

## Índice

INTRODUÇÃO .....	2
A LINGUAGEM SQL PARA DEFINIÇÃO DE DADOS .....	4
A LINGUAGEM SQL PARA MANIPULAÇÃO DE DADOS.....	8
REFERÊNCIAS .....	28
ANEXO 1 .....	30



## INTRODUÇÃO

Neste capítulo abordaremos a linguagem SQL baseada no modelo relacional e que constitui um standard para a manipulação e definição de dados.

O SQL é uma linguagem declarativa, portanto, o programador limita-se a indicar o objectivo que pretende atingir.

O primeiro passo na criação de uma BD corresponde a criar o suporte em disco para a sua estrutura. Depois, é necessário a produção de aplicações para operar os dados da BD.

Assim, iremos apresentar alguns aspectos da linguagem SQL que nos permite atingir estes objectivos.

De uma forma geral iremos utilizar as seguintes tabelas de forma a fornecer um exemplo concreto da linguagem:

### Empregados

Cod_emp	Nome	Cargo	N_Superior	Data	Ordenado	Cod_Depto
7369	Smith	Secretario	7902	13-06-1983	80.000,00 Esc.	20
7499	Allen	Vendedor	7698	15-08-1983	160.000,00 Esc.	30
7521	Ward	Vendedor	7698	26-03-1984	125.000,00 Esc.	30
7566	Jones	Gestor	7839	31-10-1983	297.500,00 Esc.	20
7654	Martin	Vendedor	7698	05-12-1983	125.000,00 Esc.	30
7698	Blake	Gestor	7839	11-06-1984	285.000,00 Esc.	30
7782	Clark	Gestor	7839	14-05-1984	245.000,00 Esc.	10
7788	Scott	Analista	7566	05-03-1984	300.000,00 Esc.	20
7839	King	Presidente		09-07-1984	500.000,00 Esc.	10
7844	Turner	Vendedor	7698	04-06-1984	150.000,00 Esc.	30
7876	Adams	Secretario	7788	04-06-1984	110.000,00 Esc.	20
7900	James	Secretario	7698	23-07-1984	95.000,00 Esc.	30
7902	Ford	Analista	7566	05-12-1983	300.000,00 Esc.	20
7934	Miller	Secretario	7782	21-11-1983	130.000,00 Esc.	10

**Departamento**

Cod_Depart	Nome_Pep	Localização
10	Contabilidade	Beja
20	Investigação	Évora
30	Vendas	Faro
40	Operações	Portalegre



## A LINGUAGEM SQL PARA DEFINIÇÃO DE DADOS

### a) Criação da estrutura

Para a definição da estrutura de dados o SQL possui uma instrução para criar tabelas. Através desta instrução podemos definir os atributos, os respectivos domínios de cada tabela e também algumas restrições de integridade. A sintaxe dessa instrução é:

```
CREATE TABLE <nome_da_tabela> (  
Atributo1 tipo [(tamanho)] [Not Null] [Índice] ...,  
Atributo2 tipo [(tamanho)] [Not Null] [Índice] ...,  
CONSTRAINT Nome <restrições> )
```

Os elementos em parêntesis rectos são opcionais. O elemento Atributo1 representa o nome do atributo da tabela. O elemento *tipo* representa o domínio de cada atributo (INTEGER, SMALLINT, DECIMAL, DOUBLE PRECISION, FLOAT, CHAR, VARCHAR, BIT, DATE, TIME, etc.). Ver anexo 1.

Exemplo:

```
CREATE TABLE Alunos (  
Cod_Aluno SMALLINT PRIMARY KEY,  
Nome CHAR(20) NOT NULL,  
Idade SMALLINT,  
Cod_Curso SMALLINT,  
FOREIGN KEY (Cod_Curso)  
REFERENCES Cursos (Cod_Curso)  
ON UPDATE CASCADE  
ON DELETE SET NULL,  
CHECK (Idade BETWEEN 17 AND 70)
```

Neste exemplo o Cod\_Aluno é a chave primária e é identificada com as palavras PRIMARY KEY. NOT NULL é a restrição mais simples e indica que o conteúdo desse atributo não é nulo para o caso do Nome. A restrição CHECK permite verificar se o conteúdo de um atributo satisfaz determinada condição. No exemplo, a restrição obriga que a idade do aluno esteja entre 17 e 70 anos.

As outras palavras chaves permite-nos estabelecer restrições de integridade referencial. Recorrendo ao exemplo observamos a restrição de integridade relativa à chave estrangeira Cod\_Curso importada da tabela Cursos. As ações ON DELETE SET NULL e ON UPDATE CASCADE existentes na tabela referenciadora (Alunos) permite que perante a remoção de um Cod\_Curso na tabela Cursos implicará colocar valores nulos às mesmas ocorrências na tabela Alunos e se o Cod\_Curso for alterado na tabela Cursos implicará a alteração do Cod\_Curso na tabela Alunos, respectivamente.

Uma outra restrição importante é a que permite identificar um ou vários atributos como chave candidata e é representada pela palavra UNIQUE. Exemplo, na tabela cujos atributos são Cod\_fornecedor e Nome\_fornecedor, o Cod\_fornecedor é a chave primária e o Nome\_fornecedor é a chave candidata. Assim,

```
CREATE TABLE Fornecedores (  
Cod_fornecedor SMALLINT PRIMARY KEY,  
Nome_fornecedor CHAR(20) NOT NULL,  
UNIQUE (Nome_fornecedor) )
```

Outro aspecto a considerar relativamente as chaves primárias é a sintaxe utilizada quando a chave primária é composta por mais do que um atributo. Considere que os atributos num\_factura e cod\_produto são a chave primária de uma tabela, então a sintaxe utilizada seria: PRIMARY KEY(num\_factura, cod\_produto).

## b) Alteração da estrutura

Outra operação que se pode realizar sobre a estrutura de dados é a alteração das tabelas existentes. Estas alterações consistem em adicionar novos atributos ou restrições de integridade, na modificação das características dos atributos e na eliminação de atributos ou das restrições de integridade existentes.

Exemplo:

**ALTER TABLE Alunos**

**ADD COLUMN nacionalidade CHAR(15)**

Comando para adicionar o atributo nacionalidade à tabela Alunos.

**ALTER TABLE Alunos**

**DROP COLUMN nacionalidade**

Comando para eliminar o atributo nacionalidade à tabela Alunos.

**DROP TABLE Alunos**

Comando para eliminar a tabela Alunos

## b) Criação de View

As *Views* são entidades que permitem a visualização do conteúdo das tabelas existentes na BD, portanto são tabelas sem existência física. A definição de cada *View* é armazenada no dicionário de dados e só quando é executado o seu conteúdo (que depende das tabelas envolvidas na definição) é calculado. Contudo, este processo é completamente transparente ao nível aplicacional, garantindo uma independência lógica.

Sintaxe:

CREATE VIEW <nome> [(<atributos>)]

AS <query>

Exemplo:

**CREATE VIEW topo\_de\_gama (Cod\_prod, designação)**

**AS (SELECT Cod\_prod, designação, preço**

**FROM Produtos**

**WHERE preço > 10.000)**

Como podemos observar as *views* depois de criadas, comportam-se como tabelas base para a consulta de dados.

Relativamente à actualização das *views* podemos dizer que genericamente todas as *views* que derivam de uma única tabela são actualizáveis, desde que o esquema da *view* inclua a

chave primária, ou alguma chave candidata, da tabela. Outro aspecto que deve ser considerado para que a *view* possa ser actualizada refere-se ao facto de que no esquema da *view* deve existir todos os atributos da tabela base que não são nulos (NOT NULL), especificamente para a *view* que pretende fazer inserções. Por último, outros aspectos a considerar para que a *view* possa ser actualizada depende da não existência de funções de agregação.

Outra operação aplicada as *views* é a de remoção, o que implica a remoção da definição da *view* do dicionário de dados do sistema.

Sintaxe:

```
DROP VIEW <nome>
```

#### d) Criação de *Index*

O *Index* é uma entidade que nos permite realizar eficientemente uma pesquisa dos dados. O *index* não é seleccionado pelo utilizador. Depois de definidos é o sistema que decide que *index* usar perante o resultado que o utilizador pretender.

Sintaxe:

```
CREATE [UNIQUE] INDEX<nome>  
ON <nome_da_tabela> ( noma_atributo1 [ASC|DES], ..)
```

Os parêntesis rectos indicam que esses elementos são opcionais. A palavra UNIQUE indica que esse índice não pode ter valores duplicados. Assim, a opção PRIMARY indica que o(s) atributo(s) envolvido(s) são a chave primária.

*Nota1: como é lógico não é necessário utilizar a opção UNIQUE quando utilizamos a opção PRIMARY.*

*Nota2: só é possível criar um index de cada vez com este comando.*

Exemplo:

```
CREATE INDEX Chave_prim  
ON Clientes (Cod_cliente);
```

Outra operação aplicada ao *index* é a de remoção.

Sintaxe:

```
DROP INDEX <nome> ON <nome_da_tabela>
```

## A LINGUAGEM SQL PARA MANIPULAÇÃO DE DADOS

A SQL possui dois grupos de comandos para a manipulação de dados, um para a consulta da BD e outro para a actualização da BD.

### a) Comandos para a actualização da BD

INSERT INTO é um dos comandos do SQL que permite a introdução de dados nas tabelas.

Sintaxe:

```
INSERT INTO <nome_tabela> [(atributo1,atributo2, atributo3,...)]
```

```
VALUES (<valores>)
```

Os parêntesis rectos indicam que esses elementos são opcionais. Os dados de cada atributo (separados por vírgulas) são introduzidos através da palavra VALUES.

Exemplos:

```
INSERT INTO Departamento
```

```
VALUES (50, 'Manutenção', 'Lisboa')
```

Resultado: Tabela Departamento

Cod_Depart	Nome_pep	Localização
50	Manutenção	Lisboa

```
INSERT INTO Departamento (Cod_Depart, Nome_pep, Localização)
```

```
VALUES (50, 'Manutenção', 'Lisboa')
```

O resultado é o mesmo que o exemplo anterior. Esta sintaxe permite inserir dados nas colunas (atributos) especificadas.

A alteração de valores em um ou mais atributos numa tabela e com critérios específicos é realizada através do comando UPDATE.

Sintaxe:

```
UPDATE <nome_tabela>
```

```
SET <atributo> = <expressão>,
```

...

```
[WHERE <condição>]
```

A palavra SET define quais são os atributos que se pretende actualizar e os novos valores para esse atributo. A palavra WHERE é opcional e é utilizada quando se pretende condicionar os tuplos.

Exemplo:

```
UPDATE Empregado  
SET Ordenado = Ordenado + 50.000  
WHERE Num_Superior=7589
```

Este conjunto de comandos soma ao ordenado mais 50.000 dos empregados cujo superior tenha o código 7589.

A eliminação de valores em um ou mais tuplos numa tabela e com critérios específicos é realizada através do comando DELETE.

Sintaxe:

```
DELETE FROM <nome_tabela>  
[WHERE <condição>]
```

Exemplo:

```
DELETE FROM Empregado  
WHERE Num_Superior=7589
```

Este conjunto de comandos elimina os empregados cujo superior tenha o código 7589.

## **b) Comandos para a consulta da BD**

As instruções fundamentais para a realização de consultas são:

```
SELECT <atributo1, atributo2,...>  
FROM <nome_tabela1, nome_tabela2,...>  
[WHERE <condição>]
```

A comando SELECT selecciona conjunto de atributos de uma(s) tabela(s) dada(s) pelo comando FROM.

Exemplo:

```
SELECT Cod_aluno, Nome, Morada  
FROM Aluno
```

O resultados deste código é:

Cod_aluno	Nome	Morada
1	Maria	Beja
2	Mário	Évora
...	..	..

O comando SELECT permite incluir expressões aritméticas e modificar o nome dos atributos. Uma expressão pode ser uma combinação de valores, operadores e funções que produzem um valor. Os operadores aritméticos que podemos incluir são:

Operadores	Descrição
+	Soma
-	Subtracção
*	Multiplicação
/	Divisão

Exemplo:

**SELECT Cod\_emp, Ordenado\*12**

**FROM Empregado;**

Cod_emp	Expr1
7369	960.000,00 Esc.
7499	1.920.000,00 Esc.
7521	1.500.000,00 Esc.
7566	3.570.000,00 Esc.
7654	1.500.000,00 Esc.
7698	3.420.000,00 Esc.
7782	2.940.000,00 Esc.
7788	3.600.000,00 Esc.
7839	6.000.000,00 Esc.
7844	1.800.000,00 Esc.
7876	1.320.000,00 Esc.

7900	1.140.000,00 Esc.
7902	3.600.000,00 Esc.
7934	1.560.000,00 Esc.

Ao observar o resultado da consulta anterior podemos verificar que o nome da coluna não tem sentido. Utilizando o comando SELECT é possível alterar o nome na coluna.

Exemplo:

**SELECT Cod\_emp, Ordenado\*12 Ordenado\_Anual  
FROM Empregado;**

Cod_emp	Ordenado_Anual
7369	960.000,00 Esc.
7499	1.920.000,00 Esc.
7521	1.500.000,00 Esc.
7566	3.570.000,00 Esc.
7654	1.500.000,00 Esc.
7698	3.420.000,00 Esc.
7782	2.940.000,00 Esc.
7788	3.600.000,00 Esc.
7839	6.000.000,00 Esc.
7844	1.800.000,00 Esc.
7876	1.320.000,00 Esc.
7900	1.140.000,00 Esc.
7902	3.600.000,00 Esc.
7934	1.560.000,00 Esc.

Por último, e ainda utilizando o comando SELECT, é possível eliminar valores duplicados utilizando a palavra DISTINCT.

*Nota: Existem funções que poderemos utilizar com o comando SELECT, como iremos verificar mais à frente neste capítulo.*

Exemplo:

**SELECT Cod\_Depto, Cargo  
FROM Empregado  
ORDER BY Cod\_Depto;**

Cod_Depar	Cargo
10	Gestor
10	Presidente
10	Secretario
20	Secretario
20	Gestor
20	Analista
20	Secretario
20	Analista
30	Vendedor
30	Vendedor
30	Vendedor
30	Gestor
30	Vendedor
30	Secretario

Nesta consulta pretende-se saber quais são os cargos que existem em cada departamento. Como podemos observar existem linhas repetidas. Para evitar esta situação utilizamos a palavra DISTINCT.

```
SELECT DISTINCT Cod_Depar, Cargo  
FROM Empregado  
ORDER BY Cod_Depar;
```

Cod_Depar	Cargo
10	Gestor
10	Presidente
10	Secretario
20	Analista
20	Gestor
20	Secretario
30	Gestor
30	Secretario
30	Vendedor

**Nota:** Repare que a palavra *DISTINCT* se encontra a seguir à palavra *SELECT*.

O comando *ORDER BY* é utilizado para ordenar tuplos. Neste exemplo, os tuplos são ordenados por código de departamento que, por defeito, é realizado de forma ascendente, apresentando os valores numéricos mais baixos primeiro. Para inverter esta sequência, utiliza-se a palavra *DESC*.

Exemplo:

```
SELECT Cod_Emp, Data
FROM Empregado
ORDER BY Data DESC;
```

Cod_emp	Data
7900	23-07-1984
7839	09-07-1984
7698	11-06-1984
7876	04-06-1984
7844	04-06-1984
7782	14-05-1984
7521	26-03-1984
7788	05-03-1984
7902	05-12-1983

7654	05-12-1983
7934	21-11-1983
7566	31-10-1983
7499	15-08-1983
7369	13-06-1983

Esta consulta apresenta-nos o código do empregado e a data da sua entrada no departamento, de forma a que as datas mais recentes sejam apresentadas primeiro. O comando WHERE corresponde ao operador de selecção da álgebra relacional. O comando WHERE é utilizado a seguir ao comando FROM e contém uma(s) condição(s) que os tuplos têm que satisfazer para que sejam visualizados. A palavra WHERE deverá possuir três elementos:

1. O nome do atributo.
2. O operador de comparação.
3. O nome de um atributo, uma constante ou uma lista de valores.

Os operadores de comparação podem ser divididos em duas categorias: lógicos e SQL. Os operadores lógicos testam as seguintes condições:

Operador	Significado
=	Igual a
>	Maior que
>=	Maior ou igual que
<	Menor que
<=	Menor ou igual que
<>	Diferente

Para listar os nomes, os cargos e departamentos de todos os empregados cujo cargo é de secretário devemos utilizar o seguinte conjunto de comandos:

```
SELECT Nome, Cargo, Cod_Depar  
FROM Empregado  
WHERE Cargo='Secretario';
```

Relativamente aos operadores SQL, existem quatro, que operam sobre todos os tipos de dados:

Operador	Significado
BETWEEN ..AND..	Entre dois valores
IN(lista)	Corresponde a qualquer valor da lista
LIKE	Cadeia de caracteres que satisfaz uma condição
IS NULL	É um valor nulo

Utilizaremos exemplos para ilustrar estes operadores.

Exemplo: Para encontrar todos os empregados que têm como superior as pessoas com código 7902, 7566 e 7788.

```
SELECT Nome, N_Superior  
FROM Empregado  
WHERE N_Superior IN (7902,7566,7788);
```

Nome	N_Superior
Smith	7902
Scott	7566
Adams	7788
Ford	7566

Exemplo: Para listar todos os nomes dos empregados que comecem por um S. Neste exercício é necessário utilizar um dos símbolos para construir a cadeia de caracteres de pesquisa. Este símbolo é %, que representa qualquer sequência com vários caracteres ou nenhum.

```
SELECT Nome
FROM Empregado
WHERE Nome LIKE 'S%';
```

Nome
Smith
Scott

Exemplo: Pretende-se encontrar o nome do empregado que não tenha superior hierárquico.

```
SELECT Nome, N_Superior
FROM Empregado
WHERE N_Superior IS NULL;
```

Nome	N_Superior
King	

Ainda sobre os operadores de comparação utilizados no comando WHERE, existem as respectivas expressões de negação:

Operador	Significado
NOT BETWEEN ..AND..	Não entre dois valores
NOT IN(lista)	Corresponde a nenhum valor da lista
NOT LIKE	Cadeia de caracteres que não satisfaz uma condição
IS NOT NULL	É um valor não nulo
<>	Diferente

Exemplo: Para encontrar os empregados cujo ordenado não esteja entre 100.000\$ e 200.000\$.

```
SELECT Nome, Ordenado  
FROM Empregado  
WHERE Ordenado NOT BETWEEN 100000 AND 200000;
```

Nome	Ordenado
Smith	80.000,00 Esc.
Jones	297.500,00 Esc.
Blake	285.000,00 Esc.
Clark	245.000,00 Esc.
Scott	300.000,00 Esc.
King	500.000,00 Esc.
James	95.000,00 Esc.
Ford	300.000,00 Esc.

Até agora realizamos consultas à BD com condições simples. Para podermos consultar dados com condições múltiplas são utilizados os operadores AND e OR, que nos permitem construir expressões lógicas compostas. O operador AND esperará que ambas as condições sejam verdadeiras para que as linhas sejam visualizadas, enquanto que o operador OR esperará que uma das condições seja verdadeira.

Pretendemos encontrar os empregados cujo cargo seja de secretário e o seu ordenado se encontre entre 100000\$ e 200000\$. Então o nosso código em SQL será:

```
SELECT Nome, Ordenado, Cargo  
FROM EMPREGADO  
WHERE Ordenado BETWEEN 100000 AND 200000  
AND Cargo='Secretário';
```

Nome	Ordenado	Cargo
Adams	110.000,00 Esc.	Secretario
Miller	130.000,00 Esc.	Secretario

Pretendemos encontrar os empregados cujo cargo seja de secretário ou o seu ordenado se encontre entre 100000\$ e 200000\$. Então o nosso código em SQL será:

```
SELECT Nome, Ordenado, Cargo
FROM Empregado
WHERE Ordenado BETWEEN 100000 AND 200000
OR Cargo='Secretário';
```

Nome	Ordenado	Cargo
Smith	80.000,00 Esc.	Secretario
Allen	160.000,00 Esc.	Vendedor
Ward	125.000,00 Esc.	Vendedor
Martin	125.000,00 Esc.	Vendedor
Turner	150.000,00 Esc.	Vendedor
Adams	110.000,00 Esc.	Secretario
James	95.000,00 Esc.	Secretario
Miller	130.000,00 Esc.	Secretario

Para por em prática todos estes conceitos efectue os seguintes exercícios:

1. Realize em linguagem SQL uma consulta que apresente todos os tipos de cargos que existem na BD INFO.
2. Realize em linguagem SQL uma consulta que apresente toda a informação dos empregados dos departamentos 10 e 20 por ordem alfabética do nome.
3. Realize em linguagem SQL uma consulta que apresente os nomes e os cargos dos secretários do departamento 20.

### **b .1.) Consultas sobre grupos de dados**

Na linguagem SQL é possível obter resultados baseados em grupos de tuplos ao contrario daquilo que temos feito até agora. Assim, existem funções de grupo que operam sobre conjuntos de tuplos. O comando GROUP BY é utilizado para dividir os tuplos de uma tabela em grupos mais pequenos. Algumas das funções de grupo são as abaixo indicadas:

Função	Valor produzido
AVG(n)	Valor médio de n
COUNT(expr)	Número de vezes que a expr toma um valor
MAX(expr)	Valor máximo de expr
MIN(expr)	Valor mínimo de expr

SUM(n)	Soma dos valores de n
--------	-----------------------

*Expr* indica os argumentos que podem ser do tipo CHAR, Number ou DATE. Todas as funções de grupo, à excepção de COUNT(\*), ignoram os valores nulos.

Torna-se ainda importante destacar que as funções de grupo por si só tratam todos os tuplos de uma tabela como um grupo.

Para calcular a média dos ordenados de todos os empregados teríamos o seguinte código em SQL:

**SELECT AVG(Ordenado)**

**FROM Empregado;**

Outro exemplo, para encontrar o ordenado mínimo recebido por um secretário, teríamos:

**SELECT MIN(Ordenado)**

**FROM Empregado**

**WHERE Cargo='secretario';**

A palavra GROUP BY é utilizada para dividir tuplos da tabela em grupos mais pequenos. As funções de grupo podem ser utilizadas para produzir informação resumida para cada grupo.

Exemplo: Pretende-se calcular o ordenado médio para cada cargo.

**SELECT Cargo, AVG(Ordenado) MédiaDeOrdenado**

**FROM Empregado**

**GROUP BY Cargo;**

Cargo	Média De Ordenado
Analista	300.000,00 Esc.
Gestor	275.833,33 Esc.
Presidente	500.000,00 Esc.
Secretario	103.750,00 Esc.
Vendedor	140.000,00 Esc.

O comando GROUP BY pode ser utilizado para obter resultados de grupos contidos em outros grupos.

Exemplo: Apresente o código em SQL que permite visualizar o ordenado mensal para cada cargo dentro de cada departamento.

**SELECT Cargo, Cod\_Depar, AVG(Ordenado) MédiaDeOrdenado**

**FROM Empregado**

**GROUP BY Cod\_Depar, Cargo**

**ORDER BY Cod\_Depar;**

Cargo	MédiaDeOrdenado	Cod_Depar
Gestor	245.000,00 Esc.	10
Presidente	500.000,00 Esc.	10
Secretario	130.000,00 Esc.	10
Analista	300.000,00 Esc.	20
Gestor	297.500,00 Esc.	20
Secretario	95.000,00 Esc.	20
Gestor	285.000,00 Esc.	30
Secretario	95.000,00 Esc.	30
Vendedor	140.000,00 Esc.	30

*NOTA: Pode observar que se incluir funções de grupo no comando SELECT, não poderá seleccionar resultados individuais a não ser que o atributo apareça no comando GROUP BY.*

Se pretender introduzir restrições ao grupo de tuplos não poderá fazê-lo com o comando WHERE, mas sim com HAVING. Esta palavra normalmente é colocada depois do comando GROUP BY, pois os grupos são formados antes de impor as restrições.

Exemplo: Apresente os cargos cujo ordenado máximo é maior ou igual a 300.000\$.

```
SELECT Cargo, MAX(Ordenado) MaxDeOrdenado
FROM Empregado
GROUP BY Cargo
HAVING MAX(Ordenado) >= 300000;
```

Cargo	MaxDeOrdenado
Analista	300.000,00 Esc.
Presidente	500.000,00 Esc.

Para por em prática estes conceitos efectue os seguintes exercícios:

- 1.- Realize em linguagem SQL uma consulta que apresente o ordenado mais baixo, mais alto e médio de todos os empregados.
- 2.- Realize em linguagem SQL uma consulta que apresente todos os departamentos que têm mais do que três empregados.

### b .2.) Subconsultas

Uma subconsulta é um comando SELECT que está encadeado noutra comando SELECT, produzindo resultados intermédios. Uma subconsulta é utilizada quando é necessário seleccionar tuplos de uma tabela com uma condição que dependa de dados da mesma tabela.

Pretende-se encontrar o empregado que ganhe o menor ordenado na empresa INFO. Como não sabemos qual é o ordenado mínimo devemos encontra-lo:

```
SELECT MIN(Ordenado)
```

```
FROM Empregado;
```

Depois devemos encontrar o empregado que ganha esse ordenado:

```
SELECT Nome, Cargo, Ordenado
```

```
FROM Empregado
```

```
WHERE Ordenado = (ordenado mínimo)
```

Assim, podemos combinar os dois códigos:

```
SELECT Nome, Cargo, Ordenado  
FROM Empregado  
WHERE Ordenado = (SELECT MIN(Ordenado)  
FROM Empregado);
```

Desta forma diríamos que existem dois blocos de consulta: a consulta externa e a consulta interna, onde a consulta interna é executada primeiro. A consulta interna tal e qual como a estamos a realizar produz apenas um resultado, no último exemplo esse resultado é o valor mínimo do ordenado.

Exemplo: Apresente o código em SQL que no indique o nome e a função dos empregados que tenham o mesmo cargo que BLAKE. Primeiro devemos obter o cargo de BLAKE e depois encontrar os empregados com esse cargo.

```
SELECT Nome, Cargo  
FROM Empregado  
WHERE Cargo = (SELECT Cargo  
FROM Empregado  
WHERE Nome='Blake');
```

Nome	Cargo
Clark	Gestor
Blake	Gestor
Jones	Gestor

A consulta que se segue pretende encontrar os empregados que ganham o menor ordenado, em cada departamento.

```
SELECT Nome, Cod_Depar, Ordenado  
FROM Empregado  
WHERE (Ordenado, Cod_Depar)IN(SELECT MIN(Ordenado), Cod_Depar  
FROM Empregado  
GROUP BY Cod_Depar);
```

Repare que a consulta interna produz mais que um valor, logo o operador utilizado é o IN pois espera-se uma lista de valores.

As subconsultas podem também ser utilizadas dentro da palavra HAVING. Suponha que se pretende obter os departamentos que tenham um ordenado médio superior ao do

departamento 30. Para obter-mos este resultado devemos calcular a média dos ordenados do departamento 30 (é apenas um valor!). Seguidamente iremos calcular a média dos ordenados de cada departamento e verificar a condição.

```
SELECT Cod_Depear, AVG(Ordenado)  
FROM Empregado  
GROUP BY Cod_Depear  
HAVING AVG(Ordenado) > (SELECT AVG(Ordenado)  
FROM Empregado  
WHERE Cod_depar = 30);
```

Para por em prática estes conceitos efectue os seguintes exercícios:

1.- Realize em linguagem SQL uma consulta (subconsulta) que apresente o nome, cargo e data dos empregado cujo ordenado é superior ao ordenado mais elevado de qualquer dos departamentos de Vendas.

2.- Realize em linguagem SQL uma consulta (subconsulta) que apresente o nome, o cargo e o ordenado dos empregado que ganham o ordenado mais elevado em cada cargo.

Outra operação a realizar sobre a BD é a junção (*join*). Uma junção é utilizada quando uma consulta SQL requer extrair dados de mais de uma tabela da BD, com base numa condição de junção. Esta condição de junção depende, geralmente, das colunas comuns que existem entre as tabelas onde se pretende realizar esta operação. Em alguns casos, pode ser necessário efectuar uma junção entre colunas da mesma tabela.

Exemplo: Realize uma consulta onde se visualize o nome do empregado, o cargo, e o nome do departamento ao qual o empregado pertence.

```
SELECT Nome, Cargo, Nome_pep  
FROM Empregado, Departamento  
WHERE Cod_Depear=Cod_Depear;
```

Nome	Cargo	Nome_Pep
Clark	Gestor	Contabilidade
King	Presidente	Contabilidade
Miller	Secretario	Contabilidade
Smith	Secretario	Investigação
Jones	Gestor	Investigação
Scott	Analista	Investigação
Adams	Secretario	Investigação
Ford	Analista	Investigação
Allen	Vendedor	Vendas
Ward	Vendedor	Vendas
Martin	Vendedor	Vendas
Blake	Gestor	Vendas
Turner	Vendedor	Vendas
James	Secretario	Vendas

Ao observar o resultado da consulta podemos verificar que para cada empregado, é mostrado o nome do departamento ao qual ele pertence. Isto é possível devido a condição que existe no comando WHERE, onde apenas são visualizados os tuplos onde o Cod\_Depar do empregado é igual ao Cod\_Depart do departamento.

O exemplo seguinte permite obter dados de mais do que uma tabela, incluindo tópicos cobertos anteriormente.

Exemplo: Apresente o nome, a localidade e o nome do departamento dos empregados cujo ordenado mensal é superior a 150000\$, ordenados pelo nome do departamento.

```
SELECT Nome, Localização, Nome_pep
FROM Empregado, Departamento
WHERE Cod_Depar=Cod_Depart
AND Ordenado > 150000
```

Nome	Localização	Nome_Pep
Allen	Faro	Vendas
Jones	Évora	Investigação
Blake	Faro	Vendas
Clark	Beja	Contabilidade
Scott	Évora	Investigação
King	Beja	Contabilidade
Ford	Évora	Investigação

O exemplo seguinte permite-nos estabelecer regras para a junção de mais de duas tabelas:  
Exemplo: Considere que tem uma BD de uma Universidade com informação dos alunos, das disciplinas e das avaliações realizadas. Assim, temos as seguintes tabelas com os respectivos atributos:

ALUNOS(Cod\_al, Nome\_al, Morada)

DISCIPLINAS(Cod\_dis, Nome\_dis, Créditos)

AVALIAÇÕES(Cod\_al, Cod\_dis, Data, Nota)

Admita que pretendia visualizar os nomes dos alunos, o nome da disciplina e a nota obtida.

```
SELECT Nome_al, Nome_dis, Nota
FROM ALUNOS, DISCIPLINAS, AVALIAÇÕES
WHERE ALUNOS.Cod_al = AVALIAÇÕES.Cod_al
AND DISCIPLINAS.Cod_dis = AVALIAÇÕES.Cod_dis;
```

Podemos concluir que para realizar a junção entre três tabelas, é necessário construir duas condições de junção, para quatro tabelas é necessário no mínimo três condições de junção e assim sucessivamente.

Outro exemplo, mostre os nomes dos empregados, cujo departamento esteja localizado em Beja.

Ainda podemos dizer que existem vários tipos de junção, *left join* e *right join*.

Vejamos um exemplo do *left join*:

```
SELECT Nome, Localização, Nome_pep
FROM Departamento left join Empregado on Cod_Depar=Cod_Depart
```

Nome	Localização	Nome_Pep
------	-------------	----------

Clark	Beja	Contabilidade
King	Beja	Contabilidade
Miller	Beja	Contabilidade
Smith	Évora	Investigação
Jones	Évora	Investigação
Scott	Évora	Investigação
Adams	Évora	Investigação
Ford	Évora	Investigação
Allen	Faro	Vendas
Ward	Faro	Vendas
Martin	Faro	Vendas
Blake	Faro	Vendas
Turner	Faro	Vendas
James	Faro	Vendas
	Portalegre	Informática

Verifique o resultado da última linha. Este tuplo tem o atributo **Nome** a *null*. Este resultado surge devido ao facto da tabela Departamento (que se encontra à esquerda do join) apresentar todos os seus tuplos mesmo que estes não possuam "ligação" com nenhum tuplo da outra tabela, neste caso da tabela Empregados.

Vejamus um exemplo de right join:

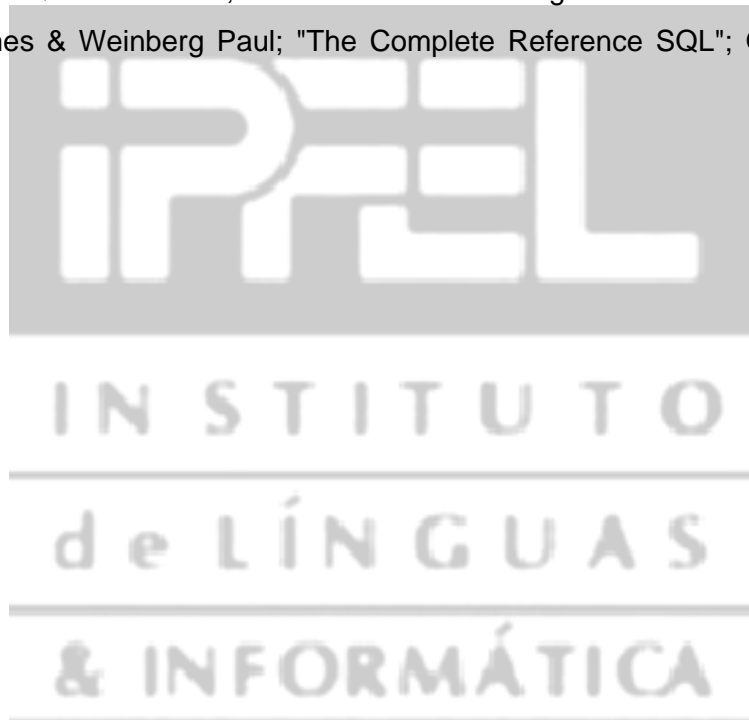
```
SELECT Nome, Localização, Nome_peg
FROM Departamento right join Empregado
on Cod_Depear=Cod_Deport
```

Nome	Localização	Nome_Pep
Clark	Beja	Contabilidade
King	Beja	Contabilidade
Miller	Beja	Contabilidade
Smith	Évora	Investigação
Jones	Évora	Investigação
Scott	Évora	Investigação
Adams	Évora	Investigação
Ford	Évora	Investigação
Allen	Faro	Vendas
Ward	Faro	Vendas
Martin	Faro	Vendas
Blake	Faro	Vendas
Turner	Faro	Vendas
James	Faro	Vendas

Este resultado surge devido ao facto da tabela Empregado (que se encontra à direita do join) apresentar todos os seus tuplos mesmo que estes não possuam "ligação" com nenhum tuplo da outra tabela, neste caso da tabela Departamento.

## REFERÊNCIAS

- Date, C. J.; "Na Introduction to Database Systems"; Sixth Edition; Addison-Wesley Publishing Company.
- Pereira, José Luís; "Tecnologia de Base de Dados"; FCA- Editora de Informática; 2.<sup>a</sup> Edição.
- Carriço, Rui Fernando, Carriço, José António; "Desenho de base de dados e linguagem SQL em Access"; CTI- Centro de Tecnologias de Informação; 1998.
- Groff, James & Weinberg Paul; "The Complete Reference SQL"; Osborne/McGraw Hill; 1998.





## ANEXO 1

### TIPOS DE DADOS SQL

TIPO	TAMANHO
TINYINT	8 bit
SMALLINT	16 bit
INTEGER	32 bit
REAL	32 bit
FLOAT(N)	64 bit
DECIMAL(P,S)	
NUMERIC(P,S)	
CHAR(N)	N<=254
VARCHAR	N<=32K
BIT	
BINARY(N)	
VARBINARY(N)	N<=32K
DATETIME	ANO, MÊS E DIA
TIMESTAMP	ANO, MÊS, DIA, HORA, MINUTO SEGUNDO.