONSTRAINT pk\_student PRIMARY KEY,

FIRST\_NAME VARCHAR(20) CONSTRAINT nn\_first\_name NOT NULL,

LAST\_NAME VARCHAR(20) CONSTRAINT nn\_last\_name NOT NULL,

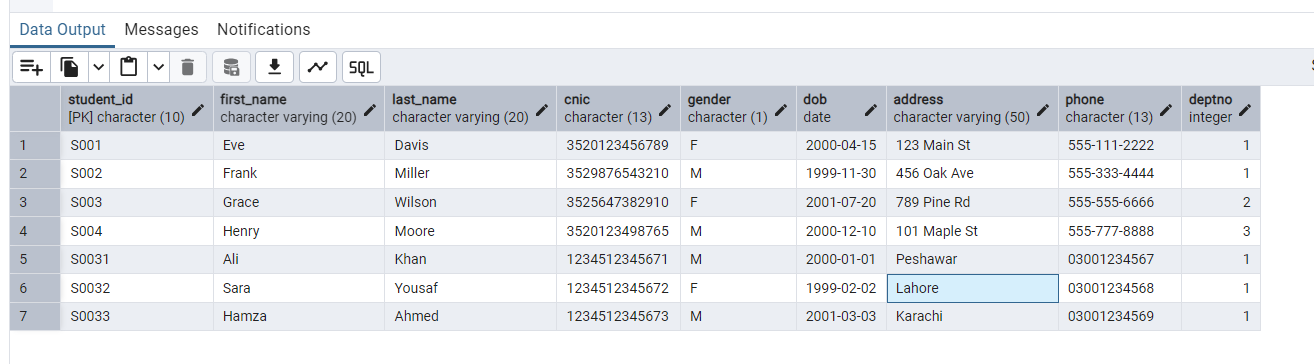
CNIC CHAR(13) CONSTRAINT uq\_cnic UNIQUE,

GENDER CHAR(1) CONSTRAINT ck\_gender CHECK (GENDER IN ('M', 'F')),

DOB DATE,

ADDRESS VARCHAR(50),

PHONE CHAR(13) CONSTRAINT nn\_phone NOT NULL,

DEPTNO NUMBER CONSTRAINT fk\_deptno REFERENCES Department(DEPTNO));

**Graduate Student**

CREATE TABLE Graduate\_Student

(

STUDENT\_ID CHAR(10) CONSTRAINT pk\_graduate\_student PRIMARY KEY REFERENCES

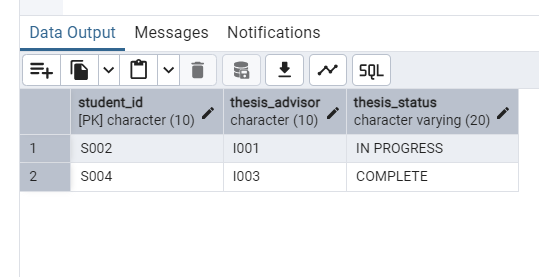
Student(STUDENT\_ID),

THESIS\_ADVISOR CHAR(10) CONSTRAINT fk\_thesis\_advisor REFERENCES

Instructor(INSTRUCTOR\_ID),

THESIS\_STATUS VARCHAR(20) CONSTRAINT ck\_thesis\_status CHECK

(THESIS\_STATUS IN ('COMPLETE', 'IN PROGRESS')) );



**Under Graduate Student**

CREATE TABLE Undergraduate\_Student

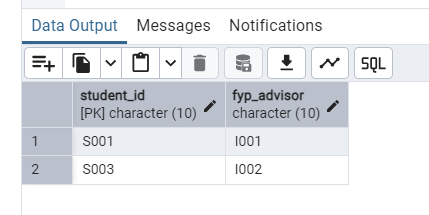
(

STUDENT\_ID CHAR(10) CONSTRAINT pk\_undergraduate\_student PRIMARY KEY REFERENCES Student(STUDENT\_ID),

FYP\_ADVISOR CHAR(10) CONSTRAINT fk\_fyp\_advisor REFERENCES

Instructor(INSTRUCTOR\_ID)

);



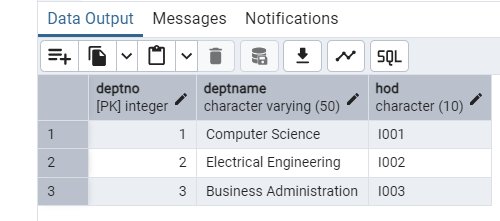
**Department**

CREATE TABLE department (

deptno INTEGER CONSTRAINT pk\_department PRIMARY KEY,

deptname VARCHAR(50) NOT NULL CONSTRAINT nn\_deptname,

hod CHAR(10) CONSTRAINT fk\_hod REFERENCES instructor(instructor\_id)

);

**Instructor**

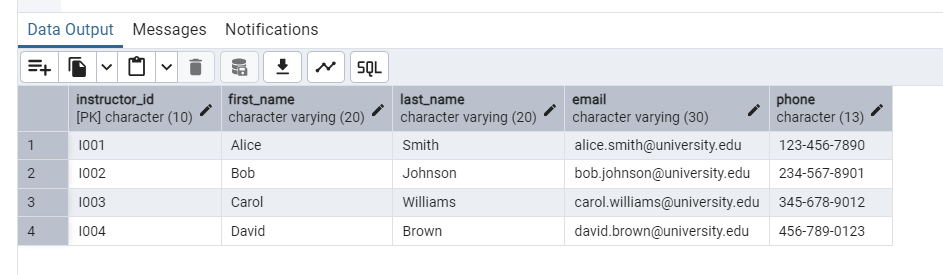
CREATE TABLE instructor (

instructor\_id CHAR(10) CONSTRAINT pk\_instructor PRIMARY KEY,

first\_name VARCHAR(20) NOT NULL CONSTRAINT nn\_instructor\_first\_name,

last\_name VARCHAR(20) NOT NULL CONSTRAINT nn\_instructor\_last\_name,

email VARCHAR(30) CONSTRAINT uq\_instructor\_email UNIQUE,

 phone CHAR(13) NOT NULL CONSTRAINT nn\_instructor\_phone);

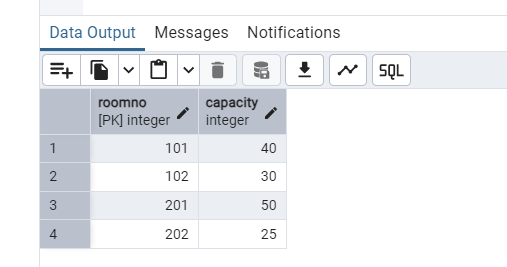
**Room**

CREATE TABLE room (

roomno INTEGER CONSTRAINT pk\_room PRIMARY KEY,

capacity INTEGER CONSTRAINT ck\_capacity CHECK (capacity > 0)

);



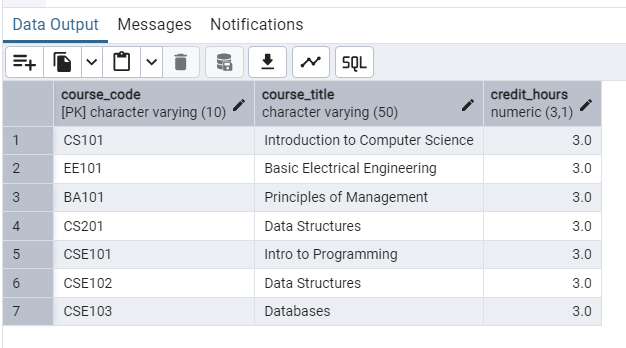
**Course**

CREATE TABLE Course

(

COURSE\_CODE VARCHAR2(10) CONSTRAINT pk\_course PRIMARY KEY,

COURSE\_TITLE VARCHAR2(50) CONSTRAINT nn\_course\_title NOT NULL,

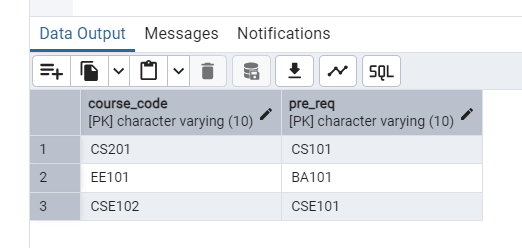
****CREDIT\_HOURS NUMBER CONSTRAINT ck\_credit\_hours CHECK (CREDIT\_HOURS BETWEEN 0.5 AND 3) );

**Pre\_Requisite\_Course**

CREATE TABLE pre\_requisite\_course (

course\_code VARCHAR(10) CONSTRAINT fk\_course\_code REFERENCES course(course\_code),

pre\_req VARCHAR(10) CONSTRAINT fk\_pre\_req REFERENCES course(course\_code),

CONSTRAINT pk\_pre\_requisite\_course PRIMARY KEY (course\_code, pre\_req));

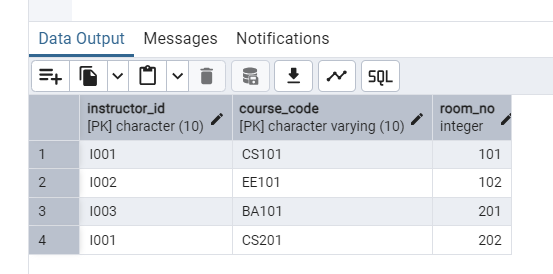
**Class**

CREATE TABLE class (

instructor\_id CHAR(10) CONSTRAINT fk\_class\_instructor REFERENCES instructor(instructor\_id),

course\_code VARCHAR(10) CONSTRAINT fk\_class\_course REFERENCES course(course\_code),

room\_no INTEGER CONSTRAINT fk\_class\_room REFERENCES room(roomno)

CONSTRAINT pk\_class PRIMARY KEY (instructor\_id, course\_code));

**Enrollment**

CREATE TABLE enrollment (

student\_id CHAR(10) CONSTRAINT fk\_enrollment\_student REFERENCES student(student\_id),

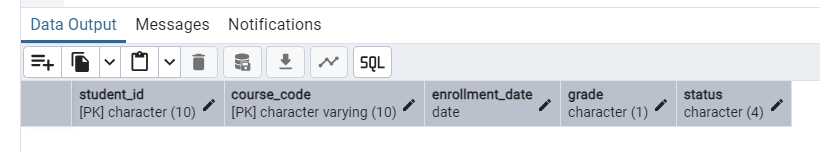
course\_code VARCHAR(10) CONSTRAINT fk\_enrollment\_course REFERENCES course(course\_code),

enrollment\_date DATE,

grade CHAR(1) CONSTRAINT ck\_grade CHECK (grade IN ('A', 'B', 'C', 'D', 'F')),

status CHAR(4) CONSTRAINT ck\_status CHECK (status IN ('PASS', 'FAIL')),

CONSTRAINT pk\_enrollment PRIMARY KEY (student\_id, course\_code)

);

**8. Views of relational schema**

**1. Student Details:**

CREATE OR REPLACE VIEW student\_details AS

SELECT

S.STUDENT\_ID,

S.FIRST\_NAME || ' ' || S.LAST\_NAME AS FULL\_NAME,

S.CNIC,

S.GENDER,

S.DOB,

S.ADDRESS,

S.PHONE,

D.DEPTNAME AS DEPARTMENT\_NAME,

H.FIRST\_NAME || ' ' || H.LAST\_NAME AS DEPARTMENT\_HEAD

FROM

Student S

JOIN Department D ON S.DEPTNO = D.DEPTNO JOIN Instructor H ON D.HOD = H.INSTRUCTOR\_ID; select \* from student\_details

### 2. Enrollment Details

CREATE VIEW Enrollment\_Details AS

SELECT

e.STUDENT\_ID,

s.FIRST\_NAME,

s.LAST\_NAME,

e.COURSE\_CODE,

c.COURSE\_TITLE,

e.ENROLLMENT\_DATE,

e.GRADE,

e.STATUS

FROM Enrollment e

JOIN Student s ON e.STUDENT\_ID = s.STUDENT\_ID JOIN Course c ON e.COURSE\_CODE = c.COURSE\_CODE select \* from enrollment\_details

**3. Graduate Students Details:**

CREATE OR REPLACE VIEW Graduate\_Student\_Details AS

SELECT gs.STUDENT\_ID,

CONCAT(s.FIRST\_NAME,CONCAT(' ',s.LAST\_NAME)) "STUDENT NAME", s.CNIC,

s.GENDER,

s.DOB,

s.ADDRESS,

s.PHONE,

CONCAT(i.FIRST\_NAME,CONCAT(' ',i.LAST\_NAME)) "ÄDVISOR NAME", gs.THESIS\_STATUS

FROM Graduate\_Student gs

JOIN Student s ON gs.STUDENT\_ID = s.STUDENT\_ID

JOIN Instructor i ON gs.THESIS\_ADVISOR = i.INSTRUCTOR\_ID;

**4. Pre-Requisite Courses Details:**

CREATE OR REPLACE VIEW Pre\_Requisite\_Course\_Details AS

SELECT

c.COURSE\_TITLE AS COURSE\_TITLE, prc2.COURSE\_TITLE AS PRE\_REQ\_TITLE

FROM Pre\_Requisite\_Course prc

JOIN Course c ON prc.COURSE\_CODE = c.COURSE\_CODE

JOIN Course prc2 ON prc.PRE\_REQ = prc2.COURSE\_CODE;

**9. Relational Data Model showing associations**

Student

Student\_ID

FirstName

LastName

CNIC

Gender

DOB

Address

PHONE#

DeptNo

Graduate

Student

Student\_ID

Thesis\_Advisor\_ID

Thesis\_Status

Under

Graduate

Student

Student\_ID

FYP\_Advisor

Instructor

Instructor\_ID

FirstName

LastName

Email

Phone#

Department

DeptNo

Dept\_Name

HOD

Course

Course\_Code

Course\_Title

Credit\_Hours

Pre-Requisite

Course

Course\_Code

Pre\_Req

Room

RoomNo

Capacity

Class

Instructor\_ID

Course\_Code

RoomNo

Enrollment

Student\_ID

Course\_Code

Enrollment\_Date

Grade

Status

## 10. Five Common Reports

### 1. QUERY TO RETRIEVE STUDENTS WHOSE THESIS IS COMPLETE

### SELECT

### S.FIRST\_NAME || ' ' || S.LAST\_NAME AS NAME,

### I.FIRST\_NAME || ' ' || I.LAST\_NAME AS ADVISOR,

### G.THESIS\_STATUS

### FROM STUDENT S

### JOIN GRADUATE\_STUDENT G ON G.STUDENT\_ID = S.STUDENT\_ID

### JOIN INSTRUCTOR I ON I.INSTRUCTOR\_ID = G.THESIS\_ADVISOR

### WHERE G.THESIS\_STATUS = 'COMPLETE';

### 2. QUERY TO RETRIEVE ROOMS THAT ARE FREE YET

### SELECT R.ROOMNO, R.CAPACITY

### FROM ROOM R

### LEFT JOIN CLASS C ON R.ROOMNO = C.ROOM\_NO

### WHERE C.ROOM\_NO IS NULL;

### 3. QUERY TO RETRIEVE DEPARTMENT ID ,NAME ,HEAD AND NUMBER OF STUDENTS IN THAT DEPARTMENT

### SELECT

### d.DEPTNO,

### d.DEPTNAME,

### i.FIRST\_NAME || ' ' || i.LAST\_NAME AS HEAD\_NAME,

### COUNT(s.STUDENT\_ID) AS STUDENT\_COUNT

### FROM Department d

### JOIN Instructor i ON d.HOD = i.INSTRUCTOR\_ID

### LEFT JOIN Student s ON d.DEPTNO = s.DEPTNO

### GROUP BY d.DEPTNO, d.DEPTNAME, i.FIRST\_NAME, i.LAST\_NAME;

### 4. LIST INSTRUCTORS WHO ARE SUPERVISING AN FYP

### SELECT DISTINCT

### I.INSTRUCTOR\_ID,

### I.FIRST\_NAME || ' ' || I.LAST\_NAME AS NAME

### FROM Instructor I

### JOIN Undergraduate\_Student U ON I.INSTRUCTOR\_ID = U.FYP\_ADVISOR;

### 5. STUDENT WHO ARE FAILED IN ANY SUBJECT

## SELECT

## S.STUDENT\_ID,

## S.FIRST\_NAME || ' ' || S.LAST\_NAME AS NAME,

## C.COURSE\_TITLE,

## I.FIRST\_NAME || ' ' || I.LAST\_NAME AS TEACHER\_NAME

## FROM STUDENT S

## JOIN ENROLLMENT E ON S.STUDENT\_ID = E.STUDENT\_ID

## JOIN COURSE C ON C.COURSE\_CODE = E.COURSE\_CODE

## JOIN CLASS CL ON E.COURSE\_CODE = CL.COURSE\_CODE

## JOIN INSTRUCTOR I ON I.INSTRUCTOR\_ID = CL.INSTRUCTOR\_ID

## WHERE E.GRADE = 'F';

## 11. Procedures

**1. PROCEDURE TO SHOW THE RESULT OF SPECIFIC STUDENT** CREATE OR REPLACE PROCEDURE Student\_Result(p\_STUDENT\_ID CHAR)

LANGUAGE plpgsql

AS $$

DECLARE

rec RECORD;

no\_enrollments BOOLEAN := TRUE;

BEGIN

FOR rec IN

SELECT e.COURSE\_CODE, c.COURSE\_TITLE, e.GRADE, e.STATUS

FROM Enrollment e

JOIN Course c ON e.COURSE\_CODE = c.COURSE\_CODE

WHERE e.STUDENT\_ID = p\_STUDENT\_ID

LOOP

no\_enrollments := FALSE;

RAISE NOTICE 'Course Code: %', rec.COURSE\_CODE;

RAISE NOTICE 'Course Title: %', rec.COURSE\_TITLE;

RAISE NOTICE 'Grade: %', rec.GRADE;

RAISE NOTICE 'Status: %', rec.STATUS;

RAISE NOTICE '-----------------------------';

END LOOP;

IF no\_enrollments THEN

RAISE NOTICE 'No enrollments found for the given Student ID.';

END IF;

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'An error occurred: %', SQLERRM;

END;

$$;2. Procedure to Insert a student

CREATE OR REPLACE PROCEDURE InsertStudent( p\_STUDENT\_ID CHAR, p\_FIRST\_NAME VARCHAR2, p\_LAST\_NAME VARCHAR2, p\_CNIC CHAR, p\_GENDER CHAR, p\_DOB DATE, p\_ADDRESS VARCHAR2, p\_PHONE CHAR,

p\_DEPTNO NUMBER

) IS

BEGIN

INSERT INTO Student (STUDENT\_ID,FIRST\_NAME,LAST\_NAME,CNIC,

GENDER,DOB,ADDRESS,PHONE,DEPTNO) VALUES

(p\_STUDENT\_ID, p\_FIRST\_NAME,p\_LAST\_NAME,p\_CNIC,p\_GENDER, p\_DOB, p\_ADDRESS,p\_PHONE,p\_DEPTNO);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found.');

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Too many Rows');

END;

/

### 3. PROCEDURE TO INSERT A RECORD IN ENROLLMENT TABLE AND

**CALCULATING GRADE BY MARKS** CREATE OR REPLACE PROCEDURE InsertGrade(

p\_student\_id CHAR,

p\_course\_id VARCHAR,

p\_marks NUMERIC

)

LANGUAGE plpgsql

AS $$

DECLARE

v\_grade CHAR(1);

v\_status CHAR(4);

BEGIN

IF p\_marks > 85 THEN

v\_grade := 'A';

v\_status := 'PASS';

ELSIF p\_marks > 75 THEN

v\_grade := 'B';

v\_status := 'PASS';

ELSIF p\_marks > 65 THEN

v\_grade := 'C';

v\_status := 'PASS';

ELSIF p\_marks > 50 THEN

v\_grade := 'D';

v\_status := 'PASS';

ELSE

v\_grade := 'F';

v\_status := 'FAIL';

END IF;

INSERT INTO Enrollment (STUDENT\_ID, COURSE\_CODE, ENROLLMENT\_DATE, GRADE, STATUS)

VALUES (p\_student\_id, p\_course\_id, CURRENT\_DATE, v\_grade, v\_status);

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Error occurred: %', SQLERRM;

END;

$$;

12. Functions

**1. FUNCTION TO GET AVERAGE GRADE POINT IN A COURSE** CREATE OR REPLACE FUNCTION GetAVGGradePoint(p\_course\_code VARCHAR)

RETURNS NUMERIC

LANGUAGE plpgsql

AS $$

DECLARE

v\_average\_grade NUMERIC;

BEGIN

SELECT AVG(

CASE GRADE

WHEN 'A' THEN 4.0

WHEN 'B' THEN 3.0

WHEN 'C' THEN 2.0

WHEN 'D' THEN 1.0

WHEN 'F' THEN 0.0

ELSE NULL

END

)

INTO v\_average\_grade

FROM Enrollment

WHERE COURSE\_CODE = p\_course\_code;

RETURN v\_average\_grade;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

$$;

### 2. FUNCTION TO GET NUMBER OF COURSES BEING TAUGT BY A SPECIFIC INSTRUCTOR

### CREATE OR REPLACE FUNCTION GetClassCountForTeacher(p\_instructor\_id CHAR)

### RETURNS INTEGER

### LANGUAGE plpgsql

### AS $$

### DECLARE

### v\_class\_count INTEGER;

### BEGIN

### SELECT COUNT(\*)

### INTO v\_class\_count

### FROM Class

### WHERE INSTRUCTOR\_ID = p\_instructor\_id;

### RETURN v\_class\_count;

### EXCEPTION

### WHEN OTHERS THEN

### RETURN -1;

### END;

### $$;

### 3. FUNCTION TO COUNT THE NUMBER OF GRADUATE STUDENTS IN A DEPARTMENT

## CREATE OR REPLACE FUNCTION GetTotalGraduateStudentsInDept(p\_deptno INTEGER)

## RETURNS INTEGER

## LANGUAGE plpgsql

## AS $$

## DECLARE

## v\_student\_count INTEGER;

## BEGIN

## SELECT COUNT(\*)

## INTO v\_student\_count

## FROM Graduate\_Student gs

## JOIN Student s ON gs.STUDENT\_ID = s.STUDENT\_ID

## WHERE s.DEPTNO = p\_deptno;

## RETURN v\_student\_count;

## EXCEPTION

## WHEN OTHERS THEN

## RETURN -1;

## END;

## $$;

## 13. Triggers

### Trigger that no more than 20 students are enrolled in a course

CREATE OR REPLACE FUNCTION trg\_CheckCourseEnrollmentLimit()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

DECLARE

v\_enrolled\_students INTEGER;

BEGIN

SELECT COUNT(\*)

INTO v\_enrolled\_students

FROM Enrollment

WHERE COURSE\_CODE = NEW.COURSE\_CODE;

IF v\_enrolled\_students >= 20 THEN

RAISE EXCEPTION 'Enrollment limit exceeded for this course. No more than 20 students can be enrolled.';

END IF;

RETURN NEW;

END;

$$;

CREATE TRIGGER trg\_CheckCourseEnrollmentLimit

BEFORE INSERT ON Enrollment

FOR EACH ROW

EXECUTE FUNCTION trg\_CheckCourseEnrollmentLimit();;

/

### Trigger to check if student has passed pre req

CREATE OR REPLACE FUNCTION trg\_EnforcePrereqCompletion()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

DECLARE

v\_prereq\_completed INTEGER;

BEGIN

SELECT COUNT(\*)

INTO v\_prereq\_completed

FROM Enrollment e

JOIN Pre\_Requisite\_Course p ON e.COURSE\_CODE = p.PRE\_REQ

WHERE e.STUDENT\_ID = NEW.STUDENT\_ID

AND p.COURSE\_CODE = NEW.COURSE\_CODE

AND e.STATUS = 'PASS';

IF v\_prereq\_completed = 0 THEN

RAISE EXCEPTION 'Prerequisite courses not completed.';

END IF;

RETURN NEW;

END;

$$;

CREATE TRIGGER trg\_EnforcePrereqCompletion

BEFORE INSERT ON Enrollment

FOR EACH ROW

EXECUTE FUNCTION trg\_EnforcePrereqCompletion();