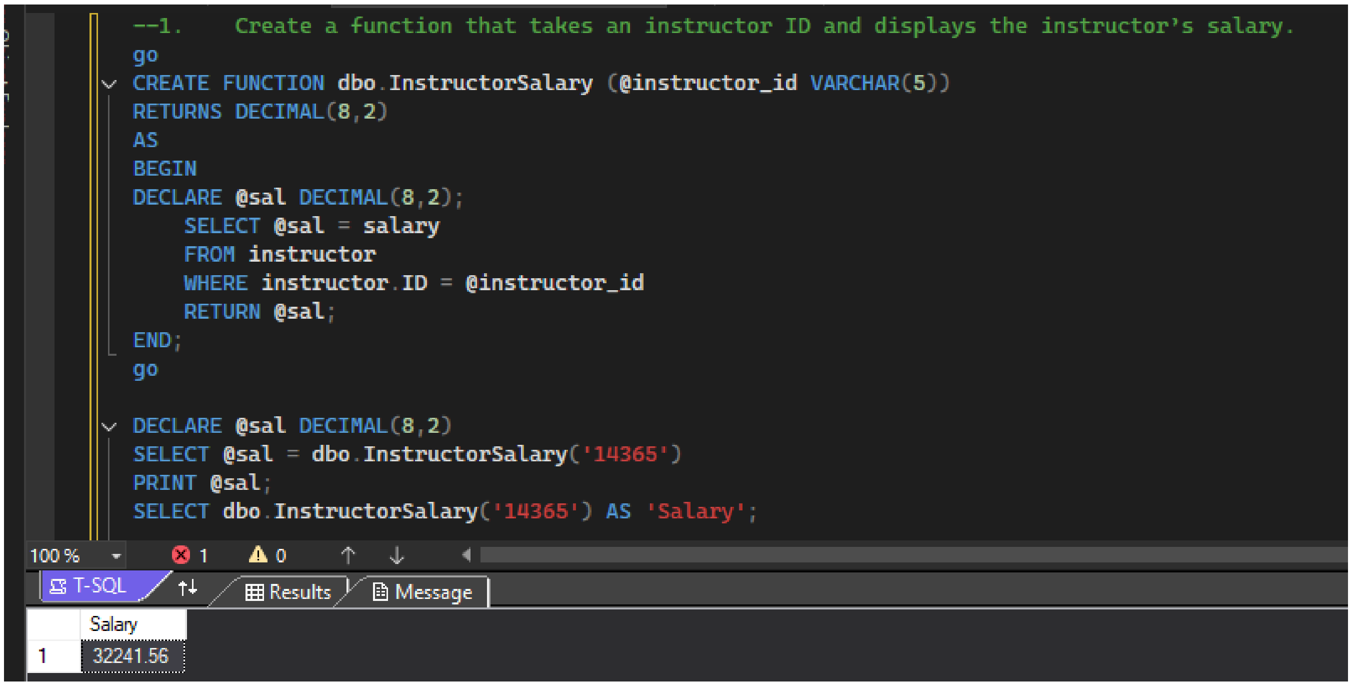
COSC 4301 – Database Theory and Practice
Lab 05

Construct the university database using DDL.sql and insert the data using largeRelationsInsertFile.sql.

Develop the following queries:

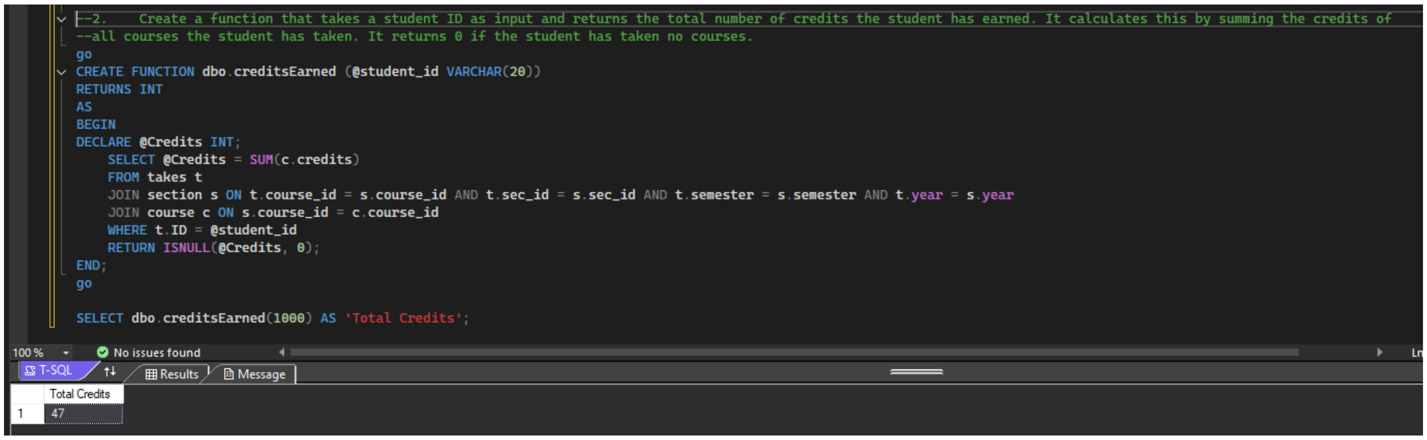
1. Create a function that takes an instructor ID and displays the instructor's salary.



```sql
--1.    Create a function that takes an instructor ID and displays the instructor's salary.
go
CREATE FUNCTION dbo.InstructorSalary (@instructor_id VARCHAR(5))
RETURNS DECIMAL(8,2)
AS
BEGIN
DECLARE @sal DECIMAL(8,2);
    SELECT @sal = salary
    FROM instructor
    WHERE instructor.ID = @instructor_id
    RETURN @sal;
END;
go

DECLARE @sal DECIMAL(8,2)
SELECT @sal = dbo.InstructorSalary('14365')
PRINT @sal;
SELECT dbo.InstructorSalary('14365') AS 'Salary';
```

```
100 %    ⊗ 1   ⚠ 0

T-SQL   Results   Message

  Salary
1 32241.56
```

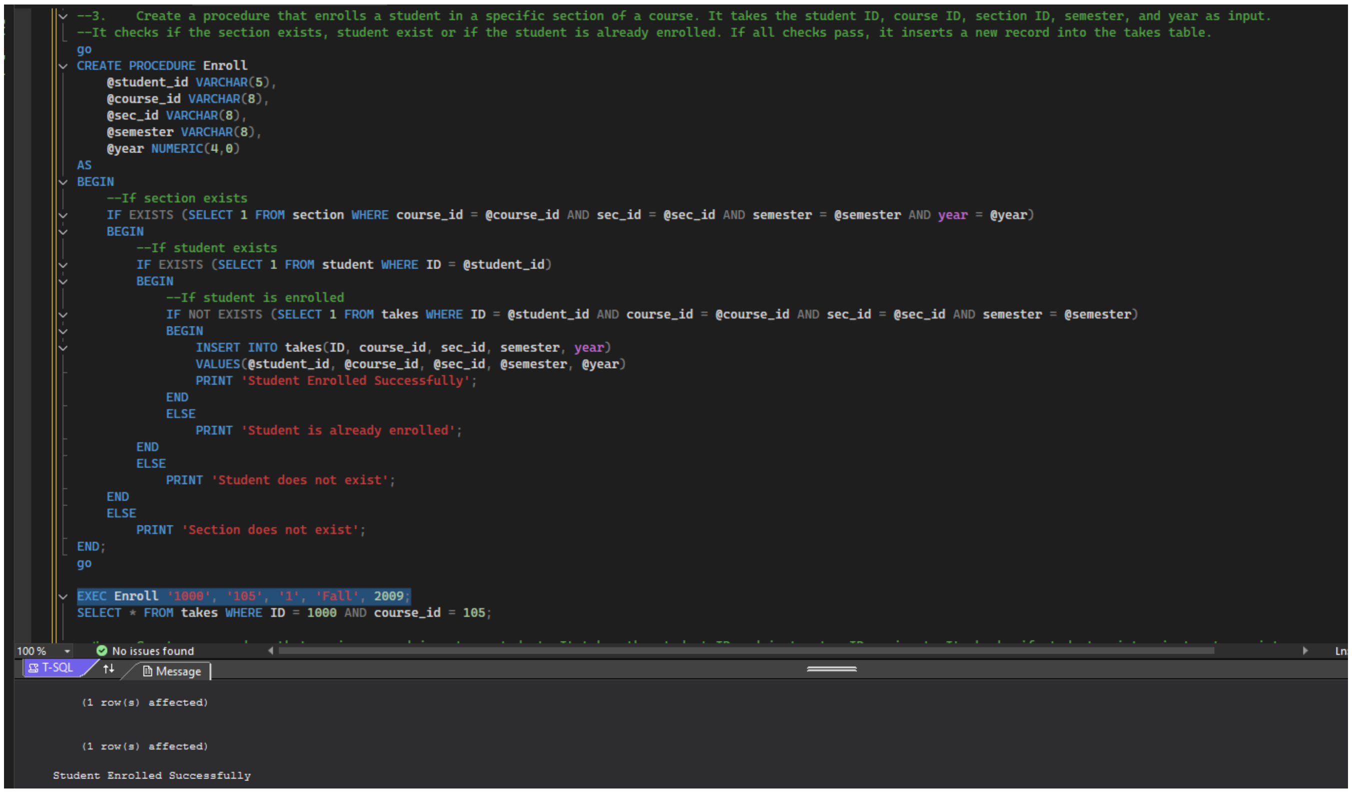
2. Create a function that takes a student ID as input and returns the total number of credits the student has earned. It calculates this by summing the credits of all courses the student has taken. It returns 0 if the student has taken no courses.



```sql
--2.    Create a function that takes a student ID as input and returns the total number of credits the student has earned. It calculates this by summing the credits of
--all courses the student has taken. It returns 0 if the student has taken no courses.
go
CREATE FUNCTION dbo.creditsEarned (@student_id VARCHAR(20))
RETURNS INT
AS
BEGIN
DECLARE @Credits INT;
    SELECT @Credits = SUM(c.credits)
    FROM takes t
    JOIN section s ON t.course_id = s.course_id AND t.sec_id = s.sec_id AND t.semester = s.semester AND t.year = s.year
    JOIN course c ON s.course_id = c.course_id
    WHERE t.ID = @student_id
    RETURN ISNULL(@Credits, 0);
END;
go

SELECT dbo.creditsEarned(1000) AS 'Total Credits';
```

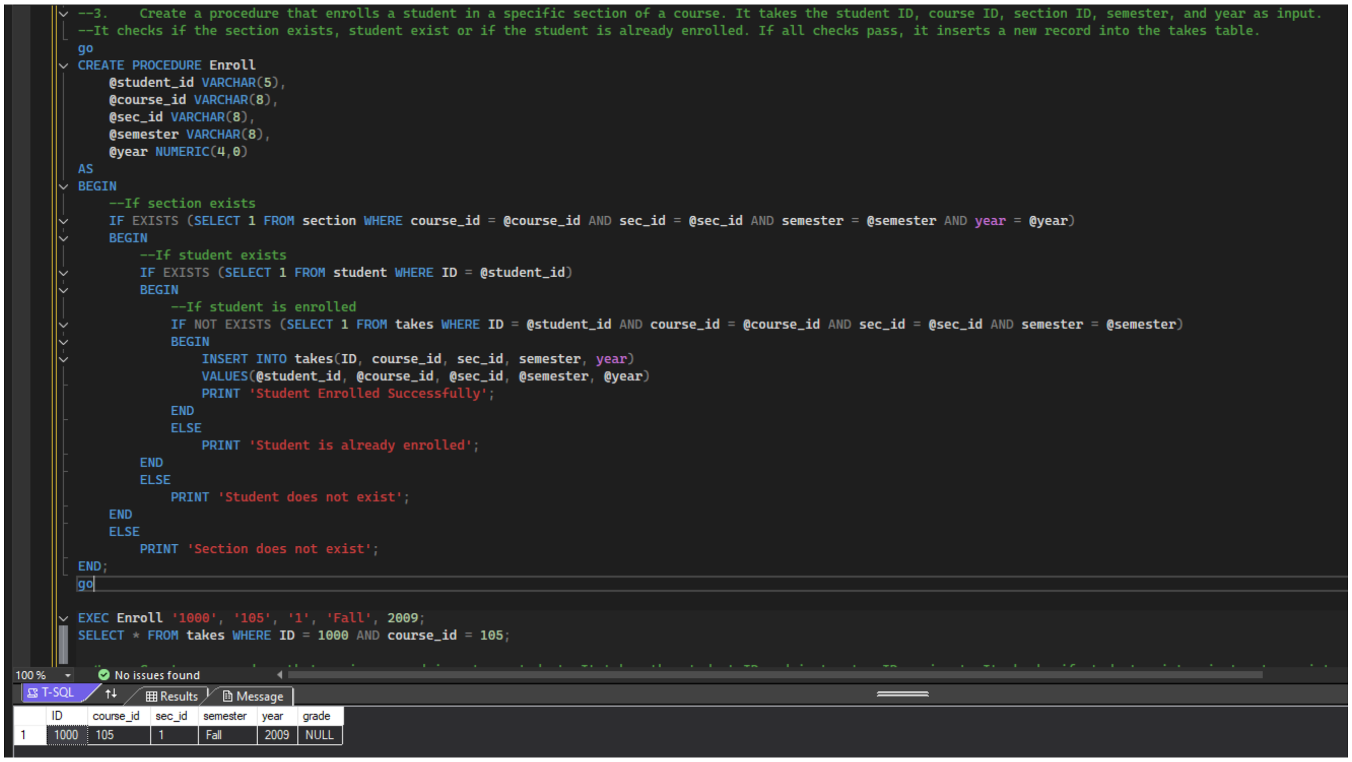
```
100 %    No issues found

T-SQL   Results   Message

  Total Credits
1 47
```

3. Create a procedure that enrolls a student in a specific section of a course. It takes the student ID, course ID, section ID, semester, and year as input. It checks if the section exists, student exist or if the student is already enrolled. If all checks pass, it inserts a new record into the takes table.



```sql
--3.   Create a procedure that enrolls a student in a specific section of a course. It takes the student ID, course ID, section ID, semester, and year as input.
--It checks if the section exists, student exist or if the student is already enrolled. If all checks pass, it inserts a new record into the takes table.
go
CREATE PROCEDURE Enroll
    @student_id VARCHAR(5),
    @course_id VARCHAR(8),
    @sec_id VARCHAR(8),
    @semester VARCHAR(8),
    @year NUMERIC(4,0)
AS
BEGIN
    --If section exists
    IF EXISTS (SELECT 1 FROM section WHERE course_id = @course_id AND sec_id = @sec_id AND semester = @semester AND year = @year)
    BEGIN
        --If student exists
        IF EXISTS (SELECT 1 FROM student WHERE ID = @student_id)
        BEGIN
            --If student is enrolled
            IF NOT EXISTS (SELECT 1 FROM takes WHERE ID = @student_id AND course_id = @course_id AND sec_id = @sec_id AND semester = @semester)
            BEGIN
                INSERT INTO takes(ID, course_id, sec_id, semester, year)
                VALUES(@student_id, @course_id, @sec_id, @semester, @year)
                PRINT 'Student Enrolled Successfully';
            END
            ELSE
                PRINT 'Student is already enrolled';
        END
        ELSE
            PRINT 'Student does not exist';
    END
    ELSE
        PRINT 'Section does not exist';
END;
go

EXEC Enroll '1000', '105', '1', 'Fall', 2009;
SELECT * FROM takes WHERE ID = 1000 AND course_id = 105;
```

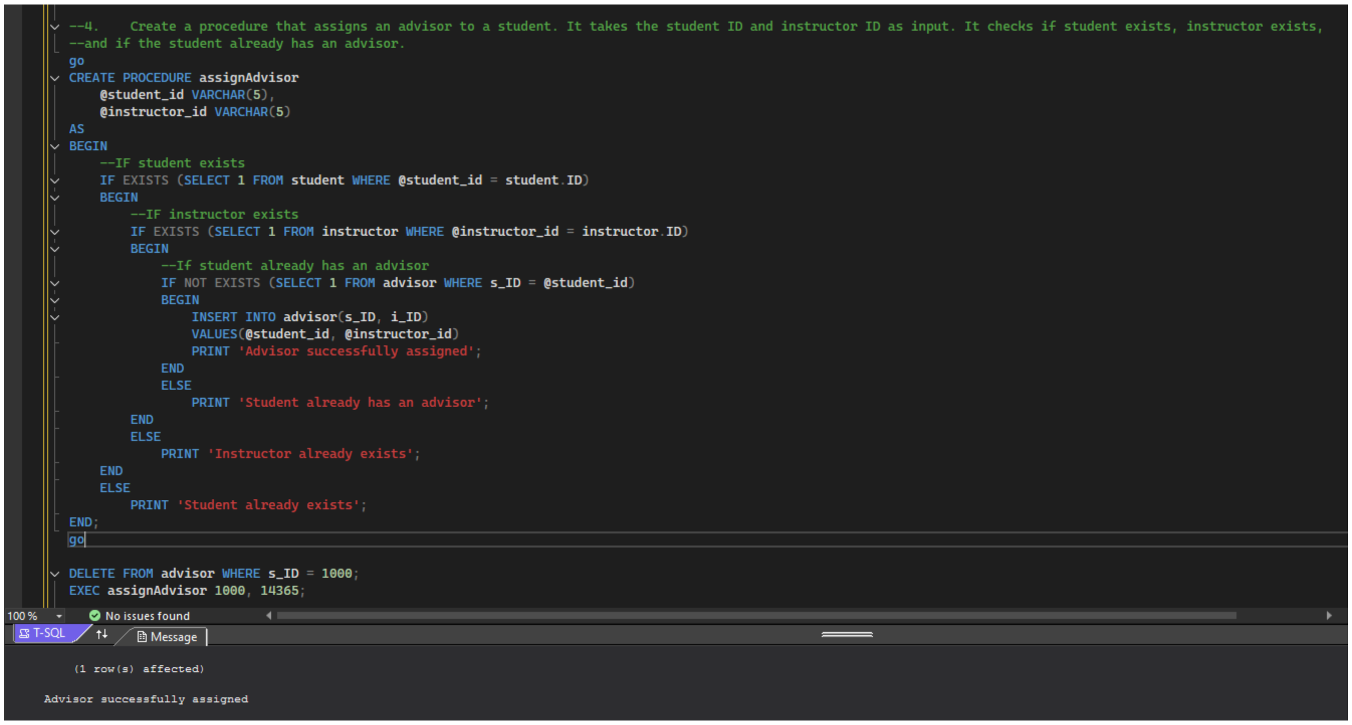
```
(1 row(s) affected)


(1 row(s) affected)

Student Enrolled Successfully
```



```sql
--3.   Create a procedure that enrolls a student in a specific section of a course. It takes the student ID, course ID, section ID, semester, and year as input.
--It checks if the section exists, student exist or if the student is already enrolled. If all checks pass, it inserts a new record into the takes table.
go
CREATE PROCEDURE Enroll
    @student_id VARCHAR(5),
    @course_id VARCHAR(8),
    @sec_id VARCHAR(8),
    @semester VARCHAR(8),
    @year NUMERIC(4,0)
AS
BEGIN
    --If section exists
    IF EXISTS (SELECT 1 FROM section WHERE course_id = @course_id AND sec_id = @sec_id AND semester = @semester AND year = @year)
    BEGIN
        --If student exists
        IF EXISTS (SELECT 1 FROM student WHERE ID = @student_id)
        BEGIN
            --If student is enrolled
            IF NOT EXISTS (SELECT 1 FROM takes WHERE ID = @student_id AND course_id = @course_id AND sec_id = @sec_id AND semester = @semester)
            BEGIN
                INSERT INTO takes(ID, course_id, sec_id, semester, year)
                VALUES(@student_id, @course_id, @sec_id, @semester, @year)
                PRINT 'Student Enrolled Successfully';
            END
            ELSE
                PRINT 'Student is already enrolled';
        END
        ELSE
            PRINT 'Student does not exist';
    END
    ELSE
        PRINT 'Section does not exist';
END;
go

EXEC Enroll '1000', '105', '1', 'Fall', 2009;
SELECT * FROM takes WHERE ID = 1000 AND course_id = 105;
```

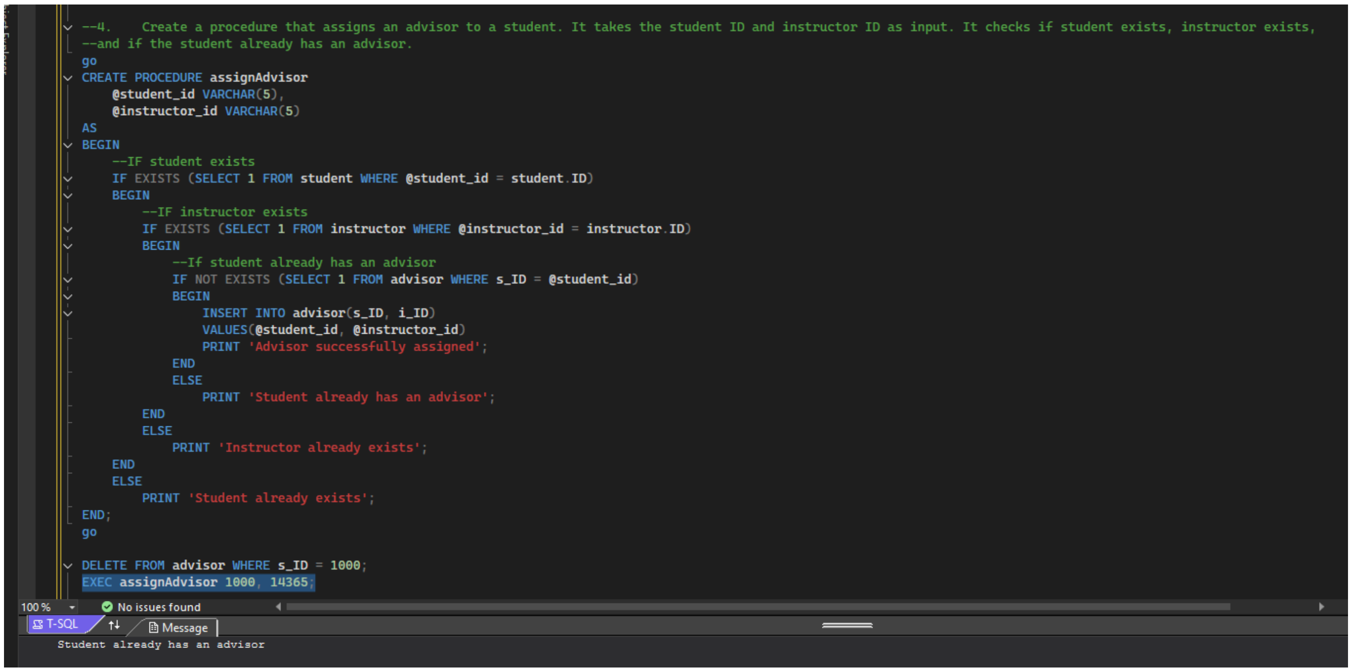
	ID	course_id	sec_id	semester	year	grade
1	1000	105	1	Fall	2009	NULL

4. Create a procedure that assigns an advisor to a student. It takes the student ID and instructor ID as input. It checks if student exists, instructor exists, and if the student already has an advisor.



```sql
--4.   Create a procedure that assigns an advisor to a student. It takes the student ID and instructor ID as input. It checks if student exists, instructor exists,
--and if the student already has an advisor.
go
CREATE PROCEDURE assignAdvisor
    @student_id VARCHAR(5),
    @instructor_id VARCHAR(5)
AS
BEGIN
    --IF student exists
    IF EXISTS (SELECT 1 FROM student WHERE @student_id = student.ID)
    BEGIN
        --IF instructor exists
        IF EXISTS (SELECT 1 FROM instructor WHERE @instructor_id = instructor.ID)
        BEGIN
            --If student already has an advisor
            IF NOT EXISTS (SELECT 1 FROM advisor WHERE s_ID = @student_id)
            BEGIN
                INSERT INTO advisor(s_ID, i_ID)
                VALUES(@student_id, @instructor_id)
                PRINT 'Advisor successfully assigned';
            END
            ELSE
                PRINT 'Student already has an advisor';
        END
        ELSE
            PRINT 'Instructor already exists';
    END
    ELSE
        PRINT 'Student already exists';
END;
go

DELETE FROM advisor WHERE s_ID = 1000;
EXEC assignAdvisor 1000, 14365;
```

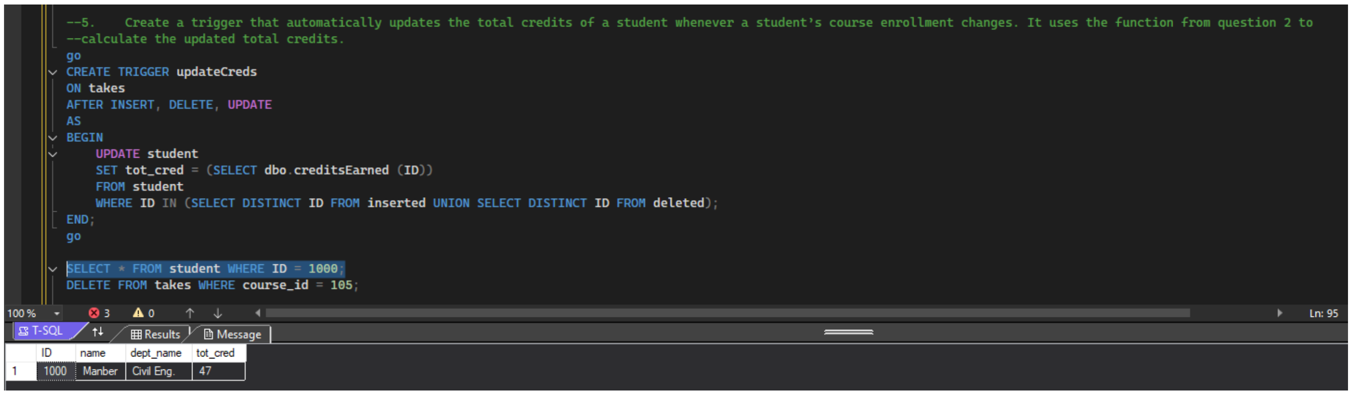
```
(1 row(s) affected)

Advisor successfully assigned
```



```sql
--4.   Create a procedure that assigns an advisor to a student. It takes the student ID and instructor ID as input. It checks if student exists, instructor exists,
--and if the student already has an advisor.
go
CREATE PROCEDURE assignAdvisor
    @student_id VARCHAR(5),
    @instructor_id VARCHAR(5)
AS
BEGIN
    --IF student exists
    IF EXISTS (SELECT 1 FROM student WHERE @student_id = student.ID)
    BEGIN
        --IF instructor exists
        IF EXISTS (SELECT 1 FROM instructor WHERE @instructor_id = instructor.ID)
        BEGIN
            --If student already has an advisor
            IF NOT EXISTS (SELECT 1 FROM advisor WHERE s_ID = @student_id)
            BEGIN
                INSERT INTO advisor(s_ID, i_ID)
                VALUES(@student_id, @instructor_id)
                PRINT 'Advisor successfully assigned';
            END
            ELSE
                PRINT 'Student already has an advisor';
        END
        ELSE
            PRINT 'Instructor already exists';
    END
    ELSE
        PRINT 'Student already exists';
END;
go

DELETE FROM advisor WHERE s_ID = 1000;
EXEC assignAdvisor 1000, 14365;
```

```
Student already has an advisor
```

5. Create a trigger that automatically updates the total credits of a student whenever a student's course enrollment changes. It uses the function from question 2 to calculate the updated total credits.

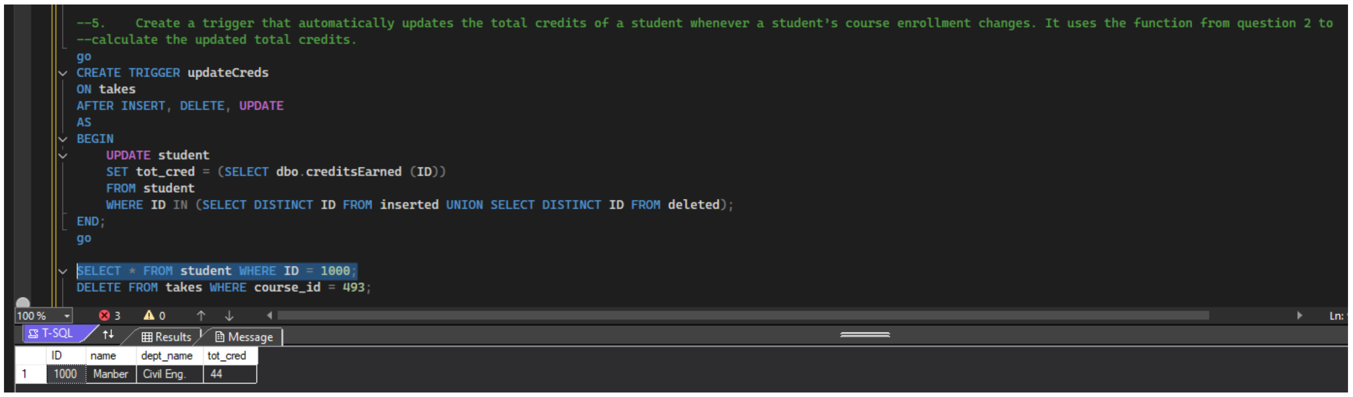


```sql
--5.   Create a trigger that automatically updates the total credits of a student whenever a student's course enrollment changes. It uses the function from question 2 to
--calculate the updated total credits.
go
CREATE TRIGGER updateCreds
ON takes
AFTER INSERT, DELETE, UPDATE
AS
BEGIN
    UPDATE student
    SET tot_cred = (SELECT dbo.creditsEarned (ID))
    FROM student
    WHERE ID IN (SELECT DISTINCT ID FROM inserted UNION SELECT DISTINCT ID FROM deleted);
END;
go

SELECT * FROM student WHERE ID = 1000;
DELETE FROM takes WHERE course_id = 105;
```

100 % 3 0 Ln: 95

T-SQL Results Message

	ID	name	dept_name	tot_cred
1	1000	Manber	Civil Eng.	47



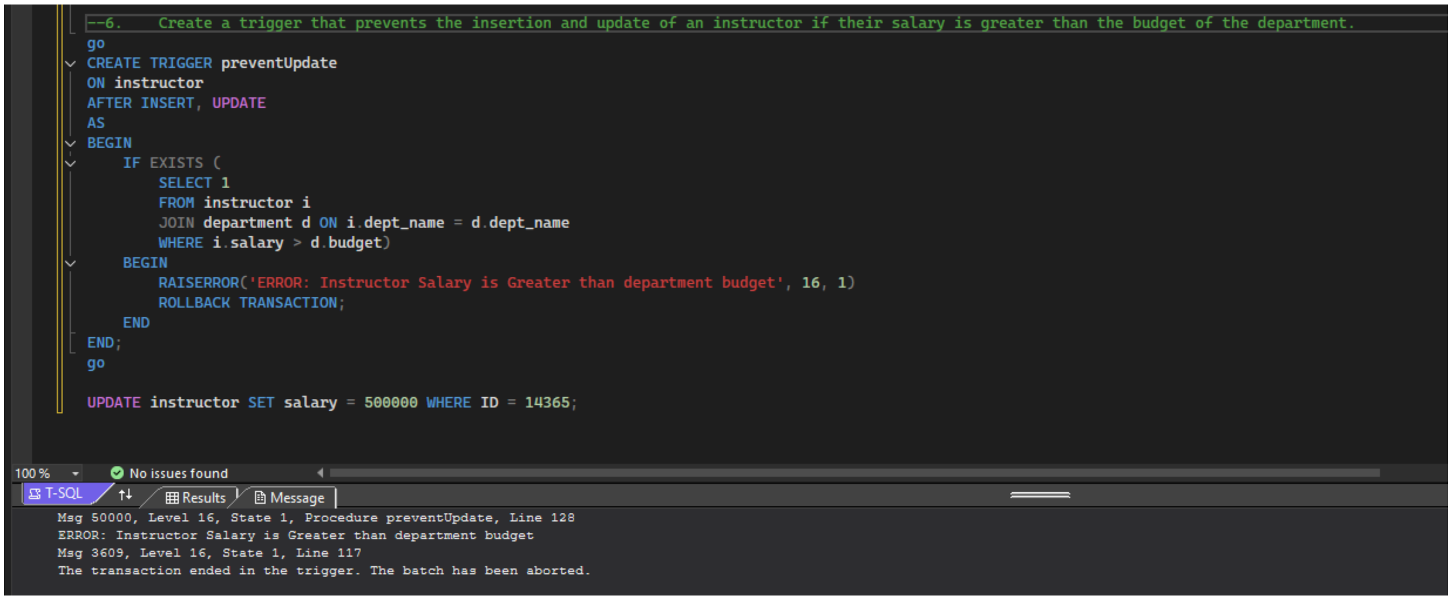
```sql
--5.   Create a trigger that automatically updates the total credits of a student whenever a student's course enrollment changes. It uses the function from question 2 to
--calculate the updated total credits.
go
CREATE TRIGGER updateCreds
ON takes
AFTER INSERT, DELETE, UPDATE
AS
BEGIN
    UPDATE student
    SET tot_cred = (SELECT dbo.creditsEarned (ID))
    FROM student
    WHERE ID IN (SELECT DISTINCT ID FROM inserted UNION SELECT DISTINCT ID FROM deleted);
END;
go

SELECT * FROM student WHERE ID = 1000;
DELETE FROM takes WHERE course_id = 493;
```

100 % 3 0

T-SQL Results Message

	ID	name	dept_name	tot_cred
1	1000	Manber	Civil Eng.	44

6. Create a trigger that prevents the insertion and update of an instructor if their salary is greater than the budget of the department.



```sql
--6.   Create a trigger that prevents the insertion and update of an instructor if their salary is greater than the budget of the department.
go
CREATE TRIGGER preventUpdate
ON instructor
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM instructor i
        JOIN department d ON i.dept_name = d.dept_name
        WHERE i.salary > d.budget)
    BEGIN
        RAISERROR('ERROR: Instructor Salary is Greater than department budget', 16, 1)
        ROLLBACK TRANSACTION;
    END
END;
go

UPDATE instructor SET salary = 500000 WHERE ID = 14365;
```

100 % No issues found

T-SQL Results Message

```
Msg 50000, Level 16, State 1, Procedure preventUpdate, Line 128
ERROR: Instructor Salary is Greater than department budget
Msg 3609, Level 16, State 1, Line 117
The transaction ended in the trigger. The batch has been aborted.
```

Code:

```
--1.	Create a function that takes an instructor ID and displays the instructor's salary.

go

CREATE FUNCTION dbo.InstructorSalary (@instructor_id VARCHAR(5))

RETURNS DECIMAL(8,2)

AS

BEGIN

DECLARE @sal DECIMAL(8,2);

	SELECT @sal = salary

	FROM instructor

	WHERE instructor.ID = @instructor_id

	RETURN @sal;

END;

go


DECLARE @sal DECIMAL(8,2)

SELECT @sal = dbo.InstructorSalary('14365')

PRINT @sal;

SELECT dbo.InstructorSalary('14365') AS 'Salary';


--2.	Create a function that takes a student ID as input and returns the total number of credits the student has earned. It calculates this by summing the credits of

--all courses the student has taken. It returns 0 if the student has taken no courses.

go

CREATE FUNCTION dbo.creditsEarned (@student_id VARCHAR(20))

RETURNS INT

AS

BEGIN

DECLARE @Credits INT;

	SELECT @Credits = SUM(c.credits)
```

```sql
	FROM takes t

	JOIN section s ON t.course_id = s.course_id AND t.sec_id = s.sec_id AND t.semester = s.semester AND t.year = s.year

	JOIN course c ON s.course_id = c.course_id

	WHERE t.ID = @student_id

	RETURN ISNULL(@Credits, 0);

END;

go


SELECT dbo.creditsEarned(1000) AS 'Total Credits';


--3.	Create a procedure that enrolls a student in a specific section of a course. It takes the student ID, course ID, section ID, semester, and year as input.

--It checks if the section exists, student exist or if the student is already enrolled. If all checks pass, it inserts a new record into the takes table.

go

CREATE PROCEDURE Enroll

	@student_id VARCHAR(5),

	@course_id VARCHAR(8),

	@sec_id VARCHAR(8),

	@semester VARCHAR(8),

	@year NUMERIC(4,0)

AS

BEGIN

	--If section exists

	IF EXISTS (SELECT 1 FROM section WHERE course_id = @course_id AND sec_id = @sec_id AND semester = @semester AND year = @year)

	BEGIN

		--If student exists

		IF EXISTS (SELECT 1 FROM student WHERE ID = @student_id)

		BEGIN
```

```
                        --If student is enrolled

                        IF NOT EXISTS (SELECT 1 FROM takes WHERE ID = @student_id AND course_id = 
@course_id AND sec_id = @sec_id AND semester = @semester)

                        BEGIN

                                INSERT INTO takes(ID, course_id, sec_id, semester, year)

                                VALUES(@student_id, @course_id, @sec_id, @semester, @year)

                                PRINT 'Student Enrolled Successfully';

                        END

                        ELSE

                                PRINT 'Student is already enrolled';

                END

                ELSE

                        PRINT 'Student does not exist';

        END

        ELSE

                PRINT 'Section does not exist';

END;

go


EXEC Enroll '1000', '105', '1', 'Fall', 2009;

SELECT * FROM takes WHERE ID = 1000 AND course_id = 105;


--4.    Create a procedure that assigns an advisor to a student. It takes the student ID and instructor ID as input. It 
checks if student exists, instructor exists,

--and if the student already has an advisor.

go

CREATE PROCEDURE assignAdvisor

        @student_id VARCHAR(5),

        @instructor_id VARCHAR(5)

AS
```

```
BEGIN
	--IF student exists
	IF EXISTS (SELECT 1 FROM student WHERE @student_id = student.ID)
	BEGIN
		--IF instructor exists
		IF EXISTS (SELECT 1 FROM instructor WHERE @instructor_id = instructor.ID)
		BEGIN
			--If student already has an advisor
			IF NOT EXISTS (SELECT 1 FROM advisor WHERE s_ID = @student_id)
			BEGIN
				INSERT INTO advisor(s_ID, i_ID)
				VALUES(@student_id, @instructor_id)
				PRINT 'Advisor successfully assigned';
			END
			ELSE
				PRINT 'Student already has an advisor';
		END
		ELSE
			PRINT 'Instructor already exists';
	END
	ELSE
		PRINT 'Student already exists';
END;
go


DELETE FROM advisor WHERE s_ID = 1000;
EXEC assignAdvisor 1000, 14365;
```

--5.	Create a trigger that automatically updates the total credits of a student whenever a student's course enrollment changes. It uses the function from question 2 to

```
--calculate the updated total credits.

go

CREATE TRIGGER updateCreds

ON takes

AFTER INSERT, DELETE, UPDATE

AS

BEGIN

	UPDATE student

	SET tot_cred = (SELECT dbo.creditsEarned (ID))

	FROM student

	WHERE ID IN (SELECT DISTINCT ID FROM inserted UNION SELECT DISTINCT ID FROM
deleted);

END;

go


SELECT * FROM student WHERE ID = 1000;

DELETE FROM takes WHERE course_id = 493;


--6.	Create a trigger that prevents the insertion and update of an instructor if their salary is greater than the budget
of the department.

go

CREATE TRIGGER preventUpdate

ON instructor

AFTER INSERT, UPDATE

AS

BEGIN

	IF EXISTS (

		SELECT 1

		FROM instructor i

		JOIN department d ON i.dept_name = d.dept_name
```

```
        WHERE i.salary > d.budget)
    BEGIN
        RAISERROR('ERROR: Instructor Salary is Greater than department budget', 16, 1)
        ROLLBACK TRANSACTION;
    END
END;
go

UPDATE instructor SET salary = 500000 WHERE ID = 14365;
```