

Database Design Specifications

1) General Scenario

The system's operations are quite practical: students can designate new faculty and instructors, teach or update courses, define semester course offerings, register for these courses, and track their progress. Once instructors set up midterms, finals, or short quizzes and enter the dates, the system automatically calculates each student's semester average and pass/fail status. It also generates clear reports on course performance, missing grades, and individual progress.

- Adding new faculty and instructors
- Creating/updating the course catalog and semester course offerings
- Saving student offerings and managing registration status
- Defining exams (type, date, weight)
- Scheduling unsaved and automatic pass/fail assignments
- Reporting on student performance, course success rates, and incomplete grades

This database is actively used to track the registration processes of department secretaries and faculty members, regularly monitor their logins, and store periodic or institutional-level reports.

Main Purpose:

- Integrate student, faculty, and courses.
- Record semester-long course selection and exam processes.
- Record automatic grade-based achievement records.
- Provide summary reports at the departmental and institutional levels.

2) Main Entities and Descriptions

- **Student:** student_id, first_last_name, email, department, registration_date. Basic student identification and contact information.
- **Instructor:** first_last_name, title, email, department. Academic personnel information responsible for courses.
- **Course:** code, description, credit, department. Course information defined at the catalog level.
- **Term:** description, start_date, end_date. Academic term information (Example: Fall 2025).
- **Course Offering:** course, semester, instructor, start_date, end_date. Version of a topic for a specific term (managed by a single faculty member).
- **Registration:** student, offering, registration_date, status. Student's registration for a specific course offering.

- **Exam:** offering, type (midterm/final/quiz), date, weight. Specific exam information and weights provided by the instructor.
- **ExamResult (Grade):** exam, student, score (0–100), pass_flag. Exam grades and pass/fail status information.

3) Functional Features

- **Definitions:** Creating and updating Student, Instructor, Course, and Term tables.
- **Course Creation:** CourseOffering — establishes relationships between course, term, and faculty.
- **Registration Procedures:** Adding or deleting students to the offering; tracking their registration status.
- **Exam & Grade Management:** Specifying the exam type, date, and weight; automatically assigning pass/fail status when grades are entered.
- **Calculations:** Weighted term grade (calculated as $\sum (\text{score} \times \text{weight}) / 100$) and a retention threshold (Example: 60).
- **Reporting:** Course success rates, section/term statistics, and incomplete grade reports (not visible in student view).

4) Constraints

- **Sequence Rule:** Registration is not possible without a prior student record; an offering cannot be opened without a course registered in the catalog.
- **Single Responsibility:** A Course Offering contains only one Instructor.
- **Registration Uniqueness:** The same student can only register for the same offering once (Unique Constraint).
- **Date Rules:** The exam must be registered within the start-end range of the offering; the exam cannot be defined before the course begins.
- **Grade Range:** The point value must be between 0 and 100 (CHECK constraint).
- **Total Weight:** The sum of all Exam weight values for a specific offering must equal 100.
- **Automatic Status:** pass_flag is automatically calculated when grades are entered, and Enrollment_status is automatically finalized at the end of the term.
- **Authorization:** Grade entry and modifications can only be performed by the assigned faculty member or authorized personnel.

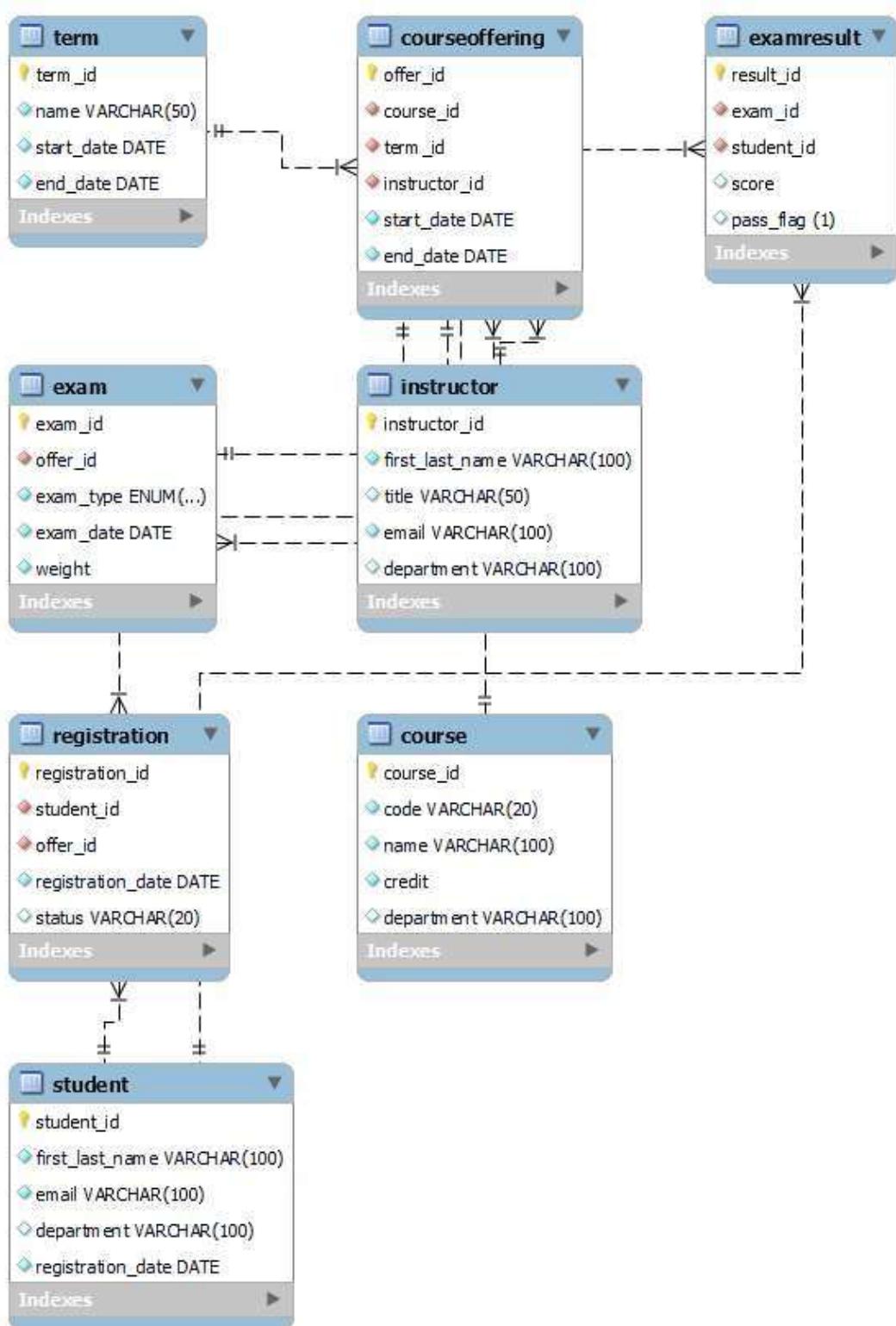
2) ER Diagram

The Entity–Relationship (ER) diagram illustrates how the main components of the Student Exam Management System are connected to each other. Each entity represents a real-world object (such as Student, Course, or Exam), while relationships define how these objects interact.

Students register for course offerings rather than directly for catalog courses. A course offering connects a specific course, a term, and one instructor, meaning that a course can be offered multiple times across different semesters, but each offering is managed by exactly one instructor. The Registration entity models the many-to-many relationship between students and course offerings, ensuring that each student–offering pair is stored only once.

Exams are defined per course offering, and each exam can have multiple results, one per student. The ExamResult entity links students and exams, storing individual scores and pass/fail status. Foreign keys are used consistently to enforce referential integrity, so no registration, exam, or grade can exist without its corresponding student, course offering, or exam record.

Overall, the ER diagram ensures a normalized, consistent structure where data redundancy is minimized and academic processes such as enrollment, assessment, and reporting are represented accurately.



3) Normalization

In this project, database normalization was applied up to Third Normal Form (3NF) in order to minimize redundancy, prevent update anomalies, and ensure data consistency. Each table represents a single logical entity such as Student, Instructor, Course, Registration, or ExamResult. Instead of storing duplicate information across multiple tables, relationships were created using foreign keys.

Normalization also improves scalability. As the number of students, courses, and exams grows, the structure remains efficient and easier to maintain. The following subsections summarize how First, Second and Third Normal Forms are satisfied in our design.

3.1 First Normal Form (1NF)

First Normal Form requires that:

1. Each table has a primary key.
2. All attributes contain atomic (single) values.
3. There are no repeating groups or repeating columns.

In our system:

- The Student table stores one row per student. There are no repeated course columns or list-type fields.
- The ExamResult table stores one row for each (*student, exam*) pair, instead of multiple exam scores inside a single row.
- All fields (names, dates, scores, weights) contain only one value per cell. Therefore, all tables comply with 1NF.

3.2 Second Normal Form (2NF)

Second Normal Form applies to tables that have composite primary keys and states that:

Every non-key attribute must depend on the entire primary key, not only a part of it.

The main example is the Registration table:

- Its logical key is $(student_id, offer_id)$.
- Attributes such as $registration_date$ and $status$ depend on the combination of the student and the course offering.

No attribute depends only on $student_id$ or only on $offer_id$.

Thus, partial dependencies are eliminated, and the table satisfies 2NF.

3.3 Third Normal Form (3NF)

Third Normal Form requires that:

There are no transitive dependencies, meaning non-key attributes do not depend on other non-key attributes.

In our design:

- In Student, the department is not derived from email or any other field.
- In Instructor, title and department do not determine each other.
- Course details (name, credit, department) exist only in the Course table and are not duplicated in Registration or Exam tables.

Because non-key attributes depend only on their primary keys, the schema satisfies 3NF.

Conclusion

In conclusion, the database is normalized up to Third Normal Form, which prevents redundant data, reduces insertion and update anomalies, and ensures consistency across all academic operations in the system.

4)Tables and Sample Data Entities

```
5 •  DROP DATABASE IF EXISTS StudentExamDB;
6 •  CREATE DATABASE StudentExamDB;
7 •  USE StudentExamDB;
8
9  /* ===== */
0  /* TABLE DEFINITIONS */
1  /* ===== */
2
3 •  CREATE TABLE Student (
4      student_id INT PRIMARY KEY AUTO_INCREMENT,
5      first_last_name VARCHAR(100) NOT NULL,
6      email VARCHAR(100) UNIQUE NOT NULL,
7      department VARCHAR(100),
8      registration_date DATE NOT NULL
9 );
0
1 •  CREATE TABLE Instructor (
2     instructor_id INT PRIMARY KEY AUTO_INCREMENT,
3     first_last_name VARCHAR(100) NOT NULL,
4     title VARCHAR(50),
5     email VARCHAR(100) UNIQUE NOT NULL,
6     department VARCHAR(100)
7 );
R
```

```
25 • Ⓜ CREATE TABLE Course (
26     course_id INT PRIMARY KEY AUTO_INCREMENT,
27     code VARCHAR(20) UNIQUE NOT NULL,
28     name VARCHAR(100) NOT NULL,
29     credit INT NOT NULL,
30     department VARCHAR(100)
31 );
32
33
34 • Ⓜ CREATE TABLE Term (
35     term_id INT PRIMARY KEY AUTO_INCREMENT,
36     name VARCHAR(50) NOT NULL,
37     start_date DATE NOT NULL,
38     end_date DATE NOT NULL
39 );
40
41
42
43 • Ⓜ CREATE TABLE CourseOffering (
44     offer_id INT PRIMARY KEY AUTO_INCREMENT,
45     course_id INT NOT NULL,
46     term_id INT NOT NULL,
47     instructor_id INT NOT NULL,
48     start_date DATE NOT NULL,
49     end_date DATE NOT NULL,
50
51     FOREIGN KEY (course_id) REFERENCES Course(course_id),
52     FOREIGN KEY (term_id) REFERENCES Term(term_id),
53     FOREIGN KEY (instructor_id) REFERENCES Instructor(instructor_id)
54 );
```

Query 1

```
57
58
59 ● - CREATE TABLE Registration (
60     registration_id INT PRIMARY KEY AUTO_INCREMENT,
61     student_id INT NOT NULL,
62     offer_id INT NOT NULL,
63     registration_date DATE NOT NULL,
64     status VARCHAR(20),
65
66     UNIQUE(student_id, offer_id),
67
68     FOREIGN KEY (student_id) REFERENCES Student(student_id),
69     FOREIGN KEY (offer_id) REFERENCES CourseOffering(offer_id)
70 );
71
72
73 ● - CREATE TABLE Exam (
74     exam_id INT PRIMARY KEY AUTO_INCREMENT,
75     offer_id INT NOT NULL,
76     exam_type ENUM('midterm','final','quiz') NOT NULL,
77     exam_date DATE NOT NULL,
78     weight INT NOT NULL CHECK (weight BETWEEN 0 AND 100),
79
80     FOREIGN KEY (offer_id) REFERENCES CourseOffering(offer_id)
81 );
82
83
84 ● - CREATE TABLE ExamResult (
85     result_id INT PRIMARY KEY AUTO_INCREMENT,
86     exam_id INT NOT NULL,
87     student_id INT NOT NULL,
88     score INT CHECK(score BETWEEN 0 AND 100),
89     pass_flag BOOLEAN,
90
91     FOREIGN KEY (exam_id) REFERENCES Exam(exam_id),
92     FOREIGN KEY (student_id) REFERENCES Student(student_id)
93 );
```

```
1 • INSERT INTO Student (first_last_name, email, department, registration_date) VALUES
2     ('Mehmet Koyuncu', 'mehmet.koyuncu@stu.edu', 'Computer Engineering', '2023-09-01'),
3     ('Mustafa Öz', 'mustafa.oz@stu.edu', 'Computer Engineering', '2023-09-01'),
4     ('Ahmet Demir', 'ahmet.demir@stu.edu', 'Software Engineering', '2022-09-05'),
5     ('Ayşe Yılmaz', 'ayse.yilmaz@stu.edu', 'Computer Engineering', '2021-09-10'),
6     ('Fatma Kara', 'fatma.kara@stu.edu', 'Information Systems', '2024-02-01'),
7     ('Elli Aydin', 'elli/aydin@stu.edu', 'Computer Engineering', '2023-02-18'),
8     ('Hakan Celik', 'hakan.celik@stu.edu', 'Software Engineering', '2022-10-01'),
9     ('Zeynep Er', 'zeynep.er@stu.edu', 'Computer Engineering', '2021-09-07'),
10    ('Baran Uçar', 'baran.ucar@stu.edu', 'Information Systems', '2023-09-15'),
11    ('Çağla Ari', 'cagla.ari@stu.edu', 'Computer Engineering', '2023-02-28'),
12    ('Berk Atay', 'berk.atay@stu.edu', 'Software Engineering', '2024-01-10'),
13    ('Deniz Can', 'deniz.can@stu.edu', 'Computer Engineering', '2023-09-01');
14
15    INSERT INTO Instructor (first_last_name, title, email, department) VALUES
16        ('Bahman Arasteh', 'Assoc. Prof.', 'bahman.arasteh@uni.edu', 'Software Engineering'),
17        ('Seda Yıldız', 'Dr.', 'seda.yildiz@uni.edu', 'Computer Engineering'),
18        ('Mert Aksoy', 'Dr.', 'mert.aksoy@uni.edu', 'Computer Engineering'),
19        ('Gökce Şahin', 'Prof. Dr.', 'gokce.sahin@uni.edu', 'Information Systems'),
20        ('Efe Yaçıcı', 'Dr.', 'efe.yacici@uni.edu', 'Software Engineering'),
21        ('Kerem İnce', 'Assist. Prof.', 'kerem.ince@uni.edu', 'Computer Engineering'),
22        ('Nazan Korkmaz', 'Dr.', 'nazan.korkmaz@uni.edu', 'Software Engineering'),
23        ('Müseyin Ateş', 'Assoc. Prof.', 'museyin.ates@uni.edu', 'Computer Engineering'),
24        ('Sinem Al', 'Dr.', 'sinem.al@uni.edu', 'Information Systems'),
25        ('Onur Taş', 'Dr.', 'onur.tas@uni.edu', 'Computer Engineering');
26
27 • INSERT INTO Course (code, name, credit, department) VALUES
28     ('CSE101', 'Introduction to Programming', 6, 'Computer Engineering'),
29     ('CSE102', 'Data Structures', 6, 'Computer Engineering'),
30     ('CSE201', 'Database Management Systems', 5, 'Software Engineering'),
31     ('CSE202', 'Operating Systems', 5, 'Computer Engineering'),
32     ('CSE203', 'Algorithms', 6, 'Computer Engineering'),
33     ('CSE204', 'Web Development', 5, 'Software Engineering'),
34     ('CSE205', 'Computer Networks', 5, 'Computer Engineering'),
35     ('CSE301', 'Machine Learning', 6, 'Computer Engineering'),
36     ('CSE302', 'Artificial Intelligence', 6, 'Software Engineering'),
37     ('CSE303', 'Mobile App Development', 5, 'Software Engineering'),
38     ('CSE304', 'Cloud Computing', 5, 'Information Systems'),
39     ('CSE305', 'Cyber Security', 6, 'Computer Engineering');
```

```
 41 •    INSERT INTO Term (name, start_date, end_date) VALUES
 42     ('Fall 2023', '2023-09-01', '2023-12-30'),
 43     ('Spring 2024', '2024-02-01', '2024-06-01'),
 44     ('Fall 2024', '2024-09-01', '2024-12-30'),
 45     ('Spring 2023', '2023-02-01', '2023-06-01'),
 46     ('Fall 2025', '2025-09-01', '2025-12-30'),
 47     ('Spring 2025', '2025-02-01', '2025-06-01');
 48
 49 •    INSERT INTO CourseOffering (course_id, term_id, instructor_id, start_date, end_date) VALUES
 50     (1, 1, 1, '2023-09-10', '2023-12-20'),
 51     (2, 1, 2, '2023-09-12', '2023-12-22'),
 52     (3, 2, 1, '2024-02-05', '2024-05-25'),
 53     (4, 2, 3, '2024-02-07', '2024-05-28'),
 54     (5, 1, 2, '2023-09-11', '2023-12-21'),
 55     (6, 2, 4, '2024-02-10', '2024-05-30'),
 56     (7, 3, 6, '2024-09-05', '2024-12-20'),
 57     (8, 3, 7, '2024-09-07', '2024-12-23'),
 58     (9, 3, 8, '2024-09-12', '2024-12-27'),
 59     (10, 2, 5, '2024-02-08', '2024-05-26'),
 60     (11, 3, 10, '2024-09-10', '2024-12-22'),
 61     (12, 1, 9, '2023-09-15', '2023-12-25');
 62
 63 •    INSERT INTO Registration (student_id, offer_id, registration_date, status) VALUES
 64     (1, 1, '2023-09-05', 'registered'),
 65     (2, 1, '2023-09-06', 'registered'),
 66     (3, 1, '2023-09-06', 'registered'),
 67     (1, 2, '2023-09-06', 'registered'),
 68     (4, 2, '2023-09-07', 'registered'),
 69     (5, 2, '2023-09-07', 'registered'),
 70     (6, 3, '2024-02-10', 'registered'),
 71     (7, 3, '2024-02-11', 'registered'),
 72     (8, 4, '2024-02-11', 'registered'),
 73     (9, 4, '2024-02-12', 'registered'),
 74     (10, 5, '2023-09-10', 'registered'),
 75     (11, 6, '2024-02-15', 'registered'),
 76     (12, 6, '2024-02-16', 'registered'),
 77     (1, 7, '2024-09-05', 'registered'),
 78     (2, 7, '2024-09-05', 'registered');
 79
```

```
76      (1, 1, '2023-11-01', 'registered');
77
78  ● INSERT INTO Exam (offer_id, exam_type, exam_date, weight) VALUES
79
80    (1, 'midterm', '2023-11-01', 40),
81    (1, 'final', '2023-12-20', 60),
82    (2, 'midterm', '2023-11-05', 50),
83    (2, 'final', '2023-12-22', 50),
84    (3, 'quiz', '2024-03-01', 20),
85    (3, 'final', '2024-05-20', 80),
86    (4, 'midterm', '2024-03-10', 40),
87    (4, 'final', '2024-05-28', 60),
88    (5, 'midterm', '2023-11-12', 50),
89    (5, 'final', '2023-12-21', 50),
90    (6, 'midterm', '2024-03-15', 40),
91    (6, 'final', '2024-05-30', 60);
92
93
94  ● INSERT INTO ExamResult (exam_id, student_id, score, pass_flag) VALUES
95    (1, 1, 78, TRUE),
96    (1, 2, 65, TRUE),
97    (1, 3, 55, FALSE),
98    (2, 1, 82, TRUE),
99    (2, 2, 60, TRUE),
100   (3, 1, 70, TRUE),
101   (4, 4, 58, FALSE),
102   (5, 6, 80, TRUE),
103   (5, 7, 68, TRUE),
104   (6, 6, 74, TRUE),
105   (6, 7, 69, TRUE),
106   (7, 8, 50, FALSE),
107   (8, 8, 62, TRUE),
108   (9, 10, 90, TRUE),
109   (10, 10, 84, TRUE);
```

5) QUERIES AND OUTPUTS

A) Stored Procedure 1- Missing Grades Report

Why / What it does

"This stored procedure is designed to ensure data integrity by identifying missing academic records. It utilizes a LEFT JOIN between the exam schedule and the student results table. By filtering for NULL values in the score column, it effectively isolates instances where a student is registered for a course offering but has not yet received a grade for a specific exam. This allows administrators to track pending evaluations efficiently."

Query Question

"Which students enrolled in a specific course offering (identified by p_offer_id) are missing scores for their scheduled exams, and what are the details of these missing assessments?"

Creation Code

```
DELIMITER $$  
• CREATE PROCEDURE sp_missing_grades_by_offer(IN p_offer_id INT)  
BEGIN  
    SELECT  
        r.offer_id, s.student_id, s.first_last_name, s.email,  
        e.exam_id, e.exam_type, e.exam_date  
    FROM Registration r  
    INNER JOIN Student s ON s.student_id = r.student_id  
    INNER JOIN Exam e ON e.offer_id = r.offer_id  
    LEFT JOIN ExamResult er  
        ON er.exam_id = e.exam_id  
        AND er.student_id = s.student_id  
    WHERE r.offer_id = p_offer_id  
        AND er.score IS NULL  
    ORDER BY s.student_id, e.exam_date;  
END$$  
DELIMITER ;
```

Run Code

- `CALL sp_missing_grades_by_offer(1);`

Query Output

The screenshot shows a MySQL Workbench interface with a result grid. The grid has columns: offer_id, student_id, first_last_name, email, exam_id, exam_type, and exam_date. There is one row of data: offer_id 1, student_id 3, first_last_name Ahmet Demir, email ahmet.demir@stu.edu, exam_id 2, exam_type final, and exam_date 2023-12-20.

	offer_id	student_id	first_last_name	email	exam_id	exam_type	exam_date
▶	1	3	Ahmet Demir	ahmet.demir@stu.edu	2	final	2023-12-20

A) Stored Procedure 2- Course Performance (HAVING + AVG)

Why / What it does

This stored procedure reports the performance of CourseOffers within a specific term. It calculates a weighted average using the Exam and ExamResult tables; then, using GROUP BY, it extracts the average for each offer and filters those above the minimum average you define using HAVING. This allows you to measure, on a term-by-term basis, which course offerings are successful.

Query Question

“In a given term (term_id), which course offerings have a weighted average of at least X for the class? (Course + Instructor + Average)”

Creation Code

```
201     DELIMITER $$  
tem  
202  
203 • CREATE PROCEDURE sp_course_performance_summary(  
204     IN p_term_id INT,  
205     IN p_min_avg DECIMAL(10,2)  
206 )  
207 BEGIN  
208     SELECT  
209         t.term_id,  
210         t.name AS term_name,  
211         co.offer_id,  
212         c.code AS course_code,  
213         c.name AS course_name,  
214         i.first_last_name AS instructor_name,  
215         AVG(er.score * e.weight) / 100 AS class_weighted_avg  
216     FROM Term t  
217     INNER JOIN CourseOffering co ON co.term_id = t.term_id  
218     INNER JOIN Course c ON c.course_id = co.course_id  
219     INNER JOIN Instructor i ON i.instructor_id = co.instructor_id  
220     INNER JOIN Exam e ON e.offer_id = co.offer_id  
221     INNER JOIN ExamResult er ON er.exam_id = e.exam_id  
222     WHERE t.term_id = p_term_id  
223     GROUP BY  
224         t.term_id, t.name,  
225         co.offer_id,  
226         c.code, c.name,  
227         i.first_last_name  
228     HAVING class_weighted_avg >= p_min_avg  
229     ORDER BY class_weighted_avg DESC;  
230 END$$  
231  
232     DELIMITER ;  
~~~
```

Run Code

- `CALL sp_course_performance_summary(1, 0);`

Query Output

Result Grid							
Filter Rows: <input type="text"/>							
	term_id	term_name	offer_id	course_code	course_name	instructor_name	class_weighted_avg
▶	1	Fall 2023	5	CSE203	Algorithms	Seda Yildiz	43.50000000
	1	Fall 2023	1	CSE101	Introduction to Programming	Bahman Arasteh	32.88000000
	1	Fall 2023	2	CSE102	Data Structures	Seda Yildiz	32.00000000

B) Stored Function 1 — Student Offer Weighted Average

Why / What it does

This stored function calculates the weighted average of a specific student (student_id) for a specific course (offer_id). It combines the weight values from the Exam table with the score values from the ExamResult table, returning a single average value using the formula:

$\text{SUM(score * weight)} / 100.$

Using LEFT JOIN + COALESCE, it avoids errors if there are unentered grades for exams (returns to null with 0).

This structure prevents the same calculation from being written repeatedly and allows for reuse within a View/Procedure.

Query Question

“What is Student A’s weighted average (term average) for Offer B?”

Creation Code

```
201      DELIMITER $$  
202  
203  •  CREATE FUNCTION fn_student_offer_average(p_student_id INT, p_offer_id INT)  
204      RETURNS DECIMAL(10,2)  
205      DETERMINISTIC  
206      READS SQL DATA  
207      BEGIN  
208          DECLARE v_avg DECIMAL(10,2);  
209  
210          SELECT  
211              COALESCE(SUM(er.score * e.weight) / 100, 0)  
212              INTO v_avg  
213              FROM Exam e  
214              LEFT JOIN ExamResult er  
215                  ON er.exam_id = e.exam_id  
216                  AND er.student_id = p_student_id  
217              WHERE e.offer_id = p_offer_id;  
218  
219          RETURN v_avg;  
220      END$$  
~~~
```

Run Code

```
221
222     DELIMITER ;
223 •  SELECT fn_student_offer_average(1, 1) AS student_weighted_avg;
224
225
```

Query Output

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	student_weighted_avg			
▶	80.40			

B) Stored Function 2 — Offer Success Rate (%)

Why / What it does

This stored function returns:the total number of registered students (COUNT),the number of students who passed (COUNT + subquery/function call),and the success rate as a percentage, for a given offer_id.It directly supports reporting the “course success rate” in the scenario. It also fulfills the requirement for using a subquery.

Query Question

“What is the success rate percentage for Offer X course launch? (Pass threshold = 60)”

Creation Code

```
DELIMITER $$

• CREATE FUNCTION fn_offer_success_rate(p_offer_id INT)
  RETURNS DECIMAL(5,2)
  DETERMINISTIC
  READS SQL DATA
BEGIN
  DECLARE v_total INT;
  DECLARE v_passed INT;

  -- Total registered students
  SELECT COUNT(*) INTO v_total
  FROM Registration
  WHERE offer_id = p_offer_id;

  -- Passed students (weighted average >= 60)
  SELECT COUNT(*) INTO v_passed
  FROM Registration r
  WHERE r.offer_id = p_offer_id
    AND fn_student_offer_average(r.student_id, r.offer_id) >= 60;

  RETURN CASE
    WHEN v_total = 0 THEN 0
    ELSE ROUND((v_passed / v_total) * 100, 2)
  END;
END$$
DELIMITER ;
```

Run Code

```
259 •   SELECT fn_offer_success_rate(1) AS success_rate_percent;
260
```

Query Output

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	success_rate_percent			
▶	66.67			

(I separated responsibilities: one function calculates a single student's average, and the other uses it to compute the course success rate. This improves readability and maintainability.)

C) View 1—v_student_exam_details

Why / What it does

This view summarizes exam performance per course offering and exam type. It provides class-level metrics (student count, average score, total weight) to support reporting and comparison across offerings.

Query Question

“For each course offering, what are the exam-level statistics (number of graded students, average score, total exam weight) grouped by exam type?”

Creation Code

```
● CREATE OR REPLACE VIEW v_offer_exam_stats AS
  SELECT
    co.offer_id,
    c.code AS course_code,
    c.name AS course_name,
    t.name AS term_name,
    e.exam_type,
    COUNT(er.score) AS graded_count,
    AVG(er.score) AS avg_score,
    SUM(e.weight) AS total_exam_weight
  FROM CourseOffering co
  INNER JOIN Course c ON c.course_id = co.course_id
  INNER JOIN Term t ON t.term_id = co.term_id
  INNER JOIN Exam e ON e.offer_id = co.offer_id
  LEFT JOIN ExamResult er ON er.exam_id = e.exam_id
  GROUP BY
    co.offer_id, c.code, c.name, t.name, e.exam_type
  HAVING COUNT(er.score) >= 1;
```

Run Code

```
! ●  SELECT *
!   FROM v_offer_exam_stats
!   ORDER BY term_name, offer_id, exam_type;
```

Query Output

	offer_id	course_code	course_name	term_name	exam_type	graded_count	avg_score	total_exam_weight
▶	1	CSE101	Introduction to Programming	Fall 2023	midterm	3	66.0000	120
	1	CSE101	Introduction to Programming	Fall 2023	final	2	71.0000	120
	2	CSE102	Data Structures	Fall 2023	midterm	1	70.0000	50
	2	CSE102	Data Structures	Fall 2023	final	1	58.0000	50
	5	CSE203	Algorithms	Fall 2023	midterm	1	90.0000	50
	5	CSE203	Algorithms	Fall 2023	final	1	84.0000	50
	3	CSE201	Database Management Systems	Spring 2024	final	2	71.0000	160
	3	CSE201	Database Management Systems	Spring 2024	quiz	2	74.0000	40
	4	CSE202	Operating Systems	Spring 2024	midterm	1	50.0000	40
	4	CSE202	Operating Systems	Spring 2024	final	1	62.0000	60

C)View 2 — v_offer_success_summary_independent

Why / What it does

This view calculates the **success rate per course offering** without relying on any stored functions. For each registered student, it computes the student's **weighted average** for that offering using an inline **subquery**, then aggregates results at the offering level to produce pass count and overall success rate.

Query Question

“What is the success rate (passed students / registered students) for each course offering, where passing is defined as weighted average ≥ 60 ? ”

Creation Code

```
● CREATE OR REPLACE VIEW v_offer_success_summary_independent AS
SELECT
    x.offer_id,
    x.course_code,
    x.course_name,
    x.term_name,
    COUNT(*) AS registered_count,
    SUM(CASE WHEN x.weighted_avg >= 60 THEN 1 ELSE 0 END) AS passed_count,
    ROUND((SUM(CASE WHEN x.weighted_avg >= 60 THEN 1 ELSE 0 END) / COUNT(*)) * 100, 2) AS success_rate_percent
FROM
    (
        SELECT
            r.offer_id,
            c.code AS course_code,
            c.name AS course_name,
            t.name AS term_name,
            r.student_id,
            (
                SELECT COALESCE(SUM(er2.score * e2.weight) / 100, 0)
                FROM Exam e2
                LEFT JOIN ExamResult er2
                    ON er2.exam_id = e2.exam_id
                    AND er2.student_id = r.student_id
                WHERE e2.offer_id = r.offer_id
            ) AS weighted_avg
        FROM Registration r
        INNER JOIN CourseOffering co ON co.offer_id = r.offer_id
        INNER JOIN Course c ON c.course_id = co.course_id
        INNER JOIN Term t ON t.term_id = co.term_id
    ) AS x
GROUP BY
    x.offer_id, x.course_code, x.course_name, x.term_name
HAVING COUNT(*) >= 1;
```

Run Code

```
● SELECT *
FROM v_offer_success_summary_independent
ORDER BY success_rate_percent DESC;
```

Query Output

Result Grid							
	offer_id	course_code	course_name	term_name	registered_count	passed_count	success_rate_percent
▶	5	CSE203	Algorithms	Fall 2023	1	1	100.00
▶	3	CSE201	Database Management Systems	Spring 2024	2	2	100.00
▶	1	CSE101	Introduction to Programming	Fall 2023	3	2	66.67
▶	2	CSE102	Data Structures	Fall 2023	3	0	0.00
▶	4	CSE202	Operating Systems	Spring 2024	2	0	0.00
▶	6	CSE204	Web Development	Spring 2024	2	0	0.00
▶	7	CSE205	Computer Networks	Fall 2024	2	0	0.00

Additional Query (RIGHT JOIN Usage)

Query Question

“List all students and their exam results (if any). Include students who have no exam result yet.”

Run Code

```
(10,10,84,TRUE);
```

- `SELECT`
 `s.student_id,`
 `s.first_last_name,`
 `s.email,`
 `er.exam_id,`
 `er.score,`
 `er.pass_flag`
 `FROM ExamResult er`
 `RIGHT JOIN Student s`
 `ON er.student_id = s.student_id`
 `ORDER BY s.student_id, er.exam_id;`

Output

	student_id	first_last_name	email	exam_id	score	pass_flag
▶	1	Mehmet Koyuncu	mehmet.koyuncu@stu.edu	1	78	1
	1	Mehmet Koyuncu	mehmet.koyuncu@stu.edu	2	82	1
	1	Mehmet Koyuncu	mehmet.koyuncu@stu.edu	3	70	1
	2	Mustafa Oz	mustafa.oz@stu.edu	1	65	1
	2	Mustafa Oz	mustafa.oz@stu.edu	2	60	1
	3	Ahmet Demir	ahmet.demir@stu.edu	1	55	0
	4	Ayse Yilmaz	ayse.yilmaz@stu.edu	4	58	0
	5	Fatma Kara	fatma.kara@stu.edu	NULL	NULL	NULL
	6	Elif Aydin	elif.aydin@stu.edu	5	80	1
	6	Elif Aydin	elif.aydin@stu.edu	6	73	1
	7	Hakan Celik	hakan.celik@stu.edu	5	68	1
	7	Hakan Celik	hakan.celik@stu.edu	6	69	1
	8	Zeynep Er	zeynep.er@stu.edu	7	50	0
	8	Zeynep Er	zeynep.er@stu.edu	8	62	1
	9	Baran Ucar	baran.ucar@stu.edu	NULL	NULL	NULL
	10	Cagla Ari	cagla.ari@stu.edu	9	90	1
	10	Cagla Ari	cagla.ari@stu.edu	10	84	1
	11	Berk Atay	berk.atay@stu.edu	NULL	NULL	NULL
	12	Deniz Can	deniz.can@stu.edu	NULL	NULL	NULL

D) Trigger 1 — trg_exam_rules_before_insert

Why / What it does

This trigger enforces key scenario constraints at the database level:

1. An exam date must fall within the related course offering's start and end dates.
2. During exam definition, the sum of all exam weights for the same offering **must not exceed 100**; the assessment plan is considered complete when the total reaches 100.
 - This trigger prevents exceeding 100 at insert time; ensuring the final total equals 100 is enforced operationally by defining the remaining exam weights accordingly.

It prevents invalid exam definitions and guarantees data integrity even if inserts are attempted manually.

Query Question

“How can we prevent inserting exams with invalid dates or weights so that exam rules are always enforced for each course offering?”

Creation Code

```
DELIMITER $$

• CREATE TRIGGER trg_exam_rules_before_insert
  BEFORE INSERT ON Exam
  FOR EACH ROW
  BEGIN
    DECLARE v_start DATE;
    DECLARE v_end DATE;
    DECLARE v_sum INT;

    -- Get offering date range
    SELECT start_date, end_date
      INTO v_start, v_end
     FROM CourseOffering
    WHERE offer_id = NEW.offer_id;

    -- Rule 1: Exam date must be within offering dates
    IF NEW.exam_date < v_start OR NEW.exam_date > v_end THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Exam date must be within the CourseOffering date range.';
    END IF;

    -- Rule 2: Total weight must not exceed 100
    SELECT COALESCE(SUM(weight), 0)
      INTO v_sum
     FROM Exam
    WHERE offer_id = NEW.offer_id;

    IF v_sum + NEW.weight > 100 THEN
      SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Total exam weights for an offering cannot exceed 100.';
    END IF;
  END$$

DELIMITER ;
```

Run Code

- `INSERT INTO Exam (offer_id, exam_type, exam_date, weight)
VALUES (1, 'midterm', '2000-01-01', 10);`
- `INSERT INTO Exam (offer_id, exam_type, exam_date, weight)
VALUES (1, 'quiz', '2023-11-15', 10);`

Query Output

✓	1	21:12:43	DROP TRIGGER IF EXISTS trg_exam_rules_before_insert	0 row(s) affected
⚠	2	21:12:55	DROP TRIGGER IF EXISTS trg_exam_rules_before_insert	0 row(s) affected. 1 warning(s): 1360 Trigger does not exist
✓	3	21:12:55	CREATE TRIGGER trg_exam_rules_before_insert BEFORE INSERT ON Exam FOR EACH ROW BEGIN DEC...	0 row(s) affected
✓	4	21:13:02	DROP TRIGGER IF EXISTS trg_exam_rules_before_insert	0 row(s) affected
✓	5	21:13:02	CREATE TRIGGER trg_exam_rules_before_insert BEFORE INSERT ON Exam FOR EACH ROW BEGIN DEC...	0 row(s) affected
✗	6	21:13:02	INSERT INTO Exam (offer_id, exam_type, exam_date, weight) VALUES (1, 'midterm', '2000-01-01', 10)	Error Code: 1644. Exam date must be within the CourseOffering date range.
✓	7	21:13:21	DROP TRIGGER IF EXISTS trg_exam_rules_before_insert	0 row(s) affected
✓	8	21:13:21	CREATE TRIGGER trg_exam_rules_before_insert BEFORE INSERT ON Exam FOR EACH ROW BEGIN DEC...	0 row(s) affected
✗	9	21:13:21	INSERT INTO Exam (offer_id, exam_type, exam_date, weight) VALUES (1, 'midterm', '2000-01-01', 10)	Error Code: 1644. Exam date must be within the CourseOffering date range.

D)Trigger 2 — `trg_examresult_set_pass_flag`

Why / What it does

This trigger automatically sets the `pass_flag` value when a new exam result is inserted.

If the student's score is 60 or higher, the student is marked as passed; otherwise, the student is marked as failed.

This ensures consistent evaluation logic at the database level and prevents manual errors.

Query Question

“How can we automatically determine whether a student passes or fails an exam when the exam score is inserted?”

Creation Code

```
● CREATE TRIGGER trg_examresult_set_pass_flag
  BEFORE INSERT ON ExamResult
  FOR EACH ROW
  BEGIN
    IF NEW.score IS NULL THEN
      SET NEW.pass_flag = NULL;
    ELSEIF NEW.score >= 60 THEN
      SET NEW.pass_flag = TRUE;
    ELSE
      SET NEW.pass_flag = FALSE;
    END IF;
  END$$

  DELIMITER ;
● INSERT INTO ExamResult (exam_id, student_id, score, pass_flag)
VALUES (1, 4, 45, NULL);
```

Run Code

```
7 ●   INSERT INTO ExamResult (exam_id, student_id, score, pass_flag)
8     VALUES (1, 4, 45, NULL);
9 ●   SELECT exam_id, student_id, score, pass_flag
10    FROM ExamResult
11   WHERE exam_id = 1 AND student_id = 4;
12
```

Query Output

Result Grid				
	exam_id	student_id	score	pass_flag
▶	1	4	45	0

E) Transaction 1 — Safe Registration (Enroll a student atomically)

Why / What it does

This transaction registers a student to a course offering **atomically** (all-or-nothing).

It first checks whether the student is already registered for the same offering. If not, it inserts the registration and commits. If the student is already registered, it rolls back to prevent duplicate/enforced-unique constraint issues.

Query Question

“How can we safely register a student to a course offering without creating duplicate registrations, ensuring the operation is atomic?”

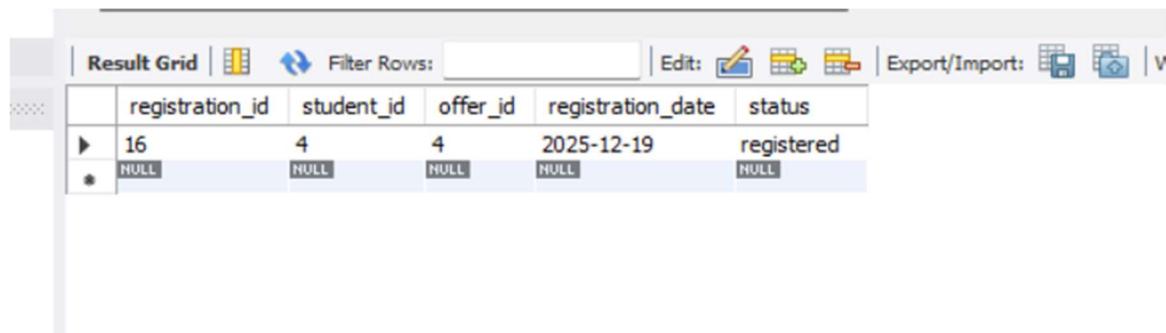
Creation Code

- `START TRANSACTION;`
- `-- If the registration already exists, do not insert`
- `INSERT INTO Registration (student_id, offer_id, registration_date, status)`
`SELECT 4, 4, CURDATE(), 'registered'`
- `WHERE NOT EXISTS (` ●
 `SELECT 1`
 `FROM Registration`
 `WHERE student_id = 4 AND offer_id = 4`
);
- `COMMIT;`

Run Code

- ▶ `SELECT *`
 `FROM Registration`
 `WHERE student_id = 4 AND offer_id = 4;`

Query Output



	registration_id	student_id	offer_id	registration_date	status
▶	16	4	4	2025-12-19	registered
*	NULL	NULL	NULL	NULL	NULL

E) Transaction 2 — Safe Grade Entry (Atomic Grade Insert)

Why / What it does

This transaction safely inserts an exam result for a student by ensuring the operation is completed as a single atomic unit.

It prevents partial inserts and guarantees data consistency during grade entry operations.

Query Question

“How can we safely insert a student’s exam score into the system using a transaction?”

Creation Code

```
• START TRANSACTION;  
•  
• INSERT INTO ExamResult (exam_id, student_id, score, pass_flag)  
VALUES (2, 4, 72, NULL);  
•  
• COMMIT;  
•
```

Run Code

```
• SELECT exam_id, student_id, score, pass_flag  
FROM ExamResult  
WHERE exam_id = 2 AND student_id = 4;  
#####extra  
• SELECT  
    e.exam_type,  
    COUNT(er.result_id) AS total_results,  
    AVG(er.score) AS avg_score  
FROM Exam e  
LEFT JOIN ExamResult er ON er.exam_id = e.exam_id  
GROUP BY e.exam_type  
HAVING COUNT(er.result_id) >= 0;
```

Query Output

Result Grid | Filter Rows: Export: Wrap Cell Content:

	exam_id	student_id	score	pass_flag
▶	2	4	72	NULL
	2	4	72	NULL
	2	4	72	NULL
	2	4	72	NULL

Result Grid | Filter Rows: Export: Wrap Cell Content:

	exam_type	total_results	avg_score
▶	midterm	6	68.0000
	final	11	70.5455
	quiz	2	74.0000

6) INTERFACE (BONUS)

Interface Overview

This section provides a brief introduction to the graphical interface of the Student Exam Control System. A more detailed description, together with the full source code, database files, and demonstration video, is included in the attached ZIP package.

The interface was designed to make the management of students, instructors, courses, registrations, and exams as simple and intuitive as possible. Each tab corresponds to a functional module in the database and performs CRUD operations (create, read, update, delete) directly on the underlying tables.

Panel Descriptions

1. Students Panel

This panel is used to create, update, and remove student records. Users can enter name, email, and department, and immediately see the changes reflected in the student list.

The system prevents duplicate emails and ensures valid registration data

The screenshot shows a Windows application window titled "Student Exam System". At the top, there is a navigation bar with links: Students, Instructors, Courses & Offerings, Registration, Exams & Grades. Below the navigation bar, there is a sub-menu for "Add / Edit Student" with three input fields: "Full Name" (containing "Ahmet Demir"), "Email" (containing "ahmet.demir@stu.edu"), and "Department" (containing "Software Engineering"). A "Save Student" button is located below these fields. Below the form, there is a table listing 15 student records. The table has columns: ID, Full Name, Email, Department, and Reg. Date. The data is as follows:

ID	Full Name	Email	Department	Reg. Date
3	Ahmet Demir	ahmet.demir@stu.edu	Software Engineering	2022-09-05
4	Ayse Yilmaz	ayse.yilmaz@stu.edu	Computer Engineering	2021-09-10
9	Baran Ucar	baran.ucar@stu.edu	Information Systems	2023-05-15
11	Berk Atay	berk.atay@stu.edu	Software Engineering	2024-01-10
10	Cagla Ari	cagla.ari@stu.edu	Computer Engineering	2023-02-20
12	Deniz Can	deniz.can@stu.edu	Computer Engineering	2023-09-01
6	Elif Aydin	elif.aydin@stu.edu	Computer Engineering	2023-02-10
5	Fatma Kara	fatma.kara@stu.edu	Information Systems	2024-02-01
7	Hakan Celik	hakan.celik@stu.edu	Software Engineering	2022-10-21
1	Mehmet Koyuncu	mehmet.koyuncu@stu.edu	Computer Engineering	2023-09-01
2	Mustafa Oz	mustafa.oz@stu.edu	Computer Engineering	2023-09-01
8	Zeynep Er	zeynep.er@stu.edu	Computer Engineering	2021-09-07

At the bottom of the table, there is a "Delete Selected" button.

2. Instructors Panel

This panel manages academic staff information such as full name, title, email, and department.

It ensures that every course offering is always assigned to a valid instructor, enforcing referential integrity in the system.

ID	Full Name	Title	Email	Department
1	Bahman Arasteh	Assoc. Prof.	bahman.arasteh@unl.edu	Software Engineering
5	Efe Yalcin	Dr.	efe.yalcin@unl.edu	Software Engineering
4	Gokce Sahin	Prof. Dr.	gokce.sahin@unl.edu	Information Systems
8	Huseyin Ates	Assoc. Prof.	huseyin.ates@unl.edu	Computer Engineering
6	Kerem Ince	Asst. Prof.	kerem.ince@unl.edu	Computer Engineering
3	Mert Aksoy	Dr.	mert.aksoy@unl.edu	Computer Engineering
7	Nazan Korkmaz	Dr.	nazan.korkmaz@unl.edu	Software Engineering
10	Onur Tas	Dr.	onur.tas@unl.edu	Computer Engineering
2	Seda Yildiz	Dr.	seda.yildiz@unl.edu	Computer Engineering
9	Sinem Ali	Dr.	sinem.ali@unl.edu	Information Systems

3. Courses & Offerings Panel

The upper section allows creation and editing of catalog courses (code, name, credit, department).

The lower section allows scheduling of specific course offerings for a particular term, with start and end dates and an assigned instructor. Only existing courses and instructors may be selected, ensuring consistency.

The screenshot shows the 'Student Exam System' application window. At the top, there is a navigation bar with links: Students, Instructors, Courses & Offerings, Registration, Exams & Grades. Below the navigation bar, there is a 'Create / Update Course' form with fields for Code, Name, Credit, and Department, followed by a 'Save Course' button. To the right of this form is a 'Course Catalog' table listing ten courses with columns for ID, Code, Name, Credit, and Department. Below the catalog is another form for 'Create Course Offering' with fields for Course, Term, Instructor, Start Date, and End Date, followed by a 'Save Offering' button. At the bottom of the panel, there is a link labeled 'Course Offerings'.

ID	Code	Name	Credit	Department
1	CSE101	Introduction to Programming	6	Computer Engineering
2	CSE102	Data Structures	6	Computer Engineering
3	CSE201	Database Management Systems	5	Software Engineering
4	CSE202	Operating Systems	5	Computer Engineering
5	CSE203	Algorithms	6	Computer Engineering
6	CSE204	Web Development	5	Software Engineering
7	CSE205	Computer Networks	5	Computer Engineering
8	CSE301	Machine Learning	6	Computer Engineering
9	CSE302	Artificial Intelligence	6	Software Engineering
10	CSE303	Mobile App Development	5	Software Engineering

4. Registration Panel

This panel registers students into available course offerings. It enforces uniqueness — the same student cannot register for the same offering twice — and stores the registration date and status. The lower list displays all registration records for monitoring.

The screenshot shows a Windows application window titled "Student Exam System". The menu bar includes "Students", "Instructors", "Courses & Offerings", "Registration", and "Exams & Grades". The "Registration" tab is selected. A sub-menu titled "Register Student to Offering" is open, containing fields for "Student" (dropdown) and "Offering" (dropdown), and a "Enroll" button. Below this, a table titled "Registrations" lists 18 entries. The columns are "Reg ID", "Student", "Offering", "Reg Date", and "Status". The data is as follows:

Reg ID	Student	Offering	Reg Date	Status
1	Mehmet Koyuncu	1 - CSE101 - Introduction to Programming (Fall 2023)	2023-09-05	registered
2	Mustafa Oz	1 - CSE101 - Introduction to Programming (Fall 2023)	2023-09-06	registered
3	Ahmet Demir	1 - CSE101 - Introduction to Programming (Fall 2023)	2023-09-06	registered
4	Mehmet Koyuncu	2 - CSE102 - Data Structures (Fall 2023)	2023-09-07	registered
5	Ayse Yilmaz	2 - CSE102 - Data Structures (Fall 2023)	2023-09-07	registered
6	Fatma Kara	2 - CSE102 - Data Structures (Fall 2023)	2023-09-07	registered
7	Elif Aydin	3 - CSE201 - Database Management Systems (Spring 2024)	2024-02-10	registered
8	Hakan Celik	3 - CSE201 - Database Management Systems (Spring 2024)	2024-02-11	registered
9	Zeynep Er	4 - CSE202 - Operating Systems (Spring 2024)	2024-02-11	registered
10	Baran Ucar	4 - CSE202 - Operating Systems (Spring 2024)	2024-02-12	registered
11	Cagla Ari	5 - CSE203 - Algorithms (Fall 2023)	2023-09-10	registered
12	Berk Atay	6 - CSE204 - Web Development (Spring 2024)	2024-02-15	registered
13	Deniz Can	6 - CSE204 - Web Development (Spring 2024)	2024-02-16	registered
14	Mehmet Koyuncu	7 - CSE205 - Computer Networks (Fall 2024)	2024-09-05	registered
15	Mustafa Oz	7 - CSE205 - Computer Networks (Fall 2024)	2024-09-05	registered
16	Hakan Celik	2 - CSE102 - Data Structures (Fall 2023)	2026-01-05	registered
17	Mehmet Koyuncu	10 - CSE303 - Mobile App Development (Spring 2023)	2026-01-05	registered
18	Elif Aydin	10 - CSE303 - Mobile App Development (Spring 2023)	2026-01-05	registered

5. Exams & Grades Panel

This panel manages the entire assessment workflow.

Exams can be defined with type, date, and weight.

Grades are entered per student, and the system automatically calculates weighted averages and determines pass/fail status.

Constraint rules ensure that exam weights total exactly 100 and grades remain between 0 and 100.

The screenshot shows a Windows application window titled "Student Exam System". The menu bar includes "Students", "Instructors", "Courses & Offerings", "Registration", "Exams & Grades", and a logo icon. The main area is divided into three sections:

- Define Exam:** Contains fields for "Offering:" (dropdown), "Exam Type (midterm/final/quiz):" (dropdown), "Exam Date (YYYY-MM-DD):" (text input), "Weight (0-100):" (text input), and a "Save Exam" button.
- Enter Grade:** Contains fields for "Exam:" (dropdown), "Student:" (dropdown), "Score (0-100):" (text input), and a "Save Grade" button.
- Course Performance Report (Weighted Average):** A table with columns "Offering:" (dropdown), "Student", "Weighted Avg", and "Pass/Fail". The "Offering:" dropdown is currently empty.