# PROCEDURES IN C#

## Introduction

This assignment tried to demonstrate the skills obtained throughout the semester in relation to the management of databases. To prove it, an application must be made in C #. This app is composed of an interface that lets you choose from a series of options. Of which, the Stored Procedures created previously in the Company Database can be chosen, with the series of characteristics that the document enunciates. Now we will proceed with the logic of each procedure.

The first thing is to check if the procedures we are going to create already exist or not. For this reason, at the beginning of the script, this sentence series is carried out:

```sql
USE CS2016B_8_Company;
GO

/*****   Drop Procedure if already exist *****/
IF (OBJECT_ID('usp_CreateDepartment') IS NOT NULL)
  DROP PROCEDURE [usp_CreateDepartment]
GO

IF (OBJECT_ID('usp_UpdateDepartmentName') IS NOT NULL)
  DROP PROCEDURE [usp_UpdateDepartmentName]
GO

IF (OBJECT_ID('usp_DeleteDepartment') IS NOT NULL)
  DROP PROCEDURE [usp_DeleteDepartment]
GO

IF (OBJECT_ID('usp_GetDepartment') IS NOT NULL)
  DROP PROCEDURE [usp_GetDepartment]
GO

IF (OBJECT_ID('usp_GetAllDepartments') IS NOT NULL)
  DROP PROCEDURE [usp_GetAllDepartments]
GO

IF (OBJECT_ID('usp_UpdateDepartmentManager') IS NOT NULL)
  DROP PROCEDURE [usp_UpdateDepartmentManager]

IF (OBJECT_ID('getEmpCount') IS NOT NULL)
  DROP FUNCTION [getEmpCount]
GO
```

Then we proceed with the procedures

PROCEDURE → **usp_CreateDepartment(DName, MgrSSN) :**

```
CREATE PROCEDURE [usp_CreateDepartment]
       @DName nvarchar(50),
       @MgrSSN numeric(9,0)

AS
--TO CATCH THE POSSIBLE EXCEPTION
IF exists(SELECT 'True' FROM Department WHERE MgrSSN = @MgrSSN)
BEGIN
THROW 50002, 'The Manager is already in assigned to other Dept.', 1;
END
ELSE IF exists(SELECT 'True' FROM Department WHERE DName = @Dname)
BEGIN
THROW 50001, 'This Name is already exist.', 1;
END
ELSE
BEGIN
--ELSE OPCION (THE REAL PROCEDURES' LOGIC)
SET NOCOUNT ON;

       DECLARE @LastChangeDate datetime = GetDate()
       DECLARE @DNumber int
       DECLARE @EmpCount int

       BEGIN TRANSACTION; --TO AUTOINCREMENT THE DNUMBER

       SET @DNumber = (SELECT COALESCE(MAX(DNumber), 0) + 1
        FROM Department WITH (UPDLOCK));

       SET @EmpCount = dbo.getEmpCount(@DNumber);

   --INSERT INTO (DEPARTMENT) TO CREATE A NEW DEP
       INSERT INTO Department
       VALUES (@DName, @DNumber, @MgrSSN, @LastChangeDate, @EmpCount)

       COMMIT TRANSACTION;
END
```

We create a procedure that will wait for 2 parameters, we check using the MgrSSN if the parameter related to the number of the manager is already assigned to another. This check is due to a Manager can not handle more than one department at a time, for this reason to avoid future problems this check is made. In the same way but with the name of the department, we perform the composition that no name is repeated.

Then three variables are declared, one to notify the current date, other to insert the number related to the department, which is defined in a way that the number will automatically increase each time a new department is inserted. And another to count the employees that works on the Dept. This last variable is defined with the return value of the function "getEmpCount (@DNumber)" (int) that is responsible for creating a relationship between the Department table and the Employee table.

By calling this function and entering the Department number as a parameter, you can return the total number of workers .

The second procedure tries to update the name of the department

PROCEDURE → **usp_UpdateDepartmentName(DNumber, DName)**

```
		CREATE PROCEDURE [usp_UpdateDepartmentName]
			@DNumber int,
			@DName nvarchar(50)

		AS

		BEGIN
			UPDATE Department
			SET DName = @DName
			WHERE DNumber = @DNumber
	END
```

As usual, the first thing is to declare the parameters that will be passed to you and then update the name of the department by filtering it by the Department number.

PROCEDURE → **usp_UpdateDepartmentManager(DNumber, MgrSSN)**

```
		CREATE PROCEDURE [usp_UpdateDepartmentManager]
			@DNumber int, @MgrSSN numeric(9,0)

		AS

		IF exists(SELECT 'True' FROM Department WHERE MgrSSN = @MgrSSN)
		BEGIN
		THROW 50004, 'Manager was assigned to a other Department.', 1;
		END
		ELSE
		BEGIN
			SET NOCOUNT ON;
			DECLARE @currentMgrSSN numeric(9,0)

			SELECT @currentMgrSSN = MgrSSN FROM Department
			WHERE DNumber = @DNumber

			DECLARE @LastChangeDate datetime = getDate()
		END

		IF exists(SELECT 'True' FROM Employee WHERE SSN = @MgrSSN)
		BEGIN TRY
			UPDATE Department SET MgrSSN = @MgrSSN, MgrStartDate
			= @LastChangeDate WHERE DNumber = @DNumber
			END TRY
			BEGIN CATCH
				THROW 50010, 'Something wrong occoured', 1;
			END CATCH
```

```
        ELSE
                THROW 50005, 'There isn´t a person with this SSN', 1;

        BEGIN
        IF exists(SELECT 'True' FROM Employee
                                WHERE Employee.SSN != @MgrSSN AND Dno =
@DNumber
                                AND Employee.SuperSSN = @currentMgrSSN)
            UPDATE Employee SET SuperSSN = @MgrSSN
                WHERE Employee.SuperSSN = @currentMgrSSN AND Dno = @DNumber
END
```

In this procedure, the objective was to update the Manager of a department. We pass by parameter the department number and the serial number of the Manager, we control those exceptions that may occur and update the field by another one (as long as that new Manager has not already assigned another department).

PROCEDURE → **usp_DeleteDepartment(DNumber)**

```
        CREATE PROCEDURE [usp_DeleteDepartment]
                @DNumber int

        AS

        BEGIN
        UPDATE Employee SET Dno = null WHERE Employee.Dno = @DNumber
        END

        BEGIN

                DELETE Works_on WHERE pno = (
                        SELECT Pno FROM Project WHERE Project.DNum = @DNumber
                )
                DELETE Project WHERE DNum = @DNumber
                DELETE Dept_Locations WHERE DNUmber = @DNumber
                DELETE Department WHERE DNumber = @DNumber

END
```

In the delete procedure, all the data related to a specific department passed by a parameter (DNumber) is deleted, except for the Dno field of the Employee table since, as is logical, even if you want to delete a Dept, you do not have to eliminate the employees who work in this one. For this reason, instead of being deleted, the Dno is changed to NULL

The last 2 procedures are practically identified and the only difference is that in the "select" of GetDepartment, the "where" condition is added in relation to the DNumber and the SSN.

PROCEDURE → **usp_GetDepartment(DNumber)**

```
            CREATE PROCEDURE [usp_GetDepartment]
                  @DNumber int

            AS

            BEGIN
                  SELECT Department.DName, Department.DNumber, Employee.Dno
                  FROM Department, Employee WHERE DNumber = @DNumber AND SSN =
      MgrSSN
            END
```

PROCEDURE → **usp_GetAllDepartments**

```
CREATE PROCEDURE [usp_GetAllDepartments]
AS
BEGIN
      SELECT * FROM Department
END
```

To finalize the documentation reacted with the .sql file, which generates all the characteristics of the procedures, it remains to define the function getEmpCount of which the beginning of the document has already been spoken.

```
CREATE FUNCTION [getEmpCount]
(
      @DNumber int --PARAM
)

RETURNS int

AS
--THE RETURN QUERY
BEGIN
RETURN
((SELECT count(SSN) from Department JOIN Employee on DNumber = Dno where Dno =
@DNumber))
END
```