Quiz 11

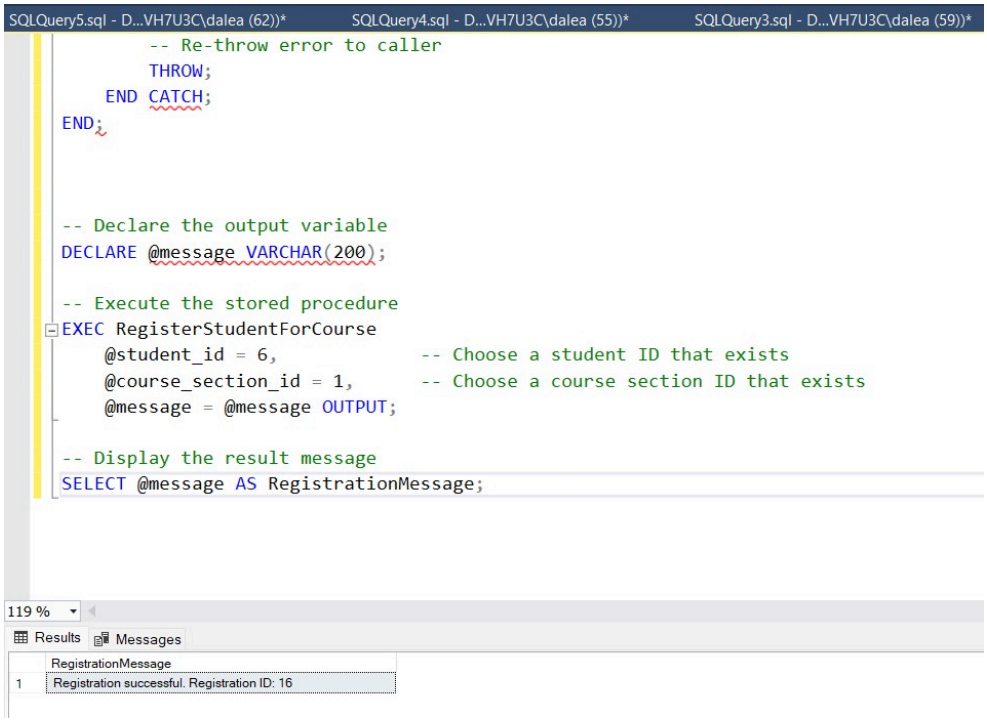
1.

```
-- Create stored procedure for student registration
CREATE OR ALTER PROCEDURE RegisterStudentForCourse
    @student_id INT,
    @course_section_id INT,
    @message VARCHAR(200) OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    -- Declare variables
    DECLARE @course_capacity INT = 40;
    DECLARE @current_enrollment INT;
    DECLARE @course_id INT;

    BEGIN TRY
        -- Start transaction
        BEGIN TRANSACTION;

        -- Get course_id for the section
        SELECT @course_id = CourseID
        FROM CourseSections
        WHERE CourseSectionID = @course_section_id;

        -- Verify student exists
        IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID =
@student_id)
        BEGIN
            THROW 50001, 'Student ID does not exist.', 1;
        END

        -- Verify course section exists
        IF NOT EXISTS (SELECT 1 FROM CourseSections WHERE
CourseSectionID = @course_section_id)
        BEGIN
            THROW 50002, 'Course section does not exist.', 1;
        END

        -- Check if student is already registered for this section
        IF EXISTS (
            SELECT 1
            FROM Registrations
            WHERE StudentID = @student_id
```

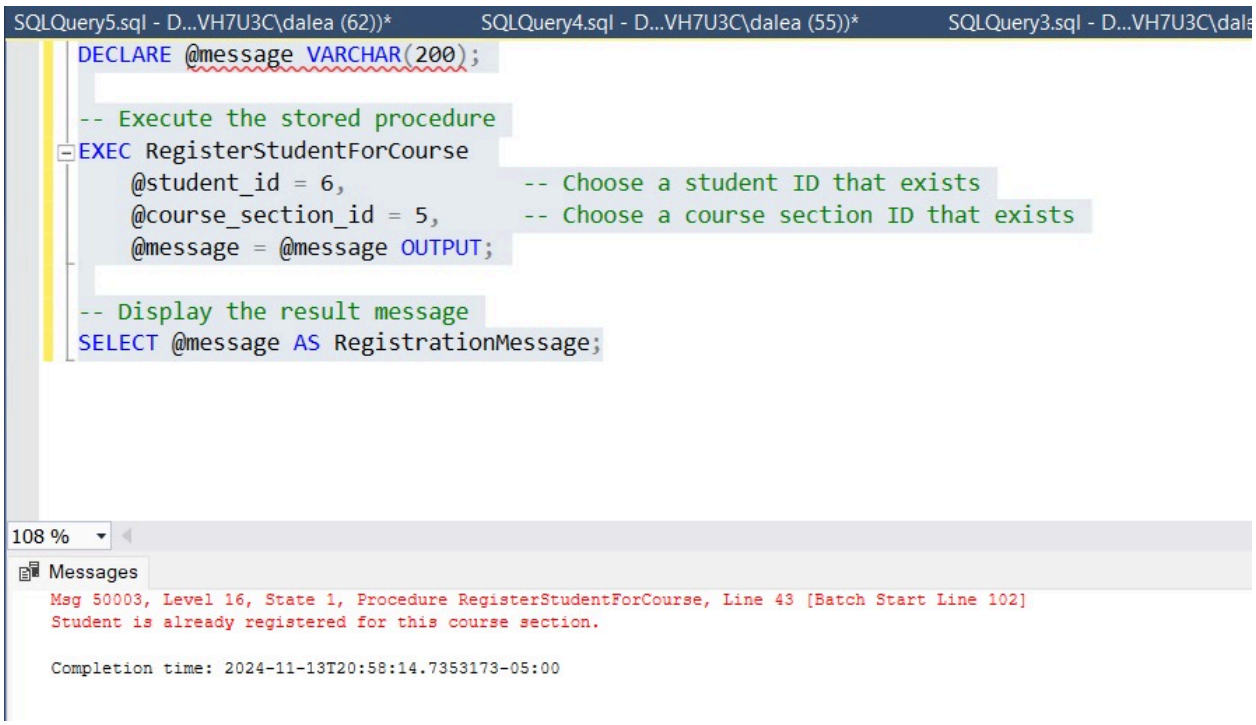
```
            AND CourseSectionID = @course_section_id
        )
        BEGIN
            THROW 50003, 'Student is already registered for this
course section.', 1;
        END

        -- Use CTE to check current enrollment
        ;WITH CurrentEnrollment AS (
            SELECT COUNT(*) as enrolled_count
            FROM Registrations
            WHERE CourseSectionID = @course_section_id
        )
        SELECT @current_enrollment = enrolled_count
        FROM CurrentEnrollment;

        -- Check if there's space available
        IF @current_enrollment >= @course_capacity
        BEGIN
            THROW 50004, 'Course section is full. Cannot register
more students.', 1;
        END

        -- Generate new registration ID
        DECLARE @new_registration_id INT;
        SELECT @new_registration_id = ISNULL(MAX(RegistrationID), 0)
+ 1
        FROM Registrations;

        -- Insert new registration
        INSERT INTO Registrations (
            RegistrationID,
            StudentID,
            CourseSectionID,
            Grade
        )
        VALUES (
            @new_registration_id,
            @student_id,
            @course_section_id,
            NULL  -- Grade will be assigned later
        );

        -- Set success message
        SET @message = 'Registration successful. Registration ID: ' +
```

```
                    CAST(@new_registration_id AS VARCHAR(10));

        -- Commit transaction
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- Rollback transaction on error
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        -- Set error message
        SET @message = ERROR_MESSAGE();

        -- Re-throw error to caller
        THROW;
    END CATCH;
END;



-- Declare the output variable
DECLARE @message VARCHAR(200);

-- Execute the stored procedure
EXEC RegisterStudentForCourse
    @student_id = 6,             -- Choose a student ID that exists
    @course_section_id = 5,      -- Choose a course section ID that
exists
    @message = @message OUTPUT;

-- Display the result message
SELECT @message AS RegistrationMessage;
```

For success

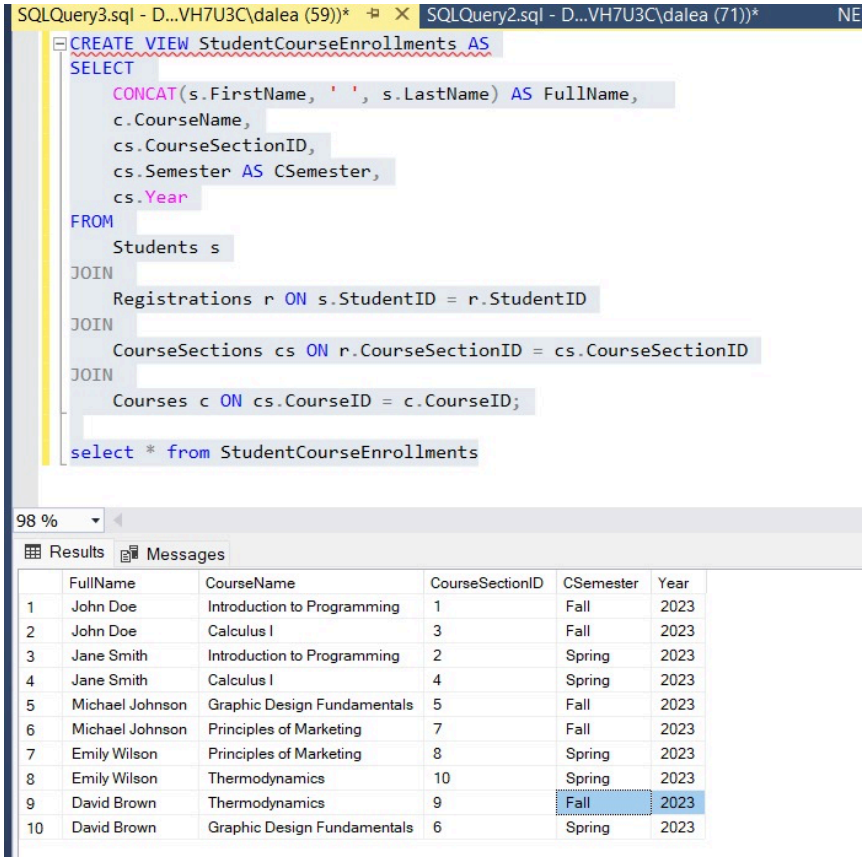

SQLQuery5.sql - D...VH7U3C\dalea (62))*
SQLQuery4.sql - D...VH7U3C\dalea (55))*
SQLQuery3.sql - D...VH7U3C\dalea (59))*
-- Re-throw error to caller
THROW;
END CATCH;
END;
-- Declare the output variable
DECLARE @message VARCHAR(200);
-- Execute the stored procedure
EXEC RegisterStudentForCourse
@student_id = 6, -- Choose a student ID that exists
@course_section_id = 1, -- Choose a course section ID that exists
@message = @message OUTPUT;
-- Display the result message
SELECT @message AS RegistrationMessage;
119 %
Results
Messages
RegistrationMessage
1
Registration successful. Registration ID: 16

For Failure


SQLQuery5.sql - D...VH7U3C\dalea (62))*
SQLQuery4.sql - D...VH7U3C\dalea (55))*
SQLQuery3.sql - D...VH7U3C\dala
DECLARE @message VARCHAR(200);
-- Execute the stored procedure
EXEC RegisterStudentForCourse
@student_id = 6, -- Choose a student ID that exists
@course_section_id = 5, -- Choose a course section ID that exists
@message = @message OUTPUT;
-- Display the result message
SELECT @message AS RegistrationMessage;
108 %
Messages
Msg 50003, Level 16, State 1, Procedure RegisterStudentForCourse, Line 43 [Batch Start Line 102]
Student is already registered for this course section.
Completion time: 2024-11-13T20:58:14.7353173-05:00

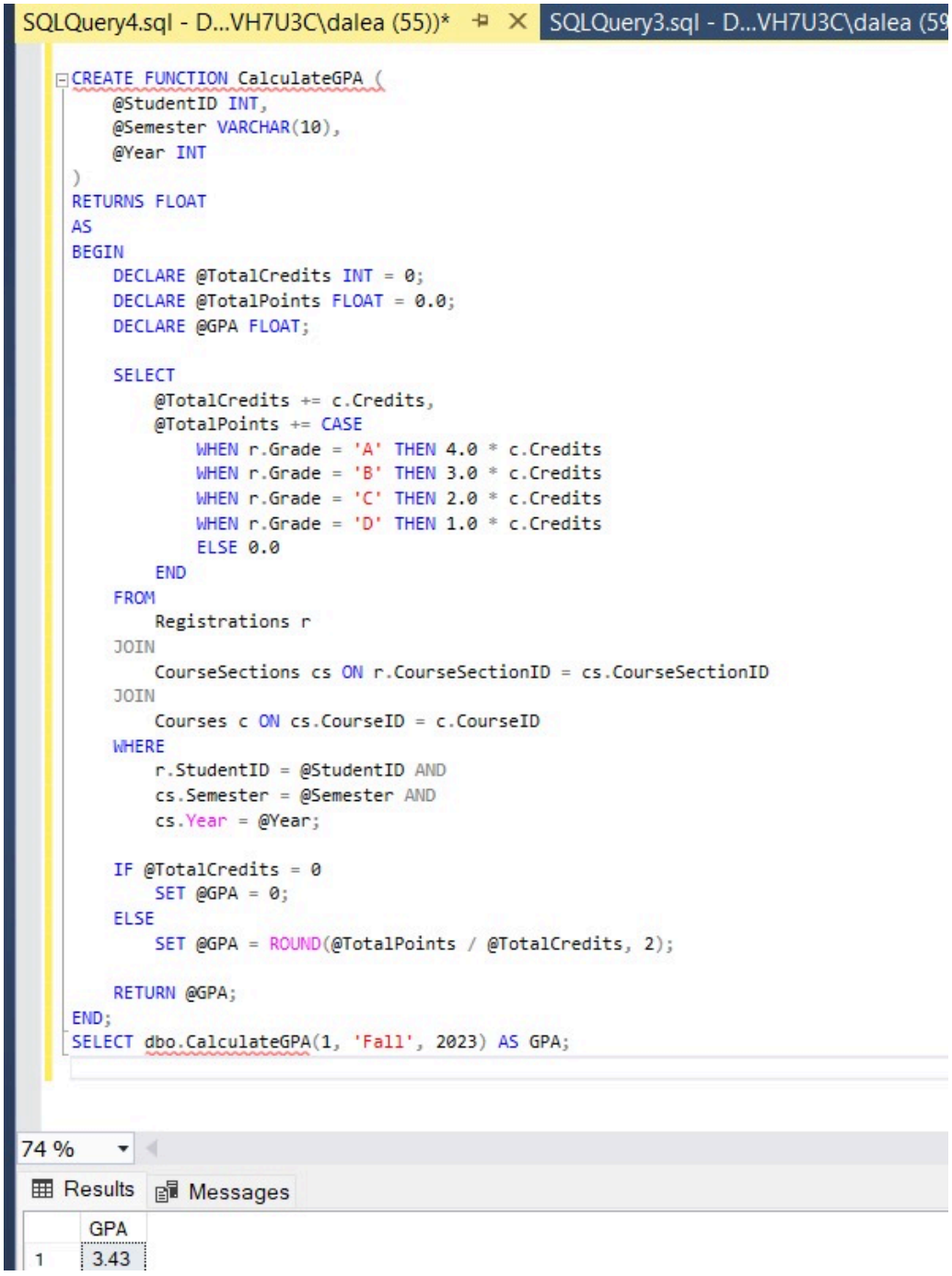
2.

```
CREATE VIEW StudentCourseEnrollments AS
SELECT
    CONCAT(s.FirstName, ' ', s.LastName) AS FullName,
    c.CourseName,
    cs.CourseSectionID,
    cs.Semester AS CSemester,
    cs.Year
FROM
    Students s
JOIN
    Registrations r ON s.StudentID = r.StudentID
JOIN
    CourseSections cs ON r.CourseSectionID = cs.CourseSectionID
JOIN
    Courses c ON cs.CourseID =c.CourseID;

select * from StudentCourseEnrollments
```



	FullName	CourseName	CourseSectionID	CSemester	Year
1	John Doe	Introduction to Programming	1	Fall	2023
2	John Doe	Calculus I	3	Fall	2023
3	Jane Smith	Introduction to Programming	2	Spring	2023
4	Jane Smith	Calculus I	4	Spring	2023
5	Michael Johnson	Graphic Design Fundamentals	5	Fall	2023
6	Michael Johnson	Principles of Marketing	7	Fall	2023
7	Emily Wilson	Principles of Marketing	8	Spring	2023
8	Emily Wilson	Thermodynamics	10	Spring	2023
9	David Brown	Thermodynamics	9	Fall	2023
10	David Brown	Graphic Design Fundamentals	6	Spring	2023

```
3.CREATE FUNCTION CalculateGPA (
    @StudentID INT,
    @Semester VARCHAR(10),
    @Year INT
)
RETURNS FLOAT
AS
BEGIN
    DECLARE @TotalCredits INT = 0;
    DECLARE @TotalPoints FLOAT = 0.0;
    DECLARE @GPA FLOAT;

    -- Calculate total credits and total grade points for the
specified semester and year
    SELECT
        @TotalCredits += c.Credits,
        @TotalPoints += CASE
            WHEN r.Grade = 'A' THEN 4.0 * c.Credits
            WHEN r.Grade = 'B' THEN 3.0 * c.Credits
            WHEN r.Grade = 'C' THEN 2.0 * c.Credits
            WHEN r.Grade = 'D' THEN 1.0 * c.Credits
            ELSE 0.0
        END
    FROM
        Registrations r
    JOIN
        CourseSections cs ON r.CourseSectionID = cs.CourseSectionID
    JOIN
        Courses c ON cs.CourseID = c.CourseID
    WHERE
        r.StudentID = @StudentID AND
        cs.Semester = @Semester AND
        cs.Year = @Year;

    -- Calculate GPA and round to 2 decimal places
    IF @TotalCredits = 0
        SET @GPA = 0; -- Avoid division by zero if no courses are
found
    ELSE
        SET @GPA = ROUND(@TotalPoints / @TotalCredits, 2);

    RETURN @GPA;
END;
```



4.

```
CREATE TABLE RegistrationAudit (
    AuditID INT PRIMARY KEY IDENTITY(1,1),
    StudentID INT,
```

```
    CourseSectionID INT,
    Action VARCHAR(10),
    Timestamp DATETIME DEFAULT GETDATE()
);

-- Create DML Trigger for Registrations table
CREATE TRIGGER LogRegistrationChanges
ON Registrations
AFTER INSERT, UPDATE
AS
BEGIN
    -- Log Inserted rows
    INSERT INTO RegistrationAudit (StudentID, CourseSectionID,
Action, Timestamp)
    SELECT i.StudentID, i.CourseSectionID, 'INSERT', GETDATE()
    FROM inserted i
    WHERE NOT EXISTS (SELECT 1 FROM deleted d WHERE i.RegistrationID
= d.RegistrationID);

    -- Log Updated rows
    INSERT INTO RegistrationAudit (StudentID, CourseSectionID,
Action, Timestamp)
    SELECT i.StudentID, i.CourseSectionID, 'UPDATE', GETDATE()
    FROM inserted i
    INNER JOIN deleted d ON i.RegistrationID = d.RegistrationID;
END;

-- Insert new registration records
INSERT INTO Registrations (RegistrationID, StudentID,
CourseSectionID, Grade)
VALUES (11, 6, 1, 'A');

INSERT INTO Registrations (RegistrationID, StudentID,
CourseSectionID, Grade)
VALUES (12, 7, 2, 'B');


-- Update an existing registration record
UPDATE Registrations
SET Grade = 'A'
WHERE RegistrationID = 11;

UPDATE Registrations
SET CourseSectionID = 3
WHERE RegistrationID = 12;
```

SQLQuery5.sql - D...VH7U3C\dalea (62))* | SQLQuery4.sql - D...VH7U3C\dalea (55))* | SQLQuery3.sql - D...VH7U3C\dalea (59))* | SQLQuery2.sql -

```sql
-- Create RegistrationAudit table to store audit logs
CREATE TABLE RegistrationAudit (
    AuditID INT PRIMARY KEY IDENTITY(1,1),
    StudentID INT,
    CourseSectionID INT,
    Action VARCHAR(10),
    Timestamp DATETIME DEFAULT GETDATE()
);


CREATE TRIGGER LogRegistrationChanges
ON Registrations
AFTER INSERT, UPDATE
AS
BEGIN
    -- Log Inserted rows
    INSERT INTO RegistrationAudit (StudentID, CourseSectionID, Action, Timestamp)
    SELECT i.StudentID, i.CourseSectionID, 'INSERT', GETDATE()
    FROM inserted i
    WHERE NOT EXISTS (SELECT 1 FROM deleted d WHERE i.RegistrationID = d.RegistrationID);

    -- Log Updated rows
    INSERT INTO RegistrationAudit (StudentID, CourseSectionID, Action, Timestamp)
    SELECT i.StudentID, i.CourseSectionID, 'UPDATE', GETDATE()
    FROM inserted i
    INNER JOIN deleted d ON i.RegistrationID = d.RegistrationID;
END;


INSERT INTO Registrations (RegistrationID, StudentID, CourseSectionID, Grade)
VALUES (11, 6, 1, 'A');

INSERT INTO Registrations (RegistrationID, StudentID, CourseSectionID, Grade)
VALUES (12, 7, 2, 'B');
```

```sql
    -- Log Updated rows
    INSERT INTO RegistrationAudit (StudentID, CourseSectionID, Action, Timestamp)
    SELECT i.StudentID, i.CourseSectionID, 'UPDATE', GETDATE()
    FROM inserted i
    INNER JOIN deleted d ON i.RegistrationID = d.RegistrationID;
END;


INSERT INTO Registrations (RegistrationID, StudentID, CourseSectionID, Grade)
VALUES (11, 6, 1, 'A');

INSERT INTO Registrations (RegistrationID, StudentID, CourseSectionID, Grade)
VALUES (12, 7, 2, 'B');

select * from RegistrationAudit

UPDATE Registrations
SET Grade = 'A'
WHERE RegistrationID = 11;

UPDATE Registrations
SET CourseSectionID = 3
WHERE RegistrationID = 12;
```

98 %

Results | Messages

	AuditID	StudentID	CourseSectionID	Action	Timestamp
1	1	6	1	INSERT	2024-11-13 20:48:44.730
2	2	7	2	INSERT	2024-11-13 20:48:44.730
3	3	6	1	UPDATE	2024-11-13 20:49:50.407
4	4	7	3	UPDATE	2024-11-13 20:49:50.407