University of Windsor
Faculty of Science

COMP 8157 Advanced Database Topics
University of Windsor, School of Computer Science
Lab 3
Weight: 3.75 %

Aim: Aim of this lab is to learn Transactions in SQL.

CONFIDENTIALITY AGREEMENT & STATEMENT OF HONESTY

I, **Jivin Varghese Porthukaran** verify that the submitted work is my own, original work, and that I did not use Generative AI tools (e.g., ChatGPT, Bard) to produce this lab report. I confirm knowing that a mark of 0 may be assigned for sharing or copying this work.

_____	**Jivin Varghese Porthukaran**	**110128868**
Student Signature	Student Name	Student I.D.

Assignment

DISCLAIMER: Wherever there is **<YourFirstName>** in the sample code snippets provided in this document, you need to replace it with your actual first name. All the tables and the stored procedure you will create as part of this quiz will have this constraint. For example, a student named Loren Ipsum would be creating the table **<YourFirstName>SampleTable** as **LorenSampleTable**.

Let's consider a scenario where you have a university registration website where students can register for courses. You have three tables in your database: **Students, Courses** and **StudentRegistration**. The **Students** table contains information about each student, such as their student ID, Full Name, Email, and total credits. The **Courses** table contains information about all the available courses provided in the university, such as the CourseID, Course name, Instructor for the course, credits with respect to the course, and the number of seats available for the registered course.

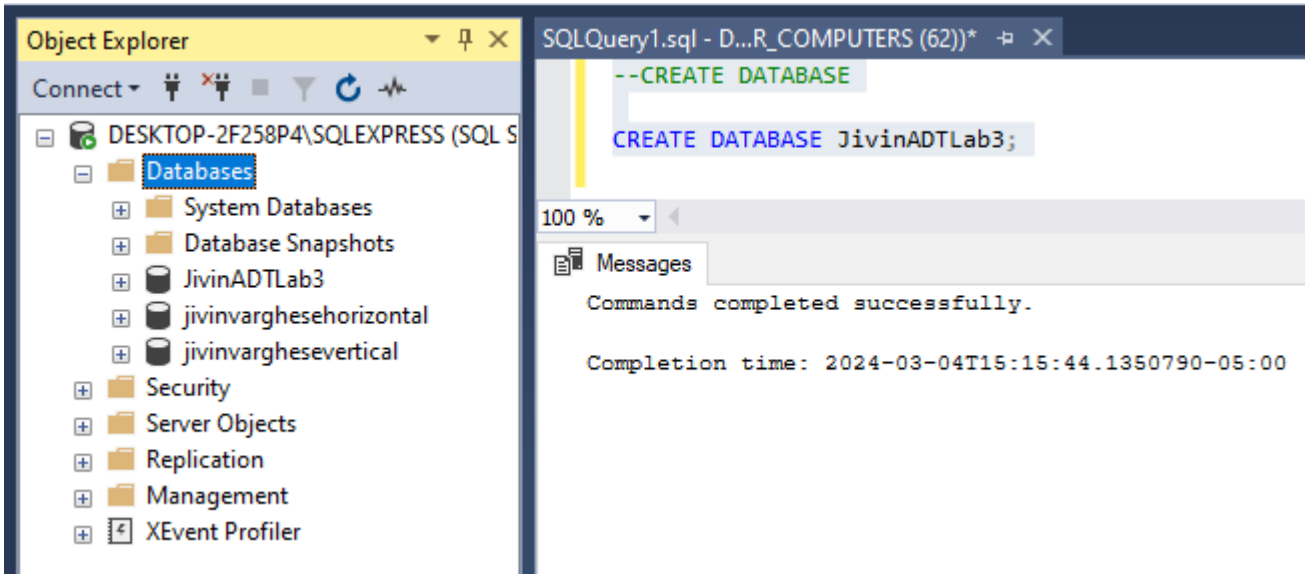
You want to ensure that the **StudentRegistration** table is updated accurately when a student registers for a course, so you need to use a transaction to ensure the integrity of the data.

The required submission is highlighted in **RED** for each of the four questions.

i.Create database <First_Name>ADTLab3. **Submit the screenshots of successful query execution.**

CODE: CREATE DATABASE JivinADTLab3;

OUTPUT:



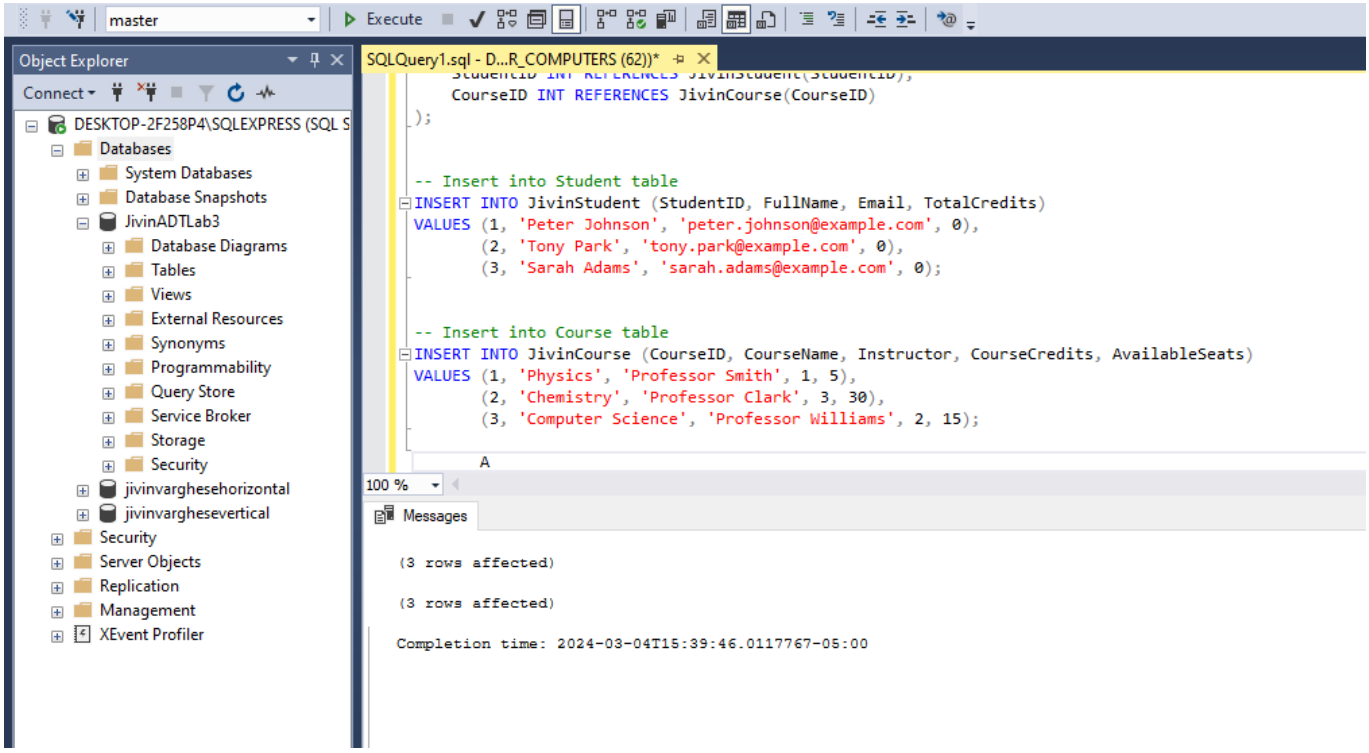
ii. Create table <First_Name>Student table, <First_Name>StudentRegistration table and <First_Name>Course table with the following schema. Insert a few rows in the **Students** and **Course** table only. **Submit the error-free and working SQL code after modifying the table names, and the screenshot of the Students, Courses, and StudentRegistration table after inserting data.**

CODE:

```
--CREATE TABLES

-- Create Students table
CREATE TABLE JivinStudent (
    StudentID INT PRIMARY KEY,
    FullName VARCHAR(100),
    Email VARCHAR(100),
    TotalCredits INT
);
```

```sql
-- Create Course table
CREATE TABLE JivinCourse (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Instructor VARCHAR(100),
    CourseCredits INT,
    AvailableSeats INT
);

-- Create StudentRegistration table
CREATE TABLE JivinStudentRegistration (
    RegistrationID INT PRIMARY KEY IDENTITY,
    StudentID INT REFERENCES JivinStudent(StudentID),
    CourseID INT REFERENCES JivinCourse(CourseID)
);


-- Insert into Student table
INSERT INTO JivinStudent (StudentID, FullName, Email, TotalCredits)
VALUES (1, 'Peter Johnson', 'peter.johnson@example.com', 0),
       (2, 'Tony Park', 'tony.park@example.com', 0),
       (3, 'Sarah Adams', 'sarah.adams@example.com', 0);


-- Insert into Course table
INSERT INTO JivinCourse (CourseID, CourseName, Instructor, CourseCredits, AvailableSeats)
VALUES (1, 'Physics', 'Professor Smith', 1, 5),
       (2, 'Chemistry', 'Professor Clark', 3, 30),
       (3, 'Computer Science', 'Professor Williams', 2, 15);
```

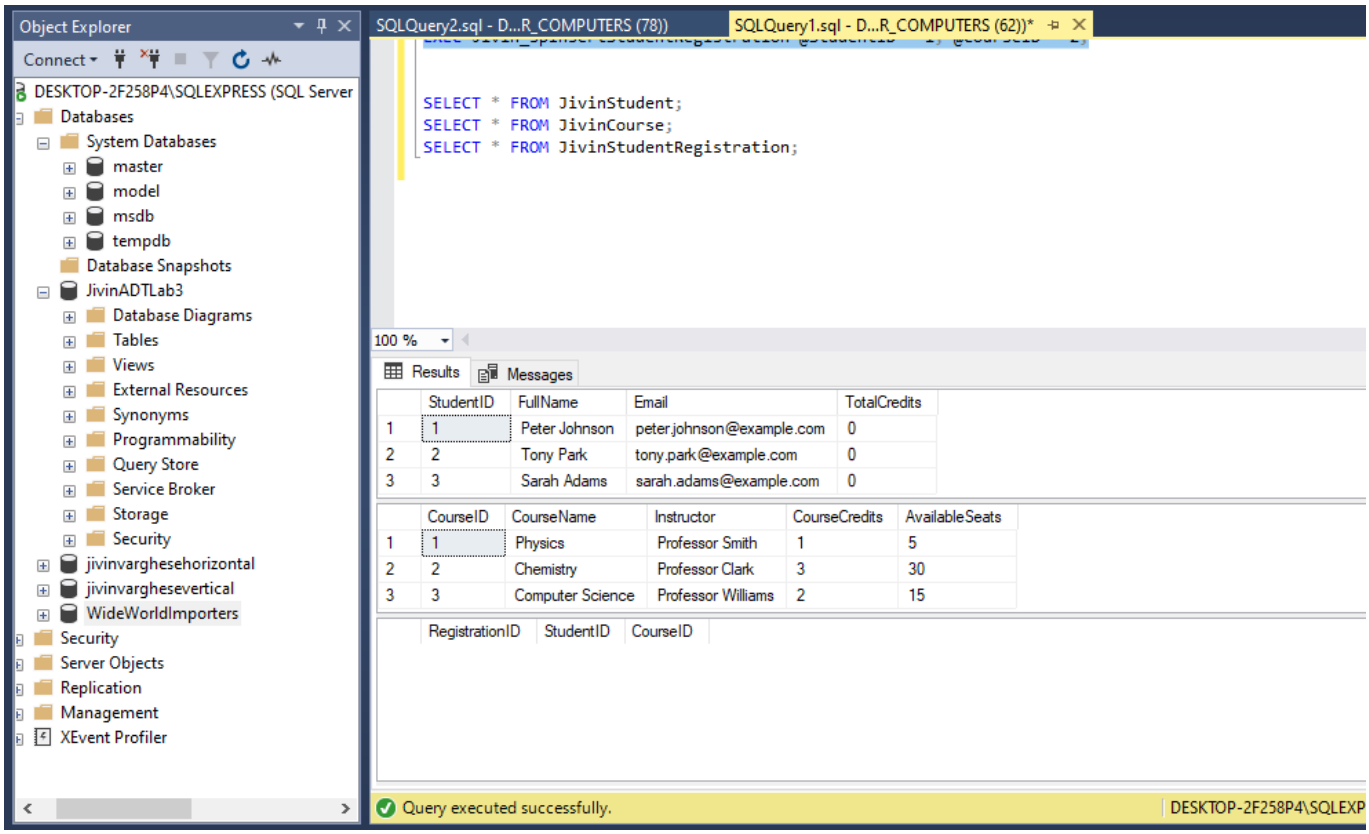
OUTPUT:



CODE:

```sql
SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:



iii.

We need to create a store procedure whenever a student registers for a course. The procedure must check the availability of seats in the course before registering the student for the course. If the student is registered the availability of the seats should be deducted and the credits of the course should be added to the student's total credits.

The structure of the stored procedure is as follows:

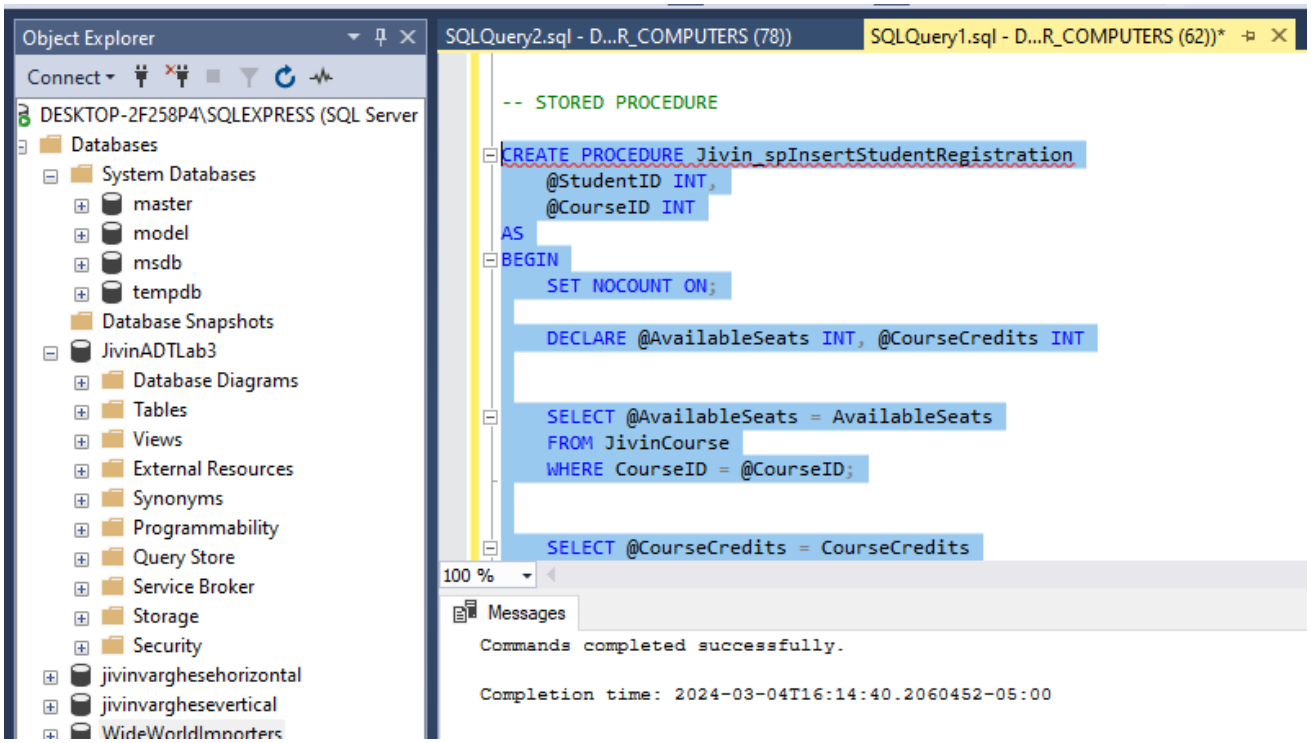
- Name of the procedure: <Your_First_Name>_spInsertStudentRegistration which takes StudentID and CourseID as input parameters.
- Check the availability of Seats in the provided course table.
- Decrease the Availability of the Seats in the Courses Table.
- Add Course credits of the Courses table to the Student Total credits in the Students table.
- Insert the record into the StudentRegistration table with RegistrationID, StudentID, and CourseID.
- If the available seats are less than or equal to 0 then the transaction should be rolled back and print the message 'Course is full. Registration failed'. **Submit the error-free working SQL code of the stored procedure.**

CODE:

```
CREATE PROCEDURE Jivin_spInsertStudentRegistration
    @StudentID INT,
    @CourseID INT
AS
BEGIN
    -- Turn off the message indicating the number of rows affected
    SET NOCOUNT ON;

    -- Declare variables to store the number of available seats and course credits
    DECLARE @AvailableSeats INT, @CourseCredits INT

    -- Retrieve the number of available seats for the specified course
    SELECT @AvailableSeats = AvailableSeats
```

```
    FROM JivinCourse
    WHERE CourseID = @CourseID;

    -- Retrieve the number of credits for the specified course
    SELECT @CourseCredits = CourseCredits
    FROM JivinCourse
    WHERE CourseID = @CourseID;

    -- Check if there are available seats for the course
    IF @AvailableSeats > 0
    BEGIN
        -- Begin a transaction to ensure data integrity
        BEGIN TRANSACTION;

        -- Decrease the available seats for the course by 1
        UPDATE JivinCourse
        SET AvailableSeats = AvailableSeats - 1
        WHERE CourseID = @CourseID;

        -- Increase the student's total credits by the number of credits for the course
        UPDATE JivinStudent
        SET TotalCredits = TotalCredits + @CourseCredits
        WHERE StudentID = @StudentID;

        -- Insert a new record into the StudentRegistration table to represent the student's
registration for the course
        INSERT INTO JivinStudentRegistration (StudentID, CourseID)
        VALUES (@StudentID, @CourseID);

        -- Commit the transaction to make the changes permanent
        COMMIT;
    END
    ELSE
    BEGIN
        -- Rollback the transaction if there are no available seats for the course
        ROLLBACK;

        -- Print a message indicating that the registration failed due to lack of available seats
        PRINT 'Course is full. Registration failed.';
    END
END;
```

OUTPUT:



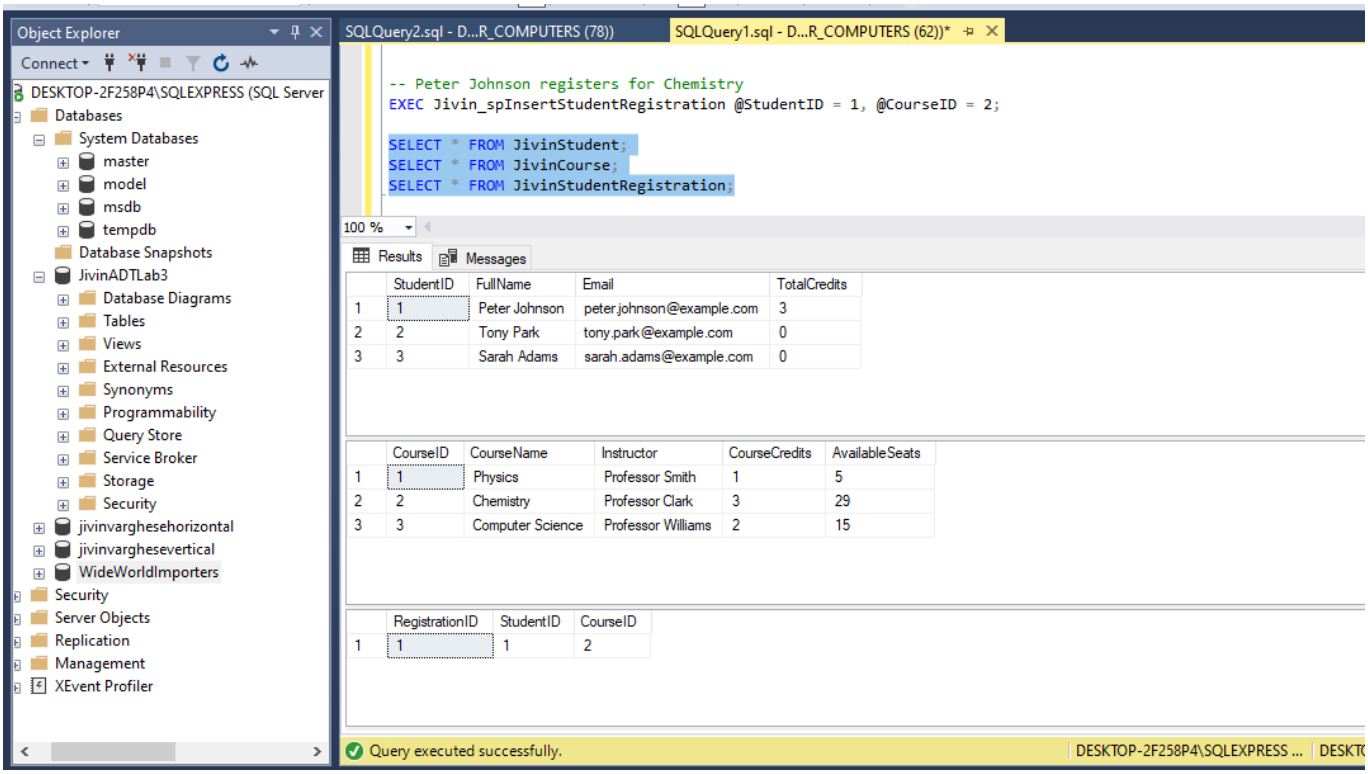
iv.Testing the solution by registering below students for the following course.

a. **Peter Johnson registers for Chemistry.**

CODE:

```
-- Peter Johnson registers for Chemistry
EXEC Jivin_spInsertStudentRegistration @StudentID = 1, @CourseID = 2;

SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:

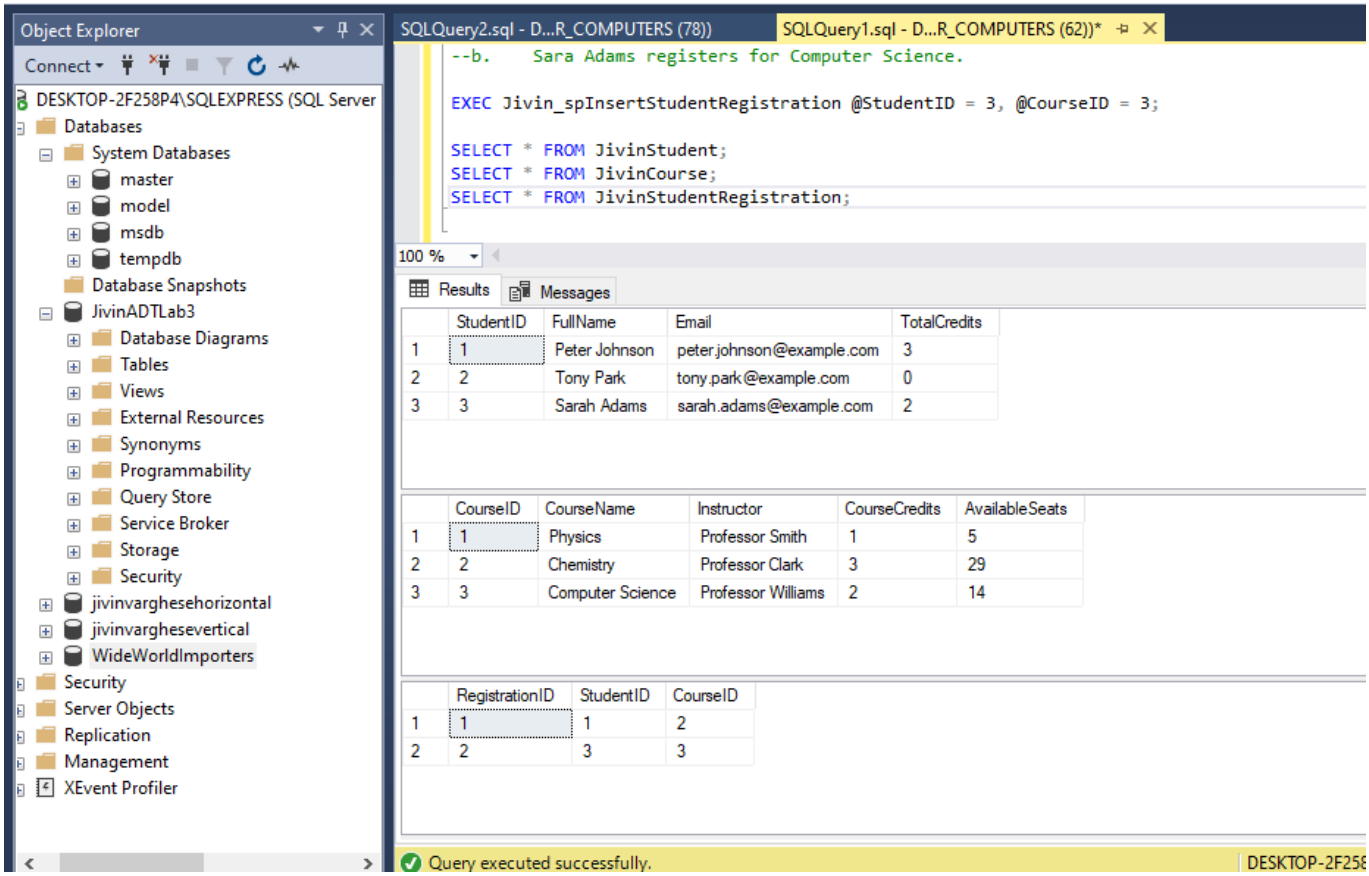


b. Sara Adams registers for Computer Science.

CODE:

```
--b.   Sara Adams registers for Computer Science.

EXEC Jivin_spInsertStudentRegistration @StudentID = 3, @CourseID = 3;

SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:

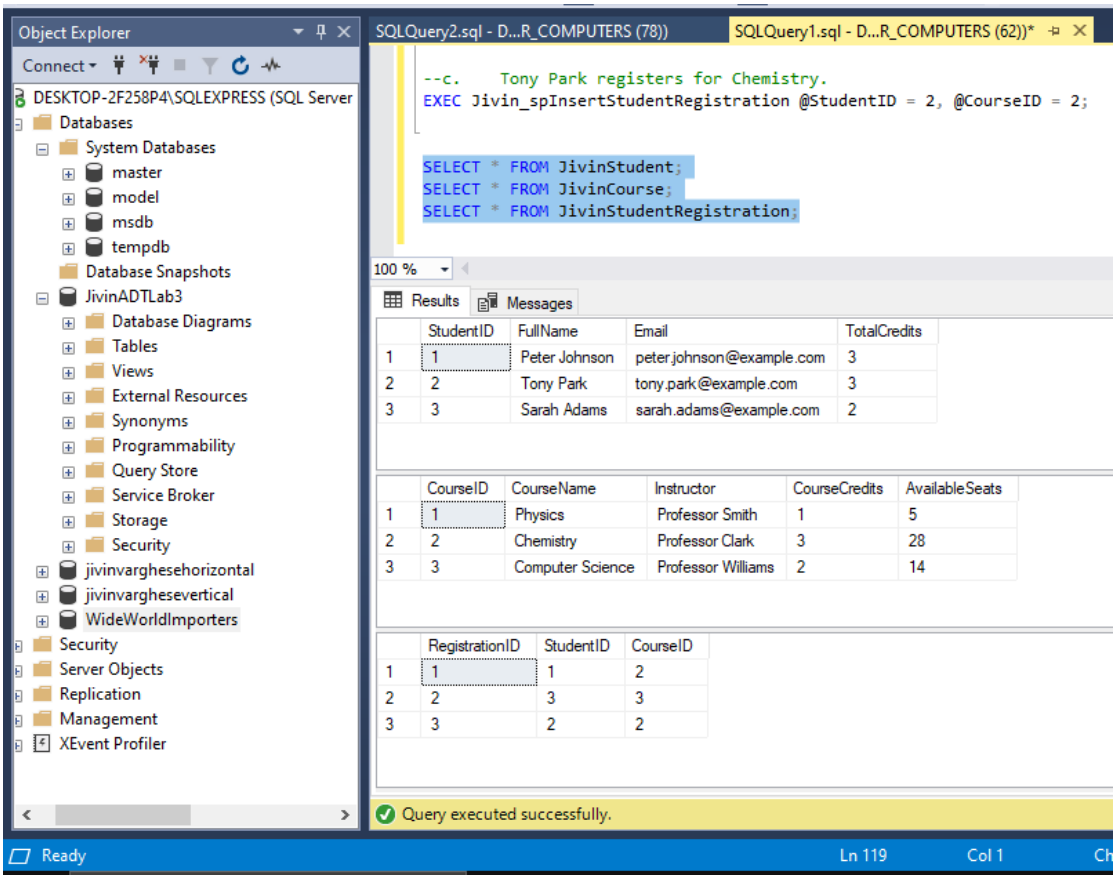


c. **Tony Park registers for Chemistry.**

CODE:

```
--b.     Sara Adams registers for Computer Science.

EXEC Jivin_spInsertStudentRegistration @StudentID = 3, @CourseID = 3;

SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:

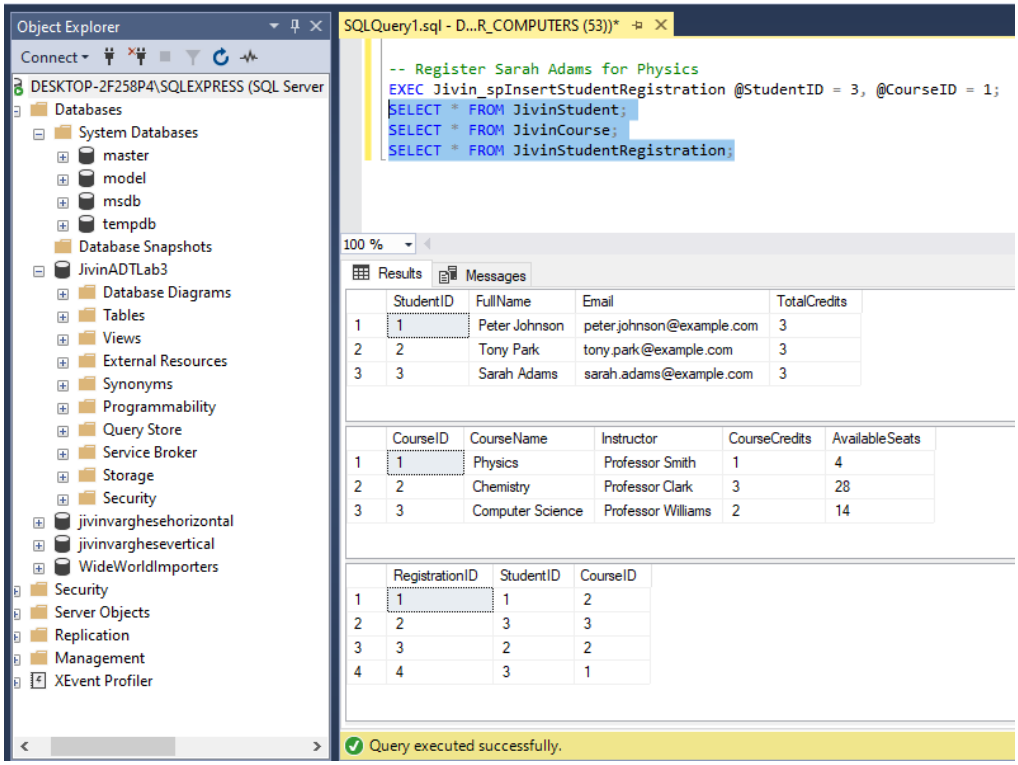


d. **Sarah Adams for Physics.**

CODE:

```
-- Register Sarah Adams for Physics
EXEC Jivin_spInsertStudentRegistration @StudentID = 3, @CourseID = 1;
SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:

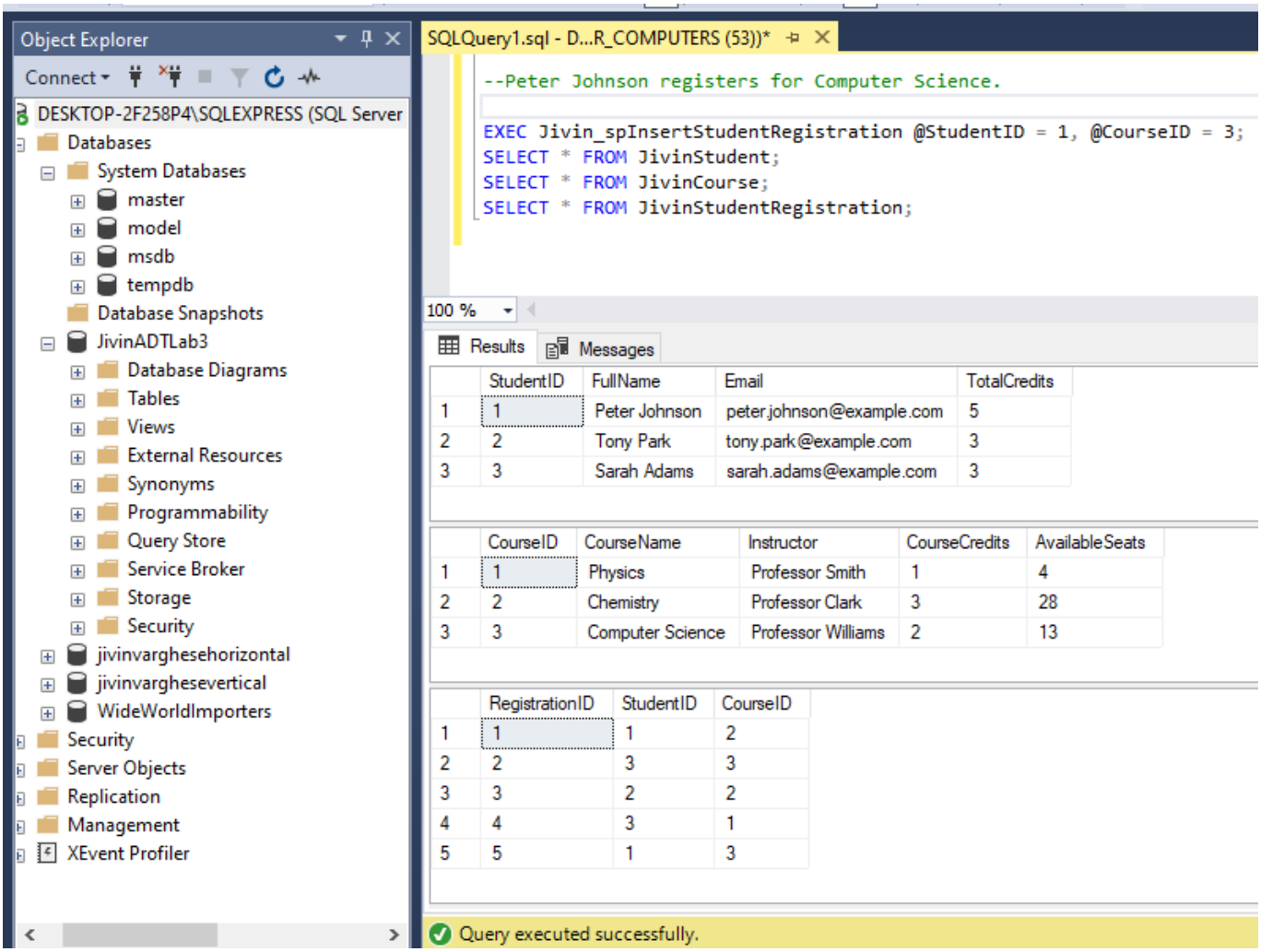


e. Peter Johnson registers for Computer Science.

CODE:

```
--Peter Johnson registers for Computer Science.

EXEC Jivin_spInsertStudentRegistration @StudentID = 1, @CourseID = 3;
SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:

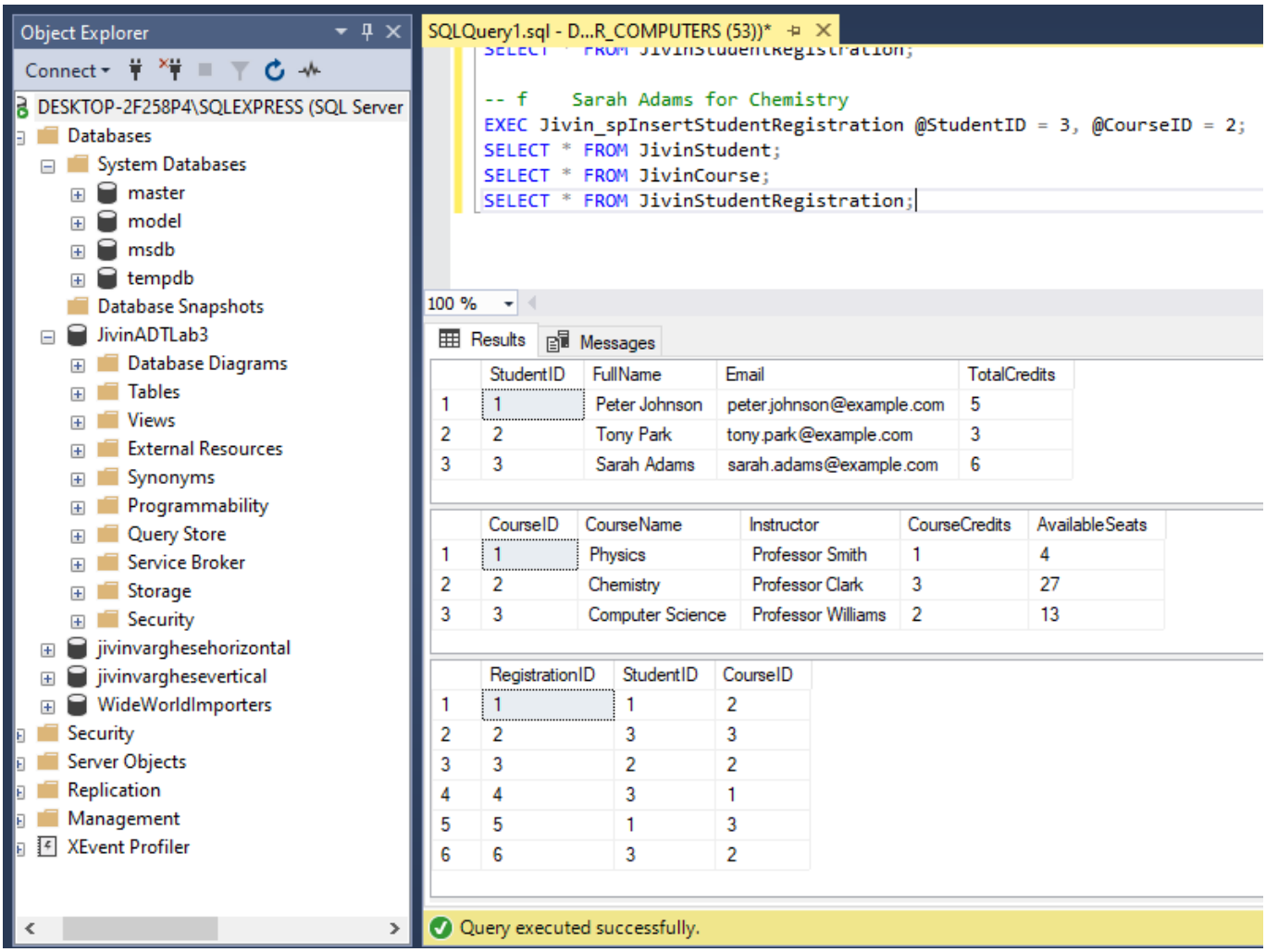


f. **Sarah Adams for Chemistry.**

CODE:

```
-- f     Sarah Adams for Chemistry
EXEC Jivin_spInsertStudentRegistration @StudentID = 3, @CourseID = 2;
SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
```

OUTPUT:


Object Explorer
Connect
DESKTOP-2F258P4\SQLEXPRESS (SQL Server
Databases
System Databases
master
model
msdb
tempdb
Database Snapshots
JivinADTLab3
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Query Store
Service Broker
Storage
Security
jivinvarghesehorizontal
jivinvarghesevertical
WideWorldImporters
Security
Server Objects
Replication
Management
XEvent Profiler
SQLQuery1.sql - D...R_COMPUTERS (53))*
-- f Sarah Adams for Chemistry
EXEC Jivin_spInsertStudentRegistration @StudentID = 3, @CourseID = 2;
SELECT * FROM JivinStudent;
SELECT * FROM JivinCourse;
SELECT * FROM JivinStudentRegistration;
100 %
Results
Messages
StudentID FullName Email TotalCredits
1 1 Peter Johnson peter.johnson@example.com 5
2 2 Tony Park tony.park@example.com 3
3 3 Sarah Adams sarah.adams@example.com 6
CourseID CourseName Instructor CourseCredits AvailableSeats
1 1 Physics Professor Smith 1 4
2 2 Chemistry Professor Clark 3 27
3 3 Computer Science Professor Williams 2 13
RegistrationID StudentID CourseID
1 1 1 2
2 2 3 3
3 3 2 2
4 4 3 1
5 5 1 3
6 6 3 2
Query executed successfully.