UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho Experimental
Fase 3 – Trabalho C

Stored Procedures
Triggers

Licenciatura em Engenharia Informática
Base de Dados

Paulo Nogueira Martins
Daniel Moreira Lopes Alexandre

Autores
Diogo António Costa Medeiros n.º 70633
Pedro Miguel Cunha da Silva n.º 70649
Rui João Barros Pinto n.º 70648

Vila Real, junho 2021

ÍNDICE

1. INTRODUÇÃO

SQL foi desenvolvida pela IBM Corporation no final da década de 70. Foi adotada como um padrão nacional pelo Instituto Nacional Americano de Padrões (ANSI) em 1986 e pela Organização Internacional de Normalização (ISO) em 1987. (Kroenke & Auer, 2016)

SQL não é uma linguagem de programação completa, como o Java ou C#. É, sim uma sublinguagem de dados pois apenas contém as declarações necessárias para criar e processar dados e metadados de uma base de dados. (Kroenke & Auer, 2016)

As declarações SQL dividem-se, habitualmente, em várias categorias, das quais se destacam:

- Declarações de linguagem de definição de dados (DDL) que são usadas na criação de tabelas, relações e outras estruturas;
- Declarações de linguagem de manipulação de dados (DLL) que são usadas para consulta, inserção, modificação, e eliminação de dados.

2. ENQUADRAMENTO TEÓRICO

INSERT é uma declaração SQL que permite adicionar um ou mais registos a uma tabela de uma base de dados. Estas declarações seguem a seguinte sintaxe:

- INSERT INTO *tabela* (*coluna1,* [*coluna2, ...*]) VALUES (*valor1,* [*valor2, ...*])

Os valores inseridos pela declaração INSERT devem satisfazer todas as restrições presentes na respetiva tabela, tais como chaves (primárias e estrangeiras), restrições CHECK e restrições NOT NULL). Por outro lado, é possível inserir um valor por defeito desde que a respetiva coluna contenha uma restrição DEFAULT, através da omissão da mesma. (Insert (SQL), 2021)

A estrutura básica duma query SQL consiste de 3 cláusulas: SELECT, FROM e WHERE, sendo este último opcional. Uma query recebe como input as relações listadas na cláusula FROM, opera sobre estas de acordo com o especificado nas cláusulas WHERE e SELECT e, de seguida, produz uma relação como resultado. (Silberschatz, Korth, & Sudarshan, 2020)

Por vezes, pretende-se consultar todos os registos, sem restringir as colunas. Nestas situações, um asterisco '*' pode ser usado para indicar que a consulta deve devolver todas as colunas das tabelas listadas. (Select (SQL), 2021)

SELECT é a instrução mais complexa em SQL, com palavras-chave e cláusulas opcionais, i.e.:

- A cláusula FROM pode incluir subcláusulas JOIN opcionais para especificar as regras de junção de tabelas: FROM tabela1 [tipo] JOIN tabela2 [ON (condição)], onde o [tipo] (opcional) pode ser INNER, LEFT [OUTER], RIGHT [OUTER] ou CROSS. (Microsoft Corporation, 2021)
- A cláusula WHERE inclui um predicado de comparação, que restringe as linhas retornadas pela consulta. Este pode ser composto por várias condições, que podem incluir operadores de comparação ou predicados do tipo LIKE, BETWEEN e IN, usando os operadores AND e OR.
- A cláusula GROUP BY projeta linhas com valores comuns num conjunto menor de linhas. GROUP BY é frequentemente usado em conjunto com funções de agregação SQL, tais como MAX, MIN, AVG, SUM ou COUNT, ou para eliminar linhas duplicadas de um conjunto de resultados. A cláusula WHERE é aplicada antes da cláusula GROUP BY.
- A cláusula HAVING inclui um predicado usado para filtrar linhas resultantes da cláusula GROUP BY. Como ela atua sobre os resultados desta cláusula, as funções de agregação podem ser usadas no predicado da cláusula HAVING.
- A cláusula ORDER BY identifica quais as colunas a usar para ordenar os dados resultantes e qual a ordem, crescente (ASC) ou decrescente (DESC).
- Sem uma cláusula ORDER BY, a ordem das linhas retornadas por uma consulta SQL é indefinida.
- A palavra-chave DISTINCT elimina dados duplicados.

Stored procedure é um programa armazenado na base de dados e compilado quando é usado. Estes podem receber parâmetros de entrada e devolver resultados e ser executados por qualquer processo que use a base de dados, desde que tenha as permissões adequadas para fazê-lo.

Trigger é um programa armazenado executado pelo SGBD quando um evento específico ocorre. Um *trigger* é chamado por um pedido SQL DML INSERT, UPDATE, ou DELETE na tabela ou vista à qual está ligado. Ao contrário dos *triggers* que estão ligados a uma dada tabela ou vista, os *stored procedures* estão ligados à base de dados. (Kroenke & Auer, 2016)

3. OBJETIVOS DO TRABALHO PRÁTICO

Para a base de dados implementada, foram solicitadas a criação de *Stored Procedures*, bem como de *Triggers* que respondessem aos cenários propostos.

4. DESENVOLVIMENTO

```
/* 1. Crie um procedimento que dados [telefone (paciente), nome e
apelido (médico), data] verifique se o médico está a operar nessa data
e caso não esteja agende uma operação para o paciente. O procedimento
deve ter como argumento de saída a especialidade do médico. */

CREATE PROCEDURE VerifyDisp (@Telefone INTEGER, @Nome VARCHAR(50),
@Apelido VARCHAR(50), @Data DATE, @Esp VARCHAR(50) OUTPUT)
AS
BEGIN
      DECLARE @ID_Pac INTEGER,
              @ID_Med INTEGER,
              @ID_Op  INTEGER,
              @ID_Enf INTEGER,
              @ID_Aux INTEGER

      SELECT @ID_Med = ID_Med, @Esp = Especialidade
      FROM NIFs N, Pessoas P, Funcionarios, Medicos
      WHERE Nome LIKE @Nome
      AND Apelido LIKE @Apelido
      AND N.NIF = P.NIF
      AND ID = ID_Func
      AND ID_Func = ID_Med

      IF (@ID_Med IS NULL)
      BEGIN
            PRINT ('O médico não existe')
            RETURN -1
```

```
END

SELECT @ID_Pac = ID_Pac
FROM NIFs N, Pessoas P, Pacientes
WHERE Telefone = @Telefone
AND N.NIF = P.NIF
AND ID = ID_Pac

IF (@ID_Pac IS NULL)
BEGIN
      PRINT ('Não existe um paciente com o telefone dado')
      RETURN -1
END

SELECT *
FROM Agendar
WHERE ID_Med = @ID_Med
AND CONVERT(date, Data_Op) = @Data

IF (@@ROWCOUNT != 0)
BEGIN
      PRINT ('O médico já tem operações agendadas para essa data')
      RETURN -1
END

SELECT @ID_Enf = E.ID_Enf
FROM Enfermeiros E
LEFT JOIN (SELECT ID_Enf
                FROM Agendar
                WHERE CONVERT(date, Data_Op) = @Data
                GROUP BY ID_Enf) SQ1
ON E.ID_Enf = SQ1.ID_Enf
WHERE SQ1.ID_Enf IS NULL
IF (@ID_Enf IS NULL)
BEGIN
      PRINT ('Não há enfermeiros disponíveis')
```

```
            RETURN -1
      END

      SELECT @ID_Aux = ID_Aux
      FROM Auxiliares

      INSERT INTO Info_Op (Data_Op)
      VALUES (@Data)
      SET @ID_Op = @@IDENTITY

      INSERT INTO Operar (ID_Op, ID_Med, ID_Enf, ID_Pac)
      VALUES (@ID_Op, @ID_Med, @ID_Enf, @ID_Pac)

      INSERT INTO Local_Op(ID_Op,ID_Med,ID_Enf,ID_Pac,Data_Op,Local_Op)
      VALUES (@ID_Op, @ID_Med, @ID_Enf, @ID_Pac, @Data, '')

      INSERT INTO Agendar (ID_Op,ID_Med,ID_Enf,ID_Pac,ID_Aux,Data_Op)
      VALUES (@ID_Op, @ID_Med, @ID_Enf, @ID_Pac, @ID_Aux, @Data)

      RETURN 1
END

/* 2. Assumindo que o salário dos enfermeiros é complementado com um
valor calculado em função das operações em que participam, das quais
recebem 5% do valor, crie um procedimento que para um dado mês e ano,
apresente [ID, nome, apelido, total que cada um recebe nesse mês]. */

CREATE PROCEDURE SalarioEnfs (@Mes INTEGER, @Ano INTEGER)
AS
BEGIN
      SELECT ID_Func AS ID_Enf, Nome, Apelido, (Salario + ISNULL(Extra,
0)) AS Total
      FROM Funcionarios, Pessoas P, NIFs N,
            (SELECT E.ID_Enf, Extra
             FROM Enfermeiros E
             LEFT JOIN (SELECT O.ID_Enf, SUM(Preco * 0.05) Extra
```

```
                              FROM Info_Op I, Operar O, Preco_Pag P
                              WHERE I.ID_Op = O.ID_Op
                              AND O.ID_Op = P.ID_Op
                              AND O.ID_Med = P.ID_Med
                              AND O.ID_Enf = P.ID_Enf
                              AND O.ID_Pac = P.ID_Pac
                              AND MONTH(Data_Op) = @Mes
                              AND YEAR(Data_Op) = @Ano
                              GROUP BY O.ID_Enf) SQ1
                ON E.ID_Enf = SQ1.ID_Enf) SQ2
      WHERE SQ2.ID_Enf = ID_Func
      AND ID_Func = ID
      AND P.NIF = N.NIF
END

/* 3. Crie um trigger que apenas deixe inserir registos no
relacionamento inquérito se o paciente tiver alergias. */

CREATE TRIGGER InserirInq
ON Descricoes -- A tabela Inqueritos referencia a tabela Descricoes
INSTEAD OF INSERT
AS
BEGIN
      INSERT INTO Descricoes (ID_Pac, Data_Inq, Descricao)
      SELECT ID_Pac, Data_Inq, Descricao
      FROM inserted,
            (SELECT PA.ID_Pac ID
             FROM Paciente_Alergia PA, Pacientes P, inserted I
             WHERE I.ID_Pac = P.ID_Pac
             AND P.ID_Pac = PA.ID_Pac
             GROUP BY PA.ID_Pac) SQ1
      WHERE ID_Pac = ID
END
```

5 CONCLUSÃO

Face ao trabalho desenvolvido, crê-se ter atingido os objetivos definidos para esta etapa, nomeadamente foram solucionados os cenários propostos.

De referir que relativamente ao primeiro cenário, dado que o protocolo apenas definiu dados para o médico, paciente e data da cirurgia, no caso do enfermeiro, optou-se por aquele que estivesse disponível na respetiva data. Quanto ao auxiliar, optou-se de forma aleatória o mesmo.

6 BIBLIOGRAFIA

Insert (SQL). (19 de maio de 2021). Obtido de Wikipédia, a enciclopédia livre: https://pt.wikipedia.org/wiki/Insert_(SQL)

Kroenke, D. M., & Auer, D. J. (2016). *Database Processing: Fundamentals, Design, and Implementation.* Edinburgh Gate, Harlow, Essex CM20 2JE, England: Pearson Education.

Microsoft Corporation. (19 de maio de 2021). *Microsoft SQL documentation - SQL Server*. Obtido de Microsoft Docs: https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15

Select (SQL). (19 de maio de 2021). Obtido de Wikipédia, a enciclopédia livre: https://pt.wikipedia.org/wiki/Select_(SQL)

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts.* 2 Penn Plaza, New York, NY 10121: McGraw-Hill Education.