

# Fundamentos de Base de Dados

## SQL - SELECT

Pedro Nogueira Ramos

(Pedro.Ramos@iscte.pt)

ISTA / DCTI

# SQL

(Structured Query Language)

Linguagem declarativa para manipulação e definição de dados especificamente para o contexto do modelo relacional.

**DDL** (Data Definition Language): CREATE TABLE, CREATE INDEX, ALTER TABLE, Etc.

**DML** (Data Manipulation Language): **SELECT**, UPDATE, **INSERT**, **DELETE**, etc.

# Select

O comando **SELECT** é composto por seis cláusulas:

**SELECT** – selecciona **colunas**

**FROM** – indica sobre que **tabelas é efectuada a pesquisa**

**WHERE** – selecciona **linhas**

**GROUP BY** – **agrupa linhas** em grupos

**HAVING** – **selecciona grupos**

**ORDER BY** – **indica critério de ordenação** da pesquisa

Apenas as **duas primeiras são obrigatórias** e tem de ser respeitada a ordem indicada. A cláusula **HAVING** necessita da cláusula **GROUP BY**.

## SELECT / FROM

Na forma mais simples a **cláusula SELECT** selecciona colunas de uma tabela (todas as linhas).

(tabela Autor)

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

**SELECT Nome**

**FROM Autor**



Nome
Vargas Llosa
Margerite Yourcenar
Paul Auster
José Saramago
Rowling

## SELECT / FROM

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

**SELECT Nome, Nacionalidade**

**FROM Autor**



Nome	Nacionalidade
Vargas Llosa	Peru
Margerite Yourcenar	França
Paul Auster	USA
José Saramago	Portugal
Rowling	USA

## SELECT / FROM - \*

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

SELECT \* FROM Autor

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

 - todos os campos

## SELECT / FROM - DISTINCT

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

SELECT Nacionalidade FROM Autor

Nacionalidade
Peru
França
USA
Portugal
USA

SELECT **DISTINCT** Nacionalidade FROM Autor

Nacionalidade
Peru
França
USA
Portugal

## SELECT / FROM / ORDER BY

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

SELECT Nome, Nacionalidade FROM Autor

**ORDER BY** Nacionalidade

Nome	Nacionalidade
Margerite Yourcenar	França
Vargas Llosa	Peru
José Saramago	Portugal
Rowling	USA
Paul Auster	USA

SELECT Nome, Nacionalidade FROM Autor

**ORDER BY** Nacionalidade, Nome

Nome	Nacionalidade
Margerite Yourcenar	França
Vargas Llosa	Peru
José Saramago	Portugal
Paul Auster	USA
Rowling	USA



## SELECT / FROM / ORDER BY –ASC, DESC

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

SELECT Nome, Nacionalidade FROM Autor  
ORDER BY Nacionalidade **DESC**

SELECT Nome, Nacionalidade FROM Autor  
ORDER BY Nacionalidade **ASC**, Nome DESC

Nome	Nacionalidade
Paul Auster	USA
Rowling	USA
José Saramago	Portugal
Vargas Llosa	Peru
Margerite Yourcenar	França

Nome	Nacionalidade
Margerite Yourcenar	França
Vargas Llosa	Peru
José Saramago	Portugal
Rowling	USA
Paul Auster	USA

## SELECT / FROM - constantes

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

**SELECT “Nome:”, Nome FROM Autor      SELECT “olá”, 1+1, 1=2 FROM Autor**

Nome:	Nome
Nome:	Paul Auster
Nome:	Rowling
Nome:	José Saramago
Nome:	Vargas Llosa
Nome:	Margerite Yourcenar

olá	1+1	1=2
olá	2	False
olá	2	False
olá	2	Fase
olá	2	False
olá	2	False

## SELECT / FROM - AS

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

**SELECT** Nome as 'Nome do Autor', Nacionalidade FROM Autor

Nome do Autor	Nacionalidade
Paul Auster	USA
Rowling	USA
José Saramago	Portugal
Vargas Llosa	Peru
Margerite Yourcenar	França

## SELECT - Nota

É importante notar que qualquer **comando SELECT *devolve* uma tabela** (um conjunto de colunas e linhas).

Sempre que, no contexto da sintaxe da linguagem SQL for referida uma tabela, ela deve ser interpretada no sentido mais lato: uma tabela *original* (definida no esquema relacional) ou o resultado de um comando SELECT.

## WHERE

A cláusula WHERE selecciona as linhas pretendidas.

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

SELECT nome

FROM Autor

WHERE Nacionalidade='USA'

**Nome**

Paul Auster

Rowling

## WHERE

Na cláusula WHERE pode constar qualquer expressão lógica que possa ser avaliada linha a linha.

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	França
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

Exemplos:

```
SELECT Nome FROM Autor WHERE Nacionalidade='USA' OR Nacionalidade='Portugal'
```

```
SELECT Nome FROM Autor WHERE IDAutor <=10
```

```
SELECT IDAutor FROM Autor WHERE Nome LIKE '%Auster%'
```

```
SELECT Nome FROM Autor WHERE IDAutor <=10 and IDAautor >=2
```

## WHERE com NULL / NOT

Em Base de Dados, a ausência de valor é explicitamente registada com o valor NULL. Existem operadores especiais para lidar com o valor NULL.

IDAutor	Nome	Nacionalidade
1	Vargas Llosa	Peru
4	Margerite Yourcenar	NULL
23	Paul Auster	USA
40	José Saramago	Portugal
7	Rowling	USA

**Nome**

Margerite Yourcenar

:SELECT Nome FROM Autor WHERE Nacionalidade **IS NULL**

**Nome**

Paul Auster

José Saramago

Vargas Llosa

Rowling

SELECT Nome FROM Autor WHERE Nacionalidade **IS NOT NULL**

FROM, mais do que uma tabela

Quando são colocadas **duas tabelas** (ou mais) na **cláusula FROM**, é executado o **produto cartesiano**.

SELECT \*


FROM **Cliente, Localidade**

(tabela Cliente)

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

(tabela Localidade)

CodPostal	Localidade
1500	Lisboa
2100	Porto
3999	Évora



Número	Nome	Cliente. CodPostal	Localidade. CodPostal	Localidade
001	João	1500	1500	Lisboa
001	João	1500	2100	Porto
001	João	1500	3999	Évora
013	Ana	2100	1500	Lisboa
013	Ana	2100	2100	Porto
013	Ana	2100	3999	Évora
056	Luís	NULL	1500	Lisboa
056	Luís	NULL	2100	Porto
056	Luís	NULL	3999	Évora



## FROM, join

Para evitar o produto cartesiano, utiliza-se a cláusula **WHERE** para efectuar um join..

**SELECT \***

**FROM** Cliente, Localidade

**WHERE** Cliente.CodPostal =  
Localidade.CodPostal



Número	Nome	Cliente. CodPostal	Localidade. CodPostal	Localidade
001	João	1500	1500	Lisboa
013	Ana	2100	2100	Porto

(tabela Cliente)

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

(tabela Localidade)

CodPostal	Localidade
1500	Lisboa
2100	Porto
3999	Évora

E se o objectivo fosse listar para **TODOS** os clientes, a sua localidade, isto é:

Número	Nome	Cliente. CodPostal	Localidade. CodPostal	Localidade
001	João	1500	1500	Lisboa
013	Ana	2100	2100	Porto
056	Luís	NULL	NULL	NULL

Ver solução mais adiante (left join).

## SELECT / GROUP BY e HAVING (I)

Recorrendo apenas às cláusulas anteriores não é possível, por exemplo, efectuar certas operações estatísticas (somatórios, médias, etc.). Tal acontece porque as operações de agregação (que envolvem vários registos) não poderem ser calculadas linha a linha. Por exemplo, o comando para listar os códigos postais associados a mais do que dois clientes não pode ser efectuado tal como de seguida se apresenta:

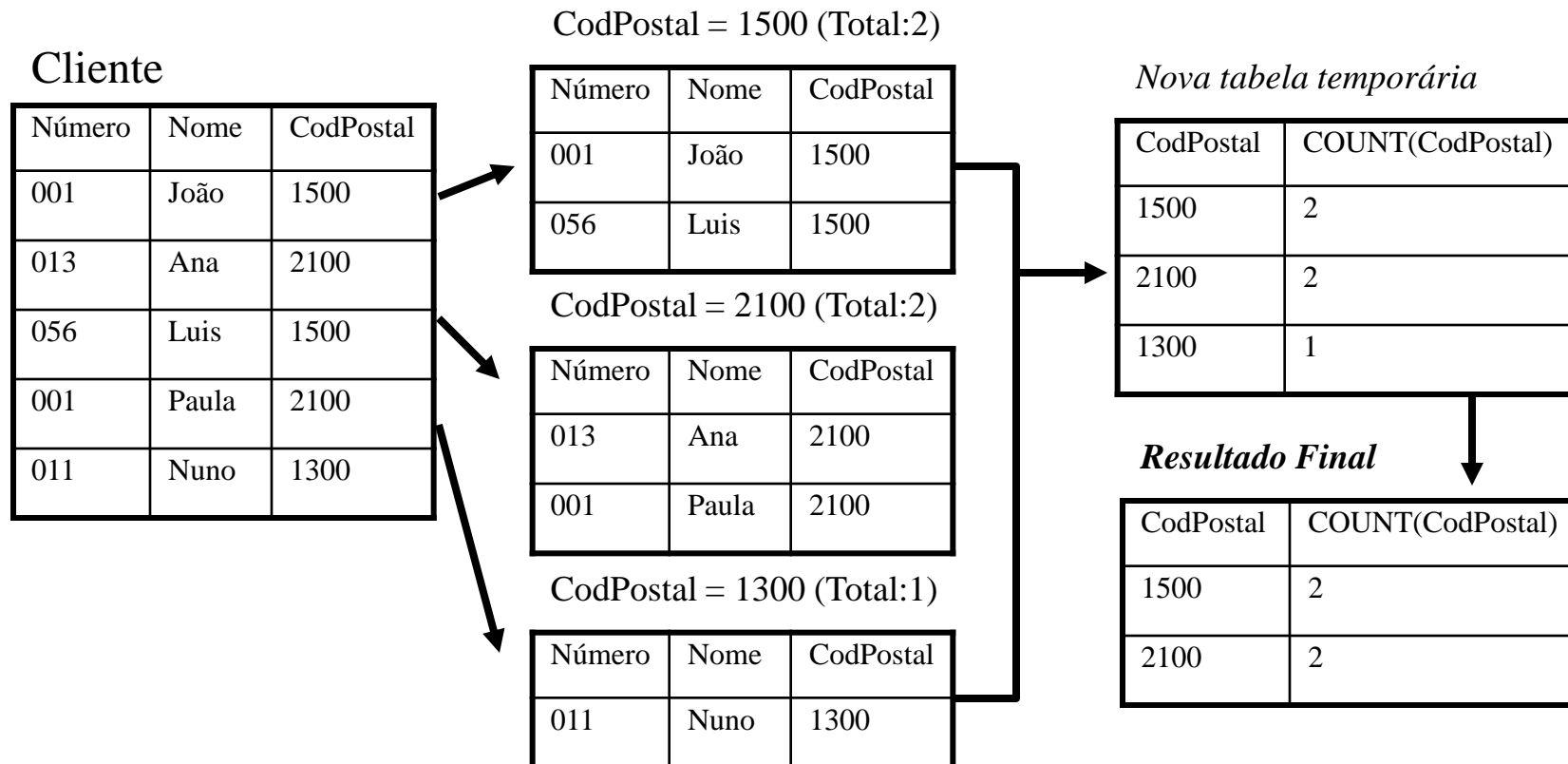
```
SELECT CodPostal FROM Cliente  
WHERE COUNT(CodPostal) > 2
```

- O comando **é incorrecto(!)** porque a cláusula WHERE é testada linha a linha e, **numa linha não é possível obter o total** de códigos postais

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luis	1500
001	Paula	2100
011	Nuno	1300

## SELECT / GROUP BY e HAVING (II)

Procedimento para obter os códigos postais associados a mais de 2 clientes.



## SELECT / GROUP BY e HAVING (III)

```
SELECT CodPostal, COUNT(CodPostal) FROM Cliente  
GROUP BY CodPostal HAVING COUNT(CodPostal) > 1;
```

Cliente

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luis	1500
001	Paula	2100
011	Nuno	1300

CodPostal = 1500 (Total:2)

Número	Nome	CodPostal
001	João	1500
056	Luis	1500

CodPostal = 2100 (Total:2)

Número	Nome	CodPostal
013	Ana	2100
001	Paula	2100

CodPostal = 1300 (Total:1)

Número	Nome	CodPostal
011	Nuno	1300

Nova tabela temporária

CodPostal	COUNT(CodPostal)
1500	2
2100	2
1300	1

Resultado Final

CodPostal	COUNT(CodPostal)
1500	2
2100	2

## SELECT / GROUP BY e HAVING (IV)

As cláusulas **GROUP BY** e **HAVING** permitem manipular **valores agregados**. A cláusula **GROUP BY** permite a **definição de grupos**. A **cláusula HAVING** é equivalente à **cláusula WHERE** só que o seu **argumento são expressões lógicas** relativas aos agrupamentos criados pela cláusula **GROUP BY**.

```
SELECT CodPostal  
FROM Cliente  
GROUP BY CodPostal  
HAVING COUNT(CodPostal) > 2;
```

A cláusula **GROUP BY** **agrupa os clientes** por código postal e a cláusula **HAVING** **selecciona os grupos** cujo número de elementos é **superior a dois**.

## SELECT / Funções de Agregação (I)

Sempre que existe uma função de agregação na cláusula SELECT, todos os restantes atributos da cláusula têm que estar incluídos na cláusula GROUP BY. O comando que de seguida se apresenta retorna o maior bilhete de identidade existente:

```
SELECT MAX(Bi) FROM Cliente;
```

Caso pretendêssemos visualizar o nome desse cliente, não poderíamos simplesmente acrescentar o atributo nome à cláusula SELECT. Pela regra anteriormente referida, teríamos que considerar a cláusula GROUP BY :

```
SELECT MAX(Bi), NOME FROM Cliente GROUP BY Nome;
```

No entanto, o resultado do comando seria a listagem de todos os nomes com a indicação do BI associado a cada nome. Mais adiante (Subqueries) apresenta-se a resolução desta interrogação.

## SELECT / Funções de Agregação (II)

Para além das funções COUNT e MAX, existem outras, tais como SUM, AVG e MIN.

Apenas a função COUNT **não necessita de argumento**: o primeiro comando retorna o total de registos de clientes enquanto o segundo devolve o total de clientes com o código postal conhecido:

```
SELECT COUNT(*) FROM Cliente;
```

```
SELECT COUNT(CodPostal) FROM Cliente;
```

Note-se que o segundo comando é **equivalente ao seguinte**:

```
SELECT COUNT(*) FROM Cliente WHERE CodPostal IS NOT NULL;
```

## SELECT - Subquerys

Uma *subquery* é um comando **SELECT** dentro de um comando **SELECT**. Muitas interrogações apenas podem ser resolvidas através de *subquerys*. Um comando **SELECT** normalmente *liga-se* a outro através da cláusula **WHERE**.

Os operadores **IN** e **EXISTS** são normalmente utilizadas nas *subquerys*.

Operador **IN** devolve verdade quando um elemento *pertence* a um conjunto.

Operador **EXISTS** devolve verdade caso a *subquery* *retorne pelo menos* uma linha.



## SELECT - Subquerys - IN

```
SELECT DISTINCT(Nome) FROM Cliente as Cliente1
```

```
WHERE NOME IN
```

```
(SELECT NOME FROM Cliente as Cliente2 WHERE  
Cliente1.bi <> Cliente2.bi);
```

Cliente1

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

Cliente2

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

João IN

Ana IN

Necessário considerar duas  
cópias virtuais da mesma  
para evitar ambiguidades.

Garantir que não encontra  
“ele próprio”.

**Detecta nomes duplicados.**

**Devolve:**

**conjunto vazio**

## SELECT - Subquerys - IN

```
SELECT DISTINCT(Nome) FROM Cliente as Cliente1  
WHERE NOME IN  
(SELECT NOME FROM Cliente as Cliente2 WHERE  
Cliente1.bi <> Cliente2.bi);
```

Necessário considerar duas  
cópias virtuais da mesma  
para evitar ambiguidades.

Cliente1

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	João	NULL

Cliente2

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	João	NULL

João IN

Ana IN

**Detecta nomes duplicados.**

**Devolve:**

**João**

## SELECT - Subqueries - EXISTS

```
SELECT DISTINCT(Nome) FROM Cliente as Cliente1  
WHERE EXISTS  
(SELECT * FROM Cliente as Cliente2 WHERE  
Cliente1.bi <> Cliente2.bi AND  
Cliente1.Nome = Cliente2.Nome);
```

Necessário considerar duas  
cópias virtuais da mesma  
para evitar ambiguidades.

Cliente1

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

Cliente2

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

João EXISTS João?

Ana EXISTS Ana?

**Detecta nomes duplicados.**

## SELECT - Subquerys – ALL, ANY

Os operadores ALL (todos) e ANY (pelo menos um) também são frequentes nas *subquerys*. O comando para retornar o maior bilhete de identidade (e nome associado) existente é:

```
SELECT Nome, BI FROM Cliente  
WHERE BI >= ALL  
      (SELECT BI FROM Cliente);
```

BI	Nome
001	João
013	Ana
056	Luís

BI	Nome
001	João
013	Ana
056	Luís

001 > todos

013 > todos.



**Devolve 056 Luís.**

Caso pretendêssemos um bilhete de identidade que não fosse o menor:

```
SELECT Nome, BI FROM Cliente  
WHERE BI > ANY  
      (SELECT BI FROM Cliente);
```

**Devolve 056 e 013**

## SELECT – Subquerys (II)

As *subquerys* também **podem ser colocadas** nas cláusula **SELECT e FROM.**

O seguinte exemplo devolve, para **cada cliente**, o **total de facturas** associadas:

```
SELECT Nome, (SELECT COUNT(*) FROM Factura  
              WHERE Factura.BI = Cliente.BI)  
FROM Cliente;
```

O seguinte exemplo **devolve o nome de todos** os clientes:

```
SELECT DISTINCT Nome  
FROM (Select * From Cliente) as Cliente;
```

## UNION

A **união de comandos SELECT** é efectuada através do operador UNION.

O seguinte comando devolve os nomes dos clientes e fornecedores :

```
SELECT Nome FROM Cliente
```

**UNION**

```
SELECT Nome FROM Fornecedor;
```

Caso **não pretendamos eliminar nomes duplicados** o comando será:

```
SELECT Nome FROM Cliente
```

**UNION ALL**

```
SELECT Nome FROM Fornecedor;
```

## UNION – LEFT JOIN

Listar para **TODOS** os clientes, a sua

(tabela Cliente)

Número	Nome	CodPostal
001	João	1500
013	Ana	2100
056	Luís	NULL

(tabela Localidade) localidade, isto é:

CodPostal	Localidade
1500	Lisboa
2100	Porto
3999	Évora

Número	Nome	Cliente. CodPostal	Localidade. CodPostal	Localidade
<b>001</b>	<b>João</b>	<b>1500</b>	<b>1500</b>	<b>Lisboa</b>
<b>013</b>	<b>Ana</b>	<b>2100</b>	<b>2100</b>	<b>Porto</b>
<b>056</b>	<b>Luís</b>	<b>NULL</b>	<b>NULL</b>	<b>NULL</b>

```
SELECT * FROM Cliente, Localidade  
WHERE Cliente.CodPostal = Localidade.CodPostal
```

### UNION

```
(SELECT Numero, Nome, NULL, NULL, NULL From Cliente  
WHERE CodPostal NOT IN  
(SELECT CodPostal FROM LOCALIDADE))
```