University Management System

Part 1: Database Design

Define Entities and Relationships

- **Departments** *-- 1-to-many with Students, Faculty, Courses*

DepartmentID INT PRIMARY KEY,
DepartmentName VARCHAR(100),
HeadOfDepartment VARCHAR(100)

- **Students** *-- many-to-many with Courses through Enrollments*

StudentID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
DOB DATE,
DepartmentID INT,
FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

- **Courses** *--many-to-many with Students through Enrollments*

CourseID INT PRIMARY KEY,
CourseName VARCHAR (50),
Credits INT,
DepartmentID INT,
FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

- **Faculty**

FacultyID INT PRIMARY KEY,
FacultyName VARCHAR(100),
Designation VARCHAR(100),
DepartmentID INT,
FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

- **Enrollments** *--junction table for Students and Courses*

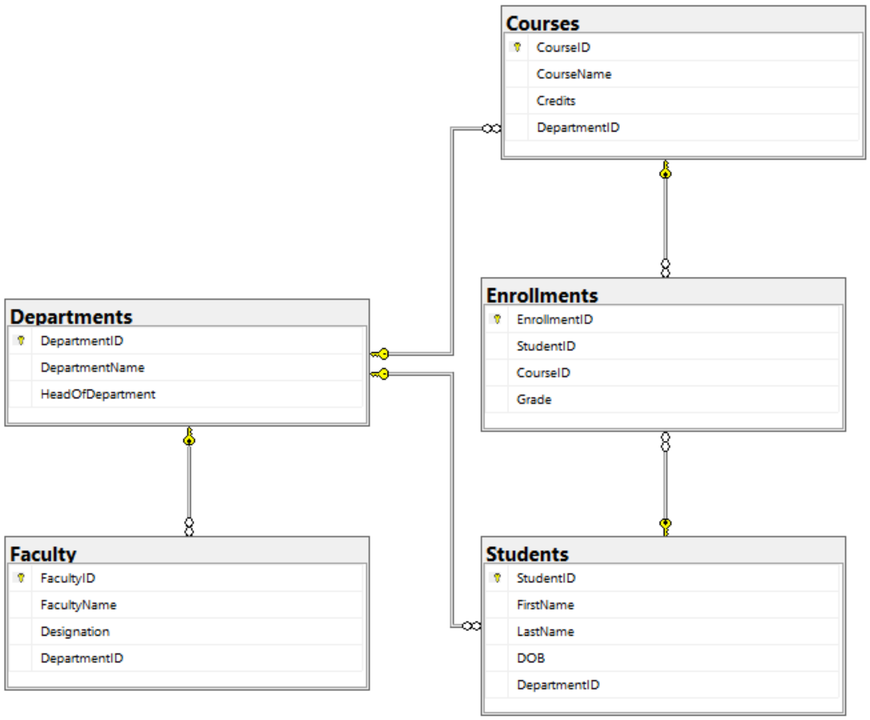
EnrollmentID INT PRIMARY KEY,
StudentID INT,
CourseID INT,
Grade DECIMAL(10,2),
FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

- **Library**

BookID INT PRIMARY KEY,

Title VARCHAR(100),
Author VARCHAR(50),
BorrowerID INT,
DueDate DATE,

Entity-Relationship Diagram



Part 2: Create the Database

1. Database Creation:
    - Create a database named UniversityManagement.
2. Define Tables:
    - Create all tables as per the entity definitions, with appropriate constraints:
        - Primary and foreign keys.
        - Unique, NOT NULL, and CHECK constraints where applicable.

3. Normalize:

- Ensure the database satisfies 3rd Normal Form (3NF).

The database design already satisfies Third Normal Form (3NF) because:

- ✓ All tables have primary keys
- ✓ All non-key attributes are fully dependent on the primary key
- ✓ There are no transitive dependencies (all non-key attributes depend only on the primary key)
- ✓ All foreign key relationships are properly established

4. Indexes:

- **Create clustered and nonclustered indexes for query optimization.**

- ✓ **All Clustered Index are already create with primary key**
- ✓ **Nonclustered Indexes:**

```
-- Create indexes for optimization
CREATE NONCLUSTERED INDEX idx_students_lastname ON Students(LastName);

-- Non-clustered index on Courses' CourseName
CREATE NONCLUSTERED INDEX idx_courses_name ON Courses(CourseName);

-- Composite index for join optimization on Enrollments
CREATE INDEX idx_enrollments_student_course ON Enrollments(StudentID, CourseID);
```

Part 3: Populate the Database

1. Insert Data:

   - Populate each table with at least 10 records of realistic sample data.
   - Ensure meaningful relationships (e.g., a student belongs to a department and is enrolled in courses offered by that department).

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Departments (DepartmentName, HeadOfDepartment) VALUES
('Computer Science', 'Dr. Alice Johnson'),
('Electrical Engineering', 'Dr. Bob Smith'),
('Mechanical Engineering', 'Dr. Carol Lee'),
('Mathematics', 'Dr. Dan Brown'),
('Business Administration', 'Dr. Eva Green'),
('Physics', 'Dr. Frank White'),
('Civil Engineering', 'Dr. Grace Liu'),
('Environmental Science', 'Dr. Henry Kim'),
('Economics', 'Dr. Irene Wang'),
('Chemistry', 'Dr. Jack Black');
SELECT * FROM Departments;
```

100 %

Results | Messages

	DepartmentID	DepartmentName	HeadOfDepartment
1	1	Computer Science	Dr. Alice Johnson
2	2	Electrical Engineering	Dr. Bob Smith
3	3	Mechanical Engineering	Dr. Carol Lee
4	4	Mathematics	Dr. Dan Brown
5	5	Business Administration	Dr. Eva Green
6	6	Physics	Dr. Frank White
7	7	Civil Engineering	Dr. Grace Liu
8	8	Environmental Science	Dr. Henry Kim
9	9	Economics	Dr. Irene Wang
10	10	Chemistry	Dr. Jack Black

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Students (StudentID, FirstName, LastName, DOB, DepartmentID) VALUES
(101, 'John', 'Doe', '2001-05-21', 1),
(102, 'Emma', 'Smith', '2000-11-10', 2),
(103, 'Liam', 'Brown', '2002-03-14', 3),
(104, 'Olivia', 'Jones', '2001-07-23', 1),
(105, 'Noah', 'Garcia', '2000-02-01', 4),
(106, 'Ava', 'Martinez', '2002-09-15', 5),
(107, 'William', 'Davis', '1999-12-30', 6),
(108, 'Sophia', 'Lopez', '2003-04-10', 7),
(109, 'James', 'Wilson', '2001-06-28', 8),
(110, 'Isabella', 'Anderson', '2000-08-05', 9);
SELECT * FROM Students
```

100 %

Results | Messages

	StudentID	FirstName	LastName	DOB	DepartmentID
1	101	John	Doe	2001-05-21	1
2	102	Emma	Smith	2000-11-10	2
3	103	Liam	Brown	2002-03-14	3
4	104	Olivia	Jones	2001-07-23	1
5	105	Noah	Garcia	2000-02-01	4
6	106	Ava	Martinez	2002-09-15	5
7	107	William	Davis	1999-12-30	6
8	108	Sophia	Lopez	2003-04-10	7
9	109	James	Wilson	2001-06-28	8
10	110	Isabella	Anderson	2000-08-05	9

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Courses (CourseID, CourseName, Credits, DepartmentID) VALUES
(301, 'Intro to Programming', 3, 1),
(302, 'Data Structures', 4, 1),
(303, 'Circuit Theory', 3, 2),
(304, 'Thermodynamics', 3, 3),
(305, 'Linear Algebra', 4, 4),
(306, 'Marketing Principles', 3, 5),
(307, 'Quantum Physics', 4, 6),
(308, 'Structural Analysis', 3, 7),
(309, 'Climate Change', 2, 8),
(310, 'Microeconomics', 3, 9);
SELECT * FROM Courses;
```

100 %

Results Messages

	CourseID	CourseName	Credits	DepartmentID
1	301	Intro to Programming	3	1
2	302	Data Structures	4	1
3	303	Circuit Theory	3	2
4	304	Thermodynamics	3	3
5	305	Linear Algebra	4	4
6	306	Marketing Principles	3	5
7	307	Quantum Physics	4	6
8	308	Structural Analysis	3	7
9	309	Climate Change	2	8
10	310	Microeconomics	3	9

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Faculty (FacultyID, FacultyName, Designation, DepartmentID) VALUES
(201, 'Johnson', 'Professor', 1),
(202, 'Smith', 'Lecturer', 2),
(203, 'Lee', 'Associate Professor', 3),
(204, 'Brown', 'Lecturer', 4),
(205, 'Green', 'Professor', 5),
(206, 'White', 'Assistant Professor', 6),
(207, 'Liu', 'Lecturer', 7),
(208, 'Kim', 'Professor', 8),
(209, 'Wang', 'Lecturer', 9),
(210, 'Black', 'Assistant Professor', 10);
SELECT * FROM Faculty;
```

100 %

Results Messages

	FacultyID	FacultyName	Designation	DepartmentID
1	201	Johnson	Professor	1
2	202	Smith	Lecturer	2
3	203	Lee	Associate Professor	3
4	204	Brown	Lecturer	4
5	205	Green	Professor	5
6	206	White	Assistant Professor	6
7	207	Liu	Lecturer	7
8	208	Kim	Professor	8
9	209	Wang	Lecturer	9
10	210	Black	Assistant Professor	10

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, Grade) VALUES
(401, 101, 301, 3.5),
(402, 102, 303, 3.7),
(403, 103, 304, 3.2),
(404, 104, 302, 3.9),
(405, 105, 305, 3.6),
(406, 106, 306, 3.1),
(407, 107, 307, 2.8),
(408, 108, 308, 3.4),
(409, 109, 309, 3.3),
(410, 110, 310, 3.0);
SELECT * FROM Enrollments;
```

100 %

Results Messages

	EnrollmentID	StudentID	CourseID	Grade
1	401	101	301	3.50
2	402	102	303	3.70
3	403	103	304	3.20
4	404	104	302	3.90
5	405	105	305	3.60
6	406	106	306	3.10
7	407	107	307	2.80
8	408	108	308	3.40
9	409	109	309	3.30
10	410	110	310	3.00

Final Project.sql -...P-4F66F8T\user (62))*

```sql
INSERT INTO Library (BookID, Title, Author, BorrowerID, DueDate) VALUES
(501, 'Database Systems', 'Elmasri & Navathe', 101, '2025-04-10'),
(502, 'Clean Code', 'Robert C. Martin', 104, '2025-04-08'),
(503, 'Digital Circuits', 'Morris Mano', 102, '2025-04-05'),
(504, 'Thermal Engineering', 'R.K. Rajput', 103, '2025-04-12'),
(505, 'Business Basics', 'Kotler', 106, '2025-04-11'),
(506, 'Quantum Mechanics', 'Griffiths', 107, NULL),
(507, 'Civil Engineering Materials', 'Neville', 108, NULL),
(508, 'Environmental Policies', 'Jane Goodall', 109, '2025-04-09'),
(509, 'Economic Theory', 'Samuelson', 110, NULL),
(510, 'Organic Chemistry', 'Solomons', NULL, NULL);
SELECT * FROM Library;
```

100 %

Results | Messages

	BookID	Title	Author	BorrowerID	DueDate
1	501	Database Systems	Elmasri & Navathe	101	2025-04-10
2	502	Clean Code	Robert C. Martin	104	2025-04-08
3	503	Digital Circuits	Morris Mano	102	2025-04-05
4	504	Thermal Engineering	R.K. Rajput	103	2025-04-12
5	505	Business Basics	Kotler	106	2025-04-11
6	506	Quantum Mechanics	Griffiths	107	NULL
7	507	Civil Engineering Materials	Neville	108	NULL
8	508	Environmental Policies	Jane Goodall	109	2025-04-09
9	509	Economic Theory	Samuelson	110	NULL
10	510	Organic Chemistry	Solomons	NULL	NULL

Part 4: Data Manipulation

1. Basic CRUD Operations:
   - Write SQL queries to:
     - Add a new student, course, and faculty member.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- Add a new student
INSERT INTO Students (StudentID, FirstName, LastName, DOB, DepartmentID)
VALUES (111, 'Mia', 'Turner', '2002-10-01', 1);

-- Add a new course
INSERT INTO Courses (CourseID, CourseName, Credits, DepartmentID)
VALUES (311, 'Web Development', 3, 1);

-- Add a new faculty member
INSERT INTO Faculty (FacultyID, FacultyName, Designation, DepartmentID)
VALUES (211, 'Parker', 'Lecturer', 1);
```

     - Update a student's department and a course's credits.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- Update a student's department
UPDATE Students SET DepartmentID = 2 WHERE StudentID = 111;

-- Update a course's credits
UPDATE Courses SET Credits = 4 WHERE CourseID = 311;
```

     - Delete a faculty member.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- Delete a faculty member
DELETE FROM Faculty WHERE FacultyID = 210;
```

2. Complex Queries:

   - Retrieve:

     - The list of all students enrolled in a specific course.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- 2.Complex Queries:
-- The list of all students enrolled in a specific course.
SELECT S.FirstName, S.LastName, C.CourseName, E.Grade
FROM Enrollments E
JOIN Students S ON E.StudentID = S.StudentID
JOIN Courses C ON E.CourseID = C.CourseID
WHERE C.CourseID = 301;
```

100 %

Results | Messages

	FirstName	LastName	CourseName	Grade
1	John	Doe	Intro to Programming	3.50

     - The department-wise count of students.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- The department-wise count of students.
SELECT D.DepartmentName, COUNT(S.StudentID) AS StudentCount
FROM Departments D
LEFT JOIN Students S ON D.DepartmentID = S.DepartmentID
GROUP BY D.DepartmentName;
```

100 %

Results | Messages

	DepartmentName	StudentCount
1	Computer Science	2
2	Electrical Engineering	2
3	Mechanical Engineering	1
4	Mathematics	1
5	Business Administration	1
6	Physics	1
7	Civil Engineering	1
8	Environmental Science	1
9	Economics	1
10	Chemistry	0

     - The details of books borrowed by students in a specific department.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- The details of books borrowed by students in a specific department
SELECT S.FirstName, S.LastName, D.DepartmentName, L.Title, L.Author, L.DueDate
FROM Library L
JOIN Students S ON L.BorrowerID = S.StudentID
JOIN Departments D ON S.DepartmentID = D.DepartmentID
WHERE D.DepartmentID = 1;
```

100 %

Results | Messages

	FirstName	LastName	DepartmentName	Title	Author	DueDate
1	John	Doe	Computer Science	Database Systems	Elmasri & Navathe	2025-04-10
2	Olivia	Jones	Computer Science	Clean Code	Robert C. Martin	2025-04-08

- Find:
  - The top 3 students with the highest GPA.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- The top 3 students with the highest GPA.
SELECT TOP 3 S.FirstName, S.LastName, AVG(E.Grade) AS GPA
FROM Students S
JOIN Enrollments E ON S.StudentID = E.StudentID
GROUP BY S.StudentID, S.FirstName, S.LastName
ORDER BY GPA DESC;
```

100 %

Results | Messages

	FirstName	LastName	GPA
1	Olivia	Jones	3.900000
2	Emma	Smith	3.700000
3	Noah	Garcia	3.600000

  - The faculty members teaching the most courses.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
SELECT TOP 3 F.FacultyName, COUNT(CF.CourseID) AS TotalCourses
FROM Faculty F
JOIN CourseFaculty CF ON F.FacultyID = CF.FacultyID
GROUP BY F.FacultyID, F.FacultyName
ORDER BY TotalCourses DESC;
```

100 %

Results | Messages

	FacultyName	TotalCourses
1	Brown	1
2	Lee	1
3	Smith	1

Part 5: Views and Stored Procedures

1. Create Views:
   - Create a view to show student enrollments with course details (e.g., StudentName, CourseName, Grade).

Final Project.sql -...P-4F66F8T\user (62))*

```sql
-- Part 5: Views and Stored Procedures
-- Create a view to show student enrollments with course details (e.g., StudentName, CourseName, Grade).
CREATE VIEW vw_StudentEnrollments AS
SELECT
    S.StudentID,
    CONCAT(S.FirstName, ' ', S.LastName) AS StudentName,
    C.CourseName,
    E.Grade
FROM Enrollments E
JOIN Students S ON E.StudentID = S.StudentID
JOIN Courses C ON E.CourseID = C.CourseID;
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2025-04-06T01:01:01.9466053-04:00
```

- Create a view to display faculty details along with the courses they teach.

```sql
-- Create a view to display faculty details along with the courses they teach.
CREATE VIEW vw_FacultyCourses AS
SELECT
    F.FacultyID,
    F.FacultyName,
    C.CourseName
FROM CourseFaculty CF
JOIN Faculty F ON CF.FacultyID = F.FacultyID
JOIN Courses C ON CF.CourseID = C.CourseID;
```

```
Commands completed successfully.

Completion time: 2025-04-06T01:04:05.0534945-04:00
```

2. Stored Procedures:

- Write a stored procedure to add a new student and enroll them in multiple courses.

```sql
-- Write a stored procedure to add a new student and enroll them in multiple courses.
CREATE PROCEDURE sp_AddStudentAndEnroll
    @StudentID INT,
    @FirstName NVARCHAR(50),
    @LastName NVARCHAR(50),
    @DOB DATE,
    @DepartmentID INT,
    @Course1ID INT,
    @Course2ID INT
AS
BEGIN
    -- Add student
    INSERT INTO Students (StudentID, FirstName, LastName, DOB, DepartmentID)
    VALUES (@StudentID, @FirstName, @LastName, @DOB, @DepartmentID);

    -- Enroll in two courses
    INSERT INTO Enrollments (StudentID, CourseID, Grade)
    VALUES
        (@StudentID, @Course1ID, NULL),
        (@StudentID, @Course2ID, NULL);
END;
```

```
Commands completed successfully.
```

- Write a stored procedure to calculate the average grade for a course.

```sql
-- Write a stored procedure to calculate the average grade for a course
CREATE PROCEDURE sp_AverageGradeForCourse
    @CourseID INT
AS
BEGIN
    SELECT
        C.CourseName,
        AVG(E.Grade) AS AverageGrade
    FROM Enrollments E
    JOIN Courses C ON E.CourseID = C.CourseID
    WHERE E.CourseID = @CourseID
    GROUP BY C.CourseName;
END;
```

```
Commands completed successfully.

Completion time: 2025-04-06T01:48:37.3007229-04:00
```

Part 6: Query Optimization

1. SARGable Queries:
    - Rewrite non-SARGable queries to ensure optimal index usage.
    - Example: Filter students based on GPA or department using indexed columns.

```sql
-- Part 6: Query Optimization
-- Rewrite non-SARGable queries to ensure optimal index usage
SELECT * FROM Students WHERE YEAR(DOB) = 2002;
```

	StudentID	FirstName	LastName	DOB	DepartmentID
1	103	Liam	Brown	2002-03-14	3
2	106	Ava	Martinez	2002-09-15	5
3	111	Mia	Turner	2002-10-01	2
4	150	Noah	Morgan	2002-03-14	2

The above query is not SARGable because the YEAR() function is applied to the DOB column, which prevents SQL from using an index on DOB.

```sql
-- to make it sarable
SELECT * FROM Students
WHERE DOB >= '2002-01-01' AND DOB < '2003-01-01';
```

	StudentID	FirstName	LastName	DOB	DepartmentID
1	103	Liam	Brown	2002-03-14	3
2	106	Ava	Martinez	2002-09-15	5
3	111	Mia	Turner	2002-10-01	2
4	150	Noah	Morgan	2002-03-14	2

Now, SQL can use an index on the DOB column for this query, improving performance.

2. Index Analysis:

- Analyze the performance impact of created indexes on SELECT and JOIN operations.

Indexing is essential for speeding up query performance, especially on **columns used in filtering (WHERE clause)**, **sorting (ORDER BY clause)**, or **joining** (JOIN).

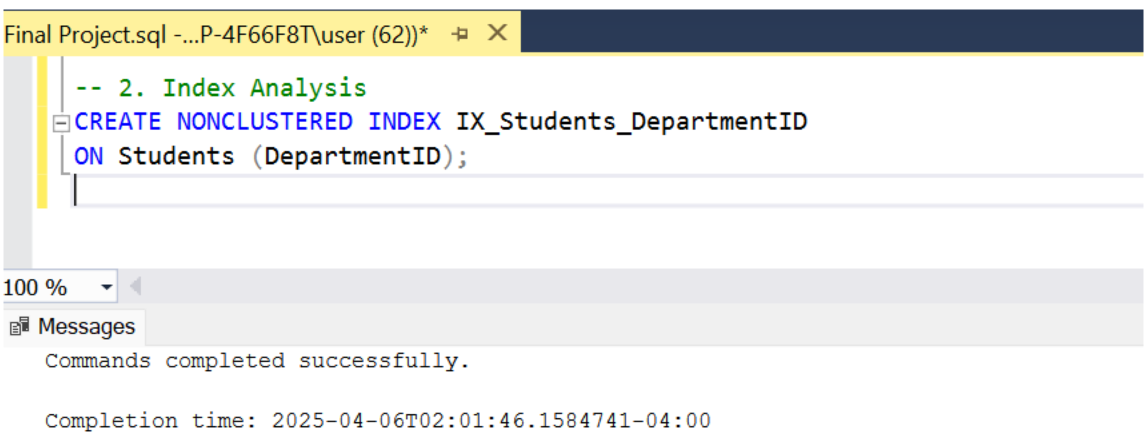
Let's review how to optimize queries by creating the right indexes.

Creating Indexes:

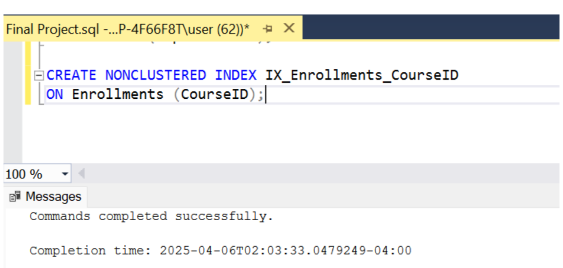
1. **Clustered Index** (Primary key by default, used for sorting).
2. **Non-Clustered Index** (Used for performance improvement in WHERE clauses, JOINs, etc.).

Let's consider creating indexes on commonly searched columns.

Example 1: Create a non-clustered index on DepartmentID in the Students table:



Example 2: Create a non-clustered index on CourseID in the Enrollments table:



This helps speed up queries that filter by DepartmentID or CourseID

Summary:

- We can ensure SARGable queries by avoiding functions on columns that are indexed.
- We can create non-clustered indexes on columns that are frequently used in WHERE, JOIN, and ORDER BY clauses.
- We use SET STATISTICS IO to analyze index usage and optimize query performance.

Part 7: Concurrency and Transactions

1. Simulate Concurrency:
   - Create a scenario where two students try to borrow the same book at the same time.
   - Implement transaction isolation levels to prevent concurrency issues.
2. Transactional Integrity:
   - Ensure that a course cannot be deleted if students are enrolled in it.
   - Implement cascading actions (e.g., ON DELETE SET NULL for Library table borrow records).

Conclusion of Part 7 Completion:

1. We have simulate concurrency by using multiple transactions (or sessions) to perform operations on the same data**. (refer to sql script to dee details of the scrip)**
2. We Do use appropriate **transaction isolation levels** to manage concurrency.
3. Finaly we ensure **transactional integrity** by implementing cascading actions for foreign key constraints to prevent deletion of records when dependencies exist.

Part 8: Reporting and Analytics

1. Generate Reports:
   - Write queries to generate:
     - A list of students with overdue books.

Final Project.sql -...P-4F66F8T\user (62))*

```sql
--Part 8: Reporting and Analytics
-- 1.   Generate Reports:
-- list of students with overdue books.
SELECT s.StudentID, s.FirstName, s.LastName, l.BookID, l.Title, l.DueDate
FROM Library l
JOIN Students s ON l.BorrowerID = s.StudentID
WHERE l.DueDate < GETDATE() AND l.BorrowerID IS NOT NULL;
```

100 %

Results | Messages

	StudentID	FirstName	LastName	BookID	Title	DueDate
1	102	Emma	Smith	503	Digital Circuits	2025-04-05

- A summary report of department-wise course enrollments.

Final Project.sql -...P-4F66F8T\user (62))*

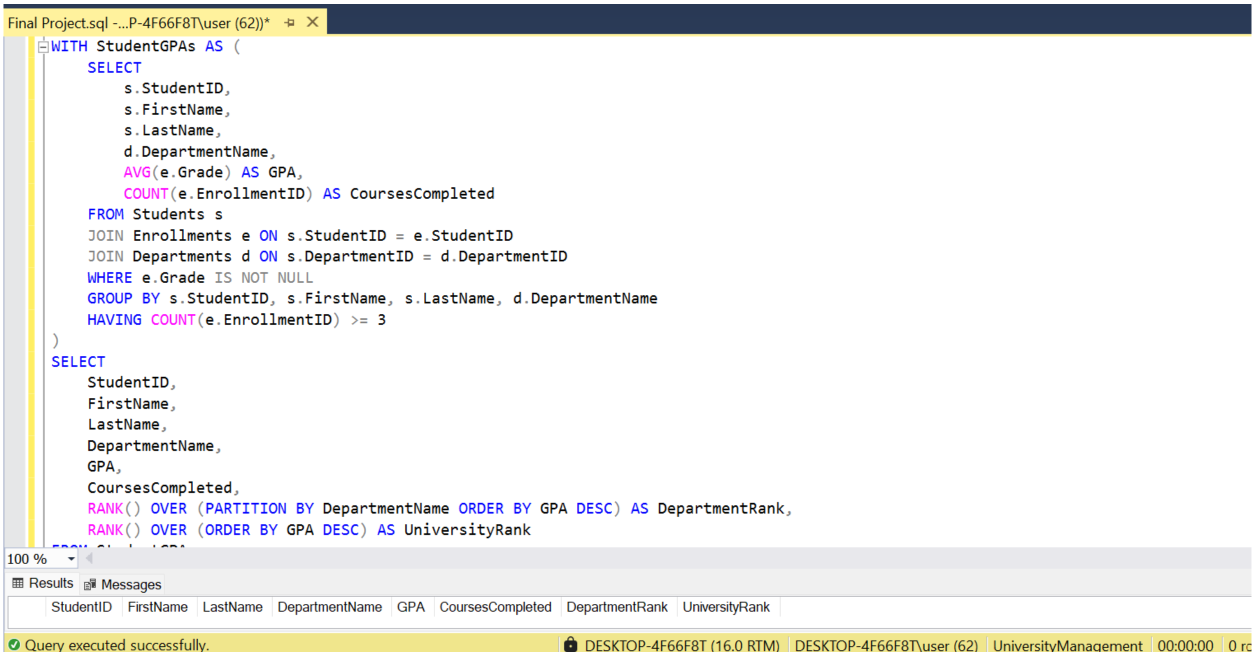
```sql
-- summary report of department-wise course enrollments.
SELECT d.DepartmentName, COUNT(e.StudentID) AS NumberOfEnrollments
FROM Enrollments e
JOIN Students s ON e.StudentID = s.StudentID
JOIN Departments d ON s.DepartmentID = d.DepartmentID
GROUP BY d.DepartmentName
ORDER BY NumberOfEnrollments DESC;
```

100 %

Results | Messages

	DepartmentName	NumberOfEnrollments
1	Computer Science	2
2	Economics	1
3	Electrical Engineering	1
4	Environmental Science	1
5	Mathematics	1
6	Mechanical Engineering	1
7	Physics	1
8	Business Administration	1
9	Civil Engineering	1

- A ranking of students by GPA, grouped by department.



2. Aggregate Functions:

   - Use aggregate functions like SUM, COUNT, AVG, and MAX in your queries.

We did used several aggregate functions, but let's explicitly list them and demonstrate their usage in different contexts:

- **COUNT**: Used to count the number of rows.
  - Example: Counting the number of enrollments per department (in the department-wise summary report).
- **AVG**: Used to calculate the average value.
  - Example: Calculating the GPA average per student (in the student GPA ranking report).
- **SUM**: Used to sum numeric values.
  - Example: You could modify the GPA report to sum the total grades per department instead of averaging them.
- **MAX**: Used to find the maximum value.
  - Example: Finding the highest GPA in each department.

Deliverable

Submit the following:

1. SQL Scripts:

    - Scripts for creating the database, tables, and relationships.
    - Scripts for populating tables with data.
    - CRUD, complex queries, views, and stored procedures.

2. Query Outputs:

    - Screenshots or exported results of all queries and reports.

3. Documentation:

    - A brief report explaining the database design, normalization process, and query optimization techniques used.