BD: Guião 9

9.1

a)

```
USE[master]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[SP_remove_employee]
        @ssn INT,
        @Status varchar(16) output,
        @Message varchar(255) output,
        @executionTime int output
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @StartTime DATETIME = GETDATE();
    BEGIN TRY
        BEGIN TRAN
            DELETE FROM Company.[Dependent] WHERE Essn=@ssn;
            DELETE FROM Company.Works_on WHERE Essn=@ssn;
            UPDATE Company.Department SET Mgr_ssn=NULL WHERE Mgr_ssn=@ssn;
            UPDATE Company.Employee SET Super_ssn=NULL WHERE Super_ssn=@ssn;
            DELETE FROM Company.Employee WHERE Ssn=@ssn;
        COMMIT TRAN
        SET @Status = 'Sucess';
        SET @Message= 'Registo apagado com sucesso'

    END TRY
    BEGIN CATCH
        ROLLBACK TRAN;
        SET @Status = 'Fail';
        SET @Message= 'Erro a apagar o Registo. ('+ ERROR_MESSAGE() + ')';
    END CATCH
    SELECT @executionTime = DATEDIFF(MILLISECOND, @StartTime,GETDATE())
END
GO
--------------------------------------------------------------------------------
----------
Devemos ter as preocupações adicionais de dar update ao Mgr_ssn no department e no
employee.
```

b)

```
USE[master]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROC [dbo].[SP_dept_managers]

AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @Managers TABLE (
    ssn int,
    yearsAsManag int
    )
            INSERT INTO @Managers (ssn, yearsAsManag)
                SELECT Employee.Ssn, DATEDIFF(Year,
Company.Department.Mgr_Start_Date, GetDate()) FROM Company.Employee
                    INNER JOIN Company.Department ON
Company.Employee.Ssn=Company.Department.Mgr_ssn
                    --ORDER BY Company.Department.Mgr_Start_Date
                    WHERE Mgr_start_date = (SELECT MIN(Mgr_start_date) from
Company.Department);

    SELECT * FROM @Managers
END
GO
```

c)

```
USE[master]
GO
CREATE TRIGGER [dbo].[T_Only_One_Dept] ON Company.Department
INSTEAD OF INSERT, UPDATE
AS
    BEGIN
        IF (SELECT count(*) FROM inserted) > 0
            BEGIN
                DECLARE @ssn AS INT;
                SELECT @ssn = Mgr_ssn FROM inserted;

                IF (@ssn) IS NULL OR ((SELECT count(*) FROM Employee WHERE
Ssn=@ssn) = 0)
                    RAISERROR('Employee Inexistent', 16, 1);
                ELSE
                    BEGIN
                        IF (SELECT COUNT(Dnumber) FROM Company.Department WHERE
Mgr_ssn=@ssn) >=1
                            RAISERROR('Not allowed to have employee managing more
than one department', 16, 1);
```

```
                        ELSE
                            INSERT INTO Company.Department SELECT * FROM inserted;
                    END
            END
    END
GO
```

d)

```
USE[master]
GO

CREATE TRIGGER vencimento_inferior ON Company.Employee AFTER INSERT, UPDATE
AS
    BEGIN
        DECLARE @vencimento_employee AS INT;
        DECLARE @vencimento_manager AS INT;
        DECLARE @ssn AS INT;
        DECLARE @dno AS INT;


        SELECT @ssn=inserted.Ssn, @vencimento_employee=inserted.Salary,
@dno=inserted.Dno FROM inserted;
        SELECT @vencimento_manager=Company.Employee.Salary FROM Company.Department
            INNER JOIN Company.Employee ON
Company.Department.Mgr_Ssn=Company.Employee.Ssn
            WHERE @dno=Company.Department.Dnumber;

        IF @vencimento_employee > @vencimento_manager
        BEGIN
            UPDATE Company.Employee SET
Company.Employee.Salary=@vencimento_manager-1
            WHERE Company.Employee.Ssn=@ssn
        END
    END

SELECT * FROM Company.Employee;
```

e)

```
USE[master]
GO

CREATE FUNCTION dbo.InfoProjects (@ssn INT) RETURNS @table TABLE([name]
VARCHAR(45), [location] VARCHAR(15))
AS
    BEGIN
        INSERT @table
            SELECT Company.Project.Pname, Company.Project.Plocation FROM
```

```
Company.Project
                INNER JOIN Company.Works_on ON
Company.Works_on.Pno=Company.Project.Pnumber
                WHERE Works_on.Essn=@ssn
        RETURN;
    END
GO
```

f)

```
CREATE FUNCTION Company.empregados_mais_bem_pagos (@dno INT)
RETURNS @table TABLE (
    [ssn] INT,
    [fname] VARCHAR(15),
    [lname] VARCHAR(15)
)
AS
BEGIN
    INSERT INTO @table ([ssn], [fname], [lname])
    SELECT e.Ssn, e.Fname, e.Lname
    FROM Company.Employee e
    JOIN (
        SELECT Dno, AVG(Salary) AS avg_sal
        FROM Company.Department d
        JOIN Company.Employee e2 ON d.Dnumber = e2.Dno
        GROUP BY Dno
    ) AS dep_avg_sal ON dep_avg_sal.Dno = e.Dno AND e.Salary > dep_avg_sal.avg_sal
    WHERE e.Dno = @dno;

    RETURN;
END

SELECT Department.Dno, AVG(Employee.Salary) AS avg_sal
    FROM Company.Department
    JOIN Company.Employee ON Department.Dno = Employee.Dnumber
    GROUP BY Department.Dno;
SELECT * FROM Company.Employee;
SELECT * FROM Company.empregados_mais_bem_pagos(1);
SELECT * FROM Company.empregados_mais_bem_pagos(2);
SELECT * FROM Company.empregados_mais_bem_pagos(3);
```

g)

```
CREATE FUNCTION dbo.employeeDeptHighAverage (@Dno INT)
RETURNS TABLE
AS
RETURN (
    SELECT p.Pname, p.Pnumber, p.Plocation, p.Dnum,
           e.Salary / 40 AS Budget,
```

```
            SUM(e.Salary / 40) OVER () AS TotalBudget
    FROM Company.Project p
    JOIN Company.Works_on w ON p.Pnumber = w.Pno
    JOIN Company.Employee e ON w.Essn = e.Ssn
    WHERE p.Dnum = @Dno
)
```

h)

```
-----------------------------------------------trigger do tipo after---------------
----------------------------------------
CREATE TRIGGER delete_to_table_after
ON Department
AFTER DELETE
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA =
'myschema' AND TABLE_NAME = 'mytable')
    BEGIN
        CREATE TABLE mytable (
            Dnumber INT,
            Dname VARCHAR(100),
            Mgr_ssn INT,
            Mgr_Start_Date DATE
        );
    END;

    INSERT INTO mytable (Dnumber, Dname, Mgr_ssn, Mgr_Start_Date)
    SELECT Dnumber, Dname, Mgr_ssn, Mgr_Start_Date
    FROM deleted;
END;

-------------------------------------------trigger do tipo instead of------------
----------------------------------------
CREATE TRIGGER delete_to_table_instead_of
ON Company.Department
INSTEAD OF DELETE
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA =
'myschema' AND TABLE_NAME = 'mytable')
    BEGIN
        CREATE TABLE mytable (
            Dnumber INT,
            Dname VARCHAR(100),
            Mgr_ssn INT,
            Mgr_Start_Date DATE
        );
    END;

    INSERT INTO mytable (Dnumber, Dname, Mgr_ssn, Mgr_Start_Date)
```

```
    SELECT Dnumber, Dname, Mgr_ssn, Mgr_Start_Date
    FROM deleted;

    DELETE FROM Department
    WHERE Dnumber IN (SELECT Dnumber FROM deleted);
END;

-----------------------------------------vantagens e desvantagens---------------
-----------------------------------------
Quanto ao trigger after este ocorre após a instrução ser executada, neste caso, a
instrução DELETE na tabela Department, ou seja, caso haja algum erro na eliminação
de dados, o trigger não é capaz de impedir a exclusão do departamento da tabela,
tornando este trigger menos eficiente e representando uma desvantagem do mesmo. No
entanto, quanto às vantagens, podem existir vários trigger after por tabela ou
podem mesmo ser utilizados para processos complexos de validação de dados
envolvendo várias tabelas. Podem também ser utilizados para realizar instruções
adicionais depois de concluída a instrução desejada.

Quanto ao trigger instead of este ocorre antes da instrução DELETE na tabela
Department que acaba por não ser executada, ao contrário do anterior, pois é
substituída pela instrução do trigger. Assim, se ocorrer algum erro durante a
exclusão do departamento da tabela, o trigger pode impedir essa mesma exclusão, ou
seja, cabe-lhe a ele efetuar ou não a instrução pretendida. No entanto, este tipo
de trigger acaba por ser menos flexível que o anterior e pode causar erros
inesperados mais facilmente.
```

i)

As stored procedures sâo pedaços de código compilados anteriormente que quando são armazenados na base de dados podem ser chamados a qualquer altura e são mais utlizados para descomplicar operações difÍceis no SQL, melhorando assim o desempenho do código.

Por outro lado, as UDFs são usadas principalmente para encapsular operações comuns, como cálculos matemáticos, formatação de strings ou manipulação de datas. Podem ser usadas no SQL como se fossem funções do sistema, tornando-as muito úteis para operações repetitivas.

```

```