Data Base Final Project

----------First the Code form Vs Code

```sql
/**
 * schema.sql: Database Table Definitions
 * ----------------------------------------------------------
 * This file creates the physical structure of the University database.
 * It uses Primary Keys for unique identification and Foreign Keys
 * to maintain relationships between tables.
 */

-- 1. DEPARTMENTS
-- LOGIC: The high-level organization (e.g., Engineering, IT).
-- Every student must belong to a department.
CREATE TABLE Department (
    Dept_ID NUMBER(3) PRIMARY KEY, -- Unique ID for the department
    Dept_Name VARCHAR2(50) NOT NULL
);

-- 2. STUDENTS
-- LOGIC: Stores personal and academic data.
-- Note: Student_ID is the Primary Key used by our Sequence.
CREATE TABLE Student (
    Student_ID NUMBER(8) PRIMARY KEY,
    Name VARCHAR2(50) NOT NULL,
    GPA NUMBER(3,2) DEFAULT 0.0,
    Total_Credits NUMBER(3) DEFAULT 0, -- This is what our TRIGGER updates
    Dept_ID NUMBER(3) REFERENCES Department(Dept_ID) -- Relationship to Department
);

-- 3. COURSES
-- LOGIC: The catalog of subjects.
-- Includes a CHECK constraint to ensure credits are between 1 and 6.
CREATE TABLE Course (
    Course_Code VARCHAR2(10) PRIMARY KEY,
    Title VARCHAR2(100),
    Credits NUMBER(1) CHECK (Credits BETWEEN 1 AND 6)
);

-- 4. SECTIONS
-- LOGIC: Specific class times/locations for a Course.
-- Capacity is used by our PL/SQL Package to prevent over-enrollment.
CREATE TABLE Section (
    Section_ID NUMBER(6) PRIMARY KEY,
    Course_Code VARCHAR2(10) REFERENCES Course(Course_Code),
    Capacity NUMBER(3)
);

-- 5. ENROLLMENT
-- LOGIC: This is a 'Join Table'.
-- It connects Students to Sections and tracks their grades.
CREATE TABLE Enrollment (
    Student_ID NUMBER(8) REFERENCES Student(Student_ID),
    Section_ID NUMBER(6) REFERENCES Section(Section_ID),
    Grade VARCHAR2(2),
    -- Composite Primary Key: Ensures a student cannot register for the
    -- same section twice.
    PRIMARY KEY (Student_ID, Section_ID)
);

-- VERIFICATION:
-- Runs a quick check to ensure all tables were created successfully.
SELECT table_name
FROM user_tables
WHERE table_name IN ('STUDENT', 'COURSE', 'SECTION', 'ENROLLMENT', 'DEPARTMENT');
```

```sql
/**
 * seed.sql: Initial Data Population
 * ---------------------------------------------------------
 * LOGIC: This file fills the tables with "Sample Data" so we can
 * test the system immediately. It follows the order of dependencies:
 * Dept -> Course -> Section -> Student.
 */

-- 1. Insert Department
-- Note: Student creation will fail if we don't create this ID (101) first.
INSERT INTO Department (Dept_ID, Dept_Name) VALUES (101, 'Computer Science');

-- 2. Insert Sample Course
-- This defines the 'Database Programming' class worth 3 credits.
INSERT INTO Course (Course_Code, Title, Credits) VALUES ('CS101', 'Database Programming', 3);

-- 3. Insert Sample Section
-- This creates 'Seat 5001' for Course CS101.
-- The 'Capacity' of 30 is what our PL/SQL Package will check during registration.
INSERT INTO Section (Section_ID, Course_Code, Capacity) VALUES (5001, 'CS101', 30);

-- 4. Insert Sample Student
-- This is our manual test student. Future students (like Ibrahim) will
-- be created via the React form using the Sequence.
INSERT INTO Student (Student_ID, Name, GPA, Dept_ID)
VALUES (20240001, 'Ahmad Ali', 3.5, 101);

-- 5. COMMIT
-- LOGIC: In Oracle, changes are 'temporary' until you COMMIT.
-- This saves everything to the permanent storage.
COMMIT;
```

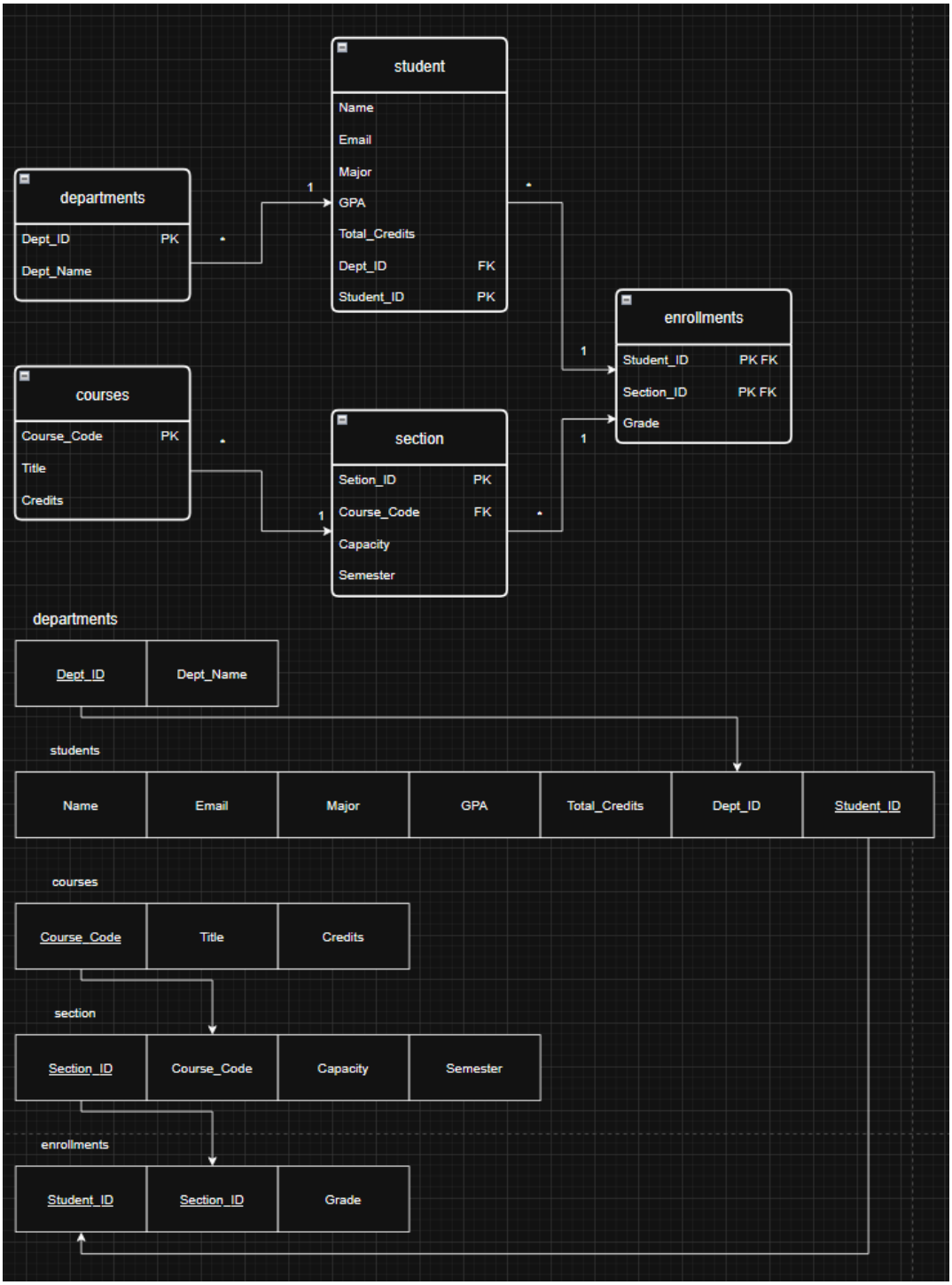
```sql
/**
 * logic.sql: PL/SQL Named Programs and Schema Updates
 * ----------------------------------------------------------
 * This file contains the automation (Triggers), unique ID generators (Sequences),
 * and business rules (Packages) for the University system.
 */

-- ==========================================================
-- 1. TRIGGER: AUTOMATIC CREDIT UPDATER
-- LOGIC: This "watches" the Enrollment table. As soon as a student
-- registers for a class, it calculates the course credits and adds
-- them to the Student's 'Total_Credits' column automatically.
-- ==========================================================
CREATE OR REPLACE TRIGGER trg_update_credits
AFTER INSERT ON Enrollment
FOR EACH ROW
DECLARE
    v_credits NUMBER;
BEGIN
    -- We join Course and Section to find how many credits this specific section is worth
    SELECT Credits INTO v_credits
    FROM Course C JOIN Section S ON C.Course_Code = S.Course_Code
    WHERE S.Section_ID = :NEW.Section_ID;

    -- Update the student record without needing manual math in the frontend
    UPDATE Student SET Total_Credits = Total_Credits + v_credits
    WHERE Student_ID = :NEW.Student_ID;
END;
/

-- ==========================================================
-- 2. SEQUENCE: STUDENT ID GENERATOR
-- LOGIC: This acts like a "number ticket machine." It ensures
-- every student gets a unique ID (Primary Key) starting from 20240002.
-- ==========================================================
CREATE SEQUENCE student_id_seq
START WITH 20240002
INCREMENT BY 1
NOCACHE;
/
```

```
30
31  -- ============================================================
32  -- 2. SEQUENCE: STUDENT ID GENERATOR
33  -- LOGIC: This acts like a "number ticket machine." It ensures
34  -- every student gets a unique ID (Primary Key) starting from 20240002.
35  -- ============================================================
36  CREATE SEQUENCE student_id_seq
37  START WITH 20240002
38  INCREMENT BY 1
39  NOCACHE;
40  /
41
42  -- ============================================================
43  -- 3. PACKAGE SPECIFICATION: THE PUBLIC INTERFACE
44  -- LOGIC: This is the "Menu." It tells the Node.js backend what
45  -- procedures are available to be called.
46  -- ============================================================
47  CREATE OR REPLACE PACKAGE Uni_Mgmt_Pkg AS
48      -- Creates a new account and generates the ID
49      PROCEDURE Add_Student(p_name IN VARCHAR2, p_email IN VARCHAR2, p_major IN VARCHAR2, p_dept IN NUMBER);
50
51      -- Registers a student after checking if the section is full
52      PROCEDURE Register_Student(p_sid IN NUMBER, p_secid IN NUMBER);
53
54      -- Deletes a student and their related enrollment records
55      PROCEDURE Delete_Student(p_sid IN NUMBER);
56
57      -- Checks GPA and returns 'Honor Roll' or 'Regular' status
58      FUNCTION Get_Status(p_sid IN NUMBER) RETURN VARCHAR2;
59  END Uni_Mgmt_Pkg;
60  /
```

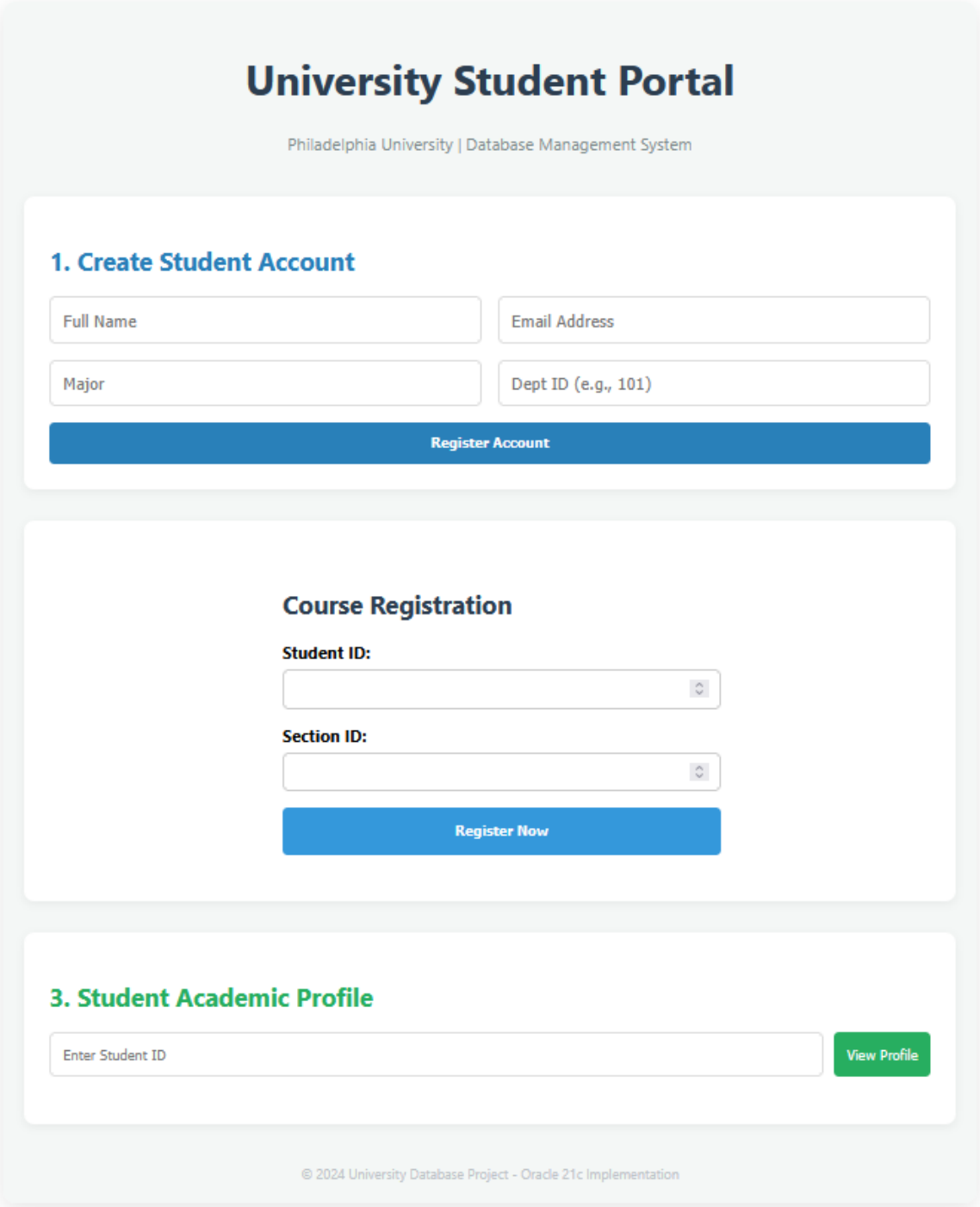
```
-- ============================================================
-- 4. PACKAGE BODY: THE PRIVATE LOGIC
-- LOGIC: This contains the actual code and business rules.
-- ============================================================
CREATE OR REPLACE PACKAGE BODY Uni_Mgmt_Pkg AS

    -- Implementation of Account Creation
    PROCEDURE Add_Student(p_name IN VARCHAR2, p_email IN VARCHAR2, p_major IN VARCHAR2, p_dept IN NUMBER) IS
    BEGIN
        -- .NEXTVAL pulls the next unique number from our ticket machine (Sequence)
        INSERT INTO Student (Student_ID, Name, Email, Major, Dept_ID)
        VALUES (student_id_seq.NEXTVAL, p_name, p_email, p_major, p_dept);
    END Add_Student;

    -- Implementation of Course Registration with Capacity Check
    PROCEDURE Register_Student(p_sid IN NUMBER, p_secid IN NUMBER) IS
        v_cur NUMBER;
        v_cap NUMBER;
    BEGIN
        -- 1. Find the limit and current number of students in the section
        SELECT Capacity INTO v_cap FROM Section WHERE Section_ID = p_secid;
        SELECT COUNT(*) INTO v_cur FROM Enrollment WHERE Section_ID = p_secid;

        -- 2. If there is space, register; if not, throw an error back to React
        IF v_cur < v_cap THEN
            INSERT INTO Enrollment (Student_ID, Section_ID) VALUES (p_sid, p_secid);
        ELSE
            -- User-defined exception: This stops the process if the class is full
            RAISE_APPLICATION_ERROR(-20001, 'Section Full');
        END IF;
    END Register_Student;

    -- Implementation of Safe Deletion
    PROCEDURE Delete_Student(p_sid IN NUMBER) IS
    BEGIN
        -- Because of Foreign Key constraints, we must delete child records (Enrollment)
        -- before deleting the parent record (Student).
        DELETE FROM Enrollment WHERE Student_ID = p_sid;
        DELETE FROM Student WHERE Student_ID = p_sid;

        COMMIT; -- Finalize the changes permanently
    END Delete_Student;

    -- Implementation of Status Check (Predefined Exception Handling)
    FUNCTION Get_Status(p_sid IN NUMBER) RETURN VARCHAR2 IS
        v_gpa NUMBER;
    BEGIN
        SELECT GPA INTO v_gpa FROM Student WHERE Student_ID = p_sid;
        IF v_gpa >= 3.5 THEN
            RETURN 'Honor Roll';
        ELSE
            RETURN 'Regular';
        END IF;
    EXCEPTION
        -- If the ID typed in React doesn't exist, this prevents a crash
        WHEN NO_DATA_FOUND THEN
            RETURN 'Student Not Found';
    END Get_Status;

END Uni_Mgmt_Pkg;
/

-- ============================================================
-- 5. SCHEMA UPDATES (MAINTENANCE)
-- LOGIC: These commands modify the tables to add columns needed
-- for the React Frontend views (like Email, Major, and Semester).
-- ============================================================

-- Add Semester column to track when a course is offered
ALTER TABLE Section ADD (Semester VARCHAR2(20) DEFAULT 'Spring 2026');

-- Add Email and Major to Student for the Profile page
ALTER TABLE Student ADD (Email VARCHAR2(100), Major VARCHAR2(50));

-- Update existing data so the 'JOIN' queries don't show empty fields
UPDATE Section SET Semester = 'Spring 2026' WHERE Semester IS NULL;
COMMIT;
```

The code was made in Vs code with oracle Database extension , and oracle database 21c XE, also to add Some backend to handle the API’s using Node.js, for the frontend we use react.

Schema \ ER


student
Name
Email
Major
GPA
Total_Credits
Dept_ID FK
Student_ID PK
departments
Dept_ID PK
Dept_Name
enrollments
Student_ID PK FK
Section_ID PK FK
Grade
courses
Course_Code PK
Title
Credits
section
Setion_ID PK
Course_Code FK
Capacity
Semester
1
*
departments
Dept_ID
Dept_Name
students
Name
Email
Major
GPA
Total_Credits
Dept_ID
Student_ID
courses
Course_Code
Title
Credits
section
Section_ID
Course_Code
Capacity
Semester
enrollments
Student_ID
Section_ID
Grade

User Interface



Test Statements :

	STUDENT_ID	NAME	GPA	TOTAL_CREDITS	DEPT_ID	EMAIL	MAJOR
1	20240001	Ahmad Ali	3.5	3	101	(null)	(null)
2	20240002	ibrahim kanan	0	3	101	ibrahim@email.com	computer science
3	20240003	Charlie Davis	3.9	24	102	(null)	(null)
4	20240004	mohammed	0	0	101	kanan	animation
5	20240005	ibrahim	0	3	101	mahmoud	CS

	Student Name	Course Name	SECTION_ID	GRADE
1	Ahmad Ali	Database Programming	5001	(null)
2	ibrahim kanan	Database Programming	5001	(null)
3	ibrahim	Database Programming	5001	(null)
4	Charlie Davis	Thermodynamics	6001	A

	DEPT_NAME	Number of Students
1	Computer Science	4
2	Mechanical Engineering	1