CSE 581

**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;) :

* (10% points) SP to use cursor(s)
* (10% points) SP to update data in a table (perform validation)
* (10% points) SP to delete data from a table
* (10% points) 1 SP of your own choice, performing a business action.

Create 1 *Function* (10% points)which can be executed by “Graders”. Not the function you did for the lab.

You will also *create a view* (named as “Benefits”) (10% points) 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.

A computer screen shot of a computer

Description automatically generated

1. You **shall** create 4 stored procedures.

A computer screen shot of a computer

Description automatically generated

1. You **shall** create 1 function.

A computer screen shot of a computer

Description automatically generated

1. You **shall** submit all your SQL code, used to create the DB objects (view, SPs and function).

**4 stored procedures:**

--to use cursor

CREATE PROCEDURE sp\_GetAllCoursesByDepartment

@DepartmentID INT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @CourseID INT, @CourseTitle VARCHAR(255);

DECLARE course\_cursor CURSOR FOR

SELECT CourseID, CourseTitle

FROM Courses

WHERE DepartmentID = @DepartmentID;

OPEN course\_cursor;

FETCH NEXT FROM course\_cursor INTO @CourseID, @CourseTitle;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Course ID: ' + CAST(@CourseID AS VARCHAR) + ', Title: ' + @CourseTitle;

FETCH NEXT FROM course\_cursor INTO @CourseID, @CourseTitle;

END;

CLOSE course\_cursor;

DEALLOCATE course\_cursor;

END;

GO

-- update data in table

CREATE PROCEDURE sp\_UpdateAnnualSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

SET NOCOUNT ON;

IF @NewSalary <= 0

BEGIN

PRINT 'Invalid salary amount.';

RETURN;

END

UPDATE EmployeeInfo

SET AnnualSalary = @NewSalary

WHERE EmployeeID = @EmployeeID;

PRINT 'Salary updated successfully.';

END;

GO

-- delete data from table

CREATE PROCEDURE sp\_DeleteEmployee

@EmployeeID INT

AS

BEGIN

SET NOCOUNT ON;

DELETE FROM EmployeeInfo

WHERE EmployeeID = @EmployeeID;

PRINT 'Employee deleted successfully.';

END;

GO

-- own choice

CREATE PROCEDURE sp\_UpdateEmployeeJob

@EmployeeID INT,

@NewJobID INT

AS

BEGIN

SET NOCOUNT ON;

IF NOT EXISTS (SELECT 1 FROM EmployeeInfo WHERE EmployeeID = @EmployeeID)

BEGIN

PRINT 'Employee does not exist.';

RETURN;

END

IF NOT EXISTS (SELECT 1 FROM Jobs WHERE JobID = @NewJobID)

BEGIN

PRINT 'The new job ID does not exist.';

RETURN;

END

IF EXISTS (SELECT 1 FROM EmployeeAndJob WHERE EmployeeID = @EmployeeID AND JobID = @NewJobID)

BEGIN

PRINT 'The employee is already assigned to this job.';

RETURN;

END

BEGIN TRY

BEGIN TRANSACTION;

UPDATE EmployeeAndJob

SET JobID = @NewJobID

WHERE EmployeeID = @EmployeeID;

COMMIT TRANSACTION;

PRINT 'Employee job updated successfully.';

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT 'An error occurred during the job update.';

END CATCH;

END;

GO

**1 Function:**

CREATE FUNCTION fn\_GetTotalStudentsInCourse(@CourseID INT)

RETURNS INT

AS

BEGIN

DECLARE @TotalStudents INT;

SELECT @TotalStudents = COUNT(\*)

FROM EnrolledStudents

WHERE CourseID = @CourseID;

RETURN @TotalStudents;

END;

GO

**1 view:**

CREATE VIEW EmpBenefits AS

SELECT

e.EmployeeID,

p.FirstName,

p.LastName,

i.Text AS InsuranceType,

c.Text AS CoverageType,

b.EmployeePremium,

b.EmployerPremium

FROM EmployeeInfo e

JOIN PersonInfo p ON e.PersonID = p.PersonID

JOIN Benefits b ON e.EmployeeID = b.EmployeeID

JOIN InsuranceType i ON b.InsuranceTypeID = i.InsuranceTypeID

JOIN CoverType c ON b.CoverTypeID = c.CoverTypeID;

GO

1. You **shall** submit a short (a single sentence) explanation (10% points) of what the purpose of each of the DB objects is.

**sp\_GetAllCoursesByDepartment**: This stored procedure retrieves and prints the Course ID and Title for all courses within a specified department using a cursor.

**sp\_UpdateAnnualSalary**: Updates the annual salary of an employee in the EmployeeInfo table based on the provided EmployeeID and new salary, with validation for a non-negative salary.

**sp\_DeleteEmployee**: Deletes an employee record from the EmployeeInfo table based on the provided EmployeeID.

**sp\_UpdateEmployeeJob**: Updates the job assignment for an employee in the EmployeeAndJob table based on the provided EmployeeID and new JobID, with validations for existing records and job existence.

**EmpBenefits View**: This view combines information from multiple tables (EmployeeInfo, PersonInfo, Benefits, InsuranceType, CoverType) to provide a comprehensive overview of employee benefits, including employee and employer premiums for insurance coverage.

**fn\_GetTotalStudentsInCourse Function**:

This function, when executed, returns the total number of students enrolled in a specified course based on the provided CourseID. The function utilizes a SELECT statement to count the number of records in the EnrolledStudents table associated with the given CourseID and returns this count as the result.

1. You **shall** execute all the SPs/function/view (provide screenshots (10% points) of execution the way you did in the previous labs (SP & function labs)). You **shall** also provide the screenshots of granting Graders privileges (5% points).

A computer screen shot of a computer

Description automatically generated

1. You **shall** submit a text file (15% points) that will run SELECTs against of your view and execution of your SPs and function.

A computer screen with a white screen

Description automatically generated

A screenshot of a computer

Description automatically generated

EXECUTE STATEMENTS FOR ALL SPs:

A computer screen shot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generatedA computer screen shot of a computer

Description automatically generated