Project 3

Your task:

You will continue working with the tables you created in project 2 in CSE581Projects database.

Create 4 *Stored Procedures* which can be executed by database role "Graders" (GRANT EXECUTE ON SCHEMA::[yourSchema] TO Graders;) :

- SP to use cursor(s)
- SP to update data in a table (perform validation)
- SP to delete data from a table
- (5% bonus point) 1 SP of your own choice, performing a business action[1].

Create 1 *Function* which can be executed by "Graders". Not the function you did for the lab.

You will also *create a view* (named as "Benefits") which can be viewed by "Graders" (GRANT SELECT ON SCHEMA::[yourSchema] TO Graders;) that shows every employee's name, ID, benefit's type, benefit coverage, employee premium and employer premium.

Your deliverables will be:

- scripts used to create all of the DB objects described above; each object needs a **short** explanation as to its purpose or goal (i.e. "This SP does this, that and the other thing..")
- screenshots demonstrating that the SPs work as expected (including valid/invalid inputs). Refer to our SP lab for more details.
- a text file with SELECTS against your views, EXECUTE against your function & stored procedures

Requirements:

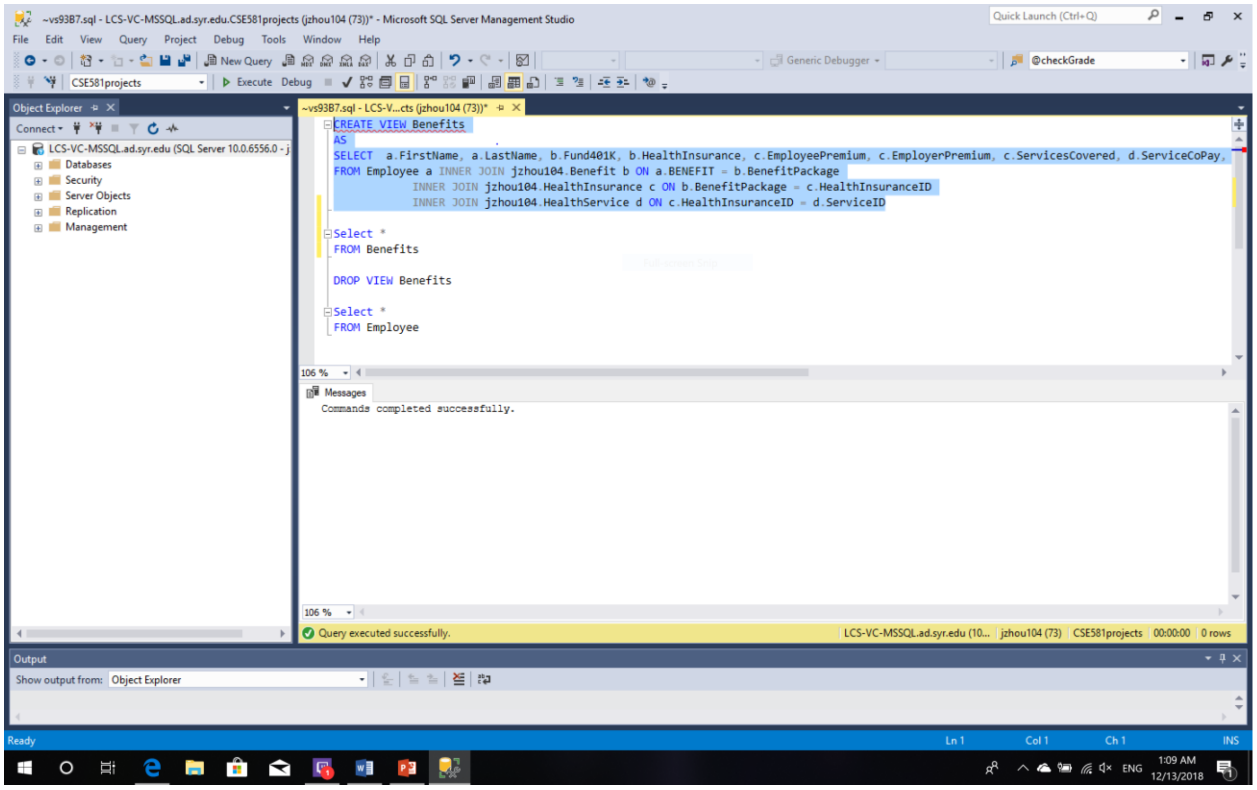
1. You **shall** create 1 view.
2. You **shall** create 4 stored procedures.
3. You **shall** create 1 function.
4. You **shall** submit all of your SQL code, used to create the DB objects (view, SPs and function).
5. You **shall** submit a short (a single sentence) explanation of what the purpose of each of the DB objects is.
6. You **shall** execute all of the SPs/function (provide screenshots of execution the way you did in the previous labs (SP & function labs)).
7. You **shall** submit a text file that will run SELECTs against of your view and execution of your SPs and function.

[1] You will get 5% bonus if this is a valid action within the scope of the business problem, and the SP is reasonably complex. In other words, a 4 line single-table select SP will get you nothing.

VIEW – View benefit packages

```
CREATE VIEW Benefits
AS
SELECT  a.FirstName, a.LastName, b.Fund401K, b.HealthInsurance, c.EmployeePremium,
c.EmployerPremium, c.ServicesCovered, d.ServiceCoPay, d.ServiceFee,
d.ServiceFullCovered, d.ServiceName, d.ServiceNotCovered, d.ServicePartiallyCovered
FROM Employee a INNER JOIN jzhou104.Benefit b ON a.BENEFIT = b.BenefitPackage
                 INNER JOIN jzhou104.HealthInsurance c ON b.BenefitPackage =
c.HealthInsuranceID
                 INNER JOIN jzhou104.HealthService d ON c.HealthInsuranceID =
d.ServiceID


Select *
FROM Benefits

DROP VIEW Benefits

Select *
FROM Employee
```

This view will fetch all relative information about the employee's benefit package.
All benefit packages are LOCKED and GROUPED so they are not customized for everybody.



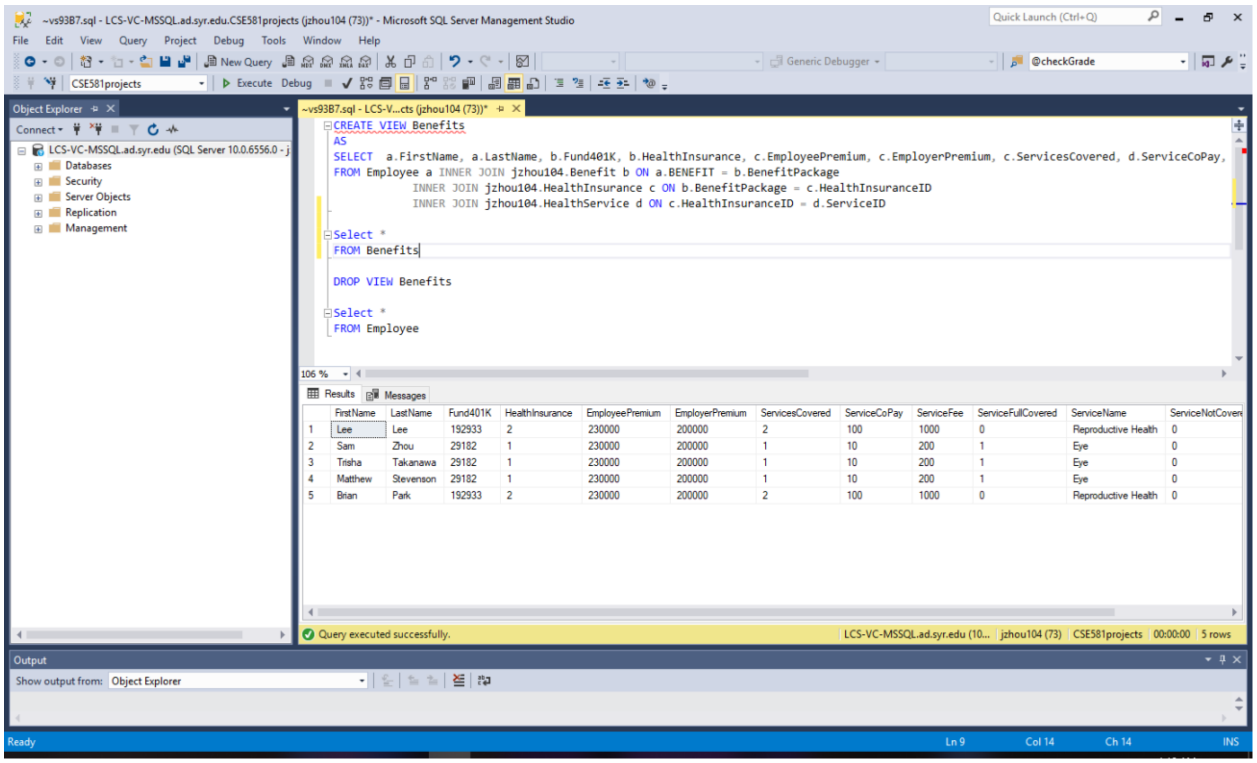
~vs93B7.sql - LCS-VC-MSSQL.ad.syr.edu.CSE581projects (jzhou104 (73))* - Microsoft SQL Server Management Studio

```
CREATE VIEW Benefits
AS
SELECT  a.FirstName, a.LastName, b.Fund401K, b.HealthInsurance, c.EmployeePremium, c.EmployerPremium, c.ServicesCovered, d.ServiceCoPay,
FROM Employee a INNER JOIN jzhou104.Benefit b ON a.BENEFIT = b.BenefitPackage
            INNER JOIN jzhou104.HealthInsurance c ON b.BenefitPackage = c.HealthInsuranceID
            INNER JOIN jzhou104.HealthService d ON c.HealthInsuranceID = d.ServiceID

Select *
FROM Benefits

DROP VIEW Benefits

Select *
FROM Employee
```

Messages

```
Commands completed successfully.
```

Query executed successfully. | LCS-VC-MSSQL.ad.syr.edu (10... | jzhou104 (73) | CSE581projects | 00:00:00 | 0 rows



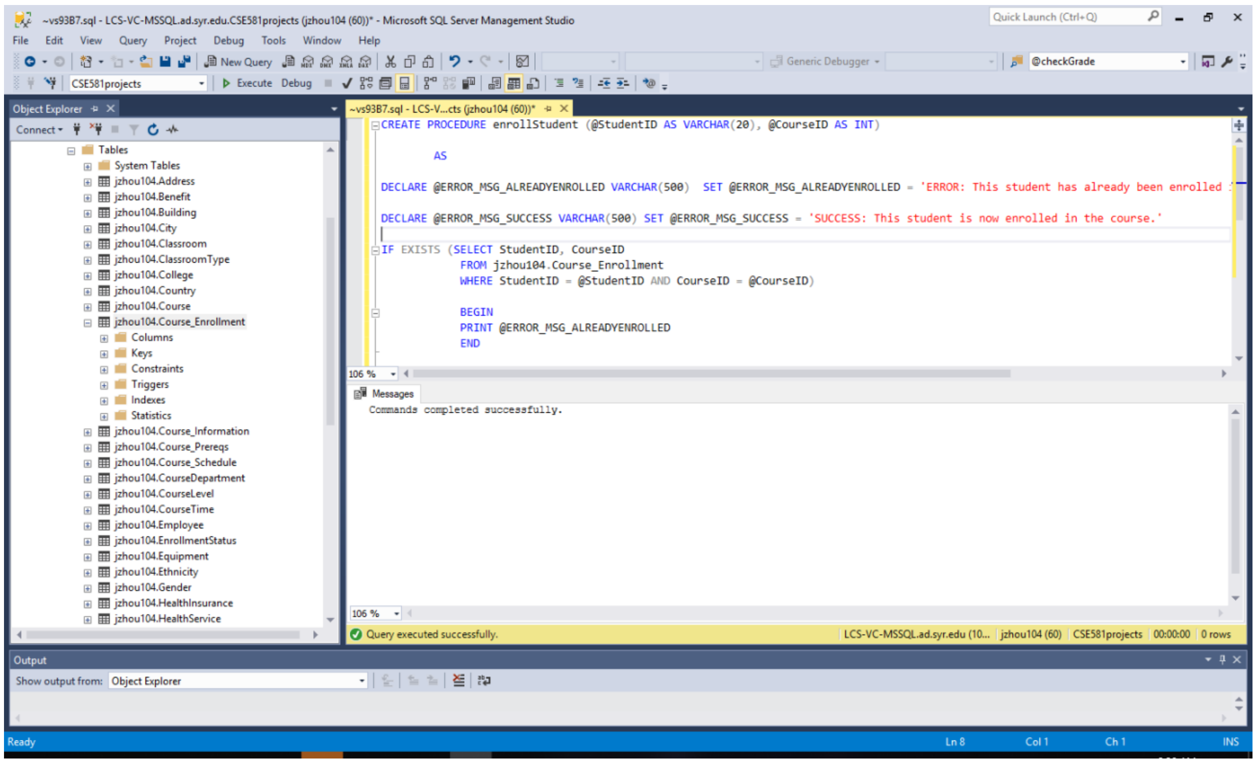
```sql
CREATE VIEW Benefits
AS
SELECT  a.FirstName, a.LastName, b.Fund401K, b.HealthInsurance, c.EmployeePremium, c.EmployerPremium, c.ServicesCovered, d.ServiceCoPay,
FROM Employee a INNER JOIN jzhou104.Benefit b ON a.BENEFIT = b.BenefitPackage
        INNER JOIN jzhou104.HealthInsurance c ON b.BenefitPackage = c.HealthInsuranceID
        INNER JOIN jzhou104.HealthService d ON c.HealthInsuranceID = d.ServiceID

Select *
FROM Benefits

DROP VIEW Benefits

Select *
FROM Employee
```

	FirstName	LastName	Fund401K	HealthInsurance	EmployeePremium	EmployerPremium	ServicesCovered	ServiceCoPay	ServiceFee	ServiceFullCovered	ServiceName	ServiceNotCovere
1	Lee	Lee	192933	2	230000	200000	2	100	1000	0	Reproductive Health	0
2	Sam	Zhou	29182	1	230000	200000	1	10	200	1	Eye	0
3	Trisha	Takanawa	29182	1	230000	200000	1	10	200	1	Eye	0
4	Matthew	Stevenson	29182	1	230000	200000	1	10	200	1	Eye	0
5	Brian	Park	192933	2	230000	200000	2	100	1000	0	Reproductive Health	0

Query executed successfully.

SP 1 – Enroll Student in a course

```sql
CREATE PROCEDURE enrollStudent (@StudentID AS VARCHAR(20), @CourseID AS INT, @SemesterID
AS INT)

        AS

DECLARE @ERROR_MSG_ALREADYENROLLED VARCHAR(500)  SET @ERROR_MSG_ALREADYENROLLED = 'ERROR:
This student has already been enrolled in the course!'

DECLARE @ERROR_MSG_SUCCESS VARCHAR(500) SET @ERROR_MSG_SUCCESS = 'SUCCESS: This student
is now enrolled in the course.'

IF EXISTS (SELECT StudentID, CourseID, SemesterID
                    FROM jzhou104.Course_Enrollment
                    WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID
= @SemesterID)

                    BEGIN
                    PRINT @ERROR_MSG_ALREADYENROLLED
                    END

ELSE IF NOT EXISTS (SELECT StudentID, CourseID FROM jzhou104.Course_Enrollment
 WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID)

      BEGIN
              INSERT INTO Course_Enrollment VALUES (@StudentID, @CourseID, @SemesterID,
1, NULL, NULL)
         END
```



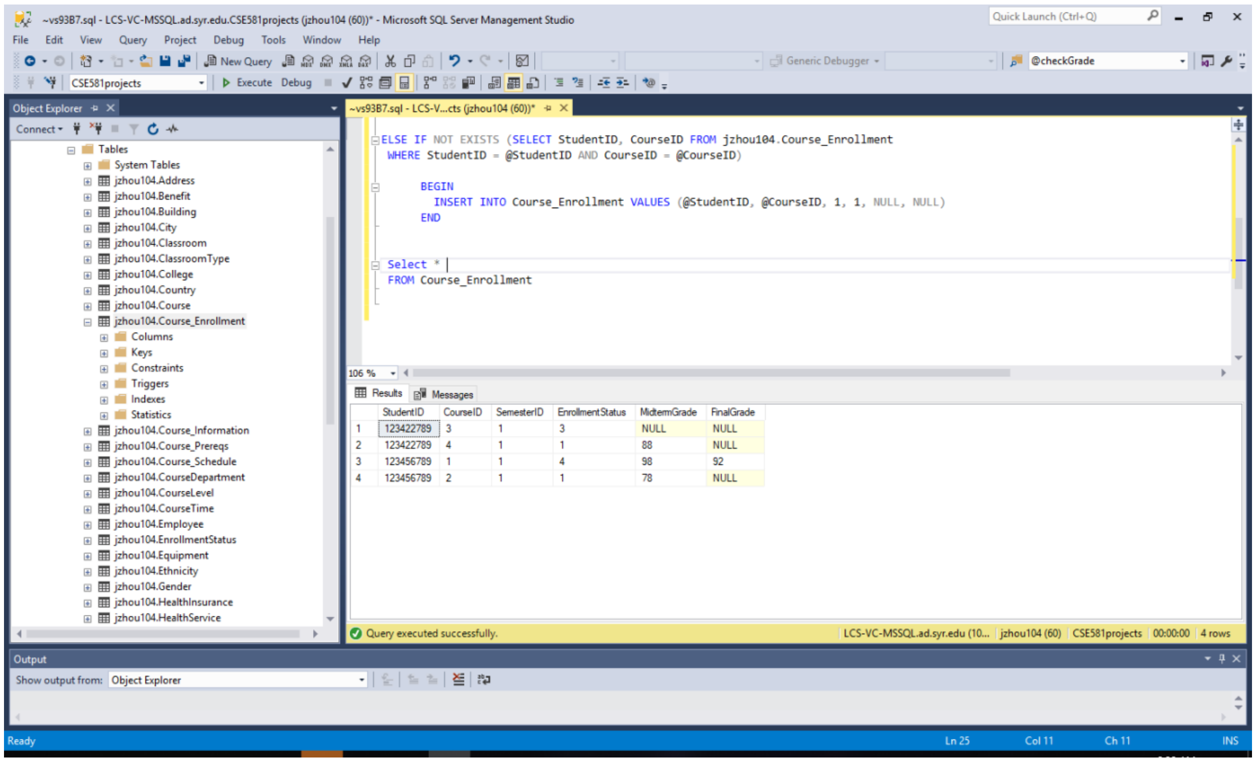
~vs93B7.sql - LCS-VC-MSSQL.ad.syr.edu.CSE581projects (jzhou104 (60))* - Microsoft SQL Server Management Studio

```sql
CREATE PROCEDURE enrollStudent (@StudentID AS VARCHAR(20), @CourseID AS INT)

        AS

DECLARE @ERROR_MSG_ALREADYENROLLED VARCHAR(500)  SET @ERROR_MSG_ALREADYENROLLED = 'ERROR: This student has already been enrolled

DECLARE @ERROR_MSG_SUCCESS VARCHAR(500) SET @ERROR_MSG_SUCCESS = 'SUCCESS: This student is now enrolled in the course.'

IF EXISTS (SELECT StudentID, CourseID
            FROM jzhou104.Course_Enrollment
            WHERE StudentID = @StudentID AND CourseID = @CourseID)

            BEGIN
            PRINT @ERROR_MSG_ALREADYENROLLED
            END
```

```
Commands completed successfully.
```

Query executed successfully. | LCS-VC-MSSQL.ad.syr.edu (10... | jzhou104 (60) | CSE581projects | 00:00:00 | 0 rows

Before Enrollment:



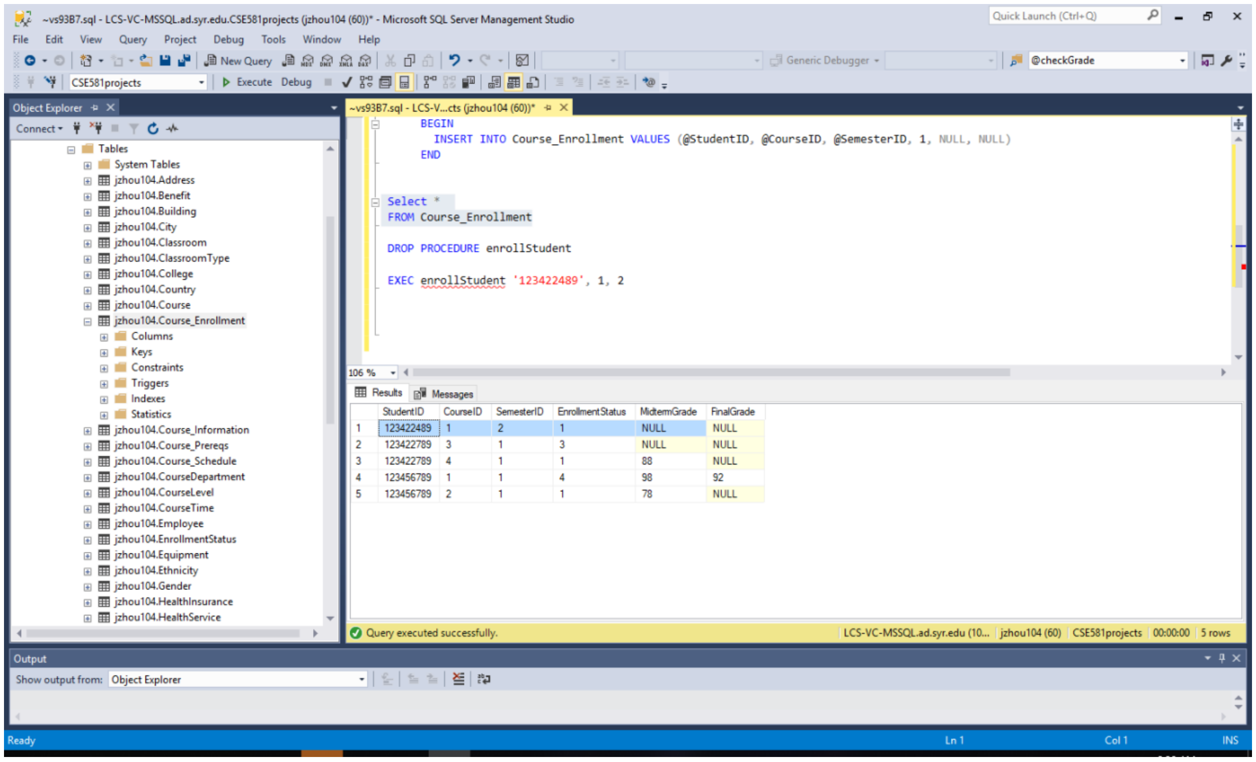
~vs93B7.sql - LCS-VC-MSSQL.ad.syr.edu.CSE581projects (jzhou104 (60))* - Microsoft SQL Server Management Studio

```sql
ELSE IF NOT EXISTS (SELECT StudentID, CourseID FROM jzhou104.Course_Enrollment
 WHERE StudentID = @StudentID AND CourseID = @CourseID)

      BEGIN
        INSERT INTO Course_Enrollment VALUES (@StudentID, @CourseID, 1, 1, NULL, NULL)
      END


 Select *
 FROM Course_Enrollment
```

	StudentID	CourseID	SemesterID	EnrollmentStatus	MidtermGrade	FinalGrade
1	123422789	3	1	3	NULL	NULL
2	123422789	4	1	1	88	NULL
3	123456789	1	1	4	98	92
4	123456789	2	1	1	78	NULL

Query executed successfully. | LCS-VC-MSSQL.ad.syr.edu (10... | jzhou104 (60) | CSE581projects | 00:00:00 | 4 rows

After Enrollment (If the entry DNE):



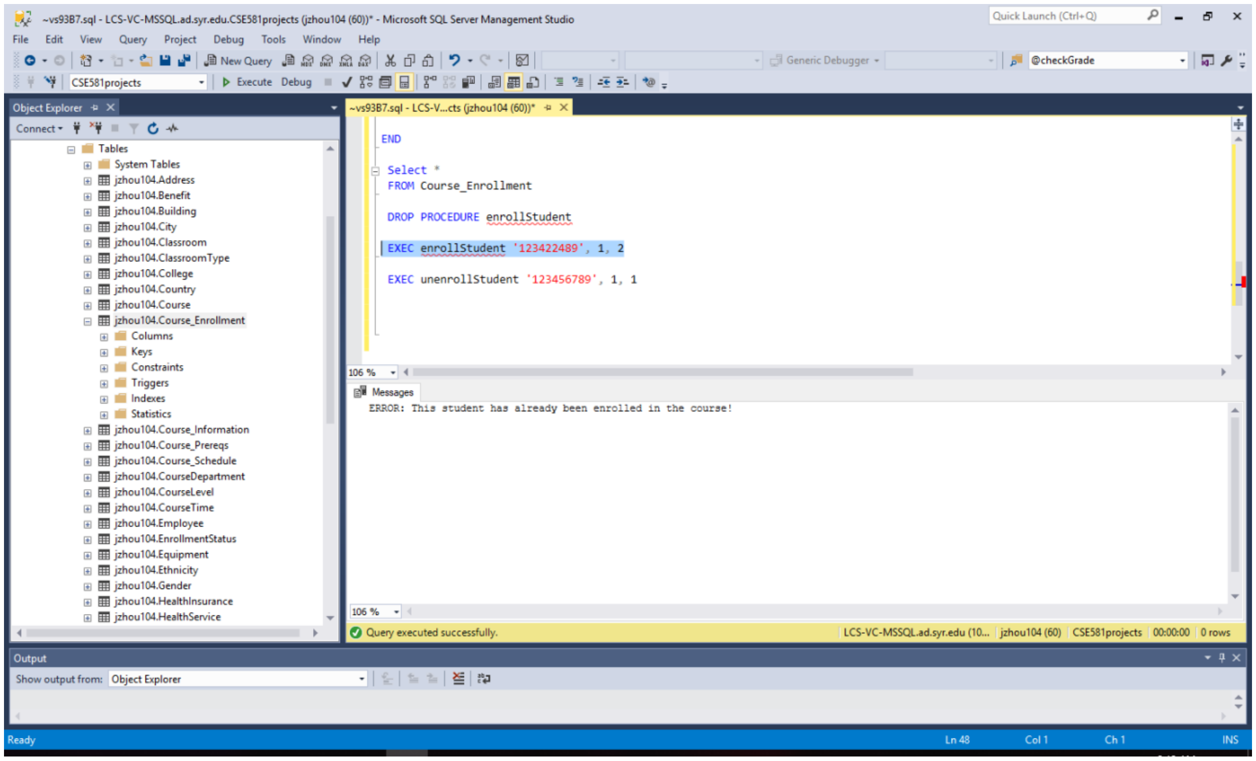
```
        BEGIN
            INSERT INTO Course_Enrollment VALUES (@StudentID, @CourseID, @SemesterID, 1, NULL, NULL)
        END

    Select *
    FROM Course_Enrollment

    DROP PROCEDURE enrollStudent

    EXEC enrollStudent '123422489', 1, 2
```

	StudentID	CourseID	SemesterID	EnrollmentStatus	MidtermGrade	FinalGrade
1	123422489	1	2	1	NULL	NULL
2	123422789	3	1	3	NULL	NULL
3	123422789	4	1	1	88	NULL
4	123456789	1	1	4	98	92
5	123456789	2	1	1	78	NULL

```
Query executed successfully.    LCS-VC-MSSQL.ad.syr.edu (10...  jzhou104 (60)  CSE581projects  00:00:00  5 rows
```

This procedure enrolls a student into a course ONLY IF non-existing rows are present. Otherwise, enrollment would be a failure.

After Enrollment (If the entry exists):



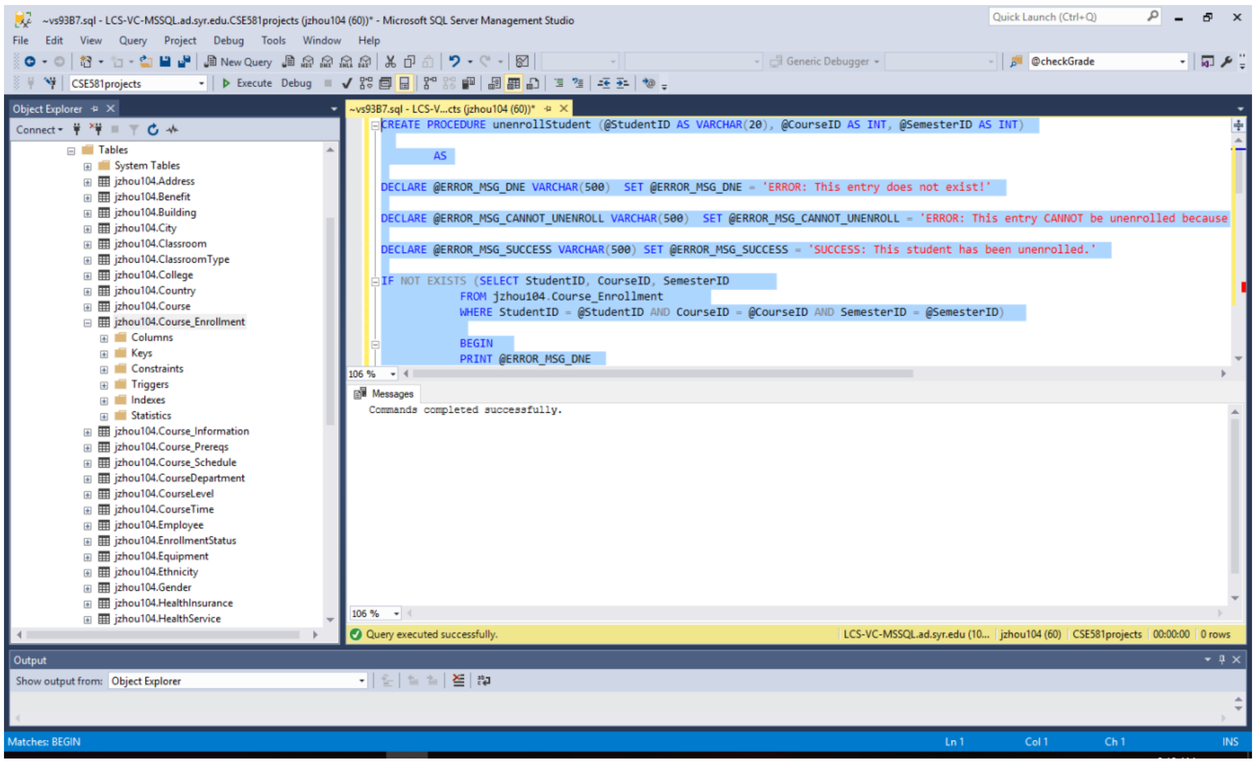
```
    END

    Select *
    FROM Course_Enrollment

    DROP PROCEDURE enrollStudent

    EXEC enrollStudent '123422489', 1, 2

    EXEC unenrollStudent '123456789', 1, 1
```

```
ERROR: This student has already been enrolled in the course!
```

```
Query executed successfully.    LCS-VC-MSSQL.ad.syr.edu (10...  jzhou104 (60)  CSE581projects  00:00:00  0 rows
```

SP 2 – Enroll Student in a course

```
CREATE PROCEDURE unenrollStudent (@StudentID AS VARCHAR(20), @CourseID AS INT,
@SemesterID AS INT)

        AS

DECLARE @ERROR_MSG_DNE VARCHAR(500)  SET @ERROR_MSG_DNE = 'ERROR: This entry does not
exist!'

DECLARE @ERROR_MSG_CANNOT_UNENROLL VARCHAR(500)  SET @ERROR_MSG_CANNOT_UNENROLL = 'ERROR:
This entry CANNOT be unenrolled because a midterm/final grade has already been entered.'

DECLARE @ERROR_MSG_SUCCESS VARCHAR(500) SET @ERROR_MSG_SUCCESS = 'SUCCESS: This student
has been unenrolled.'

IF NOT EXISTS (SELECT StudentID, CourseID, SemesterID
                    FROM jzhou104.Course_Enrollment
                    WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID
= @SemesterID)

                    BEGIN
                    PRINT @ERROR_MSG_DNE
                    END

ELSE IF EXISTS (SELECT StudentID, CourseID, SemesterID FROM jzhou104.Course_Enrollment
 WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID)

 BEGIN

 IF (SELECT MidtermGrade FROM jzhou104.Course_Enrollment
  WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID) IS
NOT NULL
OR (SELECT FinalGrade FROM jzhou104.Course_Enrollment
  WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID) IS
NOT NULL

                    BEGIN
                    PRINT @ERROR_MSG_CANNOT_UNENROLL
                    END

 ELSE

      BEGIN
             SELECT * FROM Course_Enrollment
             DELETE FROM  Course_Enrollment
             WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID =
@SemesterID
         END

END
```

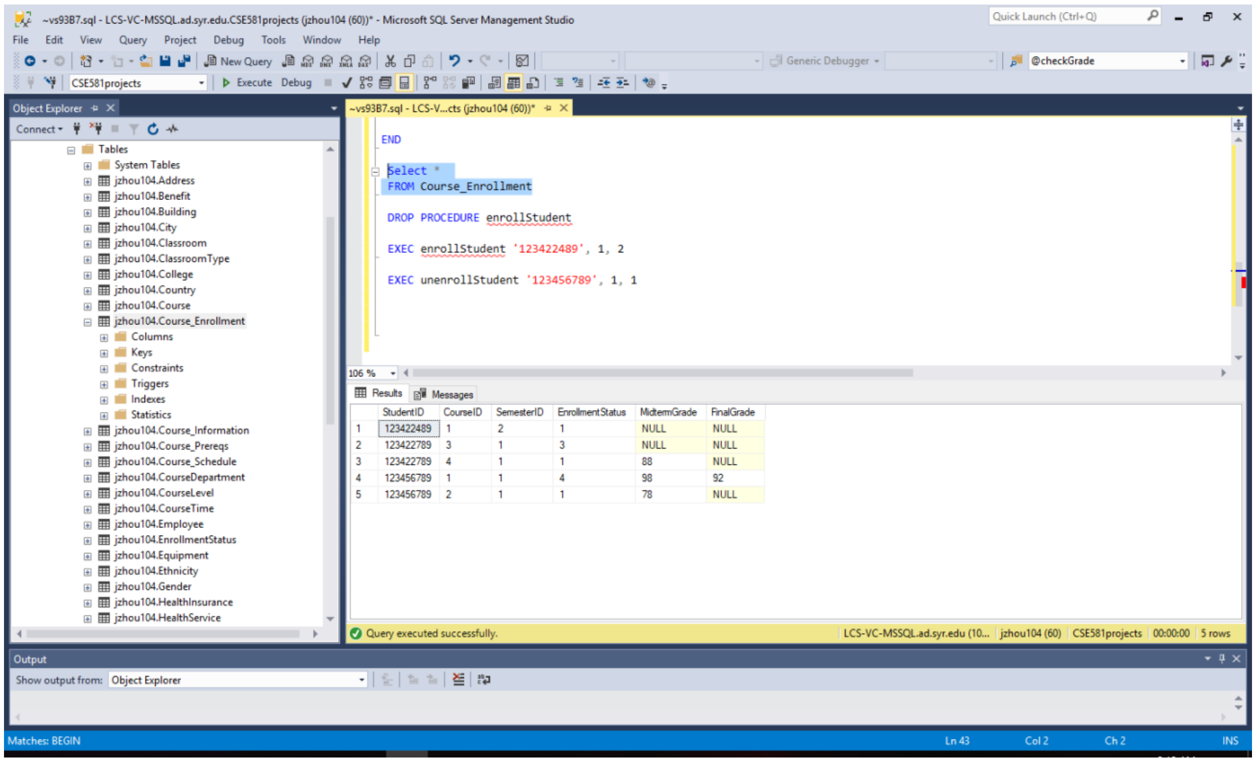


```
CREATE PROCEDURE unenrollStudent (@StudentID AS VARCHAR(20), @CourseID AS INT, @SemesterID AS INT)

        AS

DECLARE @ERROR_MSG_DNE VARCHAR(500)  SET @ERROR_MSG_DNE = 'ERROR: This entry does not exist!'

DECLARE @ERROR_MSG_CANNOT_UNENROLL VARCHAR(500)  SET @ERROR_MSG_CANNOT_UNENROLL = 'ERROR: This entry CANNOT be unenrolled because

DECLARE @ERROR_MSG_SUCCESS VARCHAR(500) SET @ERROR_MSG_SUCCESS = 'SUCCESS: This student has been unenrolled.'

IF NOT EXISTS (SELECT StudentID, CourseID, SemesterID
            FROM jzhou104.Course_Enrollment
            WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID)

            BEGIN
            PRINT @ERROR_MSG_DNE
```

```
Commands completed successfully.
```

Query executed successfully.

Before Unenrollment:

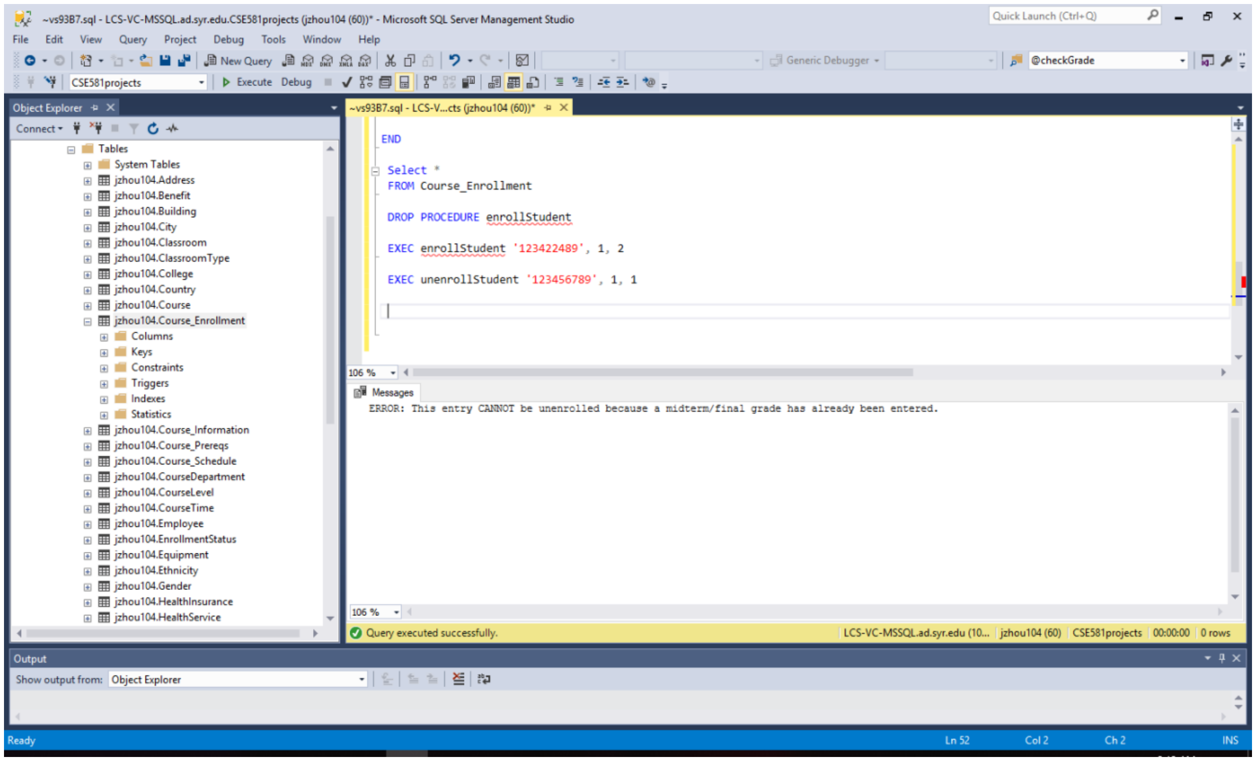


```
END

 Select *
 FROM Course_Enrollment

 DROP PROCEDURE enrollStudent

 EXEC enrollStudent '123422489', 1, 2

 EXEC unenrollStudent '123456789', 1, 1
```

	StudentID	CourseID	SemesterID	EnrollmentStatus	MidtermGrade	FinalGrade
1	123422489	1	2	1	NULL	NULL
2	123422789	3	1	3	NULL	NULL
3	123422789	4	1	1	88	NULL
4	123456789	1	1	4	98	92
5	123456789	2	1	1	78	NULL

Query executed successfully.

After Unenrollment (if the entry cannot be unenrolled)

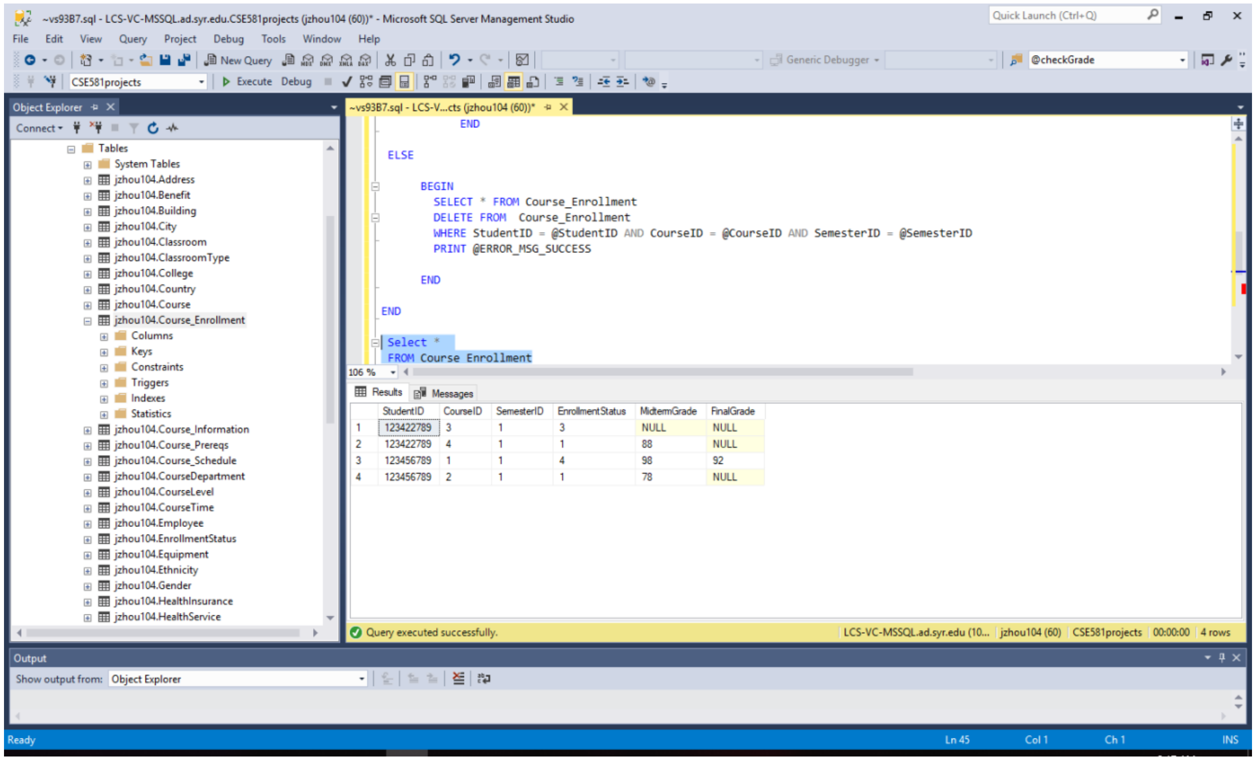


```
END

Select *
FROM Course_Enrollment

DROP PROCEDURE enrollStudent

EXEC enrollStudent '123422489', 1, 2

EXEC unenrollStudent '123456789', 1, 1
```

```
ERROR: This entry CANNOT be unenrolled because a midterm/final grade has already been entered.
```

Query executed successfully.

After Unenrollment (if the entry can be removed):



```
            END

 ELSE

     BEGIN
       SELECT * FROM Course_Enrollment
       DELETE FROM  Course_Enrollment
       WHERE StudentID = @StudentID AND CourseID = @CourseID AND SemesterID = @SemesterID
       PRINT @ERROR_MSG_SUCCESS

     END

END

Select *
FROM Course_Enrollment
```

	StudentID	CourseID	SemesterID	EnrollmentStatus	MidtermGrade	FinalGrade
1	123422789	3	1	3	NULL	NULL
2	123422789	4	1	1	88	NULL
3	123456789	1	1	4	98	92
4	123456789	2	1	1	78	NULL

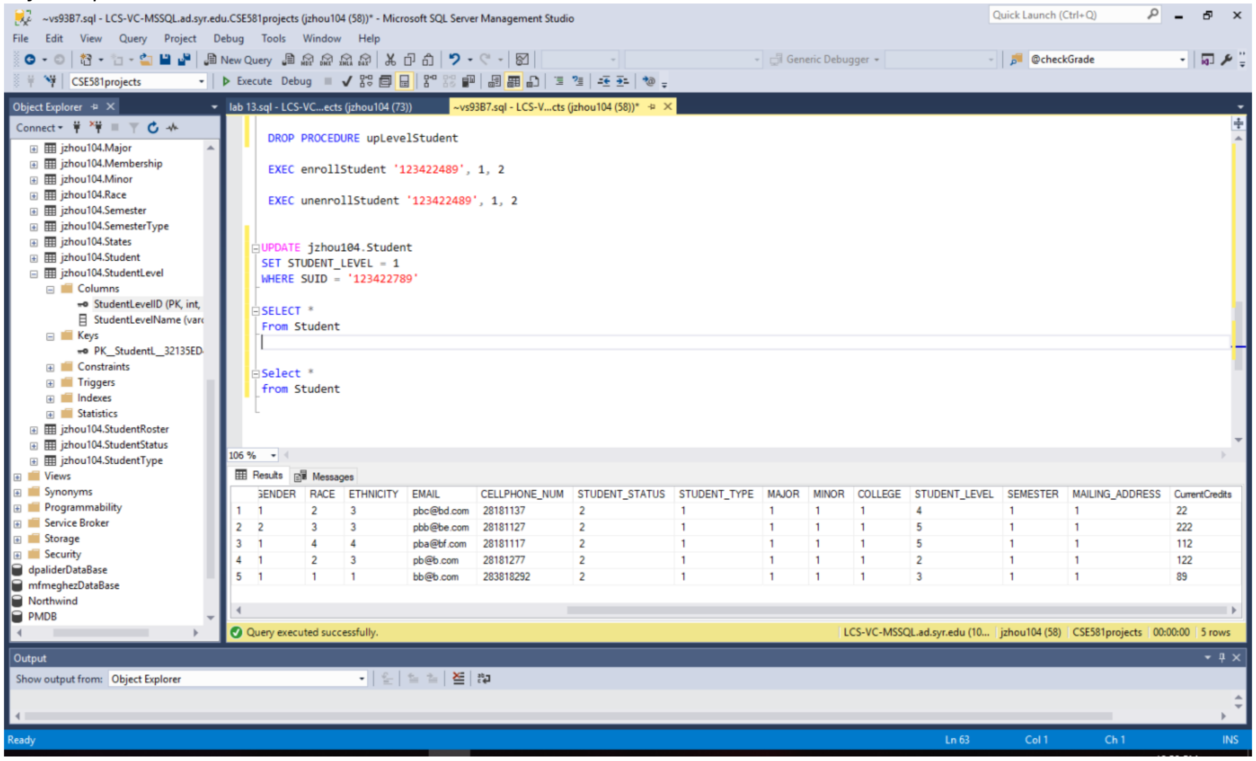
Query executed successfully.

This procedure unenrolls a student into a course ONLY IF existing rows are present and that no midterm or final grades have been entered. Otherwise, unenrollment would be a failure.

SP 3 – Upgrading Student by a Grade Level (With Cursor)

```
CREATE PROCEDURE upLevelStudent

        AS

DECLARE @ERROR_MSG_SUCCESS VARCHAR(500) SET @ERROR_MSG_SUCCESS = 'SUCCESS: All students
have been leveled up...'

DECLARE @nowGrade INT, @nowID VARCHAR(9)

DECLARE gradeCursor CURSOR FOR  (SELECT STUDENT_LEVEL
                                                    FROM jzhou104.Student)

DECLARE IDCursor CURSOR FOR  (SELECT SUID
                                                    FROM jzhou104.Student)

BEGIN
OPEN gradeCursor
FETCH gradeCursor INTO @nowGrade

OPEN IDCursor
FETCH IDCursor INTO @nowID

WHILE(@@FETCH_STATUS=0)
BEGIN
SET @nowGrade += 1


UPDATE jzhou104.Student
SET Student_Level = @nowGrade
WHERE jzhou104.Student.SUID = @nowID

FETCH NEXT FROM gradeCursor INTO @nowGrade
FETCH NEXT FROM IDCursor INTO @nowID

END

PRINT @ERROR_MSG_SUCCESS

CLOSE gradeCursor
DEALLOCATE gradeCursor

CLOSE IDCursor
DEALLOCATE IDCursor

END
```

Before UpLevel

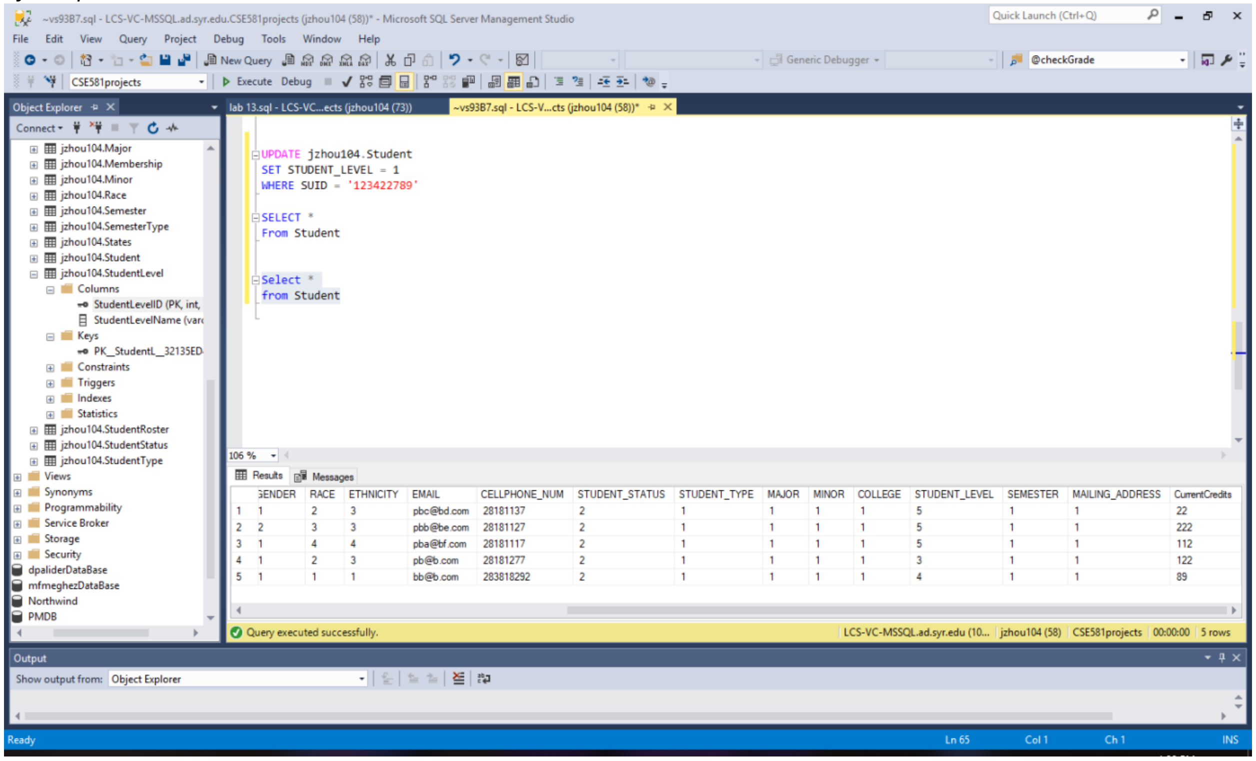


```
DROP PROCEDURE upLevelStudent

EXEC enrollStudent '123422489', 1, 2

EXEC unenrollStudent '123422489', 1, 2


UPDATE jzhou104.Student
SET STUDENT_LEVEL = 1
WHERE SUID = '123422789'

SELECT *
From Student


Select *
from Student
```

	GENDER	RACE	ETHNICITY	EMAIL	CELLPHONE_NUM	STUDENT_STATUS	STUDENT_TYPE	MAJOR	MINOR	COLLEGE	STUDENT_LEVEL	SEMESTER	MAILING_ADDRESS	CurrentCredits
1	1	2	3	pbc@bd.com	28181137	2	1	1	1	1	4	1	1	22
2	2	3	3	pbb@be.com	28181127	2	1	1	1	1	5	1	1	222
3	1	4	4	pba@bf.com	28181117	2	1	1	1	1	5	1	1	112
4	1	2	3	pb@b.com	28181277	2	1	1	1	1	2	1	1	122
5	1	1	1	bb@b.com	283818292	2	1	1	1	1	3	1	1	89

Query executed successfully. LCS-VC-MSSQL.ad.syr.edu (10... jzhou104 (58) CSE581projects 00:00:00 5 rows

After UpLevel



```
UPDATE jzhou104.Student
SET STUDENT_LEVEL = 1
WHERE SUID = '123422789'

SELECT *
From Student


Select *
from Student
```

	GENDER	RACE	ETHNICITY	EMAIL	CELLPHONE_NUM	STUDENT_STATUS	STUDENT_TYPE	MAJOR	MINOR	COLLEGE	STUDENT_LEVEL	SEMESTER	MAILING_ADDRESS	CurrentCredits
1	1	2	3	pbc@bd.com	28181137	2	1	1	1	1	5	1	1	22
2	2	3	3	pbb@be.com	28181127	2	1	1	1	1	5	1	1	222
3	1	4	4	pba@bf.com	28181117	2	1	1	1	1	5	1	1	112
4	1	2	3	pb@b.com	28181277	2	1	1	1	1	3	1	1	122
5	1	1	1	bb@b.com	283818292	2	1	1	1	1	4	1	1	89

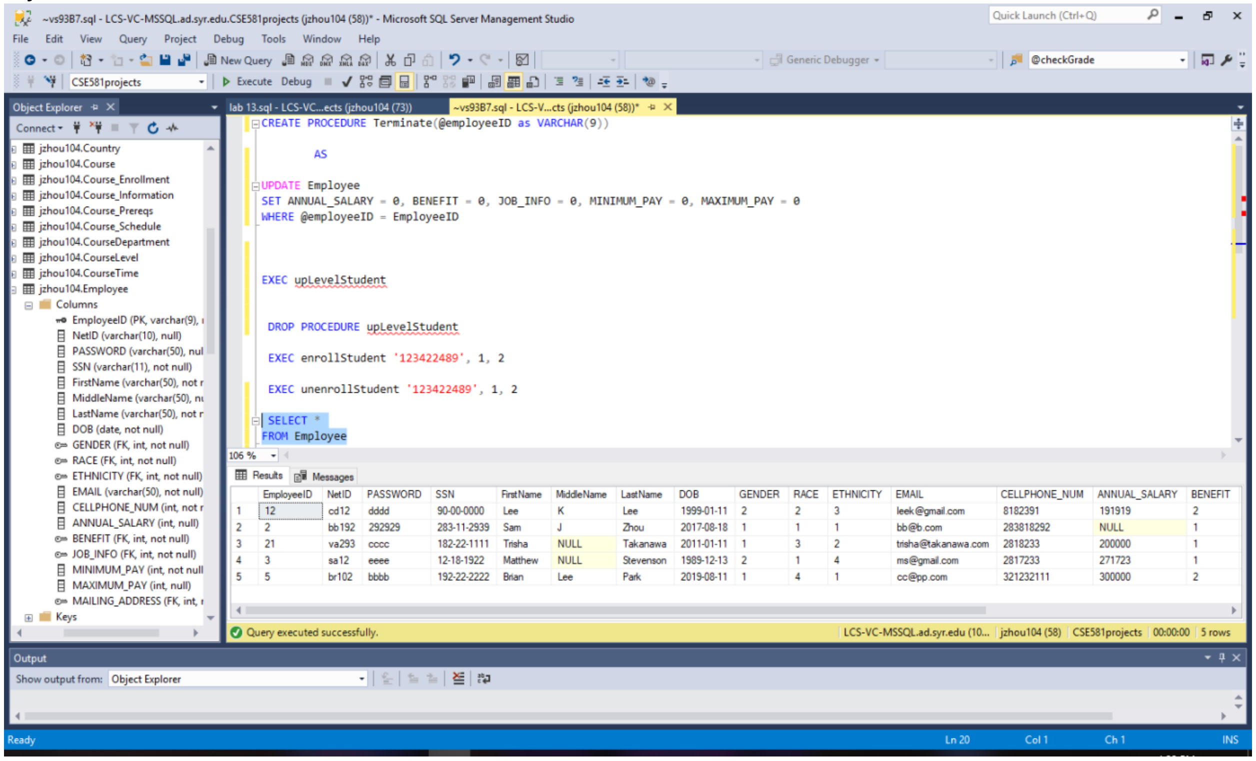
Query executed successfully. LCS-VC-MSSQL.ad.syr.edu (10... jzhou104 (58) CSE581projects 00:00:00 5 rows

This procedure upgrades all students in the database to the next grade level using 2 cursors. Specifically designed procedure can be written to make exceptions.

SP 4 – Business Procedure

```
CREATE PROCEDURE Terminate(@employeeID as VARCHAR(9))

        AS

UPDATE Employee
SET ANNUAL_SALARY = 0, BENEFIT = 0, JOB_INFO = 0, MINIMUM_PAY = 0, MAXIMUM_PAY = 0
WHERE @employeeID = EmployeeID
```

Before Termination

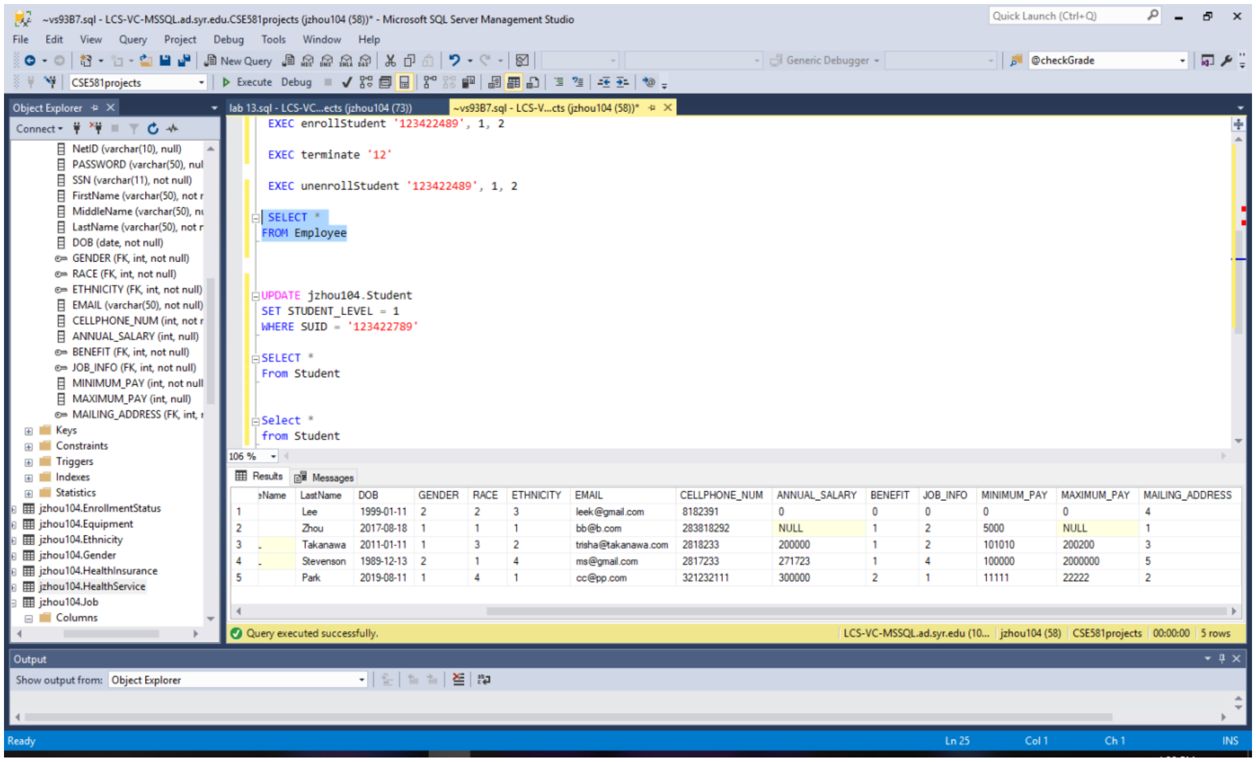


```
CREATE PROCEDURE Terminate(@employeeID as VARCHAR(9))

        AS

UPDATE Employee
SET ANNUAL_SALARY = 0, BENEFIT = 0, JOB_INFO = 0, MINIMUM_PAY = 0, MAXIMUM_PAY = 0
WHERE @employeeID = EmployeeID



EXEC upLevelStudent


 DROP PROCEDURE upLevelStudent

 EXEC enrollStudent '123422489', 1, 2

 EXEC unenrollStudent '123422489', 1, 2

 SELECT *
FROM Employee
```

	EmployeeID	NetID	PASSWORD	SSN	FirstName	MiddleName	LastName	DOB	GENDER	RACE	ETHNICITY	EMAIL	CELLPHONE_NUM	ANNUAL_SALARY	BENEFIT
1	12	cd12	dddd	90-00-0000	Lee	K	Lee	1999-01-11	2	2	3	leek@gmail.com	8182391	191919	2
2	2	bb192	292929	283-11-2939	Sam	J	Zhou	2017-08-18	1	1	1	bb@b.com	283818292	NULL	1
3	21	va293	cccc	182-22-1111	Trisha	NULL	Takanawa	2011-01-11	1	3	2	trisha@takanawa.com	2818233	200000	1
4	3	sa12	eeee	12-18-1922	Matthew	NULL	Stevenson	1989-12-13	2	1	4	ms@gmail.com	2817233	271723	1
5	5	br102	bbbb	192-22-2222	Brian	Lee	Park	2019-08-11	1	4	1	cc@pp.com	321232111	300000	2

Query executed successfully. LCS-VC-MSSQL.ad.syr.edu (10... jzhou104 (58) CSE581projects 00:00:00 5 rows

After Termination


```
NetID (varchar(10), null)
PASSWORD (varchar(50), nul
SSN (varchar(11), not null)
FirstName (varchar(50), not r
MiddleName (varchar(50), nu
LastName (varchar(50), not r
DOB (date, not null)
GENDER (FK, int, not null)
RACE (FK, int, not null)
ETHNICITY (FK, int, not null)
EMAIL (varchar(50), not null)
CELLPHONE_NUM (int, not r
ANNUAL_SALARY (int, null)
BENEFIT (FK, int, not null)
JOB_INFO (FK, int, not null)
MINIMUM_PAY (int, not null
MAXIMUM_PAY (int, null)
MAILING_ADDRESS (FK, int, r
Keys
Constraints
Triggers
Indexes
Statistics
jzhou104.EnrollmentStatus
jzhou104.Equipment
jzhou104.Ethnicity
jzhou104.Gender
jzhou104.HealthInsurance
jzhou104.HealthService
jzhou104.Job
Columns
```

```
 EXEC enrollStudent '123422489', 1, 2

 EXEC terminate '12'

 EXEC unenrollStudent '123422489', 1, 2

SELECT *
FROM Employee



UPDATE jzhou104.Student
SET STUDENT_LEVEL = 1
WHERE SUID = '123422789'

SELECT *
From Student


Select *
from Student
```

	›Name	LastName	DOB	GENDER	RACE	ETHNICITY	EMAIL	CELLPHONE_NUM	ANNUAL_SALARY	BENEFIT	JOB_INFO	MINIMUM_PAY	MAXIMUM_PAY	MAILING_ADDRESS
1		Lee	1999-01-11	2	2	3	leek@gmail.com	8182391	0	0	0	0	0	4
2		Zhou	2017-08-18	1	1	1	bb@b.com	283818292	NULL	1	2	5000	NULL	1
3	.	Takanawa	2011-01-11	1	3	2	trisha@takanawa.com	2818233	200000	1	2	101010	200200	3
4	.	Stevenson	1989-12-13	2	1	4	ms@gmail.com	2817233	271723	1	4	100000	2000000	5
5		Park	2019-08-11	1	4	1	cc@pp.com	321232111	300000	2	1	11111	22222	2

```
Query executed successfully.    LCS-VC-MSSQL.ad.syr.edu (10... | jzhou104 (58) | CSE581projects | 00:00:00 | 5 rows
```

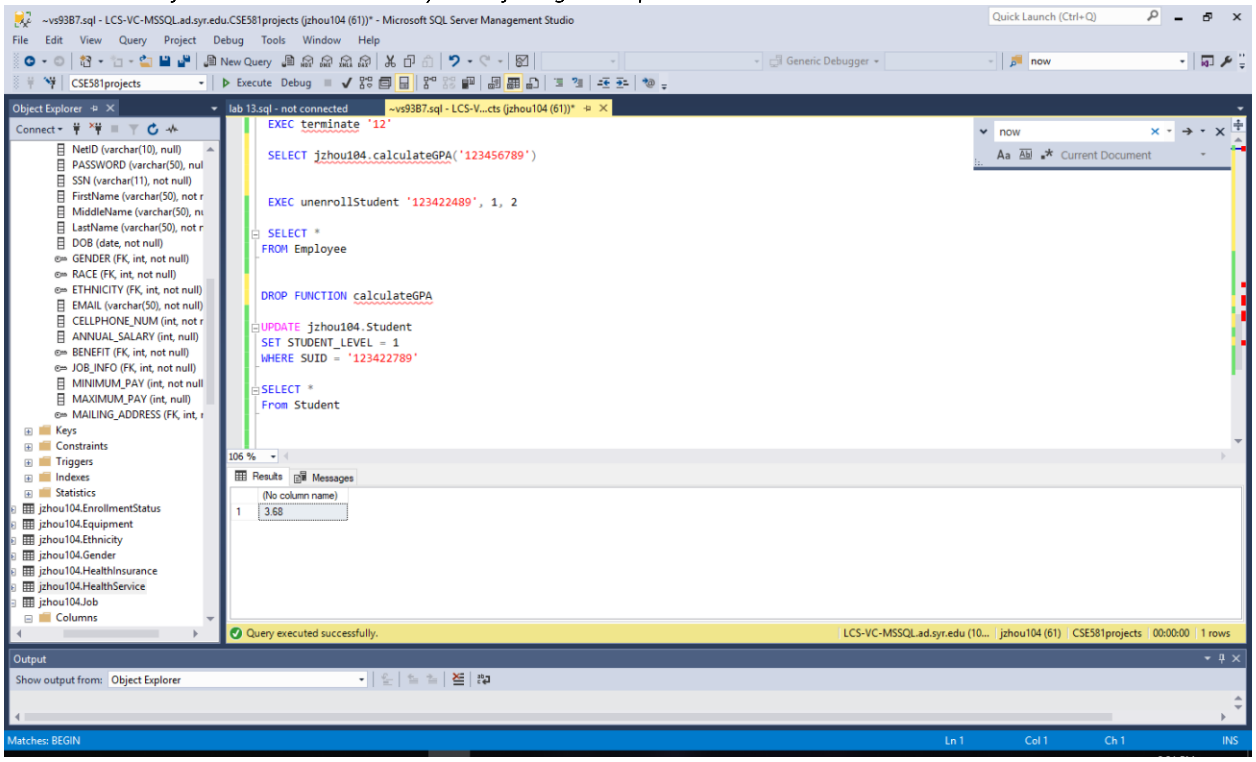
This procedure terminates an employee and respectively sets their benefits to 0. It does not remove their record entirely because we would want to keep track of past employees' records for at least 5 years in case of certain discrepancies.

FUNCTION – Caluclate Accumulative GPA

```
CREATE FUNCTION calculateGPA(@studentID AS VARCHAR(9)) RETURNS FLOAT

        AS
BEGIN
DECLARE gpaCursor CURSOR FOR  (SELECT FinalGrade
                                                    FROM jzhou104.Course_Enrollment
                                                    WHERE FinalGrade IS NOT NULL AND
@studentID = StudentID)

DECLARE @countTot FLOAT, @this FLOAT, @runningSum FLOAT, @avg FLOAT, @GPA FLOAT
SET @countTot = 0.0
SET @runningSum = 0.0

OPEN gpaCursor
FETCH gpaCursor INTO @this

WHILE(@@FETCH_STATUS=0)
BEGIN
SET @runningSum += @this
SET @countTot +=1
FETCH NEXT FROM gpaCursor INTO @this
END
```

```sql
IF(@countTot != 0.0)
BEGIN
SET @avg = @runningSum /  CAST(@countTot as DECIMAL)
SET @gpa = (@avg/25)
END

ELSE
BEGIN
SET @gpa = 4.0
END

CLOSE gpaCursor
DEALLOCATE gpaCursor

return round(@gpa, 2)

END
```

Calculate a GPA for a student who already has a final grade inputted

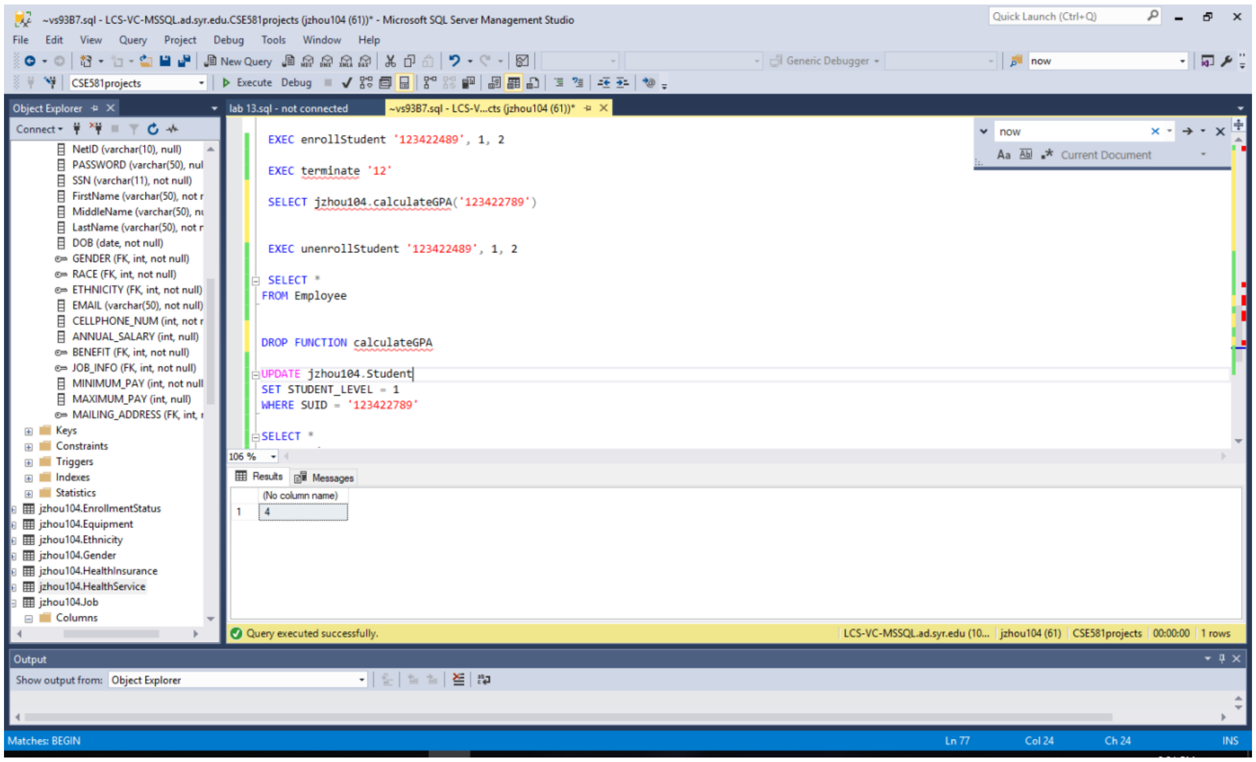

~vs93B7.sql - LCS-VC-MSSQL.ad.syr.edu.CSE581projects (jzhou104 (61))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Query Project Debug Tools Window Help
New Query
Generic Debugger
now
CSE581projects
Execute Debug
Object Explorer
Connect
NetID (varchar(10), null)
PASSWORD (varchar(50), nul
SSN (varchar(11), not null)
FirstName (varchar(50), not r
MiddleName (varchar(50), nu
LastName (varchar(50), not r
DOB (date, not null)
GENDER (FK, int, not null)
RACE (FK, int, not null)
ETHNICITY (FK, int, not null)
EMAIL (varchar(50), not null)
CELLPHONE_NUM (int, not r
ANNUAL_SALARY (int, null)
BENEFIT (FK, int, not null)
JOB_INFO (FK, int, not null)
MINIMUM_PAY (int, not null
MAXIMUM_PAY (int, null)
MAILING_ADDRESS (FK, int, r
Keys
Constraints
Triggers
Indexes
Statistics
jzhou104.EnrollmentStatus
jzhou104.Equipment
jzhou104.Ethnicity
jzhou104.Gender
jzhou104.HealthInsurance
jzhou104.HealthService
jzhou104.Job
Columns
lab 13.sql - not connected
~vs93B7.sql - LCS-V...cts (jzhou104 (61))*
now
Current Document
106 %
Results
Messages
Output
Show output from: Object Explorer
Matches: BEGIN
Ln 1
Col 1
Ch 1
INS

```sql
  EXEC terminate '12'

  SELECT jzhou104.calculateGPA('123456789')


  EXEC unenrollStudent '123422489', 1, 2

  SELECT *
FROM Employee


DROP FUNCTION calculateGPA

UPDATE jzhou104.Student
SET STUDENT_LEVEL = 1
WHERE SUID = '123422789'

SELECT *
From Student
```

```
   (No column name)
1  3.68
```

```
Query executed successfully.    LCS-VC-MSSQL.ad.syr.edu (10...   jzhou104 (61)   CSE581projects   00:00:00   1 rows
```

Calculate a GPA for a student who already has a NO classes of final grade inputted
It will be defaulted to 4.0 because we assume that the student is a freshman who has not taken any classes.



```
EXEC enrollStudent '123422489', 1, 2

EXEC terminate '12'

SELECT jzhou104.calculateGPA('123422789')


EXEC unenrollStudent '123422489', 1, 2

SELECT *
FROM Employee


DROP FUNCTION calculateGPA

UPDATE jzhou104.Student
SET STUDENT_LEVEL = 1
WHERE SUID = '123422789'

SELECT *
```

	(No column name)
1	4

Query executed successfully.

This algorithm will find out about a student's GPA. If a final grade is inputted on any of the classes, the accumulative GPA will be calculated (even with just 1 class). Otherwise, it is defaulted to 4.0 because we assume that the student is a freshman who is in the fall semester and has not finish any classes yet.