Linguagem SQL

SQL

Características actuais e Perspectivas futuras

- **Características e Componentes**
- **SQL na Manipulação de Dados**
- **SQL na Definição da Base de Dados**

História

- 1970: Codd define o Modelo Relacional
- 1974: IBM desenvolve o projecto 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 conjunção com o XML (ANSI/ISO/IEC 9075-14:2006)
- 2008: SQL:2008

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:
  - interactivamente
  - embebida em linguagens de programação

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
...	...	...	...	...	...

Comando SQL

Qual o salário do empregado ´António Abreu´ e o nome do departamento a que pertence?

```
SELECT   nome_emp, salario_base, nome_dept
FROM  Empregado, Departamento, Categoria
WHERE  nome_emp = ´António Abreu´
    AND  Empregado.cod_cat = Categoria.cod_cat
    AND  Departamento.cod_dept = Empregado. cod_dept
```

Características

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

  Existe uma clara abstracção 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 efectuam-se sobre conjuntos de dados (tabelas), não sendo necessário (nem possível) manipular linha-a-linha

Componentes

DDL (Data Definition Language)

DML (Data Manipulation Language)

TML (Transaction Manipulation Language)

DCL (Data Control Language)

SQL

Características actuais e Perspectivas futuras

- Características e Componentes

- SQL na Manipulação de Dados
- SQL na Definição da Base de Dados

SQL

Manipulação de Dados

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

Clausula SELECT e FROM

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

O símbolo * é utilizado quando se pretende seleccionar 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

Projecção

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 selecciona-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
```

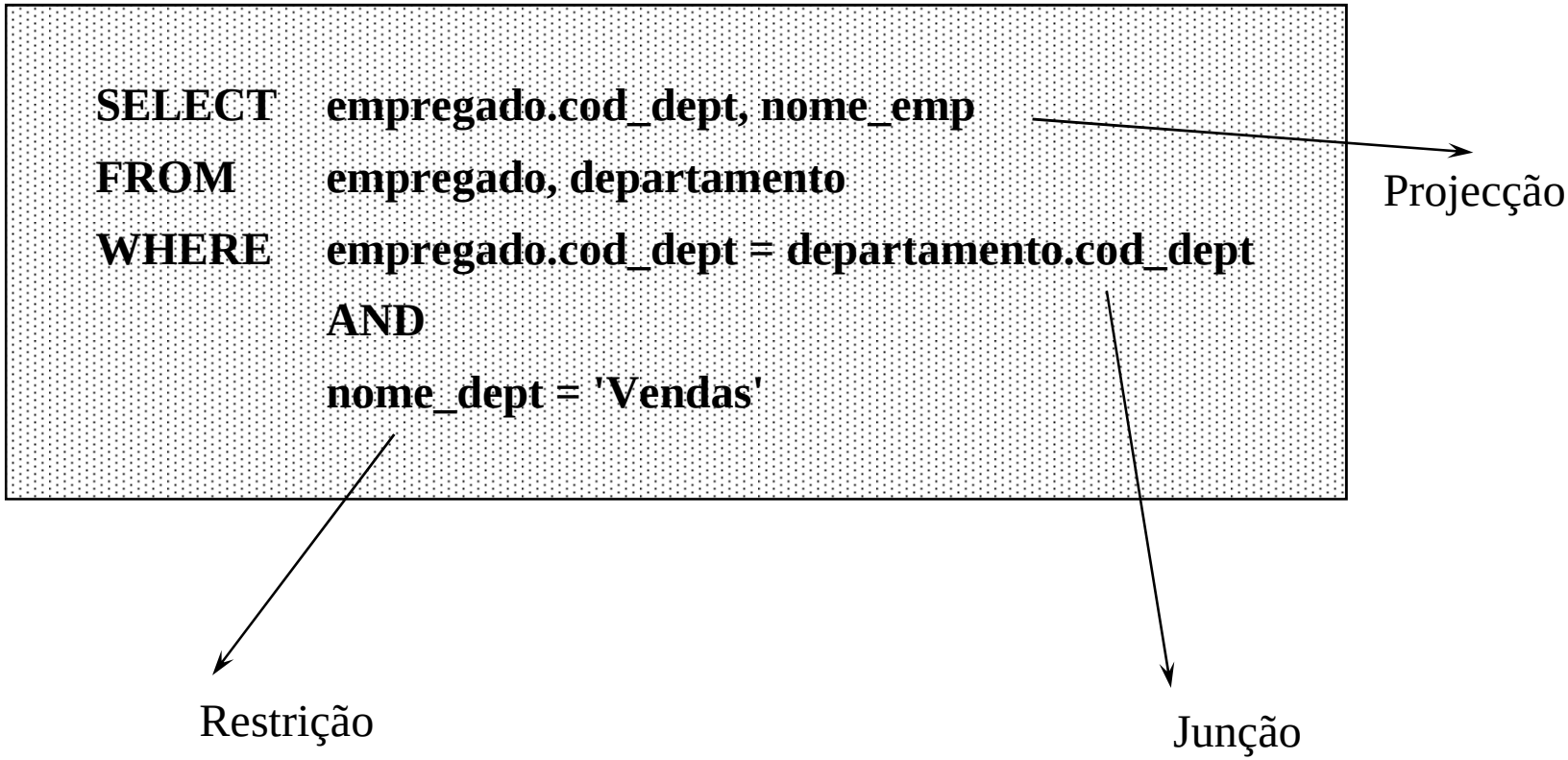
Caso o nome de uma coluna seja igual em várias tabelas então a REGRA é

Nome_Tabela.Nome_Coluna

em qualquer sítio da cláusula SELECT

Projecção, Restrição e Junção

Qual o nome dos empregados pertencentes ao departamento de Vendas



Aliases de Tabelas

Correlation Name

```
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

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

```
SELECT  E.nome, CH.nome
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 repectivo 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 respectivos 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, empregado.cod_dept, nome_dept
FROM     empregado, departamento (----> ver nota)
WHERE    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

Nota: apresenta-se acima a versão Oracle. Em SQL.92 seria :

FROM empregado RIGHT OUTER JOIN departamento on (empregado.cod_dept = departamento.cod_dept)

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
```

Intersecção

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
MINUS    (----> ver nota )
SELECT   nome, morada
FROM     fornecedor
```

Nota: apresenta-se acima a versão Oracle.
Em SQL.92 seria EXCEPT

Clausula WHERE

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

Uma condição-de-pesquisa é basicamente uma colecção 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%')
- Um teste de valor nulo (WHERE COMISSÃO **IS** NULL)
- Um predicado de IN (WHERE COD_CAT **IN** (1,2))

Predicados

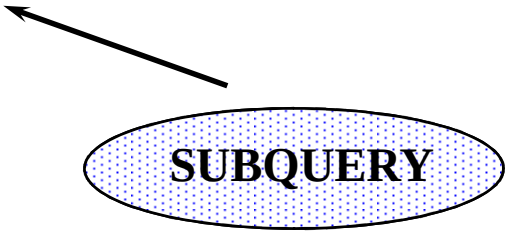
- **Os predicados podem ser utilizados num contexto estático, sendo avaliados com base em valores constantes.**

  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 usadas em:

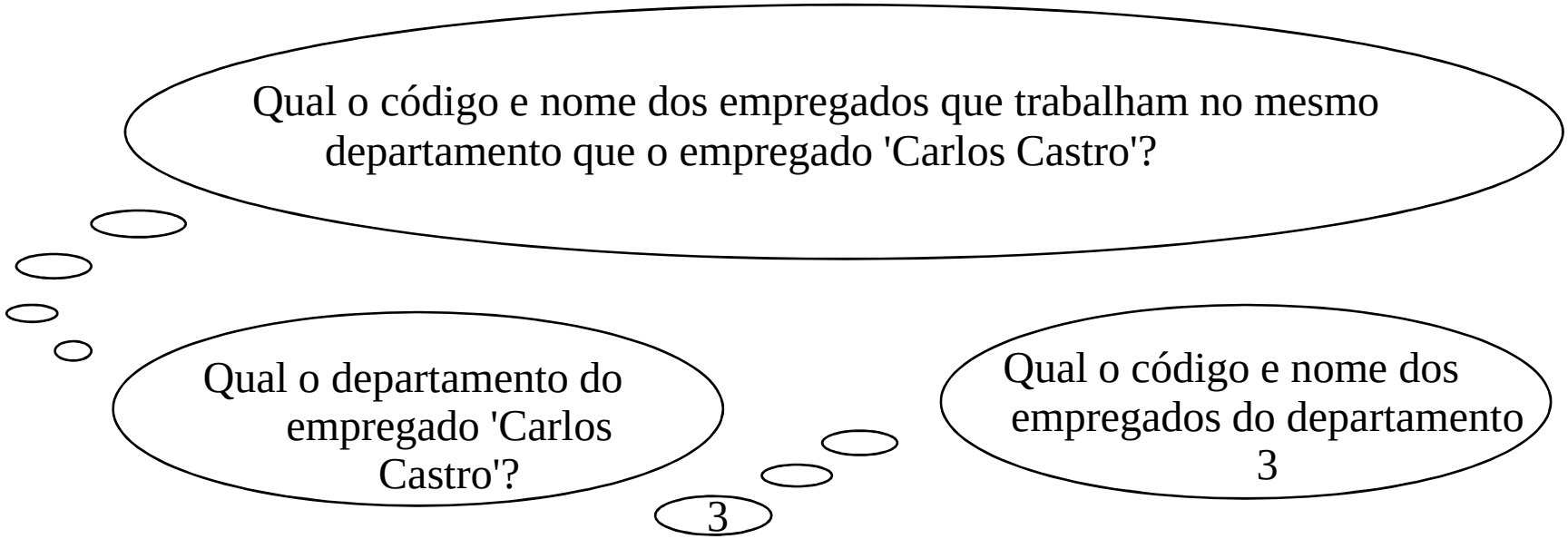
Predicados de comparação

Predicado IN

Predicados ALL ou ANY

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

Quais os empregados cujo salário é superior a **todos** os salários dos empregados do departamento 1

```
SELECT nome_emp
FROM  empregado, categoria
WHERE  empregado.cod_cat = categoria.cod_cat
          AND
          salário_base > ALL
          ( SELECT salário_base
            FROM  empregado, categoria
            WHERE empregado.cod_cat = categoria.cod_cat
                        AND cod_dept = 1
          )
```

Subqueries

Quais os empregados cujo salário é superior a **algum** dos salários dos empregados do departamento 1

```
SELECT nome_emp
FROM  empregado, categoria
WHERE  empregado.cod_cat = categoria.cod_cat
          AND
          salário_base > ANY
          ( SELECT salário_base
            FROM  empregado, categoria
            WHERE empregado.cod_cat = categoria.cod_cat
                        AND cod_dept = 1
          )
```

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)

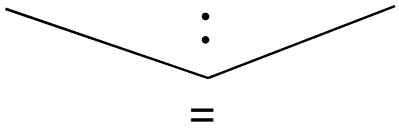
Nomes dos departamentos que têm empregados de todas as categorias?

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?

$\Leftrightarrow$

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

$\Leftrightarrow$

Nome dos departamentos: $\forall$ categoria $\in$ categorias

p(x)

$$
\begin{array}{l}
(\ \exists \text{ empregado} : \\
\qquad \text{empregado.cod\_dept} = \text{departamento.cod\_dept} \\
\qquad \text{and empregado.cod\_cat} = \text{categoria.cod\_cat} \\
)
\end{array}
$$

Divisão

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

Nome dos departamentos: $\sim\exists$ categoria $\in$ categorias (~ p(x))

$\Leftrightarrow$

Nome dos departamentos: $\sim\exists$ categoria $\in$ categorias

($\sim\exists$ empregado :

empregado.cod_dept = departamento.cod_dept

and empregado.cod_cat = categoria.cod_cat

)

Divisão

Comando SQL

```
SELECT          nome_dept
FROM   departamento
WHERE           NOT EXISTS

        ( SELECT  *
          FROM  categoria
          WHERE  NOT EXISTS

                  ( 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

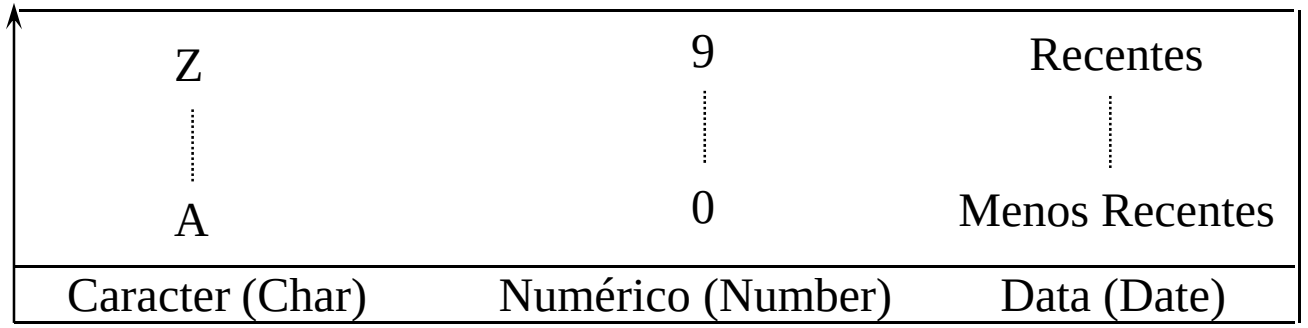
É a última clausula a ser especificada

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

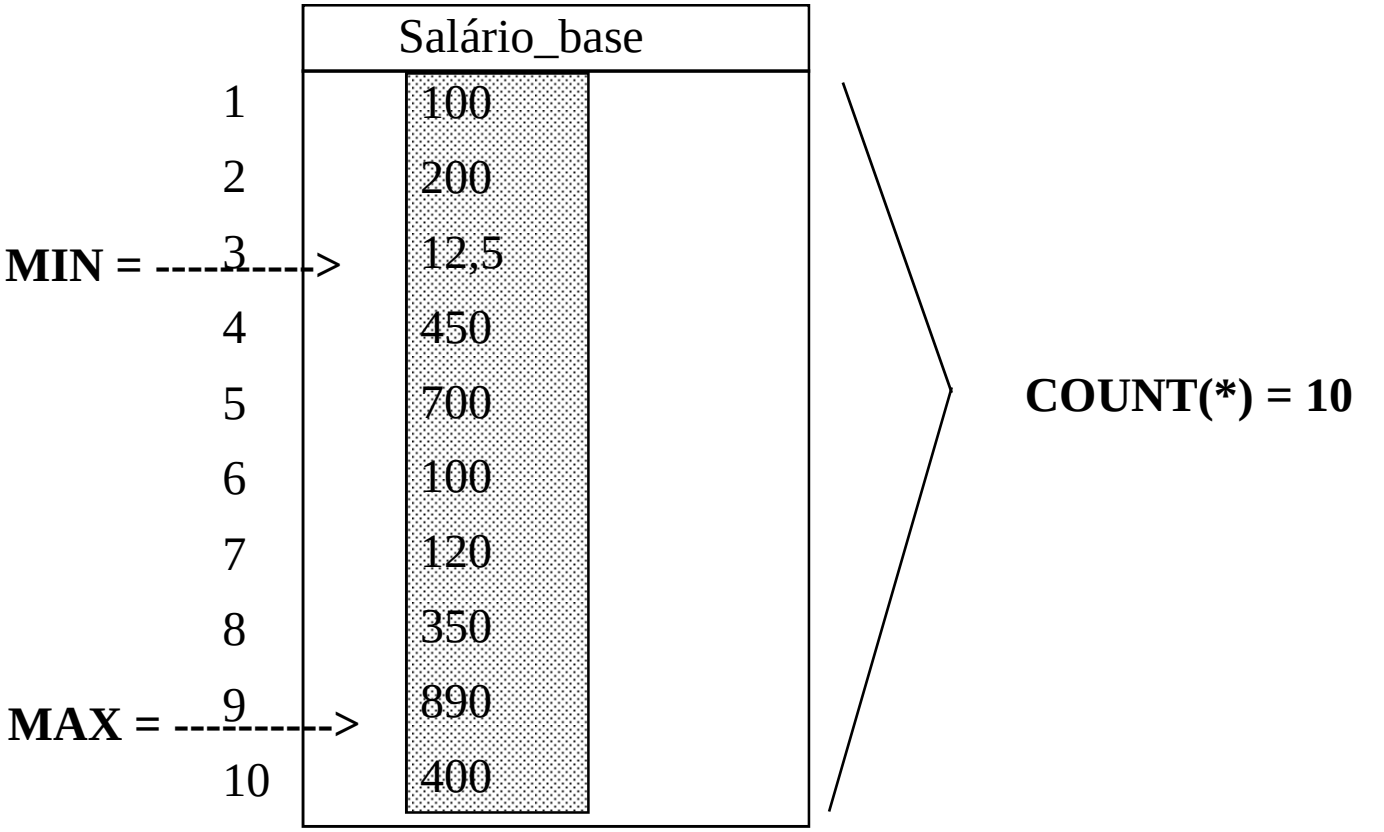
Clausula ORDER BY

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

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



Funções Agregadoras



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.

Seleccionar 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 nunca contém funções agregadoras

A cláusula HAVING deve sempre conter funções agregadoras

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

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 na clausula *from* chamamos *inline views*.

```
select *
from (
        select job, (
                        select sum(sal)
                        from emp
                        where job = a.job) totsal
         from emp a
        group by job)
where rownum <= 3 order by totsal desc;
```

JOB	TOTSAL
MANAGER	8275
ANALYST	6000
CLERK	4150

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, sal ário_base)  IN
                        (  SELECT cod_dept, max(salário_base)
                           FROM categoria, empregado
                           WHERE empregado.cod_cat = categoria.cod_cat
                           GROUP BY cod_dept
                        )
```

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, ACTUALIZAÇÕES
e REMOÇÕES

INSERT INTO tabela_nome [(coluna, coluna, ...)]
VALUES (valor, valor, ...) | 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_admissão)
SELECT cod_emp, nome_emp, data_admissão
FROM empregado
WHERE data_admissão > '1-JAN-91'
```

Update e Delete

Actualizar 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
```

SQL

Características actuais e Perspectivas Futuras

- Características e Componentes
- SQL na Manipulação de Dados
- SQL na Definição da Base de Dados

SQL

Definição da Base de Dados

CREATE	**Criação e modificação**
ALTER	**das estruturas da B.D.**
DROP	
GRANT	**Controle da segurança**
REVOKE	**da B.D.**

Definição da Base de Dados

```
CREATE TABLE nome_tabela
[ ( [ nome_coluna    tipo_dados    [restrição_coluna] ]  |
     [restrição_tabela],....) ] | [AS SELECT  comando]
```

restrição_coluna

CONSTRAINT nome_regra_coluna

[NULL | NOT NULL] | [UNIQUE | PRIMARY KEY] |

[REFERENCES tabela (coluna) [ON DELETE CASCADE]] |

[CHECK (condição)]

restrição_tabela

CONSTRAINT nome_regra_tabela

[[UNIQUE | PRIMARY KEY] (coluna,....) |

[FOREIGN KEY (coluna,...) REFERENCES tabela (coluna,...) [ON DELETE CASCADE]] |

[CHECK (condição)]

Definição da Base de Dados

(1) Definição de uma tabela com uma chave primária

```
CREATE TABLE departamento
  ( cod_dept      integer(4)  CONSTRAINT chave_dept 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(4),
    n_produto     integer(4),
    quantidade    integer(3)  NOT NULL,
CONSTRAINT chave_le 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(4)  CONSTRAINT chave_emp PRIMARY KEY,
    nome_emp    char(15)    NOT NULL,
    cod_dept    char(20)    CONSTRAINT dept_emp
                            REFERENCES departamento(cod_dept) )
```

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

```
CREATE TABLE faltas_material
  ( n_falta       integer(4),
    data_falta    date,
    n_enc         integer(4),
    n_produto     integer(4),
   CONSTRAINT chave_fme PRIMARY KEY (n_falta,data_falta),
   CONSTRAINT falta_le 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(4)  CONSTRAINT chave_enc PRIMARY KEY,
    data_enc      date        NOT NULL,
    cod_cliente   integer(4)) CONSTRAINT cli_enc REFERENCES cliente(cod_cliente),
    data_entrega date         CONSTRAINT ve_data CHECK (data_entrega > data_enc) )
```

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

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

Tipos de Dados

STANDARD SQL2

INTEGER

SMALLINT

NUMERIC

DECIMAL

REAL

DOUBLE PRECISION

FLOAT

CHARACTER
CHARACTER VARYING

BIT
BIT VARYING

DATE
TIME
TIMESTAMP

Alter Table

```
ALTER TABLE nome_tabela
ADD  novas colunas | novas restrições_coluna
```

```
ALTER TABLE nome_tabela
ALTER  definição das colunas
```

```
ALTER TABLE nome_tabela
DROP  coluna | 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  integer(4)  NOT NULL
```

```
alter table departamento
ALTER  cod_dept integer(6) NOT NULL
```

```
alter table empregado
DROP  comissão
```

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 seleccionados

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 $\neq$ Tabela Temporária

Vantagens da Utilização de Views

- SEGURANÇA

  Uma view permite restringir informação a certos grupos de utilizadores

- CONVENIÊNCIA

  É muitas vezes possível substituir uma consulta complexa por uma view, que é usada de uma forma mais simplista

Criação de Views

CREATE VIEW nome_view AS
comando_select

```
CREATE VIEW emp AS
SELECT cod_emp, nome_emp
FROM empregado
```

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 reflectem-se nas views dessa tabela

```
DROP VIEW nome_view
```

Privilégios

- **DE ESTRUTURA**

  o utilizador pode criar, alterar ou remover um objecto (ex. tabela)

- **DE CONTEÚDO**

  o utilizador pode inserir, alterar, remover ou aceder ao conteúdo de uma tabela

Estes privilégios são concedidos por utilizadores que possuem pelo menos os privilégios que estão a conceder.

Privilégios

- **GRANT**

  comando para conceder privilégios;

  pode ser dada ao concedido a possibilidade de também usar "grant's" (WITH GRANT OPTION)

- **REVOKE**

  comando para remover privilégios

EXEMPLO: Conceder privilégios de acesso a informação:

```
grant SELECT
on  EMPREGADO
to USER_A,USER_B
```

Transacções

TRANSACÇÃO

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

Exemplo: transferir 100000$00 da Conta à Ordem para a Poupança - é necessário debitar da conta à ordem e creditar na Poupança

Todas as operações da transacção devem ser:

- **EFECTIVADAS**

  utilizando o comando COMMIT

- **ANULADAS**

  utilizando o comando ROLLBACK