Project Overview:

This project involves creating a database for managing student courses. The database will have tables for students, courses, enrollments, and instructors. Students will learn and practice SQL queries covering various topics such as selection, filtering, aggregation, joins, and subqueries.

1. Database Setup

```
--Create a database named StudentCourseManagement.
CREATE DATABASE StudentCourseManagement;
USE StudentCourseManagement;
```

2. Table Creation

```
--Table Creation
CREATE TABLE Students(
    student_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    date_of_birth DATE
);

CREATE TABLE Instructors (
    instructor_id INT IDENTITY(1,1) PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);

CREATE TABLE Courses(
    course_id INT IDENTITY(1,1) PRIMARY KEY,
    course_name VARCHAR(100),
    course_description TEXT,
    instructor_id INT,
    FOREIGN KEY (instructor_id) REFERENCES Instructors(instructor_id)
);

CREATE TABLE Enrollments (
    enrollment_id INT IDENTITY(1,1) PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

3. **Insert at least 10 students, 5 courses, 3 instructors, and 15 enrollments.**

```sql
--Insert at least 10 students, 5 courses, 3 instructors, and 15 enrollments.

INSERT INTO Students (first_name, last_name, email, date_of_birth) VALUES
('Ahmed', 'Nasser', 'ahmed.nasser@alexu.com', '2001-05-12'),
('Fatima', 'Saeed', 'fatima.saeed@alexu.com', '2003-11-24'),
('Omar', 'Khaled', 'omar.khaled@alexu.com', '2002-02-18'),
('Sara', 'Hassan', 'sara.hassan@alexu.com', '2004-07-29'),
('Youssef', 'Ali', 'youssef.ali@alexu.com', '2001-09-15'),
('Mariam', 'Adel', 'mariam.adel@alexu.com', '2003-04-07'),
('Mohamed', 'Fathy', 'mohamed.fathy@alexu.com', '2002-12-10'),
('Noor', 'Ibrahim', 'noor.ibrahim@alexu.com', '2005-01-19'),
('Hala', 'Mostafa', 'hala.mostafa@alexu.com', '2004-06-03'),
('Rami', 'Ziad', 'rami.ziad@alexu.com', '2001-08-27');

INSERT INTO Instructors (first_name, last_name, email) VALUES
('Ahmed', 'Shawky', 'ahmed.shawky@alexu.edu.eg'),
('Hoda', 'Hassan', 'hoda.hassan@alexu.edu.eg'),
('Aymen', 'Khalil', 'aymen.khalil@alexu.edu.eg');


INSERT INTO Courses (course_name, course_description, instructor_id) VALUES
('Introduction to Programming', 'Learn the basics of programming with Python.', 1),
('Data Structures and Algorithms', 'An in-depth study of data structures and
  algorithms.', 2),
('Web Development', 'Building websites and web applications with HTML, CSS, and
  JavaScript.', 3),
('Database Management Systems', 'Understanding and managing relational databases.',
  3),
('Machine Learning', 'Introduction to machine learning concepts and techniques.', 1);

INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES
(4, 3, '2023-10-15'),
(2, 2, '2023-09-20'),
(3, 4, '2023-09-10'),
(4, 3, '2023-10-25'),
(5, 1, '2023-10-14'),
(6, 5, '2023-09-30'),
(7, 2, '2023-09-22'),
(8, 3, '2023-09-18'),
(9, 1, '2023-09-05'),
(10, 5, '2023-10-01'),
(4, 2, '2023-09-01'),
(10, 4, '2023-10-01'),
(9, 3, '2023-10-01'),
(4, 5, '2023-10-01'),
(2, 1, '2023-10-05'),
(6, 1, '2023-10-06'),
(8, 1, '2023-10-07'),
(9, 1, '2023-10-08'),
(10, 1, '2023-10-09');
```

4. Basic Queries

```sql
--Select all students.
SELECT *
FROM Students;

--Select all courses.
SELECT *
FROM Courses;

--Select all enrollments with student names and course names.
SELECT
    e.enrollment_id,
    s.first_name +' '+ s.last_name AS student_name,
    c.course_name

FROM Enrollments AS e
JOIN Students AS s ON e.student_id = s.student_id
JOIN Courses AS c ON e.course_id = c.course_id;
```

5. Advanced Queries

```sql
--Select students who enrolled in a specific course
SELECT
    s.student_id,
    s.first_name,
    s.last_name
FROM Students AS s
JOIN Enrollments AS e ON s.student_id = e.student_id
WHERE e.course_id = 3;

--Select courses with more than 5 students
SELECT
    c.course_id,
    c.course_name,
    COUNT(e.student_id) AS number_of_students
FROM Courses AS c
JOIN Enrollments AS e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
HAVING COUNT(e.student_id) > 5;

--Update a student's email
UPDATE Students
SET email = 'updated.email@alexu.com'
WHERE student_id = 4;

--Delete a course that no students are enrolled in
DELETE FROM Courses
WHERE course_id NOT IN (
    SELECT DISTINCT course_id
    FROM Enrollments
);
```

```sql
--Calculate the average age of students
SELECT
    AVG(
        DATEDIFF(
            YEAR, date_of_birth, GETDATE()
                )
        ) AS students_average_age
FROM Students;

--Find the course with the maximum enrollments
SELECT
    TOP 1 c.course_id,
    c.course_name,
    COUNT(e.student_id) AS number_of_enrollments
FROM Courses AS c
JOIN Enrollments AS e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name
ORDER BY number_of_enrollments DESC;



--List courses along with the number of students enrolled (use GROUP BY)
SELECT
    c.course_id,
    c.course_name,
    COUNT(e.student_id) AS number_of_students
FROM Courses AS c
JOIN Enrollments AS e ON c.course_id = e.course_id
GROUP BY c.course_id, c.course_name;
```

6. Join Queries

```sql
--Select all students with their enrolled courses (use JOIN).
SELECT
    s.first_name,
    s.last_name,
    c.course_name
FROM Students AS s
JOIN Enrollments AS e ON s.student_id = e.student_id
JOIN Courses AS c ON e.course_id = c.course_id

--List all instructors and their courses
SELECT
    i.first_name,
    i.last_name,
    c.course_name
FROM Instructors AS i
JOIN Courses AS c ON i.instructor_id = c.instructor_id
```

```sql
--Find students who are not enrolled in any course
SELECT
    s.student_id,
    s.first_name,
    s.last_name
FROM Students AS s
LEFT JOIN Enrollments AS e ON s.student_id = e.student_id
WHERE e.enrollment_id IS NULL
```

7. Subqueries and Set Operations

```sql
--Select students enrolled in more than one course.
SELECT
    s.student_id,
    s.first_name,
    s.last_name,
    COUNT(e.course_id) AS course_count
FROM Students AS s
JOIN Enrollments AS e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
HAVING COUNT(e.course_id) > 1
ORDER BY course_count DESC;


--Find courses taught by a specific instructor.
SELECT
    i.first_name + ' ' + i.last_name AS instructor_name,
    c.course_id,
    c.course_name,
    c.course_description
FROM Courses AS c
JOIN Instructors AS i ON c.instructor_id = i.instructor_id
WHERE c.instructor_id = 2;



--Select the top 3 students with the most enrollments.
SELECT
    TOP 3 s.student_id,
    s.first_name,
    s.last_name,
    COUNT(e.course_id) AS enrollment_count
FROM Students AS s
JOIN Enrollments AS e ON s.student_id = e.student_id
GROUP BY s.student_id, s.first_name, s.last_name
ORDER BY enrollment_count DESC;
```

```sql
--Use UNION to combine results of two different SELECT queries.
SELECT first_name
FROM Students
WHERE date_of_birth < '2001-12-31'
UNION
SELECT course_name
FROM Courses
WHERE course_id = 2;
```

8. Functions and Stored Procedures

```sql
--Create a stored procedure to add a new student.
CREATE PROCEDURE AddStudent
    @first_name VARCHAR(50),
    @last_name VARCHAR(50),
    @email VARCHAR(100),
    @date_of_birth DATE
AS
BEGIN
    INSERT INTO Students (first_name, last_name, email, date_of_birth)
    VALUES (@first_name, @last_name, @email, @date_of_birth);
END;

--Create a function to calculate the age of a student based on their date of birth.

CREATE FUNCTION CalculateAge
(
    @date_of_birth DATE
)
RETURNS INT
AS
BEGIN
    DECLARE @age INT;

    SET @age = DATEDIFF(YEAR, @date_of_birth, GETDATE()) -
                CASE
                    WHEN MONTH(@date_of_birth) > MONTH(GETDATE())
                         OR (MONTH(@date_of_birth) = MONTH(GETDATE())
                         AND DAY(@date_of_birth) > DAY(GETDATE()))
                    THEN 1
                    ELSE 0
                END;

    RETURN @age;
END;
```

9. Aggregate Functions and Grouping

```sql
--Calculate the total number of students.
SELECT COUNT(DISTINCT student_id) AS total_students
FROM students;

--Calculate the average, minimum, and maximum number of enrollments per course.
SELECT
    AVG(number_of_students) AS average_enrollments,
    MIN(number_of_students) AS minimum_enrollments,
    MAX(number_of_students) AS maximum_enrollments
FROM
    (SELECT
        c.course_id,
        c.course_name,
        COUNT(e.student_id) AS number_of_students
    FROM Courses AS c
    JOIN Enrollments AS e ON c.course_id = e.course_id
    GROUP BY c.course_id, c.course_name)
    AS enrollments_per_course;
```

10. Additional Tasks

```sql
--Create aliases for complex column names.
SELECT
    course_id AS CourseID,
    course_name AS CourseName
FROM Courses;


--Use CASE to categorize students based on their age.
SELECT
    first_name,
    last_name,
    CASE
        WHEN DATEDIFF(YEAR, date_of_birth, GETDATE()) < 18 THEN 'Younger than 18'
        WHEN DATEDIFF(YEAR, date_of_birth, GETDATE()) BETWEEN 18 AND 24 THEN '18 - 24
          '
        WHEN DATEDIFF(YEAR, date_of_birth, GETDATE()) BETWEEN 25 AND 34 THEN 'Bigger
          than  25'
    END AS AgeCategory
FROM Students;



--Use EXISTS to find courses with at least one enrolled student.
SELECT
    course_id,
    course_name
FROM Courses AS c
WHERE EXISTS (
    SELECT 1
    FROM Enrollments AS e
    WHERE e.course_id = c.course_id
);
```