UNIVERSIDADE da MADEIRA

ESCOLA SUPERIOR DE TECNOLOGIAS E GESTÃO

Ano Letivo 2019/2020

Curso Técnico Superior Profissional em: Tecnologias e Programação de Sistemas de Informação

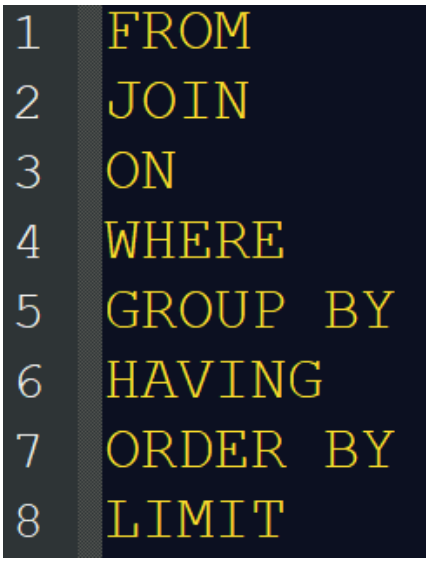
1 º Ano/2 º Semestre

Unidade Curricular: Sistemas Gestores de Bases de Dados I

Docente: Magno Andrade **Época:** Normal

SQL - MANIPULAÇÃO DE DADOS – PARTE 3

Ordem de execução de alguns comandos e cláusulas utilizados no Comando SELECT.



COMANDO SELECT – CONTINUAÇÃO

O comando tem a seguinte **sintaxe**:

Estrutura básica de expressão de SQL:

SELECT (projeção da álgebra relacional)
FROM (lista das relações)
WHERE (predicado da seleção)

Ou seja:

SELECT $a_1, a_2, a_3, \ldots a_n$ (campos pretendidos)
FROM $r_1, r_2, r_3, \ldots r_n$ (tabelas dos campos pretendidos)
WHERE p (condições a serem satisfeitas)

Cofinanciado por:

MADEIRA 14-20 REGIÃO AUTÓNOMA DA MADEIRA PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Sendo que a cláusula **WHERE** é opcional.

```
SELECT campo1, campo2, ... campon
FROM tabela 1 , tabela 2, .... tabela n
[WHERE Condição ]
[GROUP BY .....]
[HAVING ……]
[ORDER BY …….];
```

Os parêntesis rectos indicam que essa componente é opcional.

✓ JUNTAR VÁRIAS TABELAS

O Modelo Relacional estabelece claramente as regras para a divisão da informação entre tabelas, de forma a evitar a duplicação da informação.

A dispersão da informação por diferentes tabelas é facilmente manipulável através da linguagem SQL, uma vez que a ligação entre estas será realizada através das chaves estrangeiras.

A junção entre tabelas faz-se colocando na cláusula FROM o conjunto de tabelas que se pretende juntar.

```
SELECT campo1, campo2, … campoN
FROM tabela1, tabela2, … tabelaN;
```

Considerando as seguintes relações/tabelas (**pessoa** e **postal**):

pessoa_id	nome	idade	telefone	cod_postal
71	António Dias	43	789654	1000
54	Célia Morais	36	123456	1000
12	Isabel Silva	28		2040
49	José António	17	333555	9000

postal_id	local
1000	LISBOA
9000	FUNCHAL
2040	RIO MAIOR
4000	PORTO

Produto Cartesiano de Tabelas

```
SELECT *
FROM pessoa, postal;
```

O resultado obtido é o produto cartesiano de dois conjuntos de elementos.

O produto cartesiano entre as duas tabelas, associa a cada linha da tabela **pessoa** o conjunto das linhas da tabela **postal**. Dado que na tabela **pessoa** existem 4 registos e na tabela **postal** existem 4 registos, o resultado do produto cartesiano será 4*4 = 16 registos.

Cofinanciado por:

O resultado do produto cartesiano será:

pessoa_id	nome	idade	telefone	cod_postal	postal_id	local
71	António Dias	43	789654	1000	1000	LISBOA
71	António Dias	43	789654	1000	9000	FUNCHAL
71	António Dias	43	789654	1000	2040	RIO MAIOR
71	António Dias	43	789654	1000	4000	PORTO
54	Célia Morais	36	123456	1000	1000	LISBOA
54	Célia Morais	36	123456	1000	9000	FUNCHAL
54	Célia Morais	36	123456	1000	2040	RIO MAIOR
54	Célia Morais	36	123456	1000	4000	PORTO
12	Isabel Silva	28		2040	1000	LISBOA
12	Isabel Silva	28		2040	9000	FUNCHAL
12	Isabel Silva	28		2040	2040	RIO MAIOR
12	Isabel Silva	28		2040	4000	PORTO
49	José António	17	333555	9000	1000	LISBOA
49	José António	17	333555	9000	9000	FUNCHAL
49	José António	17	333555	9000	2040	RIO MAIOR
49	José António	17	333555	9000	4000	PORTO

O pretendido seria a junção entre as tabelas **pessoa** e **postal**. Para tal é necessário, na cláusula WHERE, indiciar as chaves estrangeiras de ligação.

```
SELECT *
FROM pessoa, postal
WHERE cod_postal = postal_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	nome	idade	telefone	cod_postal	postal_id	local
71	António Dias	43	789654	1000	1000	LISBOA
54	Célia Morais	36	123456	1000	1000	LISBOA
12	Isabel Silva	28		2040	2040	RIO MAIOR

49	José António	17	333555	9000	9000	FUNCHAL

✓ EQUI-JOIN

Quando todas as colunas das tabelas são apresentadas e a ligação entre as tabelas é feita através de uma igualdade, originando assim duas colunas de conteúdos exactamente iguais.

```
SELECT *
FROM pessoa, postal
WHERE cod_postal = postal_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	nome	idade	telefone	cod_postal	postal_id	local
71	António Dias	43	789654	1000	1000	LISBOA
54	Célia Morais	36	123456	1000	1000	LISBOA
12	Isabel Silva	28		2040	2040	RIO MAIOR
49	José António	17	333555	9000	9000	FUNCHAL

✓ NATURAL-JOIN

Quando todas as colunas envolvidas na ligação entre tabelas são apresentadas sem repetição de colunas.

```
SELECT pessoa.*, postal.local
FROM pessoa, postal
WHERE cod_postal = postal_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	nome	idade	telefone	cod_postal	local
71	António Dias	43	789654	1000	LISBOA
54	Célia Morais	36	123456	1000	LISBOA
12	Isabel Silva	28		2040	RIO MAIOR
49	José António	17	333555	9000	FUNCHAL

Estes dois tipos de ligação entre tabelas fazem parte de um tipo de ligação mais geral denominado de INNER JOIN.

Num INNER JOIN, apenas são apresentados os registos em que exista ligação entre as tabelas.

Cofinanciado por:

MADEIRA 14-20 · REGIÃO AUTÓNOMA DA MADEIRA · PORTUGAL 2020 · UNIÃO EUROPEIA Fundo Social Europeu

✓ INNER JOIN

Embora existem diversos tipos de ligação entre tabelas (JOIN), este é o tipo mais comum e utilizado.

Pretendemos seleccionar o **nome** e **morada completa** (**postal_id** e **localidade**) de todas as pessoas da tabela **pessoa**.

```
SELECT nome, cod_postal, local
FROM pessoa, postal
WHERE cod_postal = postal_id;
```

Após a execução do comando, será obtido o seguinte resultado:

nome	cod_postal	local
António Dias	1000	LISBOA
Célia Morais	1000	LISBOA
Isabel Silva	2040	RIO MAIOR
José António	9000	FUNCHAL

Este mesmo JOIN pode ser efectuado utilizando a cláusula existente para o efeito, o INNER JOIN:

```
SELECT nome, cod_postal, local
FROM pessoa
        INNER JOIN postal ON postal.postal_id = pessoa.cod_postal
ORDER BY pessoa.cod_postal;
```

Após a execução do comando, será obtido o seguinte resultado:

nome	cod_postal	local
António Dias	1000	LISBOA
Célia Morais	1000	LISBOA
Isabel Silva	2040	RIO MAIOR
José António	9000	FUNCHAL

✓ OUTER JOIN

Este conceito é utilizado para obter numa ligação a totalidade das linhas de uma tabela, ainda que não exista o correspondente valor na outra tabela a que está ligada pela junção.

Cofinanciado por:

MADEIRA 14-20 · REGIÃO AUTÓNOMA DA MADEIRA · PORTUGAL 2020 · UNIÃO EUROPEIA Fundo Social Europeu

Queremos seleccionar o **nome** e **morada completa** (**postal_id** e **localidade**) de todas as pessoas da tabela **pessoa**, assim como todos os códigos postais existentes na tabela **postal**.

```
SELECT nome, cod_postal, postal_id, local
FROM postal
        LEFT JOIN pessoa ON postal.postal_id = pessoa.cod_postal
```

Após a execução do comando, será obtido o seguinte resultado:

nome	cod_postal	postal_id	local
António Dias	1000	1000	LISBOA
Célia Morais	1000	1000	LISBOA
		1100	LISBOA
José António	9000	9000	FUNCHAL
Isabel Silva	2040	2040	RIO MAIOR
		2300	TOMAR
		4000	PORTO

Todas as linhas da tabela **postal** são apresentadas. Se existir correspondente na coluna **cod_postal** da tabela **pessoa**, são mostrados os dados, senão as entradas da tabela **pessoa** são preenchidas a NULL.

Quando este OUTER JOIN é realizado à direita, são considerados todos os registos da tabela da direita e apenas os registos correspondentes na tabela da esquerda.

Tendo em conta as seguintes relações (**pessoa** e **comissao**):

pessoa_id	nome	idade
71	António Dias	43
12	Isabel Silva	28
49	José António	17
85	João Silva	49

pessoa_id	mensagem_id	valor
49	10	1250
49	70	750
71	12	100
71	15	200

Pretende-se seleccionar todas as pessoas da tabela **pessoa**, assim como os correspondentes valores de **comissão**.

```
SELECT nome, valor
FROM comissao
        RIGHT JOIN pessoa ON comissao.pessoa_id = pessoa.pessoa_id
```

Cofinanciado por:

MADEIRA 14-20 REGIÃO AUTÓNOMA DA MADEIRA PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

nome	valor
António Dias	100
António Dias	200
Isabel Silva	
José António	1250
José António	750
João Silva	

✓ SELF JOIN

O Self JOIN é uma variante do INNER JOIN, em que se comparam duas colunas da mesma tabela.

```
SELECT p1.postal_id, p2.local
FROM postal.p1, postal.p2
WHERE p1.postal_id = p2.postal_id
        AND
        p2.local <> “LISBOA”;
```

✓ UNION

Uma união não é propriamente uma ligação entre tabelas. A UNION permite juntar o conteúdo de múltiplos comandos SELECT.

Queremos juntar o código e a descrição da tabela mensagens aos códigos postais.

```
SELECT mensagem_id, mensagem FROM mensagem
UNION
SELECT postal_id, local FROM postal;
```

Após a execução do comando, será obtido o seguinte resultado:

mensagem_id	mensagem
10	Comissão de vendas
30	Vendas extra
40	Refeições
1000	Lisboa
2040	Rio Maior
4000	Porto

Cofinanciado por:

MADEIRA 14-20 REGIÃO AUTÓNOMA DA MADEIRA PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Usando este comando de UNION, o número de campos a seleccionar em cada um dos comandos SELECT tem de ser igual. O nome dos campos não é relevante, mas o tipo de dados que pode ser agrupado depende de sistema para sistema.

Pretendemos juntar as tabelas **postal** (cujos locais contenham a string "OR") e **mensagem** (cujas id sejam inferiores a 30), ordenando o resultado por mensagem.

```
SELECT mensagem_id, mensagem FROM mensagem
WHERE mensagem_id < 30
UNION
SELECT postal_id, local FROM postal
WHERE local LIKE "%OR%"
ORDER BY mensagem;
```

✓ INTERSECT

Este operador retorna apenas as linhas comuns de duas ou mais consultas.

Tem a mesma funcionalidade que o INNER JOIN mas ao contrário deste não retorna duplicados e pode retornar ***NULLs***.

No entanto, o MySQL não possui este operador num comando específico. Para obtermos a mesma funcionalidade usamos diferentes cláusulas para "emular" este operador.

Sintaxe:

```
SELECT DISTINCT id FROM tabela1
INNER JOIN tabela2
        ON predicado do join;
```

✓ MINUS

Este operador compara dois resultados de consultas e retorna apenas as linhas do conjunto de resultados da primeira consulta e que não aparecem no conjunto de linhas da segunda consulta.

No entanto, o MySQL não possui este operador num comando específico. Para obtermos a mesma funcionalidade usamos diferentes cláusulas para "emular" este operador.

Sintaxe:

```
SELECT campo1, campo2 FROM tabela1
LEFT JOIN tabela2
        ON predicado do join
WHERE table2.campo IS NULL;
```

✓ RESUMO DAS JUNÇÕES

JOIN	Descrição
Produto Cartesiano	Juntar cada linha da tabela T1 com todas as linhas de T2.

INNER JOIN	Junção tradicional, em que apenas são apresentadas as linhas comuns às duas tabelas.
OUTER JOIN	Extensão do INNER JOIN ao proporcionar todos os registos de uma das tabelas, mesmo que sobre estes não exista qualquer ligação.
UNION	Todos os registos de qualquer das pesquisas (sem duplicados).
UNION ALL	Todos os registos de qualquer das pesquisas (com duplicados).
INTERSECT	Todos os registos comuns a ambas as pesquisas.
MINUS	Todos os registos da primeira pesquisa que não aparecem na segunda.

✓ AGRUPAR A INFORMAÇÃO

As funções de agregação são uma ferramenta útil quando usada para obter informação resumida sobre o resultado de um comando SELECT. No entanto, estas funções podem ser particularmente úteis no tratamento de informação de forma agrupada, não como um todo, mas em grupos mais pequenos.

Pretendemos mostrar as comissões e respectivos valores, ordenando o resultado por **pessoa_id** da tabela **comissao**.

```
SELECT pessoa_id, valor
FROM comissao
ORDER BY pessoa_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	valor
14	2600
14	400
14	3750
14	10500
25	370
25	2400
25	100

pessoa_id	valor
37	120
37	5500
37	14230
37	20
40	20
42	150
42	20

Pretendemos saber o total de valores de comissões:

```
SELECT SUM(valor) AS 'Total'
FROM comissao;
```

Após a execução do comando, será obtido o seguinte resultado:

Total
40180

No entanto, se o objectivo fosse obter a soma das comissões por cada **pessoa_id**, o resultado apresentado não seria o pretendido. Para resolver essas questões é necessário, antes de aplicar as funções de agregação, possuir a informação agrupada.

```
SELECT campo1, campo2, ... campon
FROM tabela 1 , tabela 2, .... tabela n
[WHERE Condição ]
[GROUP BY .....]
[HAVING ……]
[ORDER BY  .....]
```

✓ CLÁUSULA GROUP BY

Esta cláusula divide o resultado de um SELECT em grupos de resultados que irão ser tratados com as funções de agregação.

- **A cláusula GROUP BY é utilizada para agrupar informação.**
- **Os registos são processados em grupos de características semelhantes.**

- **As funções de agregação podem ser utilizadas para obter informação sobre cada um dos grupos.**

Pretende-se saber, para cada **pessoa_id**, o total de valores de comissões.

```
SELECT pessoa_id, SUM(valor) AS 'Total'
FROM comissao
GROUP BY pessoa_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	total
14	17250
25	2870
37	19870
40	20
42	170

Pretende-se saber, para cada **pessoa_id**, o maior valor de comissão.

```
SELECT pessoa_id, MAX(valor) AS 'Maior'
FROM comissao
GROUP BY pessoa_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	Maior
14	10500
25	2400
37	14230
40	20
42	150

Poderão ser utilizadas todas as outras funções de agregação, sobre os dados agrupados, sendo que também será possível efectuar ordenação sobre os dados agrupados e calculados.

Pretende-se saber para cada **pessoa_id**, o maior valor de comissão, efectuando a ordenação descendente por maior valor de comissão:

```
SELECT pessoa_id, MAX(valor) AS 'Maior'
FROM comissao
GROUP BY pessoa_id
ORDER BY MAX(valor) DESC;
```

Após a execução do comando, será obtido o seguinte resultado:

Maior	pessoa_id
14230	37
10500	14
2400	25
150	42
20	40

✓ CLÁUSULA HAVING

Esta cláusula serve para fazer restrições ao nível dos grupos que são processados. Esta cláusula actua sobre o resultado dos grupos, que resultam da função de agrupamento ORDER BY.

Pretende-se saber, para cada **pessoa_id**, o total de valores de comissões. No entanto, só são relevantes os totais superiores a 1000.

```
SELECT pessoa_id, SUM(valor) AS 'Total'
FROM comissao
GROUP BY pessoa_id
HAVING SUM(valor) > 1000;
```

Após a execução do comando, será obtido o seguinte resultado:

comissao_id	Total
14	17250
25	2870
37	19870

Pretende-se saber, para cada **pessoa_id**, o maior valor de comissão. No entanto, só são relevantes os valores inferiores a 3000.

```
SELECT pessoa_id, MAX(valor) AS 'Maior'
FROM comissao
GROUP BY pessoa_id
HAVING MAX(valor) < 3000;
```

Cofinanciado por:

MADEIRA 14-20 · REGIÃO AUTÓNOMA DA MADEIRA · PORTUGAL 2020 · UNIÃO EUROPEIA Fundo Social Europeu

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	Maior
25	2400
40	20
42	150

✓ WHERE vs HAVING

Utiliza-se a cláusula WHERE (aplicada a linhas individualmente) sempre que se pretende restringir os registos a considerar na selecção. A cláusula HAVING (não aplicada a linhas individualmente) serve para restringir os grupos que foram formados depois de aplicada a cláusula GROUP BY.

Pretende-se saber qual o total de comissões para cada **pessoa_id**, considerando apenas aquelas cujo valor seja superior a 1000.

```
SELECT pessoa_id, SUM(valor) AS 'Total'
FROM comissao
WHERE valor > 1000
GROUP BY pessoa_id;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	Total	
14	16850	(17250-400)
25	2400	(2870-370-100)
37	19730	(19870-120-20)

Pretende-se saber o total de comissões para cada **pessoa_id**, considerando apenas aquelas cujo valor total seja superior a 1000.

```
SELECT pessoa_id, SUM(valor) AS 'Total'
FROM comissao
GROUP BY pessoa_id
HAVING SUM(valor) > 1000;
```

Após a execução do comando, será obtido o seguinte resultado:

pessoa_id	Total
14	17250

Cofinanciado por:

MADEIRA 14-20 REGIÃO AUTÓNOMA DA MADEIRA PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

25	2870
37	19870

Nota Final

Se um comando SELECT possuir a cláusula GROUP BY, todas as colunas seleccionadas (no SELECT) têm que estar presentes na cláusula GROUP BY.

Cofinanciado por: