

# BASE DE DADOS



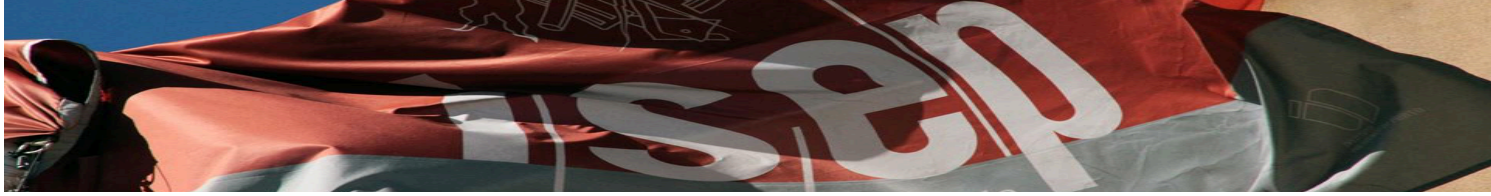
SQL –DML

Teórico-Práticas  
Ano Lectivo 2016/2017



# FUNÇÕES DE AGREGAÇÃO

- ★ Estas funções operam nos multi-conjuntos de valores de uma coluna de uma relação e retornam um valor:
  - COUNT ([DISTINCT] A): O número de valores (únicos) na coluna A
  - SUM ([DISTINCT] A): A soma de todos os valores (únicos) de A
  - AVG ([DISTINCT] A): A média de todos os valores (únicos) de A
  - MAX (A). O valor máximo existente na coluna A
  - MIN (A). O valor mínimo existente na coluna A
- ★ **Estas funções só podem ser usadas numa cláusula SELECT ou numa cláusula HAVING**
- ★ **Não é possível usar diretamente os operadores de agregação na cláusula WHERE**



# FUNÇÕES DE AGREGAÇÃO

- Se um SELECT tem uma operação de agregação, então apenas pode conter colunas com operadores de agregação, **excepto se tiver uma cláusula GROUP BY**

```
SELECT marca, Avg(Kms) FROM  
VEICULO  
GROUP BY marca
```

```
SELECT marca, Avg(Kms) FROM  
VEICULO  
GROUP BY marca  
HAVING Avg(kms) <5000
```




# SELECT - Resumo

```
SELECT localidade, Count(*) Total_empregados
FROM Empregados
WHERE salario > 3000
GROUP BY localidade
HAVING Count(*) >1
ORDER BY localidade DESC
```

Tabela Empregados

ID_EMPREGADO	NOME	ENDERECO	LOCALIDADE	SALARIO
1	Nico Seixal	Avenida da Glória 1000	Lisboa	6200
2	Ana Rita	Avenida Da Glória 1200	Lisboa	5980
3	Joana Gonçalves	Rua Antunes Almeida 114	Porto	6200
4	Paula Silva	Rua Pereira 186 1º DTO	Porto	1520
5	António Castro	Rua Direita 25 2º ESQ	Porto	2100
6	Maria Santos	Rua da Liberdade 150	Braga	1520
7	Francisco Vale	Avenida do Monte 1240	Braga	650
8	Patricia Cunha	Rua da Velhota 233 5ª ESQ	Braga	745



LOCALIDADE	TOTAL_EMPREGADOS
Lisboa	2



# OPERAÇÕES DE CONJUNTO

- ★ As operações de conjunto **Union**, **Intersect** e **Except** (**Minus em Oracle**) operam em relações e correspondem às operações  $\cup$ ,  $\cap$  e  $-$  (diferença) da álgebra relacional
- ★ Eliminam os registos repetidos; se desejarmos obter as repetições, devemos explicitar através da forma **union all**, **intersect all** e **except all**.
- ★ Tem de existir compatibilidade entre os conjuntos:
  - ★ Mesmo número de campos
  - ★ Tipos de dados compatíveis

```
SELECT * FROM veiculo  
UNION  
SELECT * FROM Automovel
```

```
SELECT * FROM veiculo  
INTERSECT  
SELECT * FROM Automovel
```

```
SELECT * FROM veiculo  
MINUS  
SELECT * FROM Automovel
```



# JUNÇÃO DE RELAÇÕES – PRODUTO CARTESIANO

- ★ Permite-nos combinar registos de relações diferentes.
- ★ Basta indicar as várias relações no operador FROM usando vírgulas.
- ★ Se as tabelas tiverem atributos com o mesmo nome, iremos ter resultados ambíguos. Neste caso devemos referenciar os atributos com o nome da tabela:
- ★ Retorna uma relação com os atributos das várias tabelas e todas as combinações possíveis de registos.

```
SELECT veiculo.nome, marca.nome  
FROM veiculo, marca;
```





## JUNÇÃO DE RELAÇÕES – JOIN

- ★ Se usarmos o produto cartesiano conjuntamente com o operador WHERE estamos a realizar a junção natural.
- ★ A condição de junção é implementada no SQL igualando a chave estrangeira de uma tabela com a respectiva chave primária da outra.
- ★ Em junções que envolvam mais de duas tabelas, deve existir pelo menos uma condição de junção para cada uma das tabelas.

```
SELECT Cliente.Nome, Profissao.Cargo  
FROM Cliente, Profissao  
WHERE Cliente.profissao=Profissao.Cod_profissao
```

**ou**

```
SELECT Cliente.nome, Profissão.Cargo  
FROM Cliente INNER JOIN Profissão  
ON Cliente.Cod_profissao = Pedido.Cod_profissao
```



# JUNÇÃO DE RELAÇÕES – JOIN

- ★ Existem várias variantes:
  - **Left Outer Join** – além dos registos que fazem match, os registos da tabela esquerda que não existam na direita também aparecem. Quando não existe numa das tabelas os valores aparecem a NULL;
  - **Rigth Outer Join** – além dos registos que fazem match, aparecem todos os restantes registos da tabela direita que não existam na esquerda;
  - **Full Outer Join** – aparecem todos os registos independentemente de fazerem ou não match.
- ★ Com o natural join apenas aparecem os que fazem match. Tem o mesmo efeito que igualar ao campos de ambas as tabelas na cláusula WHERE





# SUBQUERIES

- ★ *Uma* Subquery é uma instrução SELECT que está embebida numa cláusula de outra instrução SELECT.
- ★ Elas podem ser úteis quando precisamos de selecionar linhas de uma tabela com uma condição que depende dos dados na própria tabela.
- ★ Podemos colocar a subconsulta na cláusula WHERE, HAVING E FROM.

**SELECT**  
**FROM**  
**WHERE**

**ListaDeColunas**  
**TabelaExterna**  
**Expressao Operador**

**(SELECT**  
**FROM**  
**[WHERE**

**ListaDeColunas**  
**TabelaInterna**  
**Expressão Operador]);**



# SUBQUERIES

- ★ A *subquery* geralmente será executada primeiro, e a sua saída é usada para completar a condição de consulta para a consulta principal (ou externa).
- ★ As *subqueries* podem estar correlacionadas ou não.
- ★ As *subqueries* podem retornar uma só linha ou várias linhas
  - As que retornam um só linha só podem mencionar operadores aritméticos
  - As que retornam várias linhas podem mencionar os operadores:  
IN ( NOT IN) ; ANY(SOME); ALL ; EXISTS (NOT EXISTS)



## SUBQUERIES – Operador IN

- ✦ Seleciona as linhas em que os campos indicados antes do operador existam na *subquery*.
- ✦ Os campos indicados têm de ser no mesmo número dos campos retornados pela query e têm de ter domínios compatíveis.
- ✦ O operador NOT IN permite obter o resultado inverso.

```
SELECT nome FROM empregado  
WHERE bi NOT IN (SELECT bi_dir FROM departamento)
```



## SUBQUERIES – Operador ANY

- ★ Seleciona os resultados cujos campos indicados sejam iguais (=), maiores (>), menores(<) ou diferentes (<>) do **que pelo menos uma linha** da subquery.
- ★ Os campos indicados têm de ser no mesmo número dos campos retornados pela subquery e têm de ter domínios compatíveis.
- ★ =ANY é o mesmo que IN
- ★ SOME é o mesmo que ANY

```
SELECT nome FROM empregado
```

```
WHERE bi =ANY (SELECT bidir FROM departamento)
```



## SUBQUERIES – Operador ALL

- ★ Selecciona os resultados cujos campos indicados sejam iguais (=), maiores (>), menores(<) ou diferentes(<>) **do que todos os** tuplos da subquery.
- ★ Os campos indicados têm de ser no mesmo número dos campos retornados pela query e têm de ter domínios compatíveis.
- ★ <>ALL é o mesmo que NOT IN

```
SELECT nome FROM empregado  
WHERE salario >=ALL  
      (SELECT salario FROM empregado JOIN departamento  
        ON (numdep = num))
```



# SUBQUERIES – NÃO CORRELACIONADAS

- ★ SELECT Interior
  - Não depende de valores do SELECT exterior
  - Não referência colunas do SELECT exterior
- ★ SELECT Interior é executado em 1º lugar porque o SELECT exterior é que depende do SELECT interior.

```
SELECT A1.Nome  
FROM Aluno A1  
WHERE A1.idade= ( SELECT MIN(A2.idade)  
                  FROM Aluno A2 )
```





# SUBQUERIES – CORRELACIONADAS

- ✳ SELECT Interior depende de valores do SELECT exterior
- ✳ SELECT Interior é executado tantas vezes quanto o SELECT exterior e espera por um valor do SELECT exterior

```
SELECT Nome, Salario
FROM Pessoa P
WHERE Salario < ( SELECT SUM(Valor)
                  FROM Comissao C WHERE C.Id = P.Id )
```



# SUBQUERIES – CORRELACIONADAS

- ★ Em subqueries correlacionadas todos os operadores lógicos são aplicados, contudo usa-se o operador **EXISTS** ou **NOT EXISTS** para testar se um valor recuperado pela consulta externa existe no conjunto de valores recuperados pela consulta interna.
- ★ O operador EXISTS ou NOT EXISTS é usado para determinar se há dados numa lista de valores (restringe o conjunto de resultados de uma consulta externa para as linhas que atendam a subquery).
- ★ O operador EXISTS e NOT EXISTS retorna TRUE ou FALSE, dependendo se as queries devolvem linhas ou não;



# EXEMPLOS DE UTILIZAÇÃO DE SUBQUERIES

## Utilizando uma subconsulta como uma tabela derivada

- ✳ É um conjunto de registos dentro de uma consulta que funciona como uma tabela;
- ✳ Ela toma o lugar da tabela na cláusula FROM;
- ✳ É otimizada com o resto da consulta;

```
SELECT Medicos.*, C.hora  
FROM Medicos, ( SELECT codm, hora  
                FROM Consultas  
                WHERE data = '06/06/13') C  
WHERE Médicos.codm = C.codm
```



# EXEMPLOS DE UTILIZAÇÃO DE SUBQUERIES

## Utilizando uma subconsulta como uma expressão

- ★ É executada uma vez para toda a instrução;

```
SELECT numemp, nome, (SELECT MIN(datapedido)
FROM pedidos WHERE rep = numemp)
FROM empregados;
```

## Utilizando uma subconsulta na cláusula Where e Having

- ★ Numa cláusula WHERE/ HAVING temos sempre uma condição e a subquery atua operando dentro dessa condição;

```
SELECT dep_no NDepartamento, COUNT(*) TotalEmpregados
FROM empregados
GROUP BY dep_no
HAVING COUNT(*) = (SELECT MAX(COUNT(*)) FROM empregados GROUP BY dep_no);
```