LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

Disciplina de Gestão de Dados e de Bases de Dados

Ano Letivo 2020/2021

Linguagem SQL

SQL

Caraterísticas e Componentes

- **SQL na Manipulação de Dados**
- **SQL na Definição da Base de Dados**
- **SQL no Controlo da Base de Dados**
- **SQL na Gestão de Transações**

História

- 1970: Codd define o Modelo Relacional
- 1974: IBM desenvolve o projeto SYSTEM/R com a linguagem SEQUEL
- 1979: É lançado o primeiro SGBD comercial (ORACLE)
- 1981: É lançado o SGBD INGRES
- 1983: IBM anuncia o DB2
- 1986, 1987: É ratificada a norma SQL que fica conhecida como SQL-86 (ANSI X3.135-1986 e ISO 9075:1987)
- 1989: É ratificada a norma SQL-89 quer pela ANSI quer pela ISO
- 1992: É ratificada a norma: SQL2
- 1999: É ratificada a norma SQL1999, anteriormente conhecida como SQL3
- SQL:2003
- 2006: SQL:2006, define a forma como o SQL pode ser usado em combinação com o XML (ANSI/ISO/IEC 9075-14:2006)

História (cont.)

- 2008: SQL:2008
- 2011: SQL:2011, introduz suporte temporal
- 2016: SQL:2016, introduz JavaScript Object Notation or JSON (entre outras novas funcionalidades)
- 2019: SQL:2019, Adiciona a Part 15, multidimensional arrays (MDarray type and operators)

Structured Query Language, o que é ?

- ✓ **SQL é uma linguagem normalizada para definição, acesso, manipulação e controlo de Bases de Dados Relacionais**
- ✓ **Na maioria dos SGBDR, esta linguagem pode ser utilizada:**
  - **interativamente**
  - **embebida em linguagens de programação**

Caraterísticas

- **Linguagem não procedimental em que se especifica O QUÊ e não COMO**

  Existe uma clara abstração perante a estrutura física dos dados, isto é, não é necessário especificar caminhos de acesso nem algoritmos de pesquisa física

- **Operações sobre estruturas lógicas**

  As operações efetuam-se sobre conjuntos de dados (tabelas), não sendo necessário (nem possível) manipular linha-a-linha

Componentes

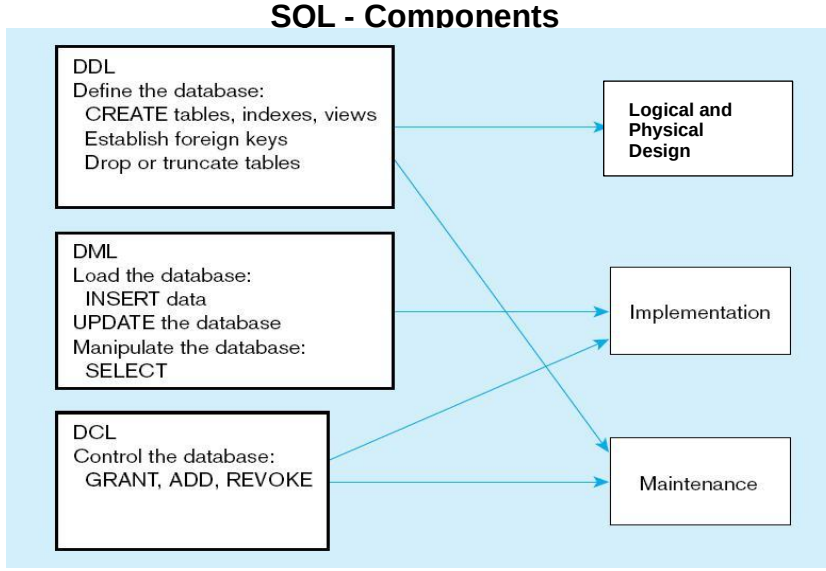
DDL (Data Definition Language)

DML (Data Manipulation Language)

DCL (Data Control Language)

TCL (Transaction Control Language)

SQL - Components



SQL

Manipulação de Dados

SELECT	**Acesso aos dados da B.D.**
INSERT **UPDATE** **DELETE**	**Manipulação dos dados da B.D.**

Esquema Relacional

Empregado (cod-emp**,** nome_emp, data_admissão, *cod_cat*, *cod_dept*, *cod_emp_chefe*)

Departamento (cod-dept, nome_dept, localização)

Categoria (cod-cat, designação, salario_base)

Base de Dados Relacional

Categoria

cod_cat	designação	salario_base
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160
...	...	...

Departamento

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
...	...	...

Empregado

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1
...	...	...	...	...	...
20	Manuel Matos	7-Feb-90	3	2	2
...	...	...	...	...	...

Clausula SELECT e FROM

```
SELECT    [ DISTINCT ] coluna, ... | *
FROM      tabela
```

O símbolo ***** é utilizado quando se pretende selecionar todos os atributos da tabela especificada na clausula FROM

DISTINCT é aplicado a todas as colunas especificadas na clausula SELECT e elimina as repetições existentes

Projeção

Empregado

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1
...	...	...	...	...	...
20	Manuel Matos	7-Feb-90	3	2	2
...	...	...	...	...	...

Clausulas
Select
From

```
SELECT   cod_emp, nome_emp
FROM     empregado
```


Restrição

Categoria

cod_cat	designação	salario_base
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160
...	...	...

Clausula Where

```
SELECT   *
FROM     categoria
WHERE    salario_base > 200
```


Junção

Empregado

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1
...	...	...	...	...	...
20	Manuel Matos	7-Feb-90	3	2	2
...	...	...	...	...	...

A partir do produto cartesiano seleciona-se somente as linhas que satisfazem a condição

EMPREGADO.COD_DEPT= DEPTARTAMENTO.COD_DEPT

Departamento

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
...	...	...

Junção

```
SELECT   nome_emp, empregado.cod_dept, nome_dept
FROM     empregado, departamento
WHERE    empregado.cod_dept = departamento.cod_dept
```

Caso o nome de uma coluna seja igual em várias tabelas então a REGRA é

Nome_Tabela.Nome_Coluna

em qualquer local da cláusula SELECT

Junção

OU

```
SELECT   nome_emp, empregado.cod_dept, nome_dept
FROM     empregado [INNER] JOIN  departamento
ON       empregado.cod_dept = departamento.cod_dept
```

OU

```
SELECT   nome_emp, cod_dept, nome_dept
FROM     empregado NATURAL JOIN  departamento
```


Projeção, Restrição e Junção

Qual o nome dos empregados pertencentes ao departamento de Vendas

```
SELECT    nome_emp, nome_dept
FROM      empregado, departamento
WHERE     empregado.cod_dept = departamento.cod_dept
          AND
          nome_dept = 'Vendas'
```

Projeção

Restrição

Junção

Aliases de Tabelas

```
SELECT   cod_emp, D.cod_dept, nome_dept
FROM     empregado E, departamento D
WHERE    E.cod_dept = D.cod_dept
```

Particularmente útil quando se pretende usar a mesma tabela com significados diferentes

SELF JOIN

Pretende-se o nome de cada empregado e o nome do respetivo chefe

```
SELECT  E.nome_emp, CH.nome_emp
FROM    empregado E, empregado CH
WHERE   E.cod_emp_chefe = CH.cod_emp
```


Junções Múltiplas

Categoria

cod_cat	designação	salario_base
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160
...	...	...

Departamento

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
...	...	...

Empregado

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1
...	...	...	...	...	...
20	Manuel Matos	7-Feb-90	3	2	2
...	...	...	...	...	...

Junções Múltiplas

Para cada categoria listar o nome dos empregados, salário_base e repetivo departamento

```
SELECT   categoria.cod_cat, nome_emp, nome_dept, salario_base
FROM     empregado, departamento, categoria
WHERE    empregado.cod_dept = departamento.cod_dept
         AND
         empregado.cod_cat = categoria.cod_cat
```


Junção "Outer" (Outer Join)

Empregado

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1

?

Quais os departamentos e respetivos empregados.

Nesta listagem deverão aparecer todos os departamentos, mesmo os que não têm empregados.

Departamento

cod_dept	nome_dept	localização
...	...	...
6	Marketing	Lisboa

Junção Outer à Direita (Right Outer Join)

```
SELECT   nome_emp, departamento.cod_dept, nome_dept
FROM     empregado right outer join  departamento
on       empregado.cod_dept  = departamento.cod_dept
```

cod_emp	nome_emp	cod_dept	nome_dept
1	António Abreu	1	Contabilidade
2	Bernardo Bento	2	Vendas
3	Carlos Castro	3	Investigação
		6	Marketing

União

Suponha que tem as seguintes tabelas:

CLIENTE (nome, morada)

FORNECEDOR (nome, morada)

Pretende uma listagem com os nomes e moradas quer dos clientes, quer dos fornecedores

```
SELECT   nome, morada
FROM     cliente
UNION
SELECT   nome, morada
FROM     fornecedor
```


Interseção

Suponha que com as tabelas anteriores

Pretende uma listagem com os nomes e moradas dos clientes que também são fornecedores

```
SELECT   nome, morada
FROM     cliente
INTERSECT
SELECT   nome, morada
FROM     fornecedor
```


Diferença

Suponha que com as tabelas anteriores

Pretende uma listagem com os nomes e moradas dos clientes que não são fornecedores

```
SELECT   nome, morada
FROM     cliente
EXCEPT
SELECT   nome, morada
FROM     fornecedor
```

Em Oracle tem de se utilizar **MINUS** em vez de **EXCEPT**

Clausula WHERE

```
SELECT    [ DISTINCT ] coluna, ...| *
FROM      tabela, [tabela,....]
WHERE     condição-de-pesquisa
```

Uma condição-de-pesquisa é basicamente uma coleção de predicados, combinados através dos operadores booleanos AND, OR, NOT e parêntesis.

Predicados

Um predicado pode ser:

- Um predicado de comparação (WHERE NOME_EMP = ´Manuel Silva´)
- Um predicado de BETWEEN (WHERE COD_EMP **BETWEEN** 1 AND 5)
- Um predicado de LIKE (WHERE NOME_EMP **LIKE** ' M%' |' Manu_l'
- Um teste de valor nulo (WHERE COMISSÃO **IS** NULL)
- Um predicado de IN (WHERE COD_CAT **IN** (1,2))

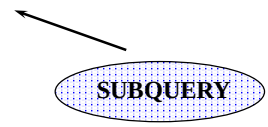
Predicados

- **Os predicados podem ser utilizados num contexto estático, sendo avaliados com base em valores constantes.**

  Ex: WHERE COD_CAT IN (1,2)

- **Podem também ser avaliados com base em valores dinâmicos, a retirar da base de dados**

Ex: WHERE COD_CAT IN
(SELECT COD_CAT FROM CATEGORIA)

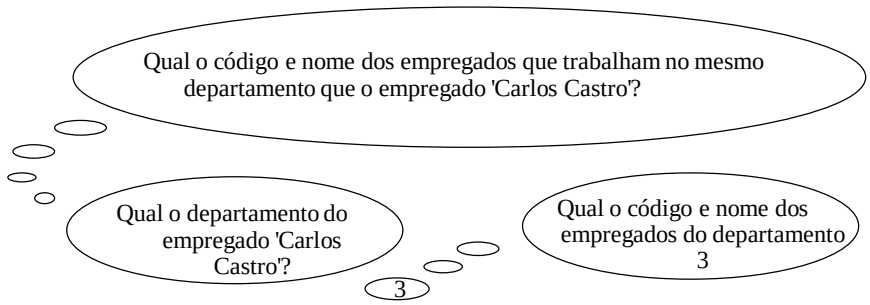


Predicados utilizados em Subqueries

As subqueries são sobretudo usadas em:

Predicados de comparação
Predicado IN
Predicado EXISTS

Subqueries




```
SELECT    cod_dept
FROM      empregado
WHERE     nome_emp = 'Carlos Castro'
```

```
SELECT  cod_emp, nome_emp
FROM    empregado
WHERE   cod_dept = 3
```


Subqueries

Integração das duas Queries

```
SELECT  cod_emp, nome_emp
FROM    empregado
WHERE   cod_dept =  ( SELECT cod_dept
                      FROM empregado
                      WHERE nome_emp = 'Carlos Castro')
```


Subqueries

Quais os nomes dos empregados que trabalham nos departamentos de Lisboa

```
SELECT  cod_emp, nome_emp
FROM    empregado
WHERE   cod_dept IN  (  SELECT cod_dept
                        FROM departamento
                        WHERE localização = 'Lisboa'
                     )
```


Subqueries Correlacionadas vs Não Correlacionadas

- **Subqueries Não Correlacionadas:**
  - **Não dependem de dados da query exterior**
  - **São executadas uma única vez**
- **Subqueries Correlacionadas:**
  - **Utilizam dados da query exterior**
  - **São executadas para cada linha da query exterior**
  - **Utilizam o operador EXISTS**

Operador EXISTS

Nome dos departamentos que têm empregados (pelo menos um)

```
SELECT nome_dept
FROM  departamento
WHERE EXISTS
        ( SELECT  *
          FROM  empregado
          WHERE departamento.cod_dept = empregado.cod_dept
    )
```

A condição é VERDADEIRA se o resultado da subquery não for vazio

Operador NOT EXISTS

Nome dos departamentos que não têm empregados

```
SELECT nome_dept
FROM  departamento
WHERE  NOT EXISTS
        ( SELECT  *
          FROM  empregado
          WHERE departamento.cod_dept = empregado.cod_dept
    )
```

A condição é VERDADEIRA se o resultado da subquery for vazio

Divisão (exemplo)

Nomes dos departamentos que têm empregados de todas as categorias?

Empregado

cod_emp	nome_emp	cod_cat	cod_dept
1	António Abreu	1	01
2	Bernardo Bento	1	02
3	Carlos Castro	3	03
4	Diogo Dado	2	02
5	Ernesto Eco	3	02

Categoria

cod_cat	designação	salario_base
1	CategoriaA	300
2	CategoriaB	250
3	CategoriaC	160

:

=

cod_dept
02

Divisão

Nomes dos departamentos que têm empregados de todas as categorias?

⇔

Nome dos departamentos para os quais, qualquer que seja a categoria, existe algum empregado desse departamento e dessa categoria

⇔

Nome dos departamentos: ∀ categoria ∈ categorias

p(x)

```
( ∃ empregado :
  empregado.cod_dept  = departamento.cod_dept
  and empregado.cod_cat = categoria.cod_cat
)
```


Divisão

Sabendo que: $\forall x : p(x) \Leftrightarrow \sim\exists x : \sim p(x)$

Nome dos departamentos: ~∃ categoria ∈ categorias (~ p(x))

⇔

Nome dos departamentos: ~∃ categoria ∈ categorias

(~∃ empregado :
empregado.cod_dept = departamento.cod_dept
and empregado.cod_cat = categoria.cod_cat
)

Divisão

Comando SQL

```
SELECT          nome_dept  → Quociente
FROM   departamento
WHERE           NOT EXISTS

        ( SELECT  *  → Divisor
          FROM  categoria
          WHERE  NOT EXISTS

Dividendo →     (SELECT     *
                  FROM      empregado
                  WHERE     empregado.cod_dept  = departamento.cod_dept
                            and empregado.cod_cat = categoria.cod_cat ))
```


Clausula ORDER BY

A clausula **ORDER BY** é usada para ordenar os dados referentes a uma ou mais colunas

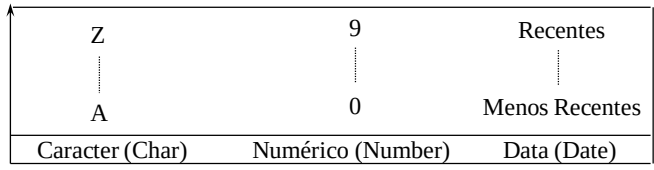
É a última clausula a ser especificada

```
SELECT      [ DISTINCT ] coluna, ... | *
FROM        tabela
WHERE       condição
ORDER BY    coluna [ASC | DESC ], ...
```

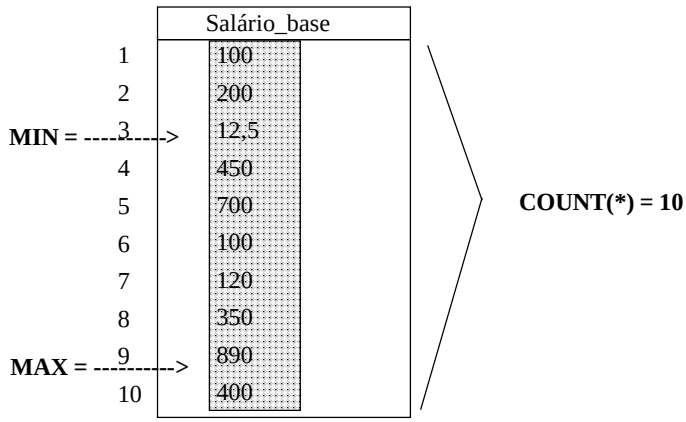

Clausula ORDER BY

```
SELECT      *
FROM        empregado
ORDER BY    nome_emp
```

Por **defeito**, os dados são ordenados **ascendentemente**



Funções Agregadoras



SUM = • **AVG = SUM / COUNT**

Funções Agregadoras

```
SELECT  MAX(salario_base)
FROM    categoria
```

```
SELECT  MIN(salario_base)
FROM    categoria
```

```
SELECT  COUNT(*)
FROM    categoria
```

```
SELECT  SUM(salario_base)
FROM    categoria, empregado
WHERE   empregado.cod_cat = categoria.cod_cat
```

```
SELECT  AVG(salario_base)
FROM    categoria, empregado
WHERE   empregado.cod_cat = categoria.cod_cat
```


Funções Agregadoras com Restrições

```
SELECT    AVG(salario_base)
FROM      empregado, categoria
WHERE     cod_dept = 1
          and
          empregado.cod_cat = categoria.cod_cat
```

Média dos salários dos empregados do departamento cujo código é 1

Agrupamentos

Cod_dept	Salário_base	
1	120	120
1	250	
1	150	
1	300	
1	250	
2	100	100
2	150	
2	230	
3	300	160
3	400	
3	200	
3	160	

Para cada departamento qual o salário minímo?

```
SELECT      cod_dept, min(salario_base)
FROM        empregado, categoria
WHERE       empregado.cod_cat = categoria.cod_cat
GROUP BY    cod_dept
```


Agrupamentos Múltiplos

Cod_dept	Cod_cat	Salário_base	
1	A	120	120
1	A	250	
1	B	150	150
1	B	300	
1	B	250	
2	A	100	100
2	B	150	150
2	B	230	
3	B	300	300
3	B	400	
3	C	200	160
3	C	160	

Para cada categoria de cada departamento qual o salário minímo?

```
SELECT      cod_dept, cod_cat, min(salario_base)
FROM        empregado, categoria
WHERE       empregado.cod_cat = categoria.cod_cat
GROUP BY    cod_dept, cod_cat
```


Agrupamentos Múltiplos

```
SELECT      [ DISTINCT ] coluna, ... | *
FROM        tabela, ...
WHERE       condição
GROUP BY    coluna,  ...
```

Qualquer coluna que não seja uma função agregadora só pode estar na cláusula SELECT se estiver na claúsula GROUP BY

```
SELECT      COD_DEPT, min(salario_base)
FROM        empregado, categoria
WHERE       empregado.cod_cat = categoria.cod_cat
GROUP BY    COD_DEPT
```


Restrições sobre Grupos

Cod_dept	Salário_base	
1	120	AVG = 214
1	250	
1	150	120
1	300	
1	250	
2	100	AVG = 160
2	150	
2	230	100
3	300	AVG = 265
3	400	
3	200	160
3	160	

Para cada departamento qual o salário mínimo.

Selecionar apenas os departamentos cujo salário médio seja superior a 200

```
SELECT      cod_dept, min(salario_base)
FROM        empregado, categoria
WHERE       empregado.cod_cat = categoria.cod_cat
GROUP BY    cod_dept
HAVING      avg (salario_base) > 200
```


Cláusula HAVING

```
SELECT      [ DISTINCT ] coluna, ... | *
FROM        tabela, ...
WHERE       condição
GROUP BY    coluna,  ...
HAVING      condição
```

WHERE OU HAVING ?

A cláusula WHERE aplica-se a linhas

A cláusula HAVING aplica-se a grupos (Group By)

Subqueries com Funções Agregadoras

Qual o nome do empregado que tem o maior salário

```
SELECT  empregado.cod_emp, nome_emp
FROM    empregado, categoria
WHERE   empregado.cod_cat = categoria.cod_cat and
        salário_base  =  ( SELECT max(salário_base)
                           FROM categoria, empregado
                           WHERE empregado.cod_cat = categoria.cod_cat
                         )
```


Subqueries com Agrupamentos

Para cada departamento qual o empregado que tem o maior salário

```
SELECT  cod_dept, cod_emp, nome_emp
FROM    empregado, categoria
WHERE   empregado.cod_cat = categoria.cod_cat and
        (cod_dept, salario_base)  IN
                        ( SELECT cod_dept, max(salario_base)
                          FROM categoria, empregado
                          WHERE empregado.cod_cat = categoria.cod_cat
                          GROUP BY cod_dept
                        )
```


USO de Grouping Sets, Rollup, Cube, Grouping

Grouping Sets e Rollup

```
SELECT fact_1_id,
       fact_2_id,
       SUM(sales_value) AS sales_value
FROM   dimension_tab
GROUP BY Grouping Sets (fact_1_id,
(fact_1_id,Fact_2_id),())
ORDER BY fact_1_id, fact_2_id;
```

```
SELECT fact_1_id,
       fact_2_id,
       SUM(sales_value) AS sales_value
FROM   dimension_tab
GROUP BY ROLLUP (fact_1_id, fact_2_id)
ORDER BY fact_1_id, fact_2_id;
```

FACT_1_ID	FACT_2_ID	SALES_VALUE
1	1	4799.08
1	2	4371.3
1	3	4620.13
1	4	6403.59
1	5	4993.41
1	-	25187.51
2	1	5285.17
2	2	5301.6
2	3	4025.97
2	4	5475.51
2	5	4776.22
2	-	24864.47
-	-	50051.98

Grouping Sets e CUBE

```
SELECT fact_1_id,
       fact_2_id,
       SUM(sales_value) AS sales_value
FROM   dimension_tab
       GROUP BY CUBE (fact_1_id, fact_2_id)
ORDER BY fact_1_id, fact_2_id;
```

```
SELECT fact_1_id,
       fact_2_id,
       SUM(sales_value) AS sales_value
FROM   dimension_tab
GROUP BY Grouping Sets (fact_1_id, Fact_2_id,
(fact_1_id,Fact_2_id),())

ORDER BY fact_1_id, fact_2_id;
```

FACT_1_ID	FACT_2_ID	SALES_VALUE
1	1	4799.08
1	2	4371.3
1	3	4620.13
1	4	6403.59
1	5	4993.41
1	-	25187.51
2	1	5285.17
2	2	5301.6
2	3	4025.97
2	4	5475.51
2	5	4776.22
2	-	24864.47
-	1	10084.25
-	2	9672.9
-	3	8646.1
-	4	11879.1
-	5	9769.63
-	-	50051.98

GROUPING

```
SELECT fact_1_id,
       fact_2_id,
       SUM(sales_value) AS sales_value,
       GROUPING(fact_1_id) AS f1g,
       GROUPING(fact_2_id) AS f2g
FROM   dimension_tab
GROUP BY CUBE (fact_1_id, fact_2_id)
ORDER BY fact_1_id, fact_2_id;;
```

FACT_1_ID	FACT_2_ID	SALES_VALUE	F1G	F2G
1	1	4799.08	0	0
1	2	4371.3	0	0
1	3	4620.13	0	0
1	4	6403.59	0	0
1	5	4993.41	0	0
1	-	25187.51	0	1
2	1	5285.17	0	0
2	2	5301.6	0	0
2	3	4025.97	0	0
2	4	5475.51	0	0
2	5	4776.22	0	0
2	-	24864.47	0	1
-	1	10084.25	1	0
-	2	9672.9	1	0
-	3	8646.1	1	0
-	4	11879.1	1	0
-	5	9769.63	1	0
-	-	50051.98	1	1

Função RANK

- The RANK() function assigns a rank to each row within a partition of a result set.
- The rows within a partition that have the same values will receive the same rank.
- The rank of the first row within a partition is one

```
RANK() OVER (
 [PARTITION BY partition_expression, ... ]
ORDER BY sort_expression [ASC | DESC], ...
)
```

RANK

RANK() function assigns ranks to the products by their list prices:

```
SELECT productid, name, listprice,
RANK () OVER ( ORDER BY listprice DESC ) price_rank
FROM Product;
```

Results Messages

	productid	name	listprice	price_rank
1	749	Road-150 Red, 62	3578,2700	1
2	750	Road-150 Red, 44	3578,2700	1
3	751	Road-150 Red, 48	3578,2700	1
4	752	Road-150 Red, 52	3578,2700	1
5	753	Road-150 Red, 56	3578,2700	1
6	771	Mountain-100 Silver, 38	3399,9900	6
7	772	Mountain-100 Silver, 42	3399,9900	6
8	773	Mountain-100 Silver, 44	3399,9900	6
9	774	Mountain-100 Silver, 48	3399,9900	6

57

RANK

```
SELECT productid, name, color, listprice,
RANK () OVER ( PARTITION BY color
ORDER BY listprice DESC )
FROM Product
```

Results Messages

	productid	name	color	listprice	(No column name)
52	775	Mountain-100 Black, 38	Black	3374,99…	1
53	776	Mountain-100 Black, 42	Black	3374,99…	1
54	777	Mountain-100 Black, 44	Black	3374,99…	1
55	778	Mountain-100 Black, 48	Black	3374,99…	1
56	793	Road-250 Black, 44	Black	2443,35…	5
57	794	Road-250 Black, 48	Black	2443,35…	5
58	795	Road-250 Black, 52	Black	2443,35…	5
59	796	Road-250 Black, 58	Black	2443,35…	5
60	782	Mountain-200 Black, 38	Black	2294,99…	9
61	783	Mountain-200 Black, 42	Black	2294,99…	9
62	784	Mountain-200 Black, 46	Black	2294,99…	9
63	680	HL Road Frame - Black…	Black	1431,50…	12

Inline Views

Enquanto até à norma SQL-89 a utilização de *nested queries* se resumia à claúsula where dos comandos SELECT, com a SQL 92 a utilização destas *queries* foi liberalizada, **podendo surgir em qualquer ponto do comando select, desde que produza um resultado adequado a essa localização.** Às subqueries nas claúsulas *select* ou *from* chamamos *inline views*.

Inline Views

Para cada departamento qual o empregado que tem o maior salário

```
SELECT emp.cod_dept, cod_emp, nome_emp, salario_base
  FROM                (empregado emp natural join categoria cat ) join
      ( SELECT cod_dept, max(salario_base) salmax
                  FROM categoria, empregado
                  WHERE empregado.cod_cat = categoria.cod_cat
                  GROUP BY cod_dept) dep
                on emp.cod_dept= dep.cod_dept and
                cat.salario_base=dep.salmax
```


Comando SELECT

```
SELECT      [ DISTINCT ] coluna, ... | *
FROM        tabela, ...
WHERE       condição
GROUP BY    coluna, ...
HAVING      condição
ORDER BY    coluna [ASC | DESC ], ...
```


Manipulação da Base de Dados

INSERÇÕES, ATUALIZAÇÕES
e REMOÇÕES

```
INSERT INTO tabela_nome [ (coluna, coluna, ....)]
VALUES (valor1, valor2, ...) | comando SELECT

UPDATE tabela_nome SET lista_de_atribuições
[WHERE condição]

DELETE FROM tabela_nome
[WHERE condição]
```


Insert

```
INSERT INTO DEPARTAMENTO VALUES (4,'Marketing','Lisboa')
```

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
...	...	...

→

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
4	Marketing	Lisboa
...	...	...

Cópia de Valores de outras Tabelas

```
INSERT INTO EMP_HIST
(cod_emp, nome_emp, data_admissao)
SELECT cod_emp, nome_emp, data_admissao
FROM empregado
WHERE data_admissao > 01-Jan-81
```


Insert

```
INSERT INTO DEPARTAMENTO (cod_dept, nome_dept) VALUES (5,'Produção')
```

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
4	Marketing	Lisboa
...	...	...

→

cod_dept	nome_dept	localização
1	Contabilidade	Lisboa
2	Vendas	Porto
3	Investigação	Coimbra
4	Marketing	Lisboa
5	Produção	

Insert

Filme (fID, titulo, ano, realizador)
Cada filme é identificado por um número fID, um título, ano de produção e realizador.

Critico (cID, nome)
Cada crítico é identificado por um número cID e pelo seu nome.

Classificacao (cID, fID, estrelas, dataClassificacao)
Cada classificação é caraterizada pelo nº do crítico, nº do filme, nº de estrelas atribuídas (1-5) e data da classificação.

Questão
Insira uma classificação de 5 estrelas atribuída pelo critico Daniel Morgado em todos os filmes da base de dados. Deixe a NULL a data da classificação.

Resposta

```
Insert into Classificacao (cID,fID,estrelas)

Select (select cid from critico where nome='Daniel Morgado'),  fID,5
from  Filme
```

Jennifer Widom, Stanford University

Update e Delete

Atualizar o código do chefe do empregado Bernardo Bento

```
UPDATE empregado
SET    cod_emp_chefe=2
WHERE nome_emp = 'Bernardo
      Bento'
```

Apagar todos os empregados que trabalham no departamento 2

```
DELETE FROM
      empregado
WHERE cod_dept = 2
```


Update

Filme (fID, titulo, ano, realizador)
Cada filme é identificado por um número fID, um título, ano de produção e realizador.

Critico (cID, nome)
Cada crítico é identificado por um número cID e pelo seu nome.

Classificacao (cID, fID, estrelas, dataClassificacao)
Cada classificação é caraterizada pelo nº do crítico, nº do filme, nº de estrelas atribuídas (1-5) e data da classificação.

Questão:
Para todos os filmes cuja média de classificação é 4 ou superior, adicione 25 ao ano de produção.

Resposta:

```
Update Filme
set ano=ano+25
where fID in
(select fID
from Classificacao
group by fID
having avg (estrelas)>=4)
```

Jennifer Widom, Stanford University

Delete

Filme (fID, titulo, ano, realizador)
Cada filme é identificado por um número fID, um título, ano de produção e realizador.

Critico (cID, nome)
Cada crítico é identificado por um número cID e pelo seu nome.

Classificacao (cID, fID, estrelas, dataClassificacao)
Cada classificação é caraterizada pelo nº do crítico, nº do filme, nº de estrelas atribuídas (1-5) e data da classificação.

Question:
Apague todas as classificações inferiores a 4, de filmes produzidos antes de 1970 ou depois de 2000

Resposta:

```
Delete from Classificacao
Where fID in (select fID
from Filme where ano <1970 or ano >2000 )
and estrelas <4
```

Jennifer Widom, Stanford University

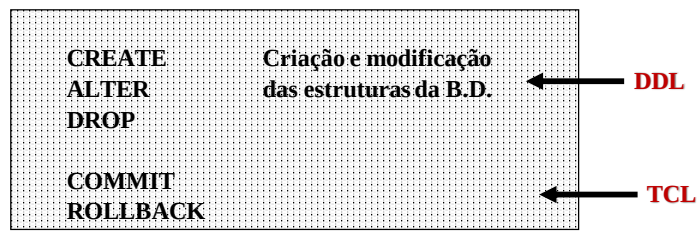
SQL

Caraterísticas atuais e Perspetivas futuras

- Caraterísticas e Componentes
- ☞ SQL na Definição da Base de Dados (DDL)
- SQL na Manipulação de Dados (DML)
- SQL no Controlo da Base de Dados (DCL)
- SQL no Controlo de Transações (TCL)

SQL

Definição da Base de Dados e Controlo de Transações



Definição da Base de Dados

(1) Definição de uma tabela com uma chave primária

```
CREATE TABLE departamento
 ( cod_dept     INTEGER PRIMARY KEY,
   nome_dept    char(15)    NOT NULL,
   data_adm     date        NOT NULL,
   localização  char(20) )
```

(2) Definição de uma tabela com uma chave primária composta

```
CREATE TABLE linha_enc
 ( n_enc        INTEGER,
   n_produto    INTEGER,
   quantidade   INTEGER NOT NULL,
 PRIMARY KEY (n_enc, n_produto) )
```


Definição da Base de Dados

(3) Definição de uma tabela com uma chave estrangeira

```
CREATE TABLE empregado
 ( cod_emp     INTEGER PRIMARY KEY,
   nome_emp    char(15)    DEFAULT 'Desconhecido',
   cod_dept    char(20)    REFERENCES departamento(cod_dept) [ON DELETE
 [CASCADE|SET NULL|SET DEFAULT]]
 [ON UPDATE [CASCADE|SET NULL|SET DEFAULT]] )
```

(4) Definição de uma tabela com uma chave estrangeira composta

```
CREATE TABLE faltas_material
 ( n_falta      INTEGER,
   data_falta   date,
   n_enc        INTEGER,
   n_produto    INTEGER,
  PRIMARY KEY (n_falta,data_falta),
  FOREIGN KEY (n_enc, n_produto) REFERENCES linha_enc(n_enc, n_produto) )
```

Definição da Base de Dados

(5) Definição de uma tabela com uma regra de verificação

```
CREATE TABLE encomenda
 ( n_enc        INTEGER [CONSTRAINT chave_enc ] PRIMARY KEY,
   data_enc     date        NOT NULL,
   cod_cliente  INTEGER [CONSTRAINT cli_enc ] REFERENCES cliente(cod_cliente),
   data_entrega date,
  CHECK (data_entrega > data_enc) )
```

(6) Definição de uma tabela com valores selecionados de outra tabela

```
CREATE TABLE emp_dept1
AS SELECT cod_emp,  nome_emp, data_admissao
FROM empregado
WHERE cod_dept = 1
```


Definição da Base de Dados

(7) Definição de uma tabela com CHAVE SURROGATE

```
CREATE TABLE departamento
 ( cod_dept     INTEGER   GENERATED ALWAYS AS IDENTITY
   [START WITH 10 INCREMENT BY 10] PRIMARY KEY ,
   nome_dept   char(15)    NOT NULL,
   data_adm    date        NOT NULL,
   localização  char(20) )
```

Note:
The identity column is not inherited by the CREATE TABLE AS SELECT statement
https://www.oracletutorial.com/oracle-basics/oracle-identity-column/

Definição da Base de Dados
Criação de Índices

Definição de um Índice

CREATE [UNIQUE] INDEX nome_indice
ON Table (attribute)

Exemplo

CREATE INDEX indice_empregado_nome
ON empregado (nome_emp)

Alguns Tipos de Dados

NUMBER [(precision, scale)]
Ex: Number (5,2)

INTEGER

BINARY_FLOAT
32-bit floating point number (5 bytes)

BINARY_DOUBLE
64-bit floating point number (9 bytes)

CHAR
VARCHAR2

DATE

http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330

Alguns Tipos de Dados

LOB (*Large objects*) datatypes :

BLOB
A binary large object. Maximum size is (4 gigabytes - 1) * (database block size –usually 16K to 32K).

CLOB
A character large object containing single-byte or multibyte characters. Maximum size is (4 gigabytes - 1) * (database block size).

BFILE
Contains a locator to a large binary file stored outside the database. Maximum size is 4 gigabytes.

http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330

Data Type Date

Select Sysdate from dual – para saber o formato da data

Normalmente é MM-DD-YYYY

Algumas Funções aplicáveis a Date

Para apresentar a data no formato pretendido

Select **TO_CHAR** (data_aquisicao,'DD-MM-YYYY') from passagem

Para converter uma string para uma data

Insert into passagem values (1, **TO_DATE** ('25-01-1999','dd-mm-yyyy'), 10, 1, 50000);

Sem usar o TO_DATE a data tem de ser inserida no formato de SYSDATE

http://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330

Alter Table

```
ALTER TABLE nome_tabela
ADD [nova coluna] | [CONSTRAINT] nova
    restrição_coluna]
```

```
ALTER TABLE nome_tabela
MODIFY definição de coluna
```

```
ALTER TABLE nome_tabela
DROP [COLUMN coluna|
    CONSTRAINT restrição_coluna]
```

Não se pode modificar uma coluna contendo valores nulos para NOT NULL.

Só se pode adicionar uma coluna NOT NULL a uma tabela que não contenha nenhuma linha. Solução: Adicione como NULL, preencha-a completamente e depois mude para NOT NULL

Pode-se decrementar o tamanho de uma coluna e o tipo de dados, caso essa coluna contenha valores nulos em todas as linhas.

Nota: Disponível em quase todos os SGBDR existentes no mercado.

Alter Table

```
alter table empregado
ADD comissão  NUMBER(4)
```

```
alter table passageiro
ADD CONSTRAINT intervalo CHECK
(num_passageiro < 100000)
```

```
alter table empregado MODIFY
(cod_emp number(15))
```

```
alter table empregado
DROP  COLUMN
comissão
```


Tabelas interessantes do Sistema

USER_CONS_COLUMNS - Constraints existentes na BD

Ex: select * from user_cons_columns where table_name='DEPARTAMENTO'

ALL_INDEXES – índices existentes na BD

Ex: SELECT index_name, index_type, visibility, status
FROM all_indexes
WHERE table_name = 'DEPARTAMENTO'

View

cod_emp	nome_emp	data_admissão	cod_cat	cod_dept	cod_emp_chefe
1	António Abreu	13-Jan-75	1	1	1
2	Bernardo Bento	1-Dec-81	1	2	1
3	Carlos Castro	4-Jun-84	3	3	1
...	...	...	...	...	...
20	Manuel Matos	7-Feb-90	3	2	2
...	...	...	...	...	...

Não contêm informação própria

É uma imagem de uma tabela através de uma "janela" a partir da qual se pode visualizar e alterar os campos selecionados

Não ocupam espaço físico e por isso são vulgarmente denominadas tabelas virtuais

Assemelham-se a tabelas e com algumas restrições são tratadas como tal

View ≠ Tabela Temporária

Criação de Views

```
CREATE VIEW nome_view AS
comando_select
WITH CHECK OPTION
```

```
CREATE VIEW vCategoria AS
SELECT *
from categoria
where salario_base<300
With check option
```

No comando select podem-se utilizar todas as clásulas excepto a claúsula Order By

Podem-se definir views à custa de outras views

As alterações na tabela original refletem-se nas views dessa tabela

```
DROP VIEW nome_view
```


Atualização de Views

Restrições à Atualização de Views segundo a Norma SQL

1. **Select (sem Distinct) sobre uma única tabela T**
2. **Os atributos que não pertencem à view podem ser nulos ou ter um valor por defeito**
3. **Eventuais subqueries não podem referenciar T**
4. **Não utilização de GROUP BY ou de funções de agregação**

Materialized Views

1. CREATE MATERIALIZED VIEW LOG ON **empregado** WITH PRIMARY KEY
2. CREATE MATERIALIZED VIEW **vemp**

   BUILD IMMEDIATE (default)

   REFRESH [FAST|COMPLETE|FORCE] WITH PRIMARY KEY FOR UPDATE

   AS SELECT cod_emp, nome_emp

   FROM empregado

FAST : A fast refresh is attempted. If materialized view logs are not present against the source tables in advance, the creation fails.
COMPLETE: The table segment supporting the materialized view is truncated and repopulated completely using the associated query.
FORCE : A fast refresh is attempted. If one is not possible a complete refresh is performed.

SQL

Caraterísticas atuais e Perspetivas Futuras

- Caraterísticas e Componentes
- SQL na Manipulação de Dados (DML)
- SQL na Definição da Base de Dados (DDL)
- SQL no Controlo da Base de Dados (DCL)
- ☞ Transaction Control Language (TCL)

Transaction Control Language (TCL)

Transações

TRANSAÇÃO

Unidade de trabalho, que para ser realizada pode necessitar de várias operações

Exemplo: transferir 500 € da Conta Poupança para a Conta à Ordem - é necessário debitar da conta poupança e creditar na à ordem

Todas as operações da transação devem ser:

- **EFETIVADAS**
  utilizando o comando COMMIT
- **ANULADAS**
  utilizando o comando ROLLBACK

Transações - Exemplo

```
START TRANSACTION; (não suportado pelo Oracle)
UPDATE savings_accounts
SET balance = balance - 500
WHERE account = 3209;

UPDATE checking_accounts
SET balance = balance + 500
WHERE account = 3208;

[COMMIT|ROLLBACK];
```


Oracle Application Express

https://apex.oracle.com/en/

