**Advanced database project**

**University course enrollment & enrollment system**

|  |  |
| --- | --- |
| Member1 | ندى عماد محمود 22010283 |
| Member2 | شهد حسام زكريا 22010351 |
| Member3 | غادة السيد فرج 22010371 |
| Member4 | علا رجب سعد على 22010367 |

Member1

The database scheme:

Students, courses, course instructors, enrollments, departments, instructors, grades

And relationships:

Students with enrollments (one to many)

Courses with enrollments (one to many)

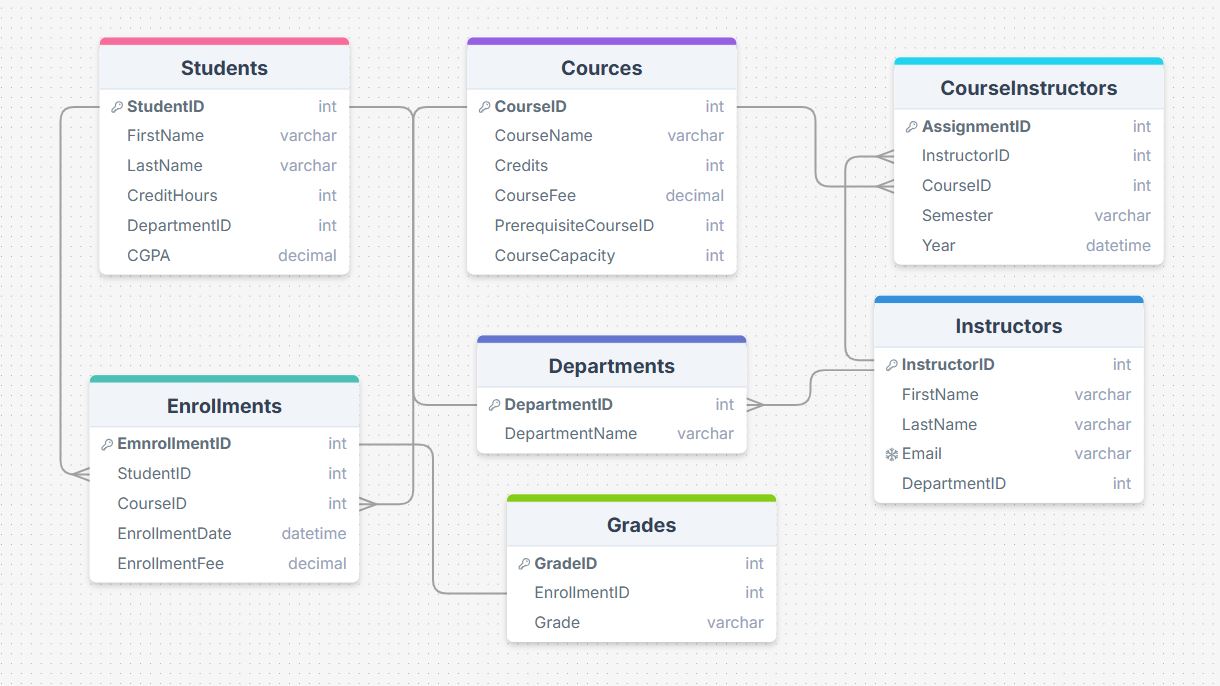
Courses with course instructors (one to many)

Instructors with departments (one to many)

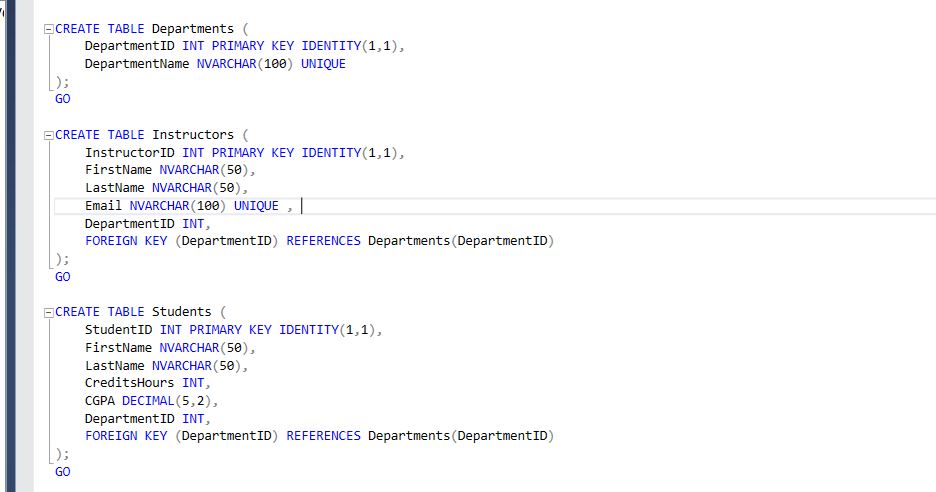
Instructors to course instructors (one to many)

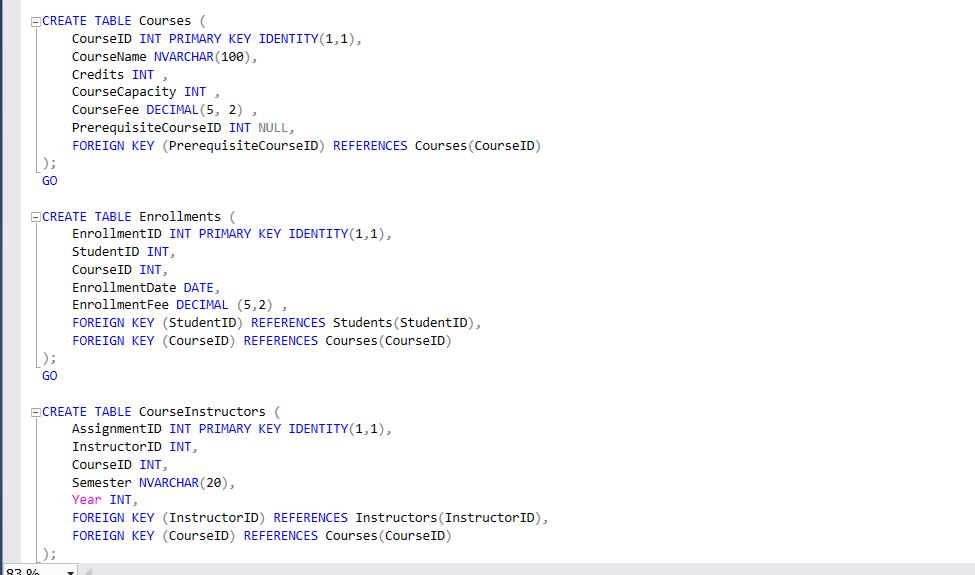
Students with departments (one to one)

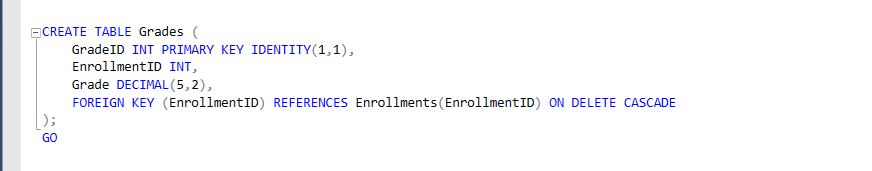
Grades with enrollments (one to one)



**The created tables**

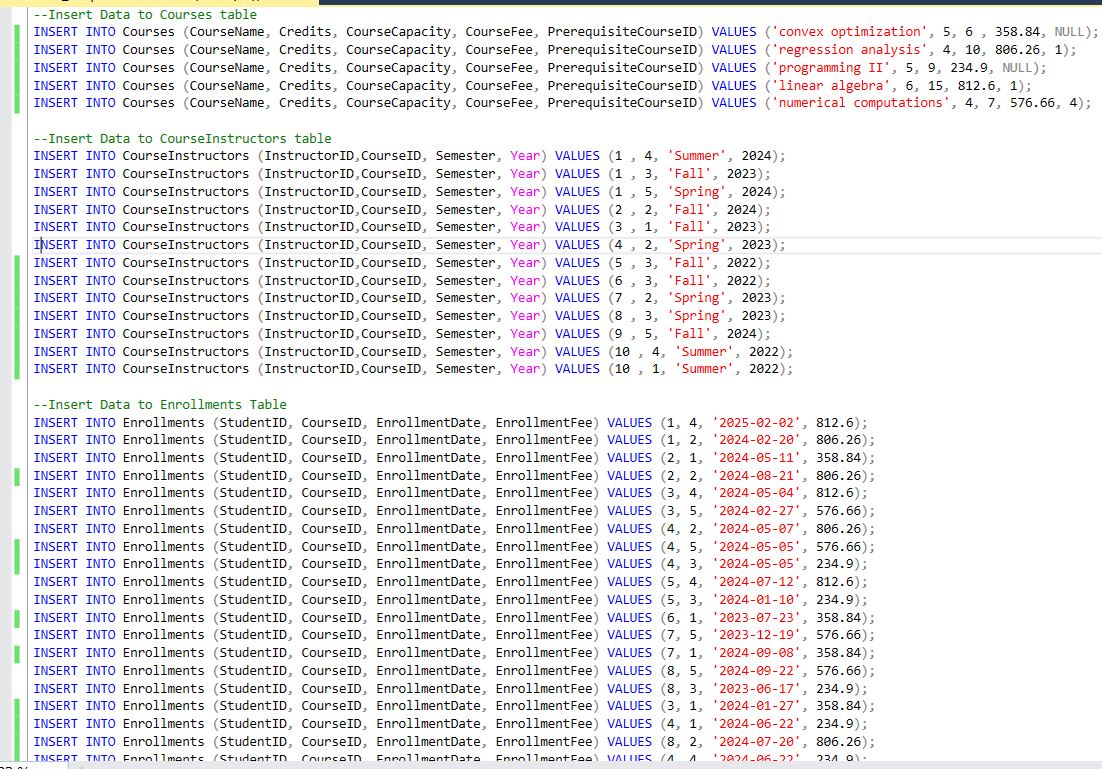




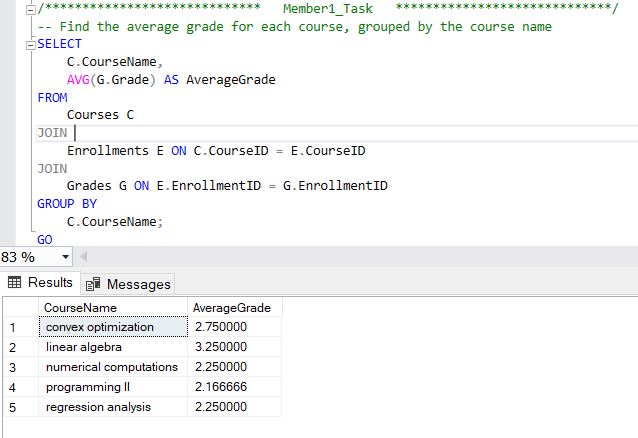


**Populate the tables**

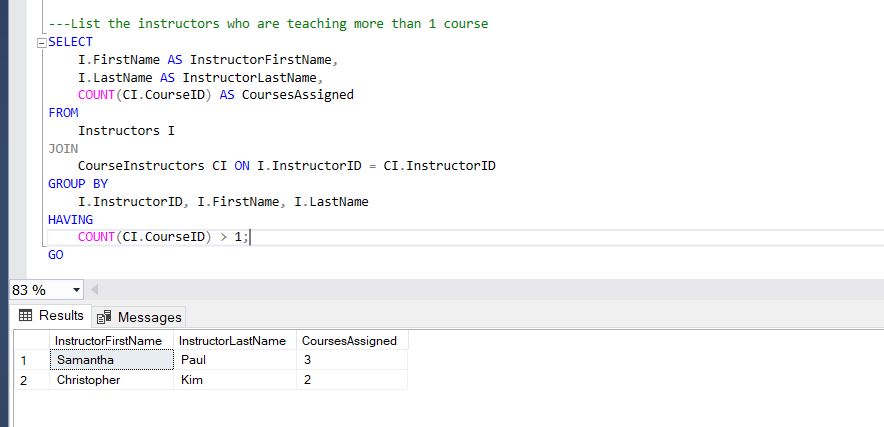




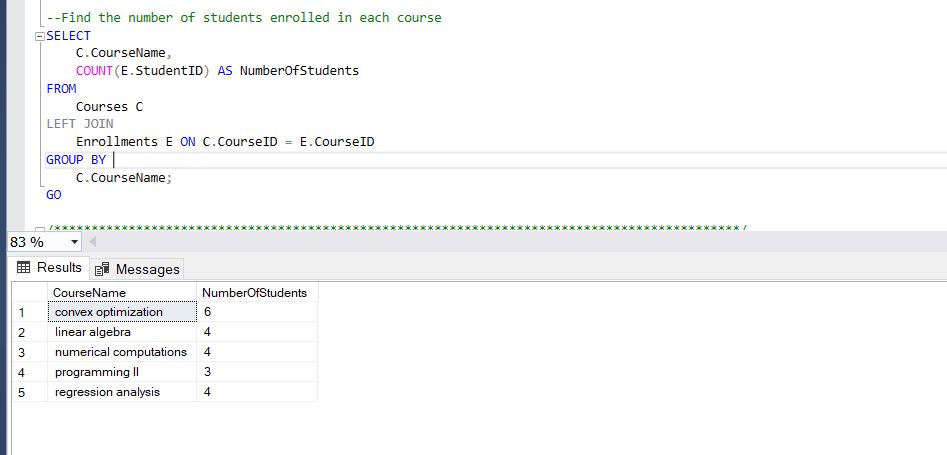
**2) Find the average grade for each course, grouped by the course name**



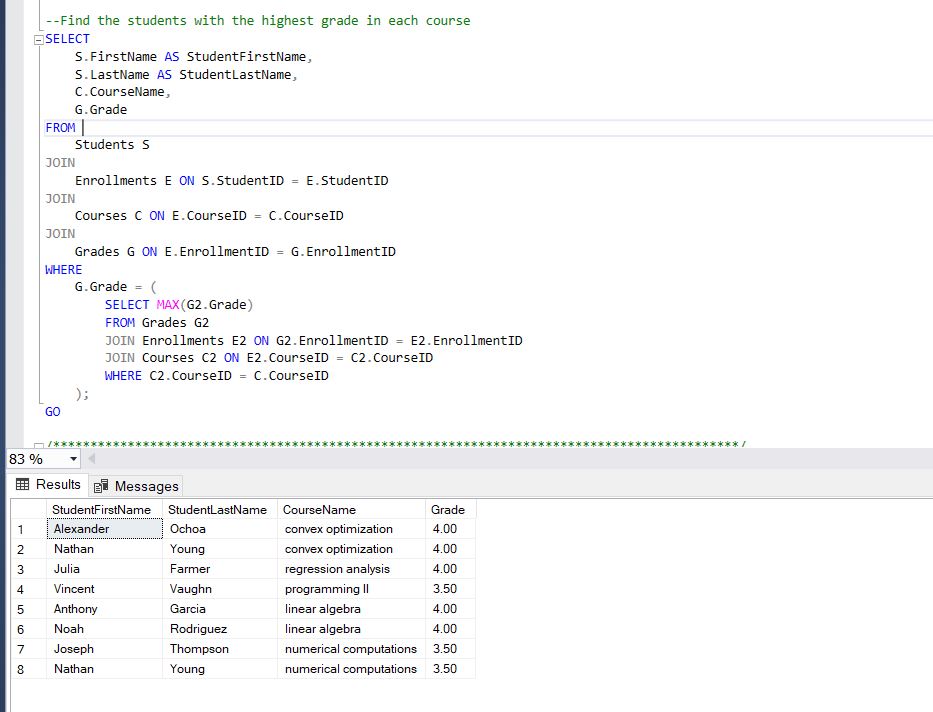
**List the instructors who are teaching more than 1 course**



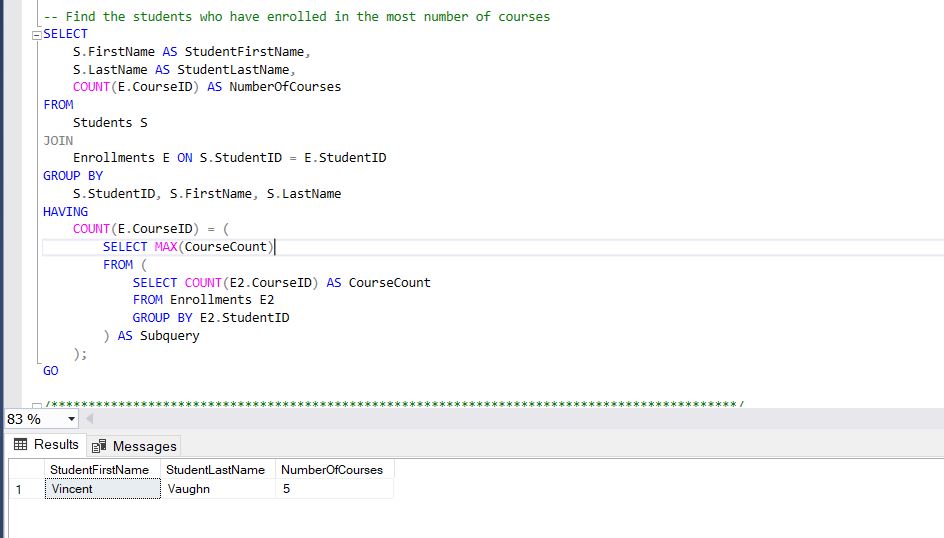
**Find the number of students enrolled in each course**



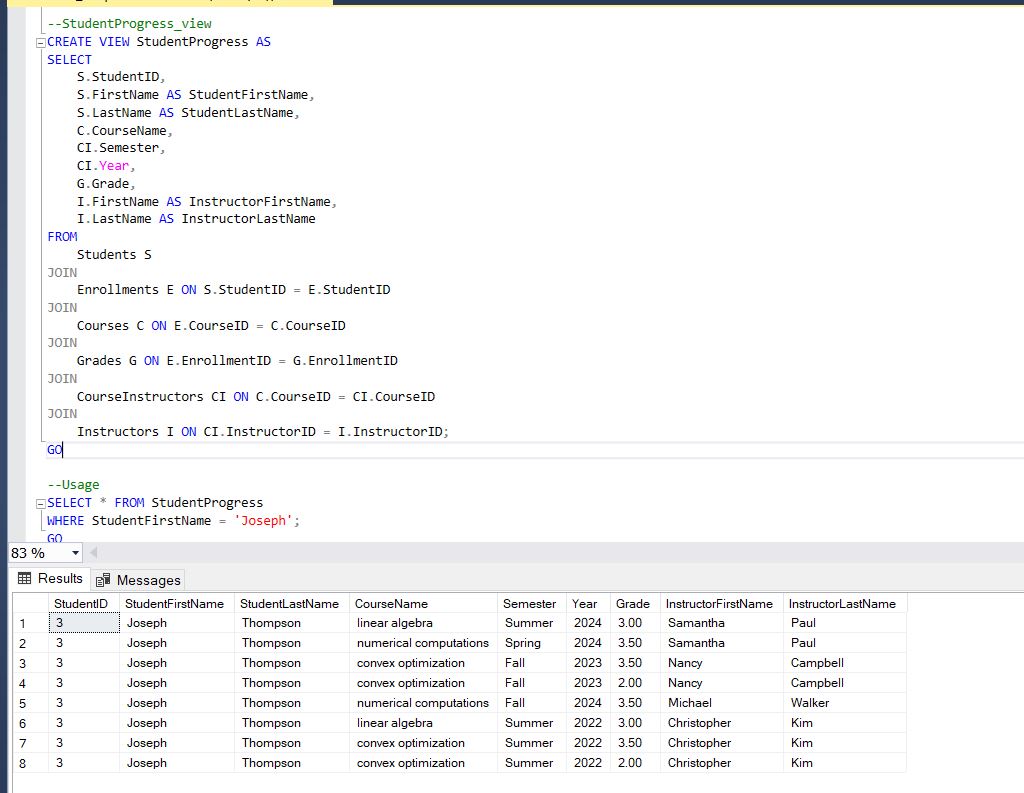
**Find the students with the highest grade in each course (Sub queries)**



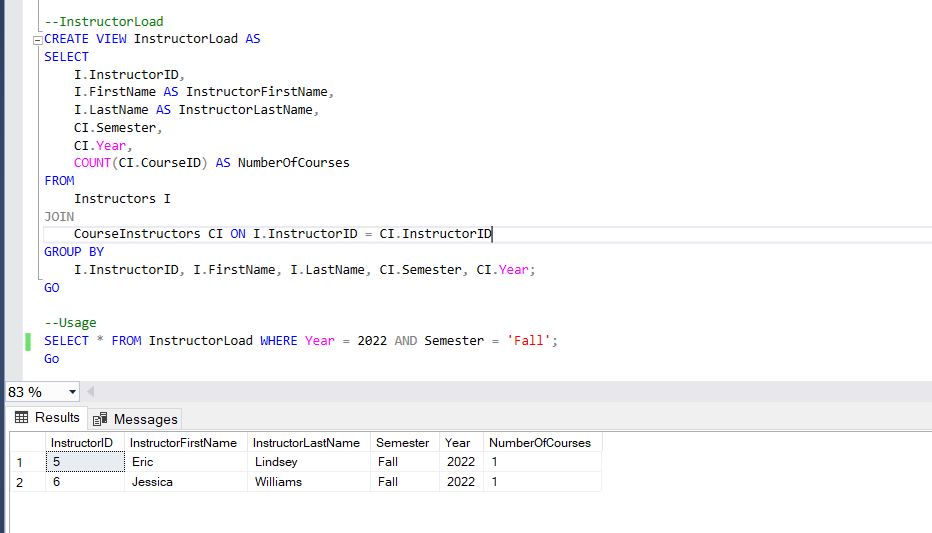
**Find the students who have enrolled in the most number of courses**



**StudentProgress\_view**



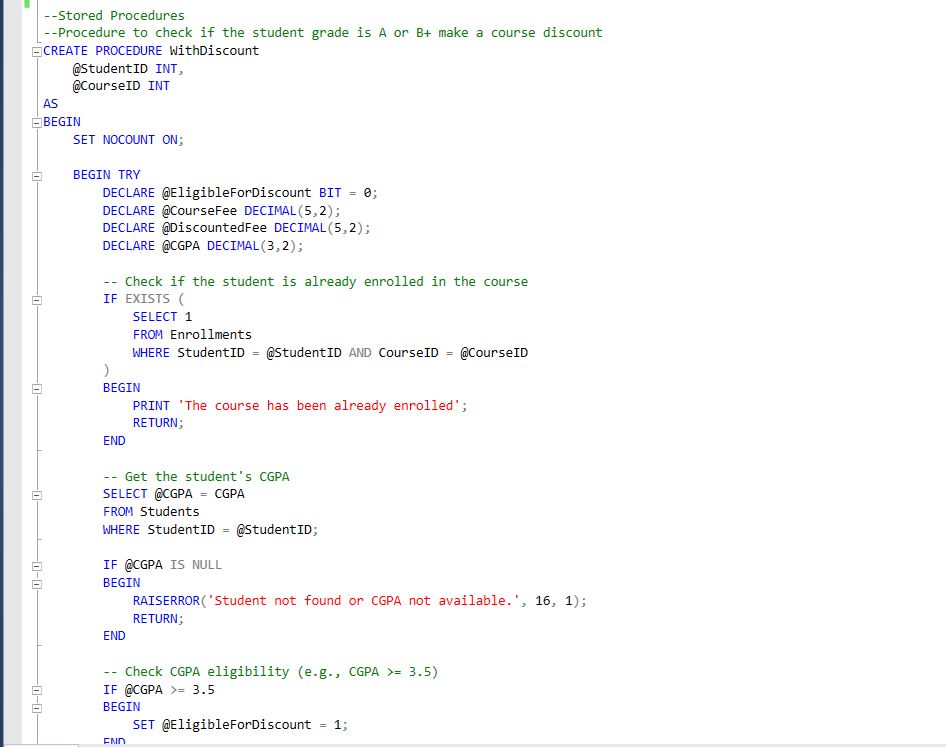
**InstructorLoad view**

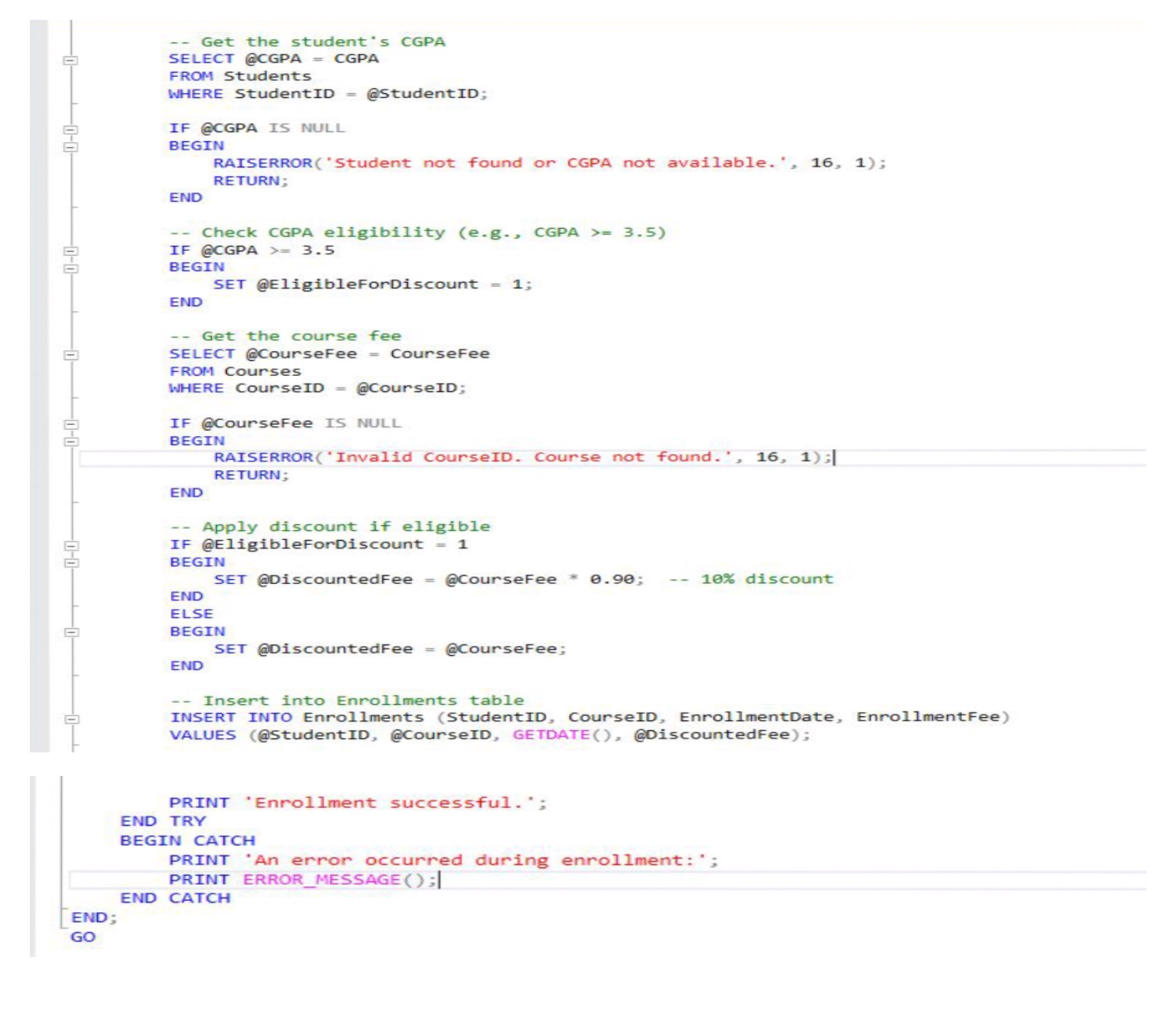


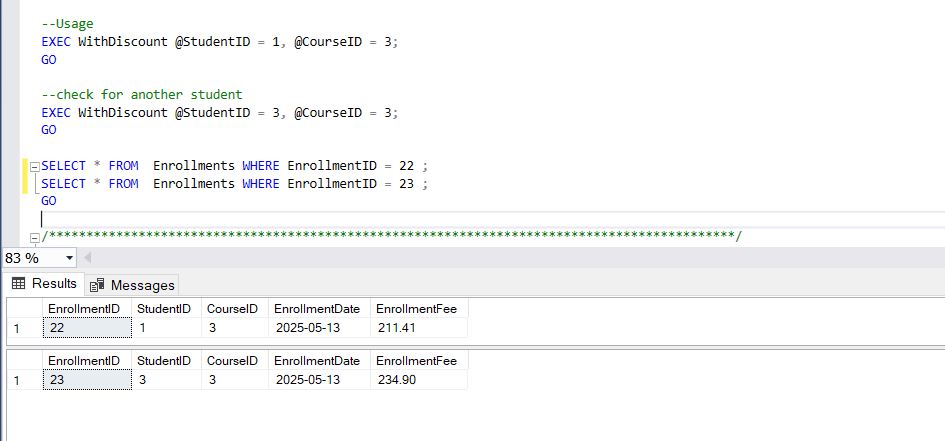
Member2

Stored Procedures try and catch error handling

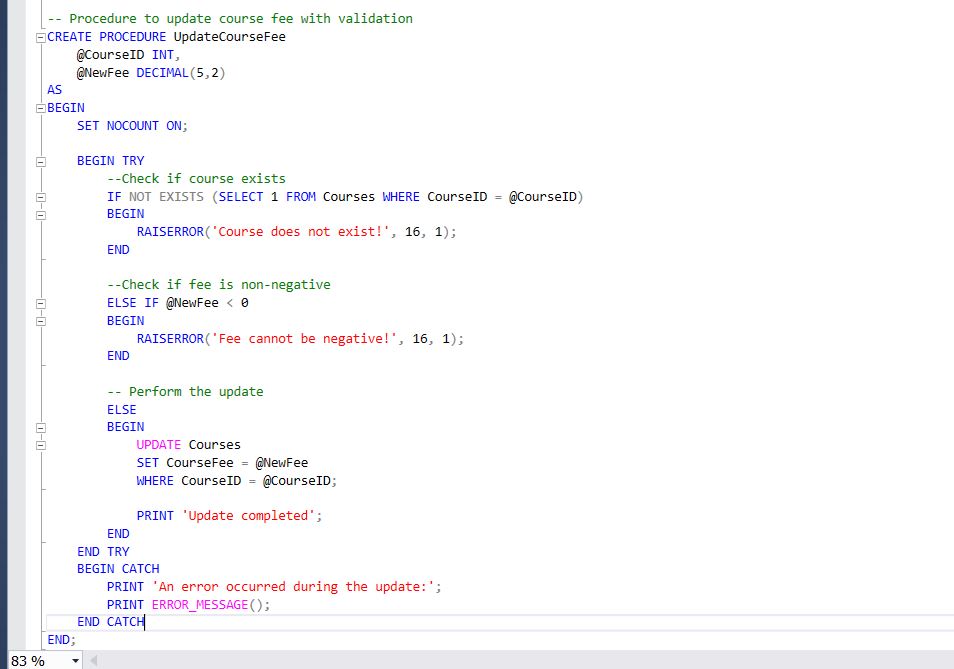
**The first Procedure to check if the student grade is A or B+ make a course discount with**





Two test cases were conducted: one for a student whose CGPA satisfies the condition (CGPA > 3.5), and another for a student whose CGPA falls below this threshold. Both students were enrolled in the same course. The first received the discount as expected, while the second did not behavior of the trigger logic.

**Second Procedure to update course fee with validation**



Here are two checks before and after the update

A white background with black text

AI-generated content may be incorrect.

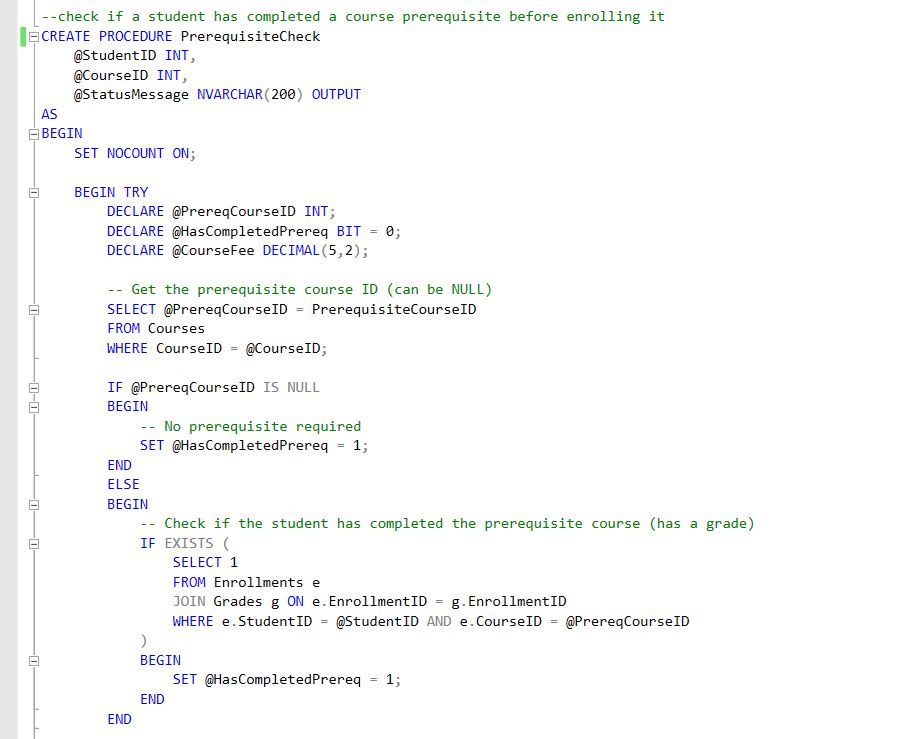
A screenshot of a computer

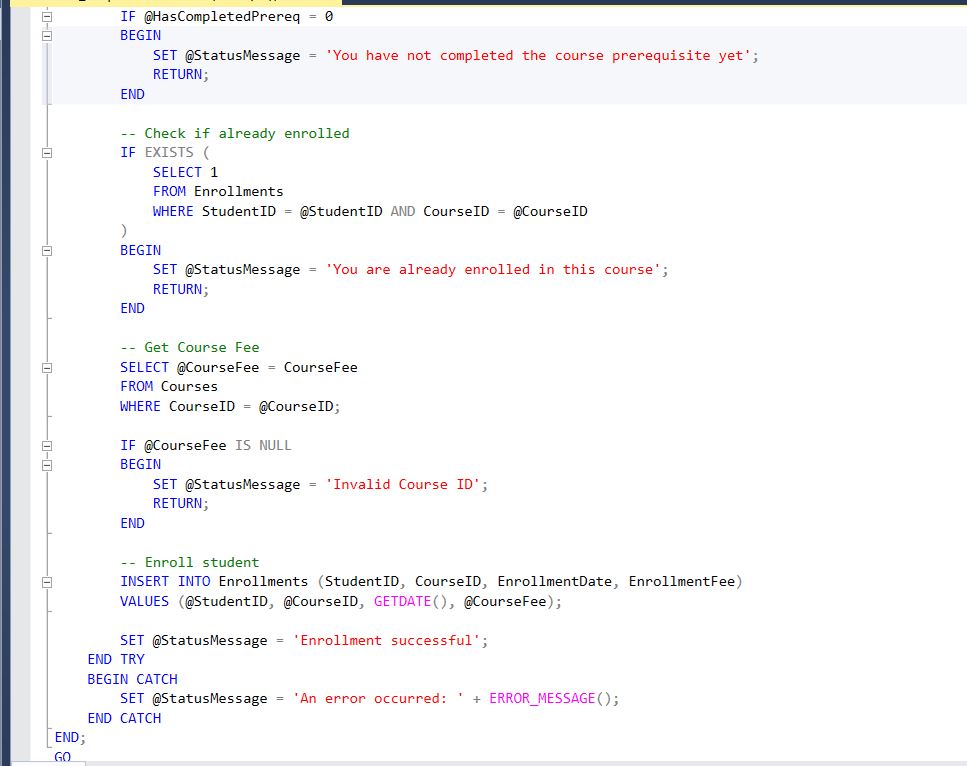
AI-generated content may be incorrect.

A screen shot of a computer

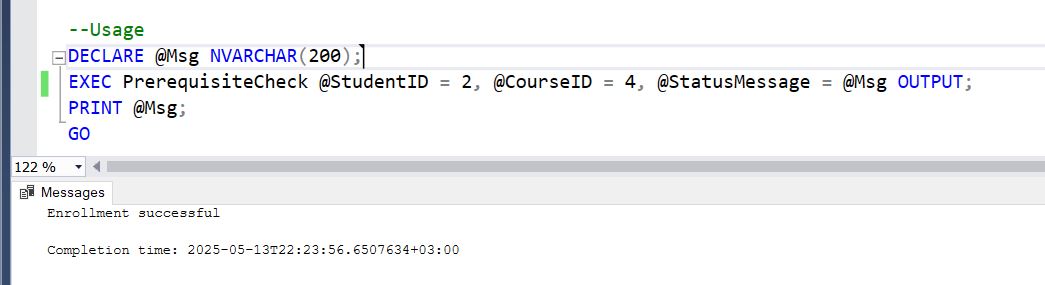
AI-generated content may be incorrect.

**The Third procedure to check if a student has completed a course prerequisite before enrolling it**





Two tests one for the student who has completed the course prerequisite, and one haven’t

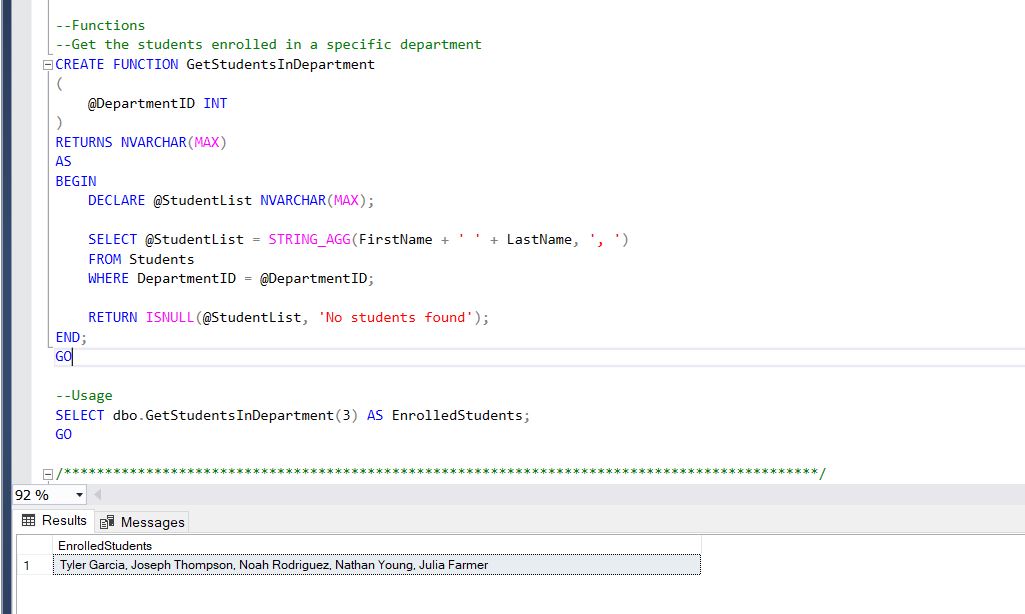


A screenshot of a computer

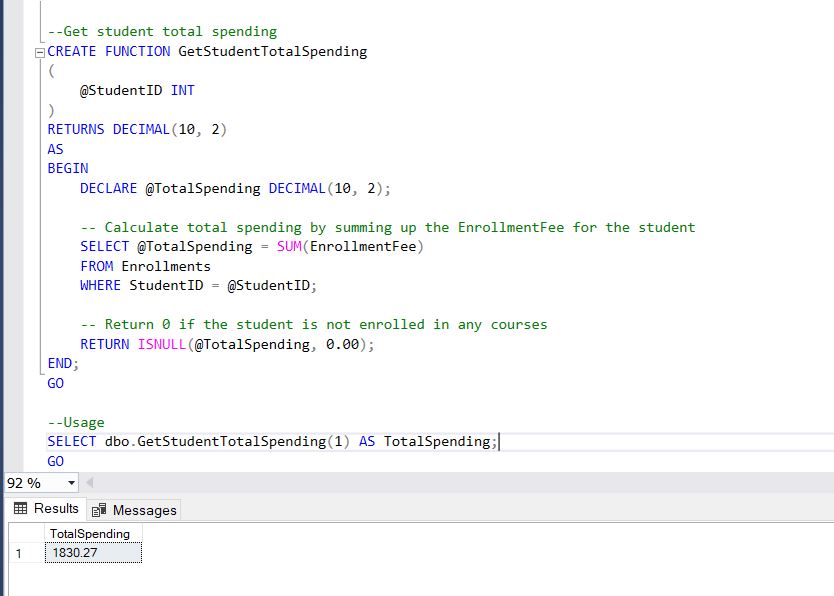
AI-generated content may be incorrect.

**Functions**

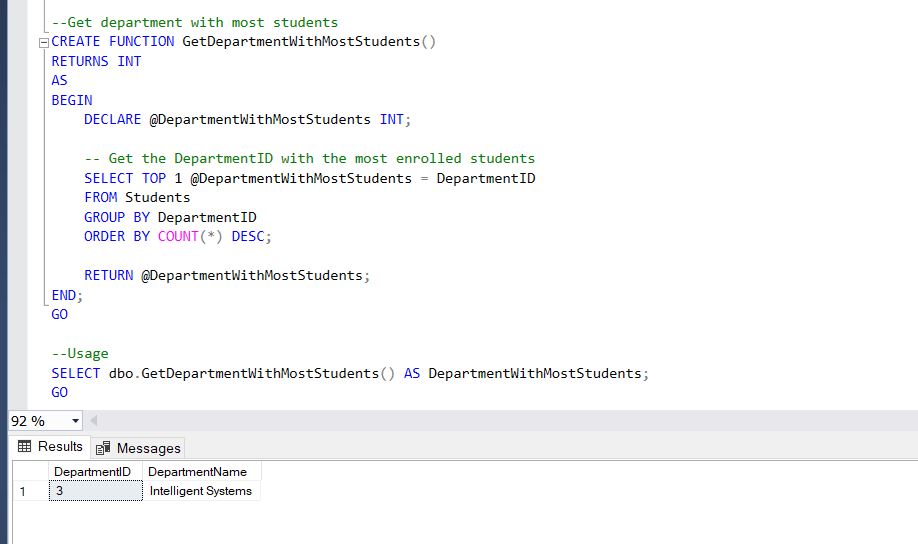
**The first function to Get the students enrolled in a specific department**



**The second function to Get student total spending on courses**

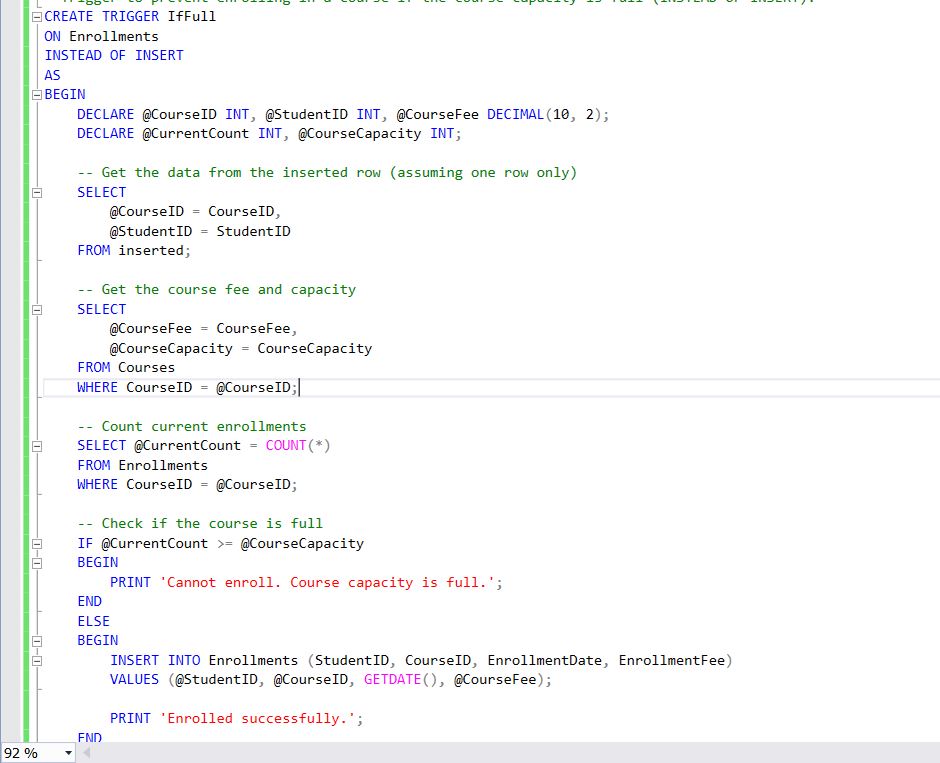


**The Third function to Get department with most students**

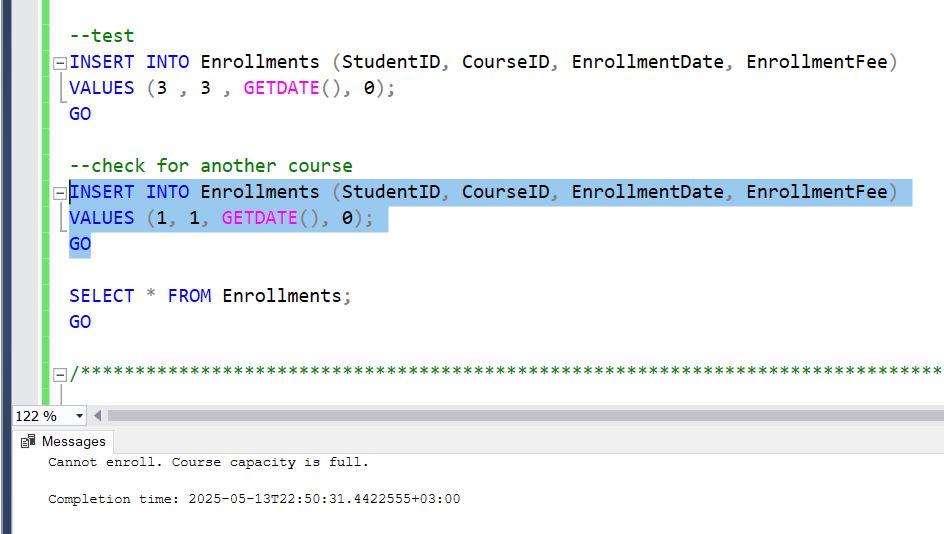


**Triggers**

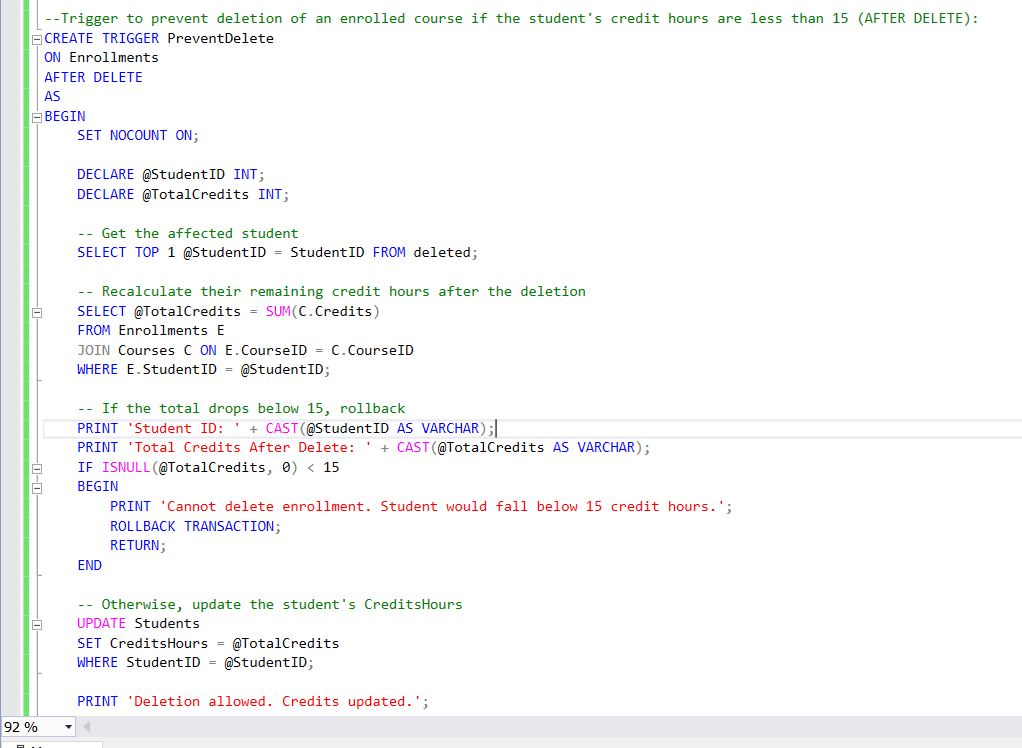
**1)Trigger to prevent enrolling in a course if the course capacity is full (INSTEAD OF INSERT):**

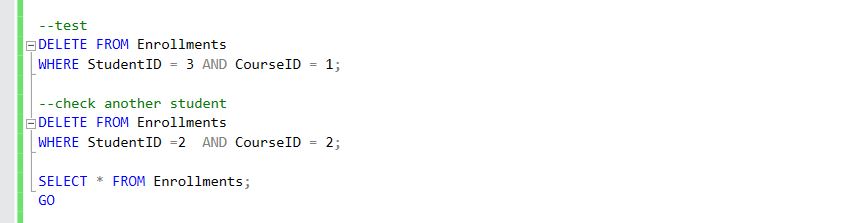
The trigger checks the current number of enrollments for the specified course. If the count exceeds or equals the course's capacity, the trigger prevents the insertion to enforce the course enrollment limit.

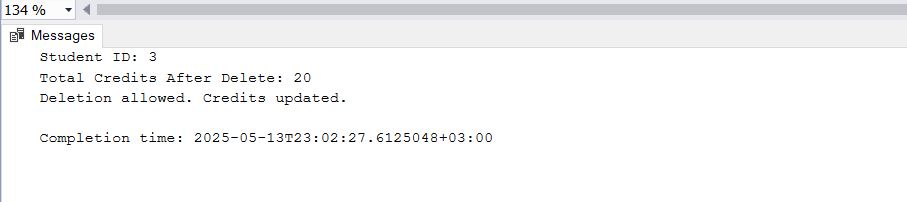
The first student has enrolled successfully, but the other student couldn’t as the capacity is full

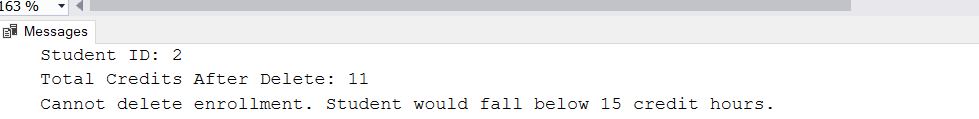


**2) Trigger to prevent deletion of an enrolled course if the student's credit hours are less than 15 (AFTER DELETE):**

The trigger first calculates the student's total credit hours by summing the credits of all currently enrolled courses. It then evaluates the impact of the intended deletion by comparing the total credits before and after the removal. If the resulting credit hours fall below 15, the deletion is prevented to ensure academic requirements are maintained.







Member3

the use of transactions and concurrency control techniques in a university database system using SQL Server. The examples demonstrate how to maintain data consistency, handle concurrent user operations, and avoid typical concurrency problems such as dirty reads, lost updates, and transaction conflicts.

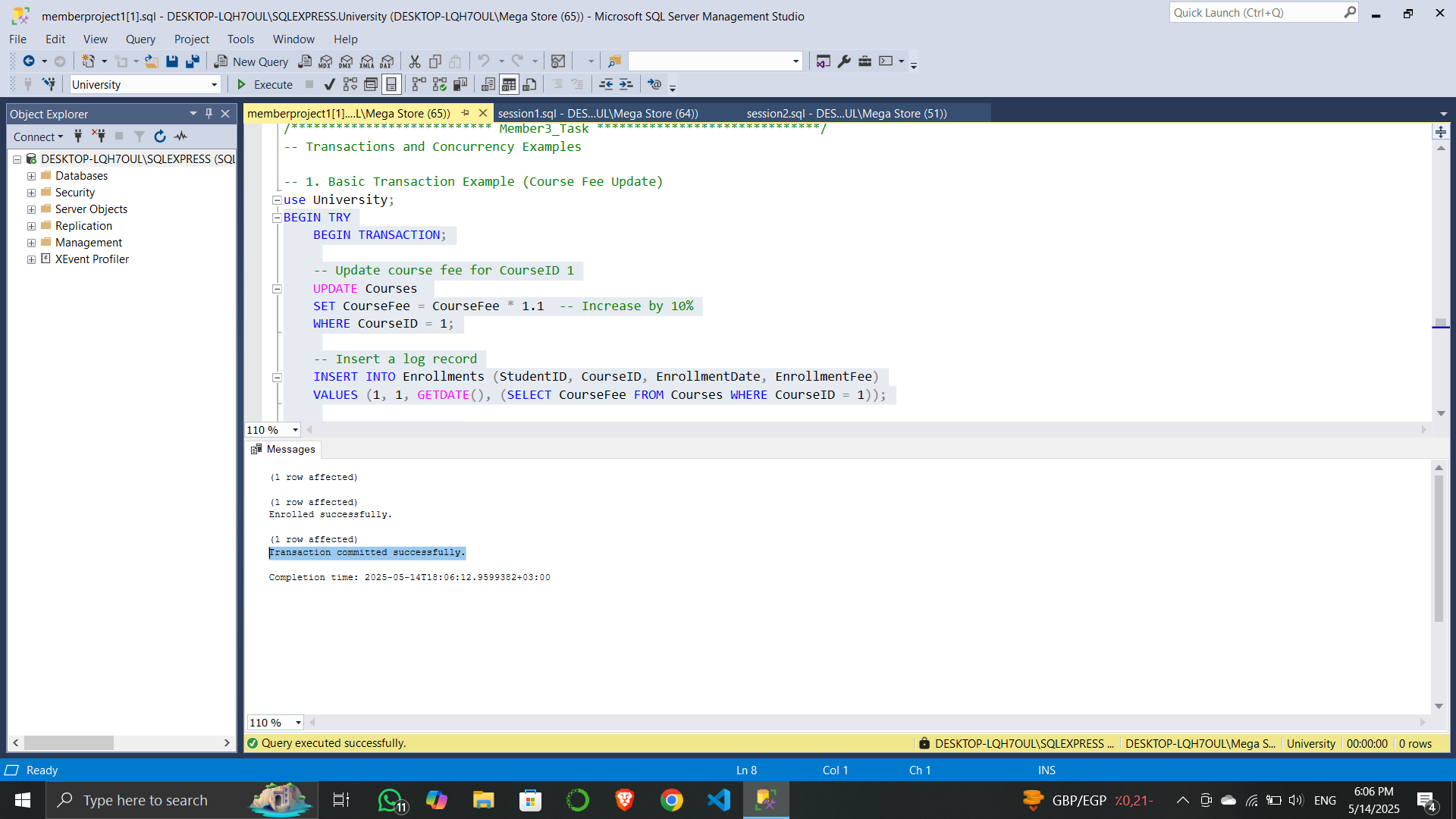
**1. Basic Transaction: Course Fee Update**

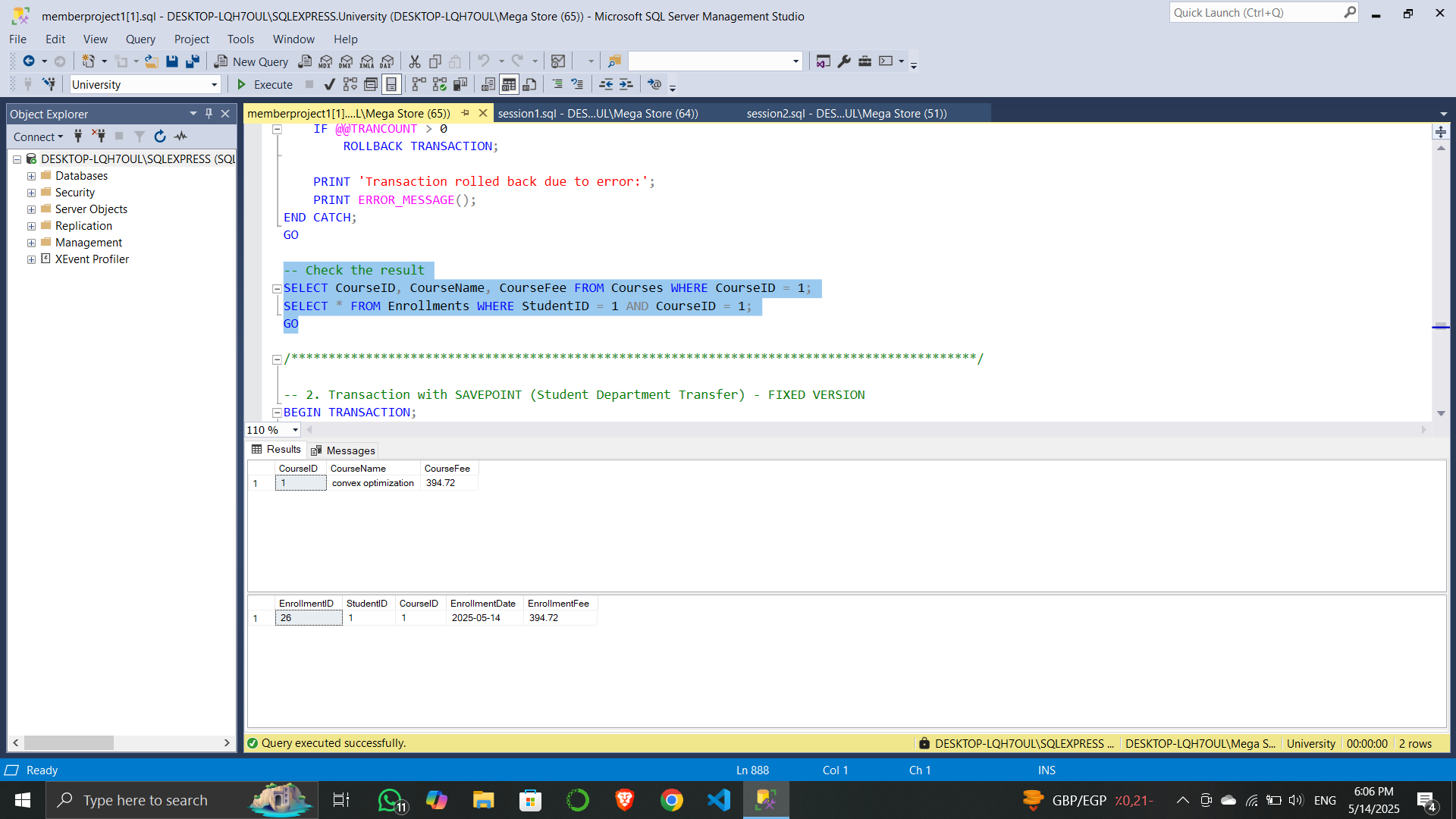
This example demonstrates a simple transaction that increases the course fee by 10% for a specific course (CourseID = 1) and records this action in the Enrollments table.

* **BEGIN TRANSACTION** ensures all changes are committed together.
* **TRY/CATCH** is used for error handling to rollback changes if something goes wrong.
* **COMMIT** confirms the update and insertion.

**Goal**: Maintain consistency between course updates and student enrollments.





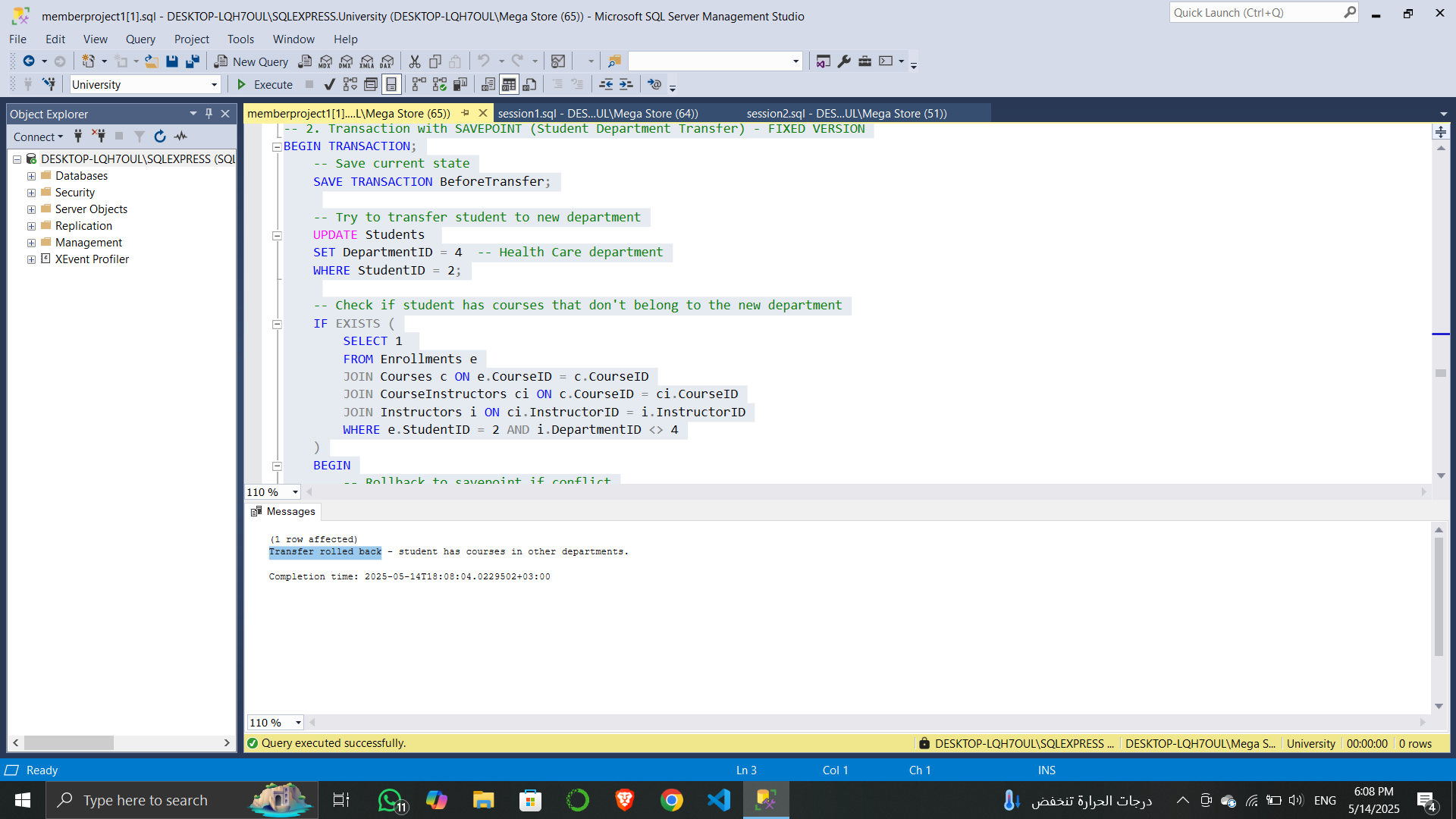


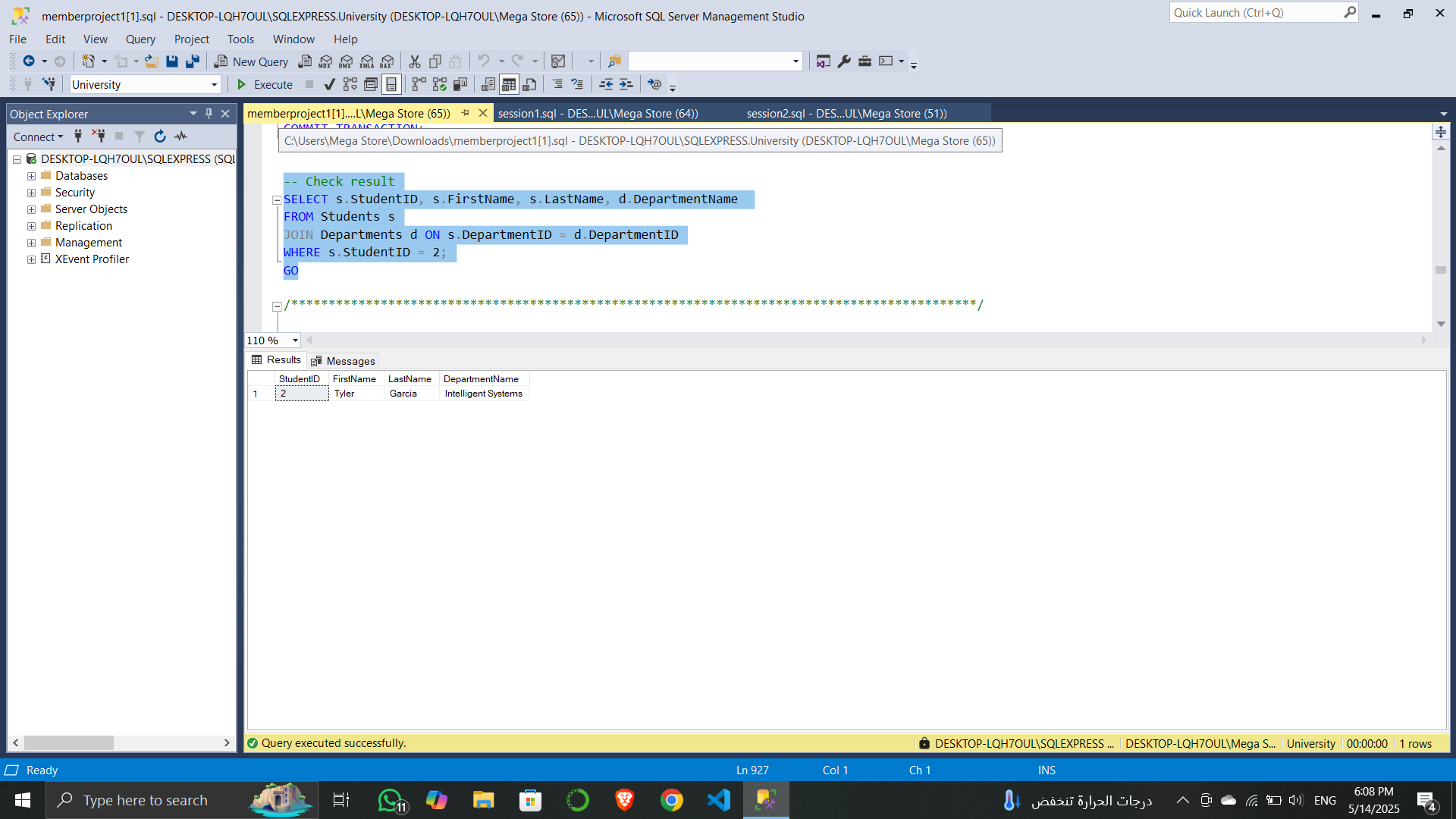
**2. SAVEPOINT Transaction: Student Department Transfer**

This scenario handles conditional rollback using **SAVEPOINT**.

* A student is transferred to a new department.
* If the student is enrolled in courses outside the new department, the operation rolls back to the savepoint.

**Goal**: Ensure that department transfers do not conflict with existing course enrollments.



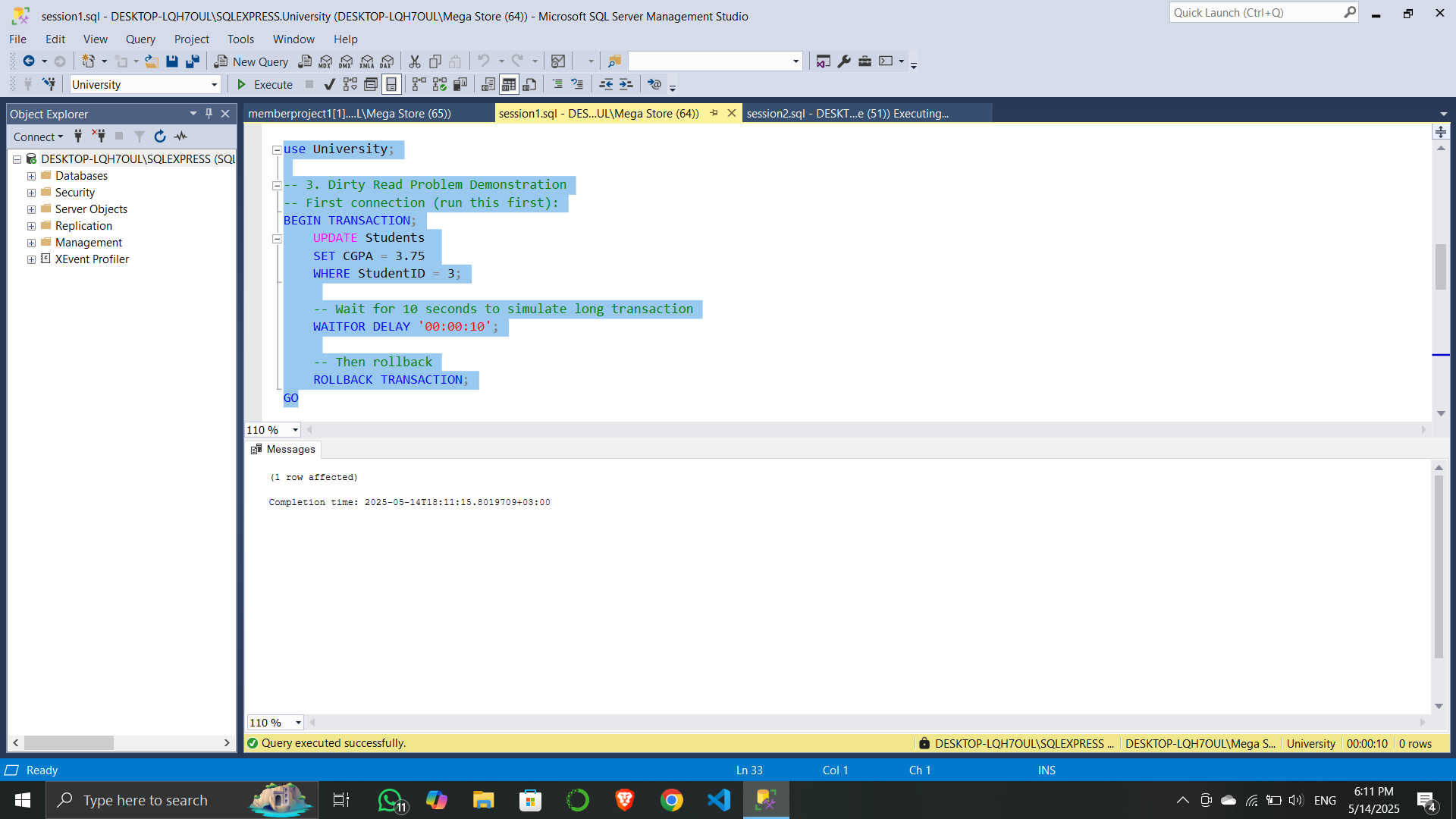


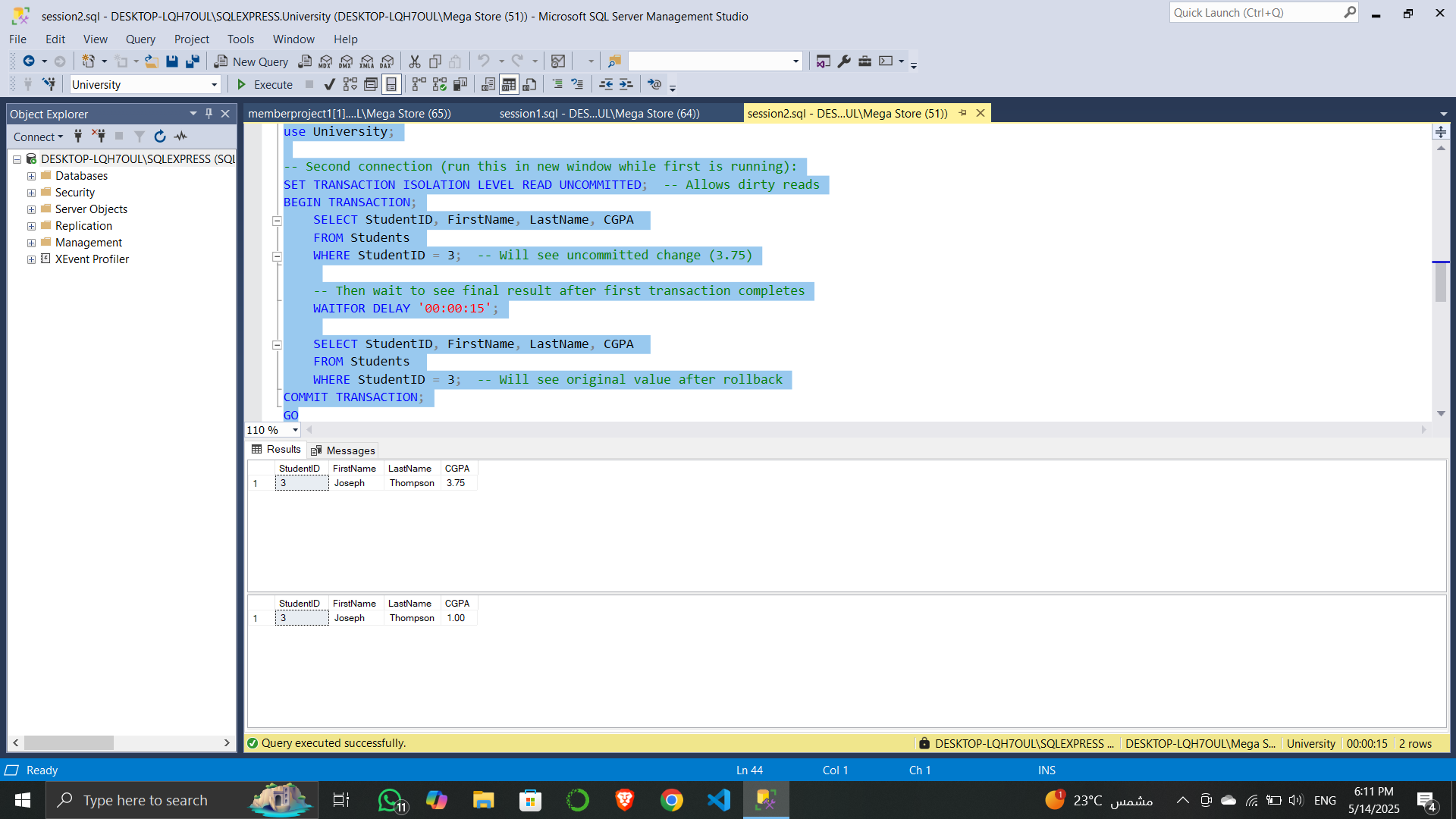
**3. Dirty Read Problem**

This simulation demonstrates a dirty read:

* **Session 1** updates a student's CGPA and waits without committing.
* **Session 2** (using READ UNCOMMITTED) reads the uncommitted value.
* After **Session 1** rolls back, **Session 2** observes inconsistent data.

**Goal**: Show how uncommitted reads lead to temporary, invalid data being read.



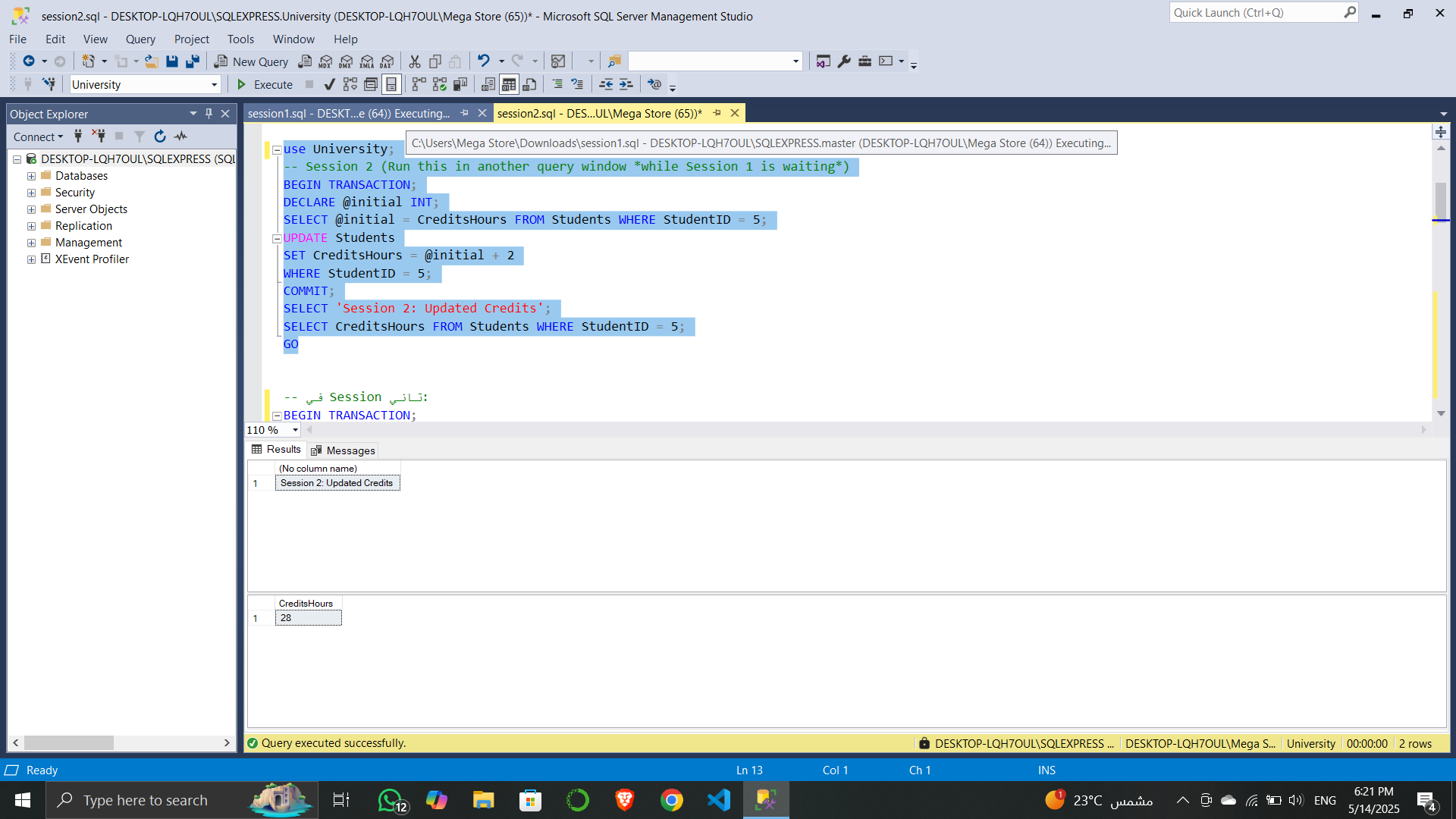


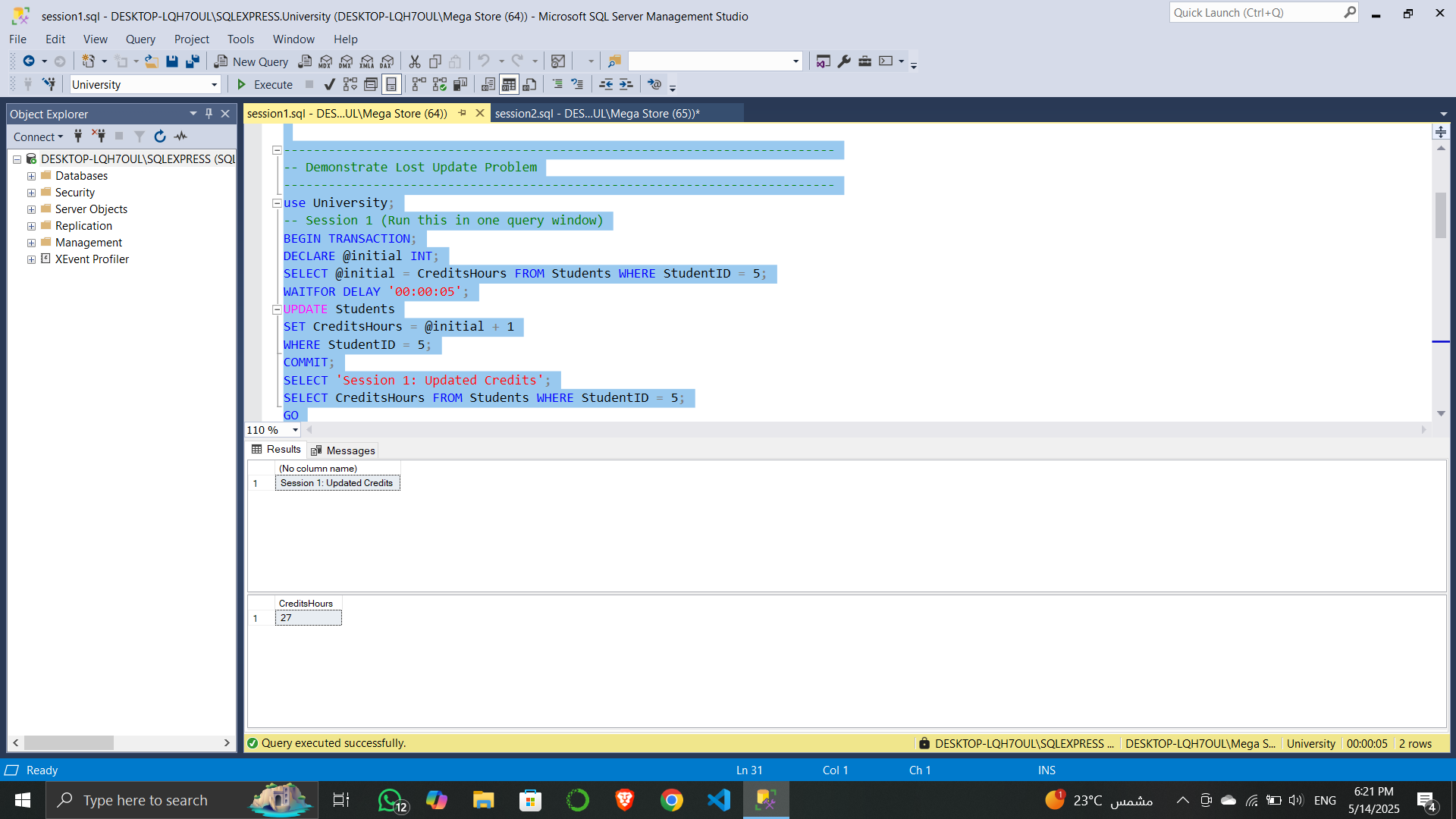
**4. Lost Update Problem**

This simulation shows how updates can be lost when multiple sessions read and update the same data concurrently.

* **Session 1** and **Session 2** both read a student's credit hours.
* Each session updates the value based on the initial read.
* Final result reflects only one session's update.

**Goal**: Illustrate the need for concurrency control mechanisms to avoid overwriting each other’s updates.



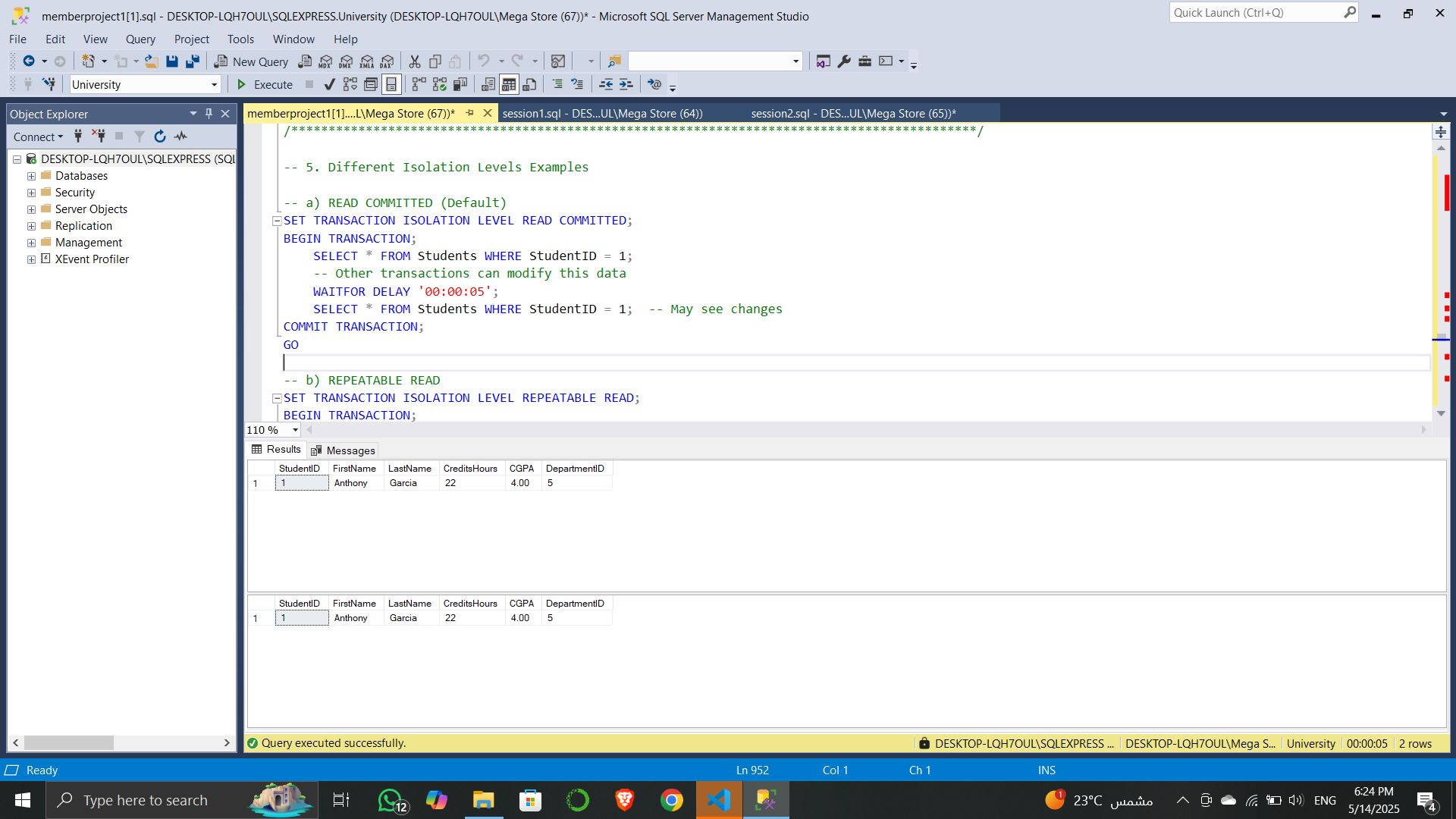


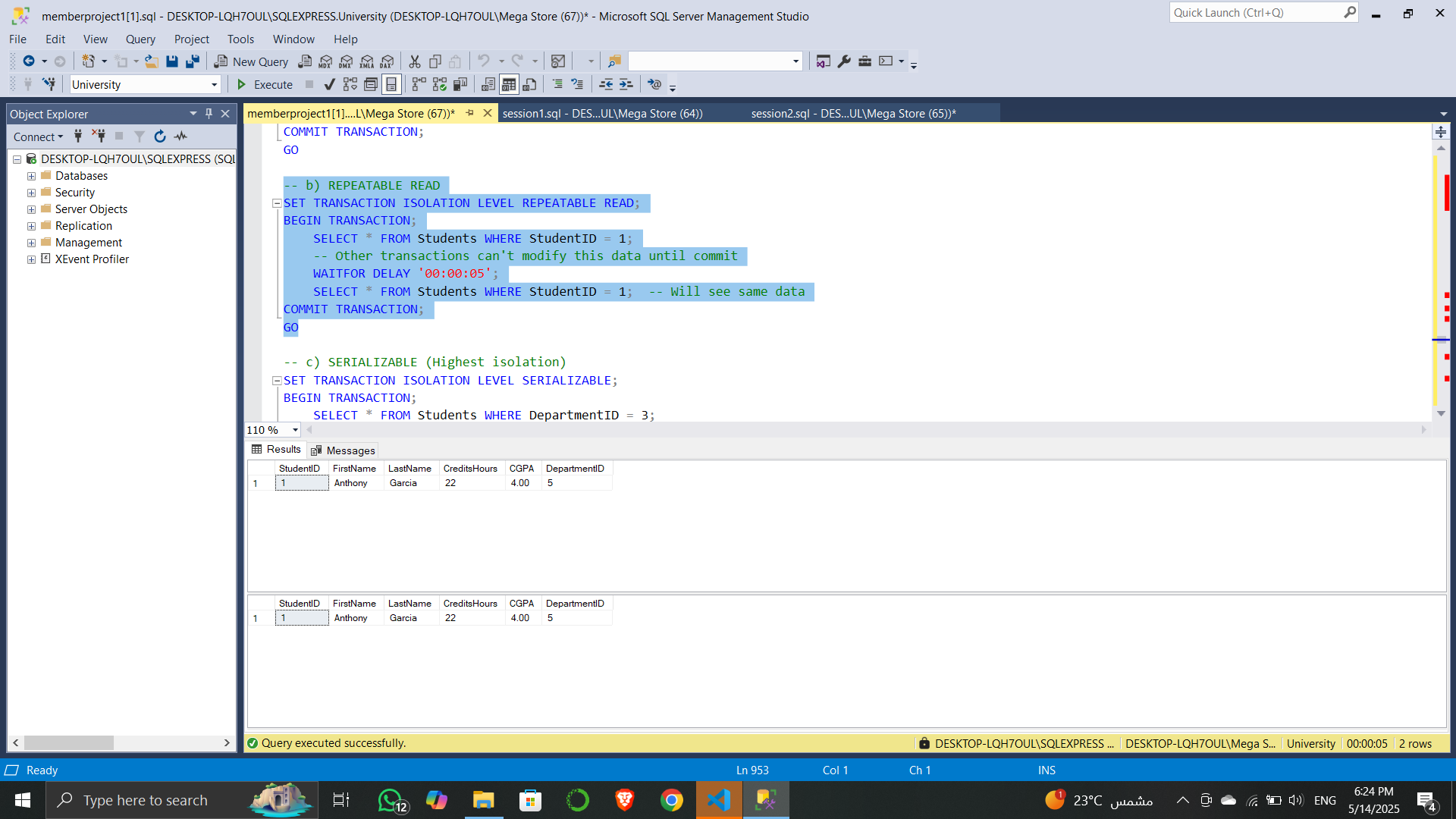
**5. Isolation Levels Demonstration**

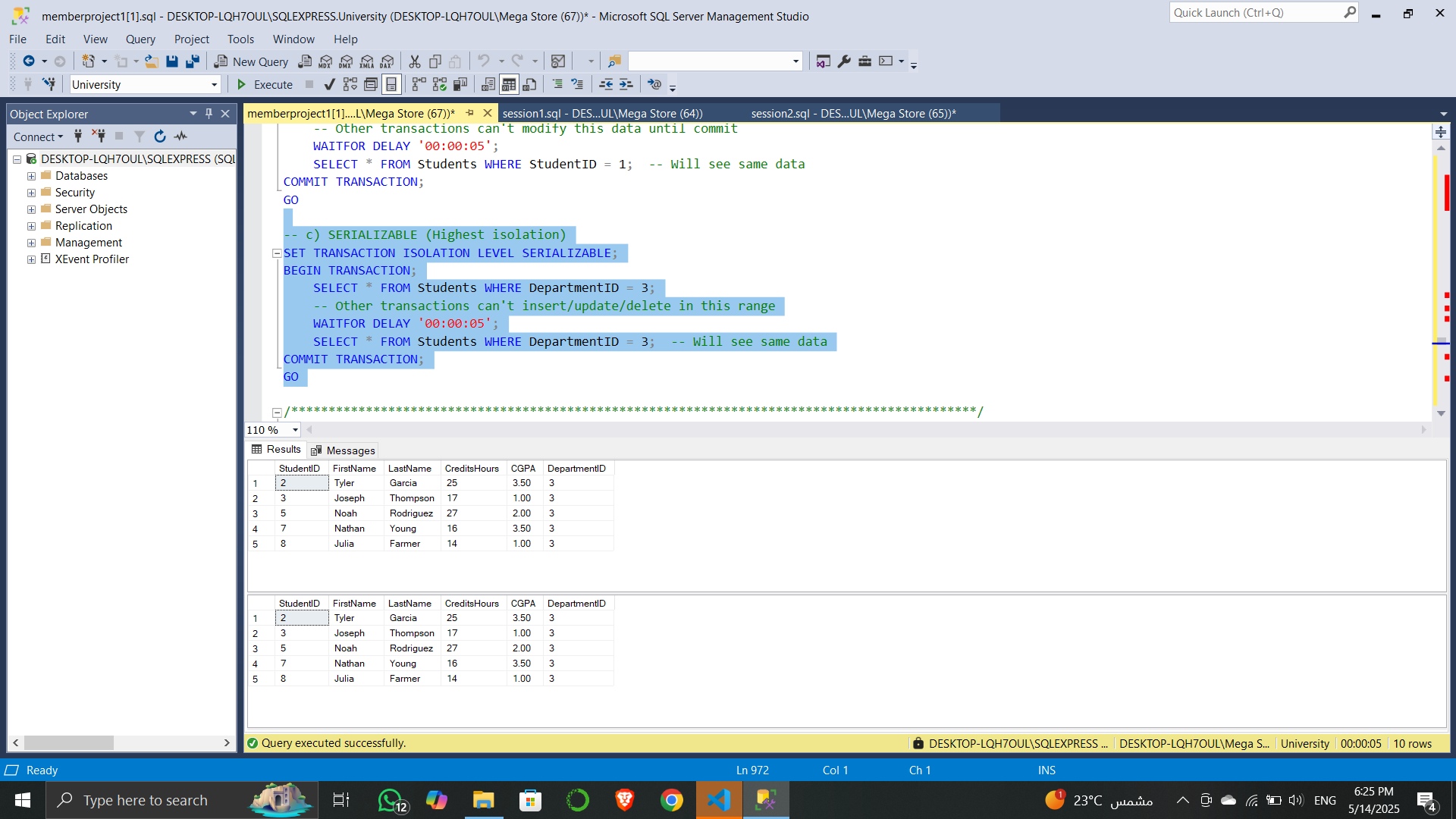
Illustrates different transaction isolation levels in SQL Server and their impact on concurrent access.

* **READ COMMITTED**: Default level. Allows other transactions to change data between reads.
* **REPEATABLE READ**: Prevents others from modifying data read during the transaction.
* **SERIALIZABLE**: Highest isolation. Prevents others from inserting, updating, or deleting rows in the range.

**Goal**: Understand and demonstrate how isolation levels affect transaction visibility and locking behavior.





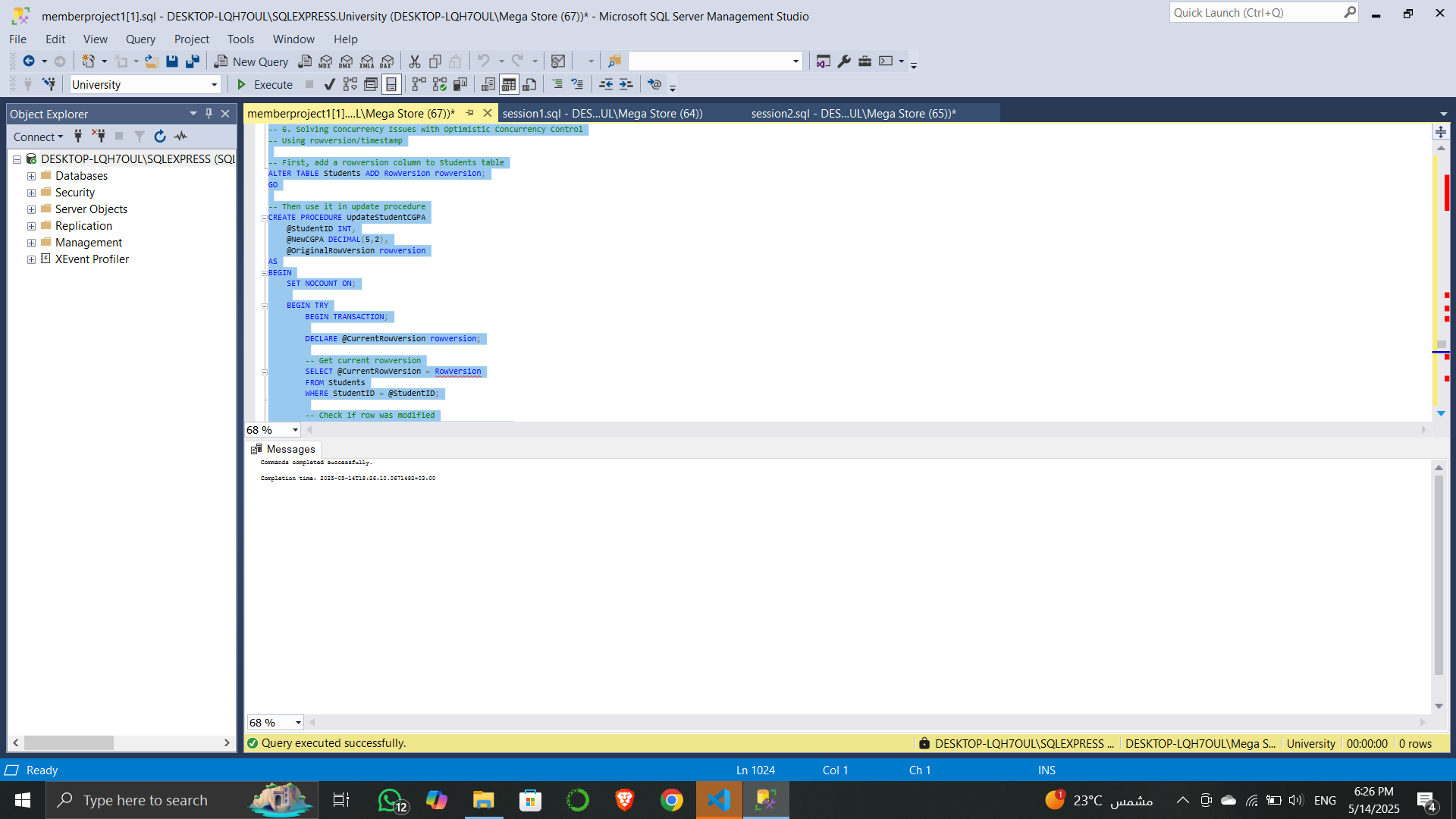


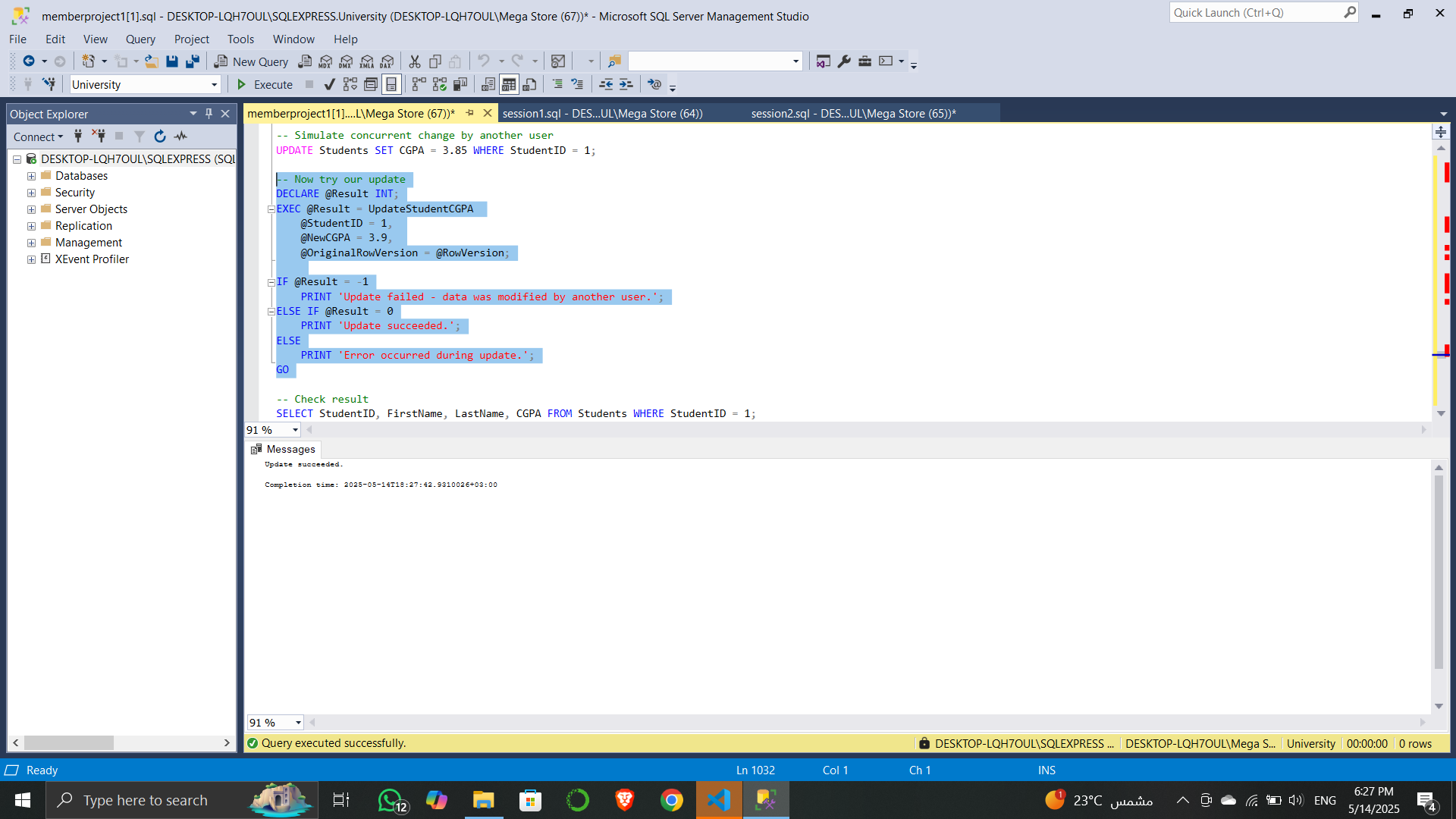
**6. Optimistic Concurrency Control**

This section introduces optimistic concurrency using a rowversion column.

* A new column RowVersion is added to the Students table.
* A stored procedure UpdateStudentCGPA updates a student's CGPA only if the RowVersion hasn’t changed since it was last read.

**Goal**: Prevent data conflicts due to simultaneous updates by different users.





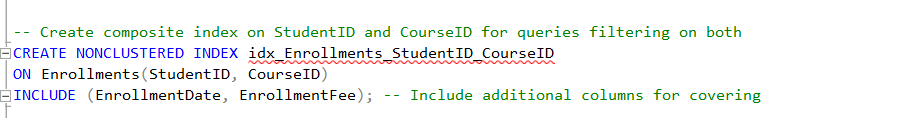
**Memeber4**

he purpose of this report is to demonstrate the implementation and effectiveness of indexes in improving query performance in a university database system. The focus is on commonly accessed tables such as Enrollments, Students, Courses, and Grades.

The following **non-clustered indexes** were created to optimize queries involving frequent filtering, joins, and lookups:

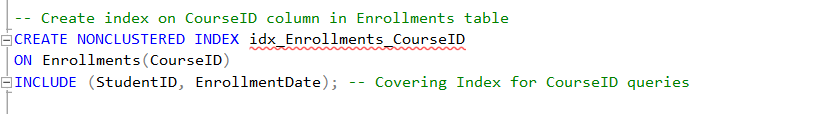
**1\_Enrollments Table**

* **idx\_Enrollments\_StudentID\_CourseID**



Used for queries that filter by both StudentID and CourseID, while also fetching EnrollmentDate and EnrollmentFee directly from the index (covering index).

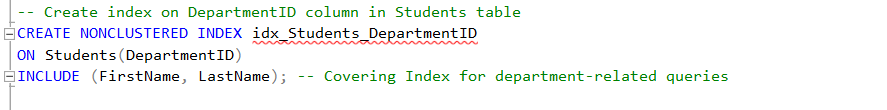
**idx\_Enrollments\_CourseID**

****

Optimizes lookups by CourseID, commonly used for filtering enrollments by course.

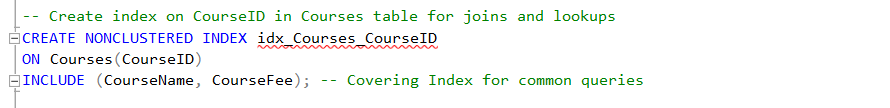
**2-Students Table**

**idx\_Students\_DepartmentID**

****

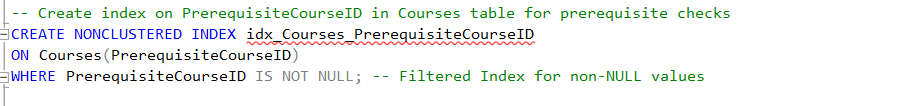
Speeds up queries filtering students by their department

**3-Courses Table**

****

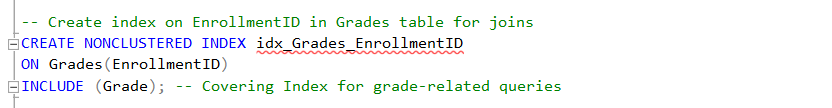
Commonly used for joins and course info retrieval.

**idx\_Courses\_PrerequisiteCourseID**

****

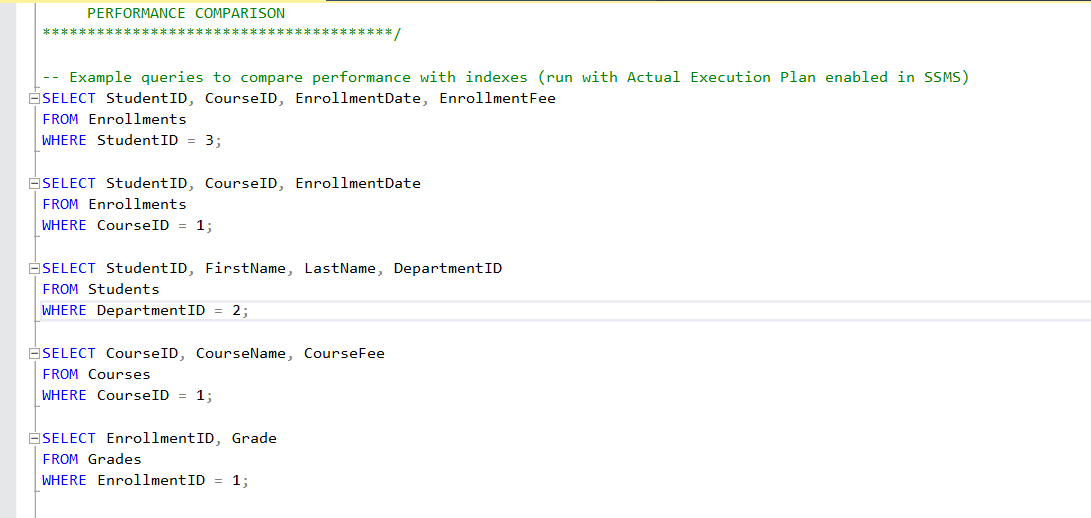
A filtered index for prerequisite checks, reducing storage and improving performance for relevant queries.

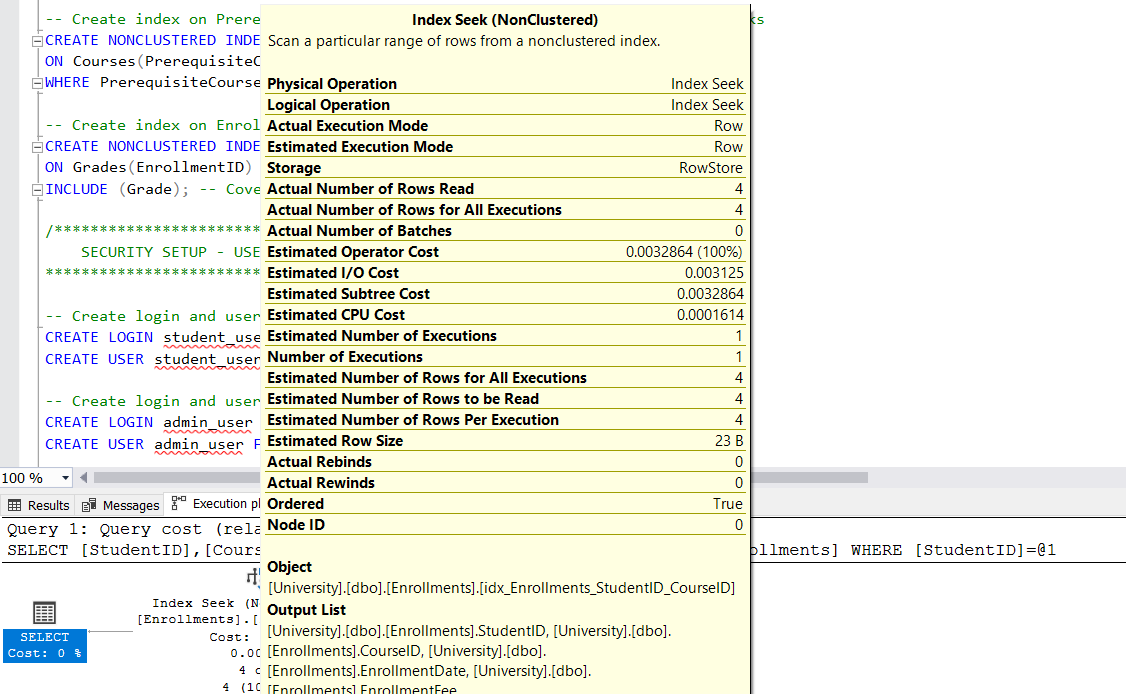
**4-Grades Table**

****

Optimizes queries fetching grades linked to a specific enrollment.

**Test This**

****

****

📊 Execution Plan Analysis

A query was executed to retrieve records based on StudentID and CourseID from the Enrollments table. The following observations were made from the execution plan:

* Physical Operation: Index Seek

Indicates the query engine used the non-clustered index directly and efficiently to locate matching rows.

* Estimated Operator Cost: 0.0032864

Very low, showing minimal resource usage.

* Estimated Number of Rows Read: 4

Indicates that the engine had to read only the relevant 4 rows instead of scanning the entire table.

* Estimated Row Size: 23 B

Efficient memory usage.

* Storage: RowStore

Normal row-based storage engine used in SQL Server.

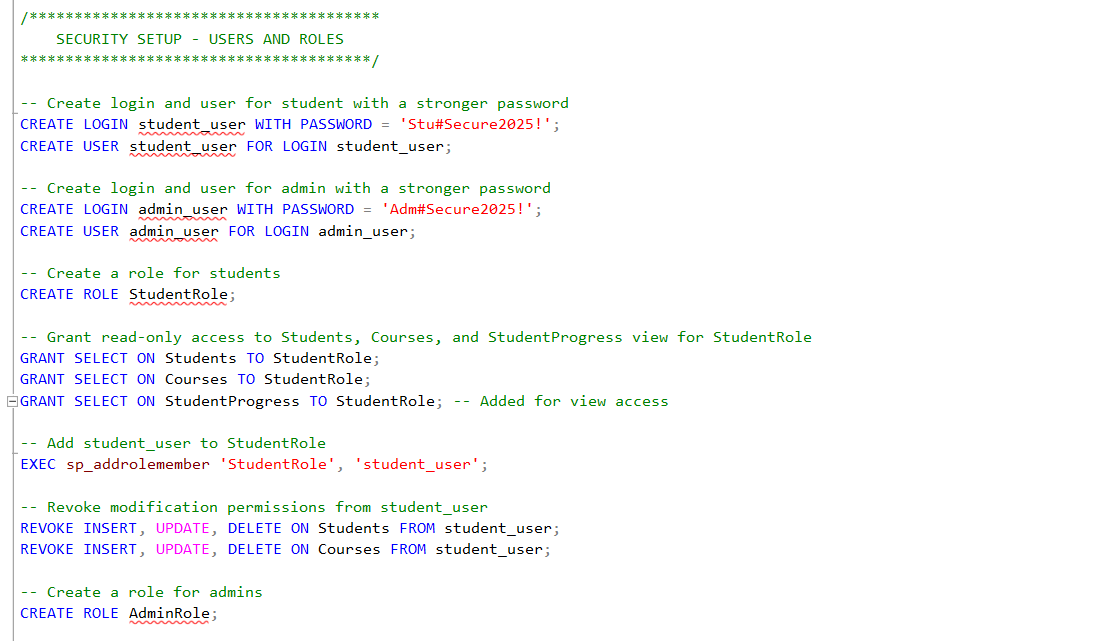
* Index Used: [idx\_Enrollments\_StudentID\_CourseID]

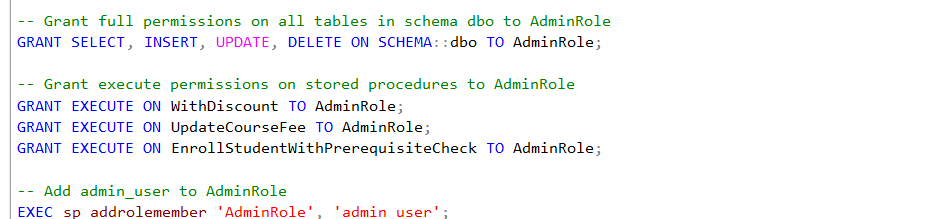
Confirms that our composite index was utilized as expected.

Conclusion from Execution Plan:

The use of Index Seek instead of Table Scan clearly demonstrates that the created composite index (StudentID, CourseID) is effective. It improves performance by reducing the number of rows read, lowers I/O cost, and minimizes CPU usage.

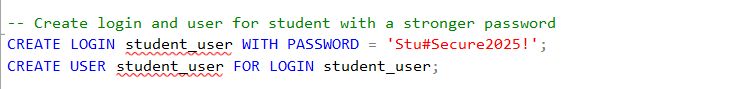
**🔒 Security Measures in SQL Server Database**

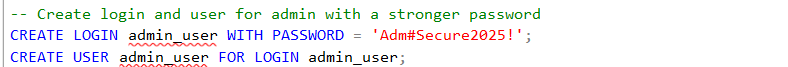
****

****

📝 Objective

The goal of this security setup is to implement role-based access control (RBAC) within the university database to:

* Protect sensitive data from unauthorized modification,
* Restrict access to only necessary operations,
* Separate duties between regular users (students) and privileged users (admins).
* **👥 Users and Roles Defined**
* **🧑‍🎓 Student User**
* ****
*  **Purpose**: Represents a student accessing the system.
*  **Security**: Uses a strong password policy to resist brute-force attacks.
* **🛡️ Admin User**

****

**Purpose: Represents an administrator with higher privileges.**

**Security: Also uses a strong password with complexity rules enforced.**

**🔐 . Role Creation and Privilege**

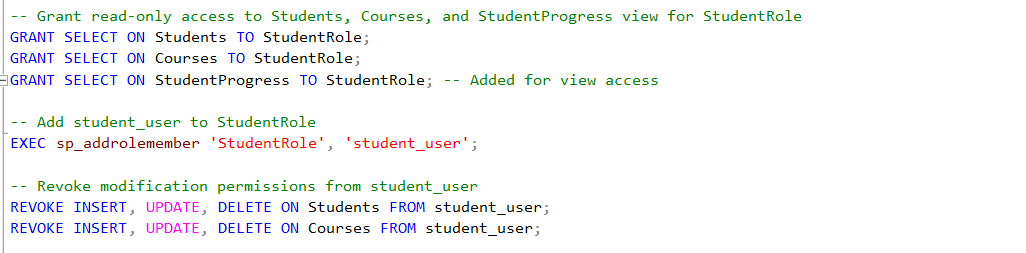
**🎓 StudentRole**

**Granted Permissions:**

* SELECT on Students, Courses, and StudentProgress view.
* Read-only access for informational purposes.

**Revoked Permissions**:

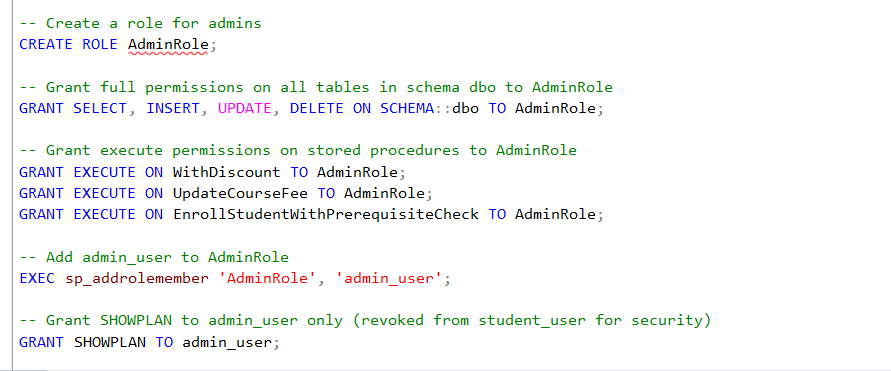
* Prevents students from modifying data.



**User Association**:



**AdminRole**

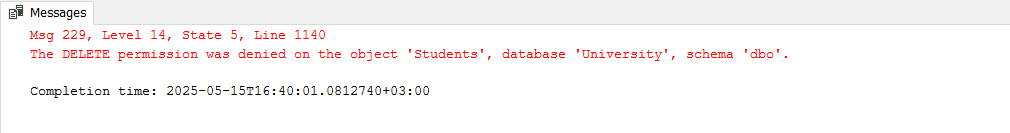
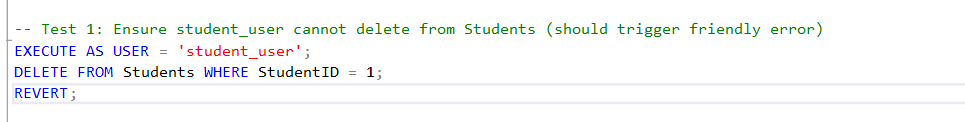
****

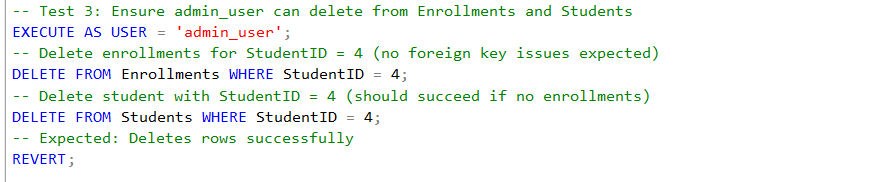
**Granted Permissions:**

* Full DML (Data Manipulation Language) access on all tables in the dbo schema
* Execution of key stored procedures

**SHOWPLAN Permission**:

* Enables admins to view query execution plans for optimization
* **🧪 4. Functional Testing**
* **🔎 Test 1: Unauthorized Delete (Student)**

****

* **Result**: Fails due to revoked DELETE permission.
* **✅ Test 2: Admin Delete Access**
* **Result: Executes successfully as AdminRole has full permissions.**