

Datawarehouse

Filipe Fidalgo

ffidalgo@ipcb.pt

Sumário

- Modelo Relacional
- SQL
- Álgebra Relacional
- Recuperação, Transacções e *Tunnig*

Modelo Relacional

Fase1: Modelo de entidades e Relacionamento

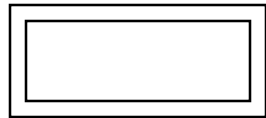
Fase2: Modelo Relacional

Fase3: Implementação SQL

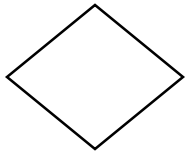
Modelo E/R - Notação



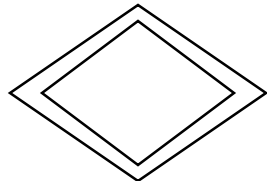
Entidade



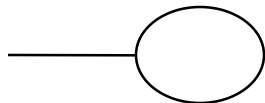
Entidade Fraca



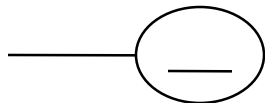
Relacionamento



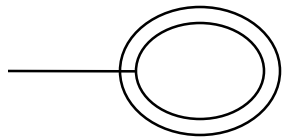
Relacionamento
Identificador



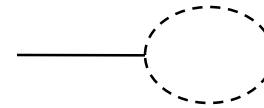
Atributo



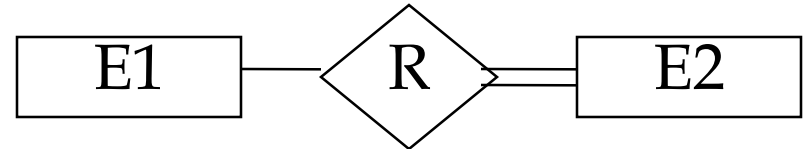
Atributo Chave



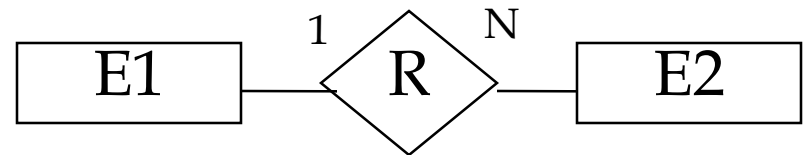
Multivalor



Atributo Derivado

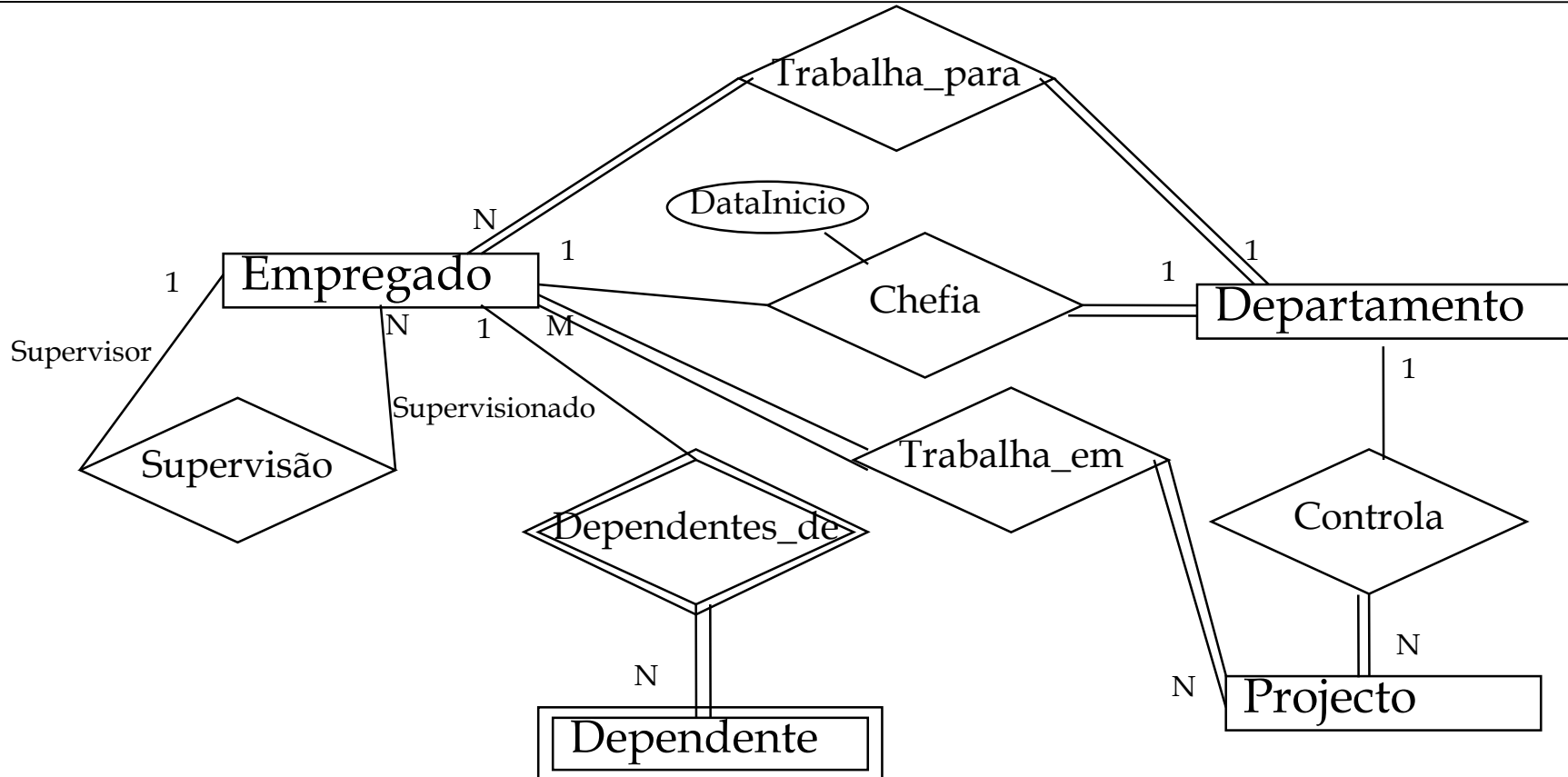


Participação total de E2 em R



Razão de cardinalidade 1:N para E1:E2 em R

Modelo E/R - Exemplo



DEPARTAMENTO(Numero, Nome, {Localidades}, NumEmp)

PROJECTO(Numero, ,Nome, Localidade)

EMPREGADO(Nss, Nome, Sexo, Morada, Salário, Data_Nascimento)

DEPENDENTE(Nome_Dependente, Sexo, Data_Nascimento, Parentesco)

Modelo Relacional

Relacionamento binário de grau 1:1 e participação obrigatória de ambas as entidades:	É apenas necessária uma tabela A chave primária dessa relação pode ser a chave primária de qualquer das entidades
Relacionamento binário 1:1 e participação obrigatória de apenas uma das entidades	São necessárias duas tabelas, uma para cada entidade; A chave primária de cada entidade serve de chave primária na tabela correspondente; A chave primária da entidade com participação não obrigatória tem que ser utilizada como atributo na tabela correspondente à entidade cuja participação é obrigatória
Relacionamento binário de grau 1:1 e participação não obrigatória de ambas as entidades:	São necessárias três tabelas, uma para cada entidade e a terceira para o relacionamento; A chave primária de cada entidade serve de chave primária na tabela correspondente; A tabela que corresponde ao relacionamento terá entre os seus atributos as chaves primárias das duas entidades
Relacionamento binário de grau 1:N e participação obrigatória do lado N	São necessárias duas tabelas, uma para cada entidade; A chave primária de cada entidade serve de chave primária na tabela correspondente; A chave primária da entidade do lado 1 tem que ser usada como atributo na tabela correspondente à entidade do lado N
Relacionamento binário de grau 1:N e participação não obrigatória do lado N	São necessárias três tabelas, uma para cada entidade e uma terceira para o relacionamento; A chave primária de cada entidade serve de chave primária na tabela correspondente; A tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada uma das entidades
Relacionamento binário de grau M:N	São sempre necessárias três tabelas, uma para cada entidade e uma para o relacionamento, independentemente do tipo de participação; A chave primária de cada entidade serve de chave primária na tabela correspondente; A tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada uma das entidades.
Relacionamento ternário e superior	São sempre necessárias quatro tabelas, uma para cada entidade e uma quarta para o relacionamento; A chave primária de cada entidade serve de chave primária na tabela correspondente; A tabela relativa ao relacionamento terá de ter entre os seus atributos as chaves primárias de cada uma das outras tabelas Num relacionamento de grau n são necessárias n+1 relações

Implementação SQL

Criação do Script SQL:

```
Create table socio (  
    Num_Socio int,  
    nome varchar(50),  
    local varchar(50),  
    data_nasc date,  
    Sexo varchar(50),  
    Primary key (num_socio))
```

ALTERNATIVA

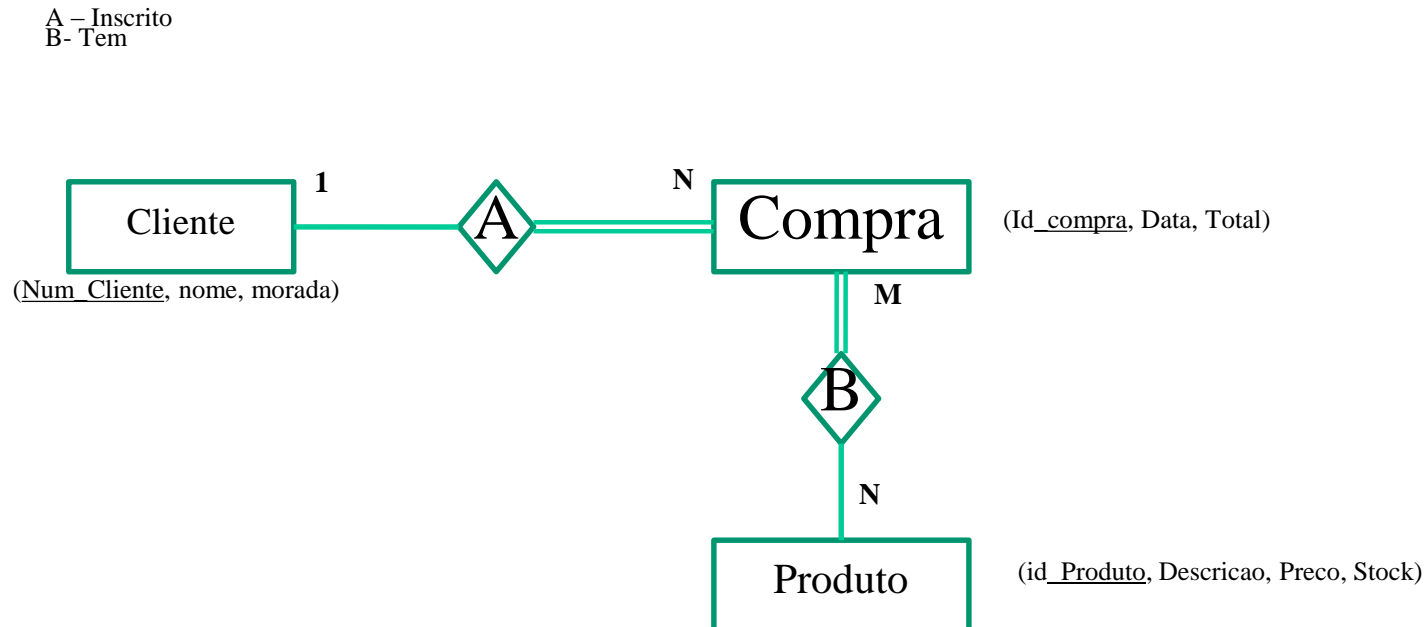
```
Create table socio (  
    Num_Socio int,  
    nome varchar(50),  
    local varchar(50),  
    data_nasc date,  
    Sexo varchar(50));
```

```
Alter table quota add constraint  
Tb_quota_pk primary key  
(AnoCivil)
```

Exercício

Criar um modelo de Entidade e Relacionamento
que dê suporte a um sistema de compras básico
(Cliente, Produtos e Compras)

Modelo E/R



Modelo Relacional

Cliente (Id_Cliente, nome, morada)

Produto (id_Produto, Descricao, Preco, Stock)

Compra (Id_compra, Data, Total, Id_Cliente(FK))

Linha_Compra (Id_Compra (FK), id_Produto
(FK), quantidade, preco)

Script SQL

```
Create table Cliente (  
  
  Id_Cliente int,  
  
  Nome varchar(50) not null,  
  
  Morada varchar(50),  
  
  Primary key(id_cliente));
```

Script SQL

Create table Produto ... (id_Produto, Descricao,
Preco, Stock)

Create table Compra ... (Id_compra, Data, Total,
Id_Cliente(FK))

Script SQL

```
Create table Linha_Compra (  
  
  Id_Compra int,  
  
  id_Produto int,  
  
  Quantidade int,  
  
  Preco decimal(4,2),  
  
  Primary key(id_produto, id_compra),  
  
  Foreign key (id_compra) references Compra(id_compra),  
  
  Foreign key (id_produto) references Produto(id_produto))
```


SQL

SQL (Structured Query Language)

A linguagem SQL é constituída por três sublinguagens:

DML - Data Manipulation Language (Select, Insert, Update, Delete,...)

DDL - Data Definition Language (Create, Alter, Drop,...)

DCL - Data Control Language (Grant, Revoke,...)

SQL – Comando Select

A sintaxe do comando SELECT é:

```
SELECT Campo1, Campo2, ..., Campo n, *  
FROM Tabela1, Tabela2, ..., Tabela k  
[WHERE Condição]  
[GROUP BY ...]  
[HAVING ...]  
[ORDER BY ...]
```

Nota: Os parêntesis rectos quando presentes na sintaxe de comandos, indicam que a componente é facultativa.

SQL - SELECT

- Seleccionando todos os registos:
Select Codigo, localidade
From postal

OU

Select *
From postal

SQL - SELECT

- A ordem da lista dos atributos pode ser redefinida pelo utilizador:

Select * from pessoa;

Igual a

**Select id, nome, IDADE, SaLaRiO, Telefone, Cod_Postal
from pessoa;**

ID	NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL
42	António Dias	43	74000	789654	1500
5	Célia Moraes	26	170000	123456	1100
32	Florinda Simões	35	147000		4000
37	Isabel Espada	28	86000		1100
49	José António	17	210000		1500
14	Nascimento Augusto	35	220000	456123	2300
25	Paulo Viegas	32	95000		1500

SQL - SELECT

Podemos alterar a ordem, repetir campos, etc.


**Select id, IDADE, SaLaRiO, Telefone, Cod_Postal, id,
nome from pessoa;**

ID	IDADE	SALARIO	TELEFONE	COD_POSTAL	ID	NOME
42	43	74000	789654	1500	42	António Dias
5	26	170000	123456	1100	5	Célia Morais
32	35	147000		4000	32	Florinda Simões
37	28	86000		1100	37	Isabel Espada
49	17	210000		1500	49	José António
14	35	220000	456123	2300	14	Nascimento Augusto
25	32	95000		1500	25	Paulo Viegas

SQL - WHERE

Quando se utiliza a clausula WHERE, só fazem parte do resultado, os tuplos que tornam a condição usada na clausula WHERE verdadeira.

Select * from pessoa where idade=35;



ID	NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL
32	Florinda Simões	35	147000		4000
14	Nascimento Augusto	35	220000	456123	2300

SQL - WHERE

Operadores a usar nas condições:
Operadores Relacionais

Operador	Descrição	Exemplo	Resultado
=	Igual a	$7 = 5$	Falso
>	Maior que	$7 > 5$	Verdadeiro
<	Menor que	$7 < 5$	Falso
>=	Maior ou igual que	$7 >= 5$	Verdadeiro
<=	Menor ou igual que	$7 <= 5$	Falso
<> Ou !=	Diferente	$7 <> 5$	Verdadeiro

SQL - WHERE

Select id, nome, salario, idade
From pessoa
Where idade >= 18;

ID	NOME	SALARIO	IDADE
42	António Dias	74000	43
5	Célia Morais	170000	26
32	Florinda Simões	147000	35
37	Isabel Espada	86000	28
14	Nascimento Augusto	220000	35
25	Paulo Viegas	95000	32

SQL - WHERE

Operadores a usar nas condições:
Operadores Lógicos

Operador	Exemplo
AND	Cond1 AND Cond2
OR	Cond1 OR Cond2
NOT	NOT Cond

SQL - WHERE

Select id, nome, idade, salario
From pessoa
Where idade \geq 30 **AND** idade \leq 40

ID	NOME	IDADE	SALARIO
32	Florinda Simões	35	147000
14	Nascimento Augusto	35	220000
25	Paulo Viegas	32	95000

SQL - WHERE

Select id, nome, idade, salario
From pessoa
Where idade < 30 **OR** idade > 40

Ou

Select id, nome, idade, salario
From pessoa
Where NOT (idade >= 30 **AND** idade <= 40)

ID	NOME	IDADE	SALARIO
42	António Dias	43	74000
5	Célia Morais	26	170000
37	Isabel Espada	28	86000
49	José António	17	210000

SQL - WHERE

CUIDADO COM AS PRECEDÊNCIAS!!!

Select id, nome, idade, salario
From pessoa
Where NOT (idade < 30 **AND** idade > 40)

ID	NOME	IDADE	SALARIO
42	António Dias	43	74000
5	Célia Morais	26	170000
32	Florinda Simões	35	147000
37	Isabel Espada	28	86000
49	José António	17	210000
14	Nascimento Augusto	35	220000
25	Paulo Viegas	32	95000

DIFERENTE

Select id, nome, idade, salario
From pessoa
Where NOT idade < 30 **AND** idade > 40

ID	NOME	IDADE	SALARIO
42	António Dias	43	74000

IGUAL

Select id, nome, idade, salario
From pessoa
Where (NOT (idade < 30)) **AND** idade > 40

SQL - BETWEEN

O operador BETWEEN permite especificar intervalos de valores.

Select ...

From ...

Where valor [NOT] BETWEEN valor1 AND valor2

Na realidade as condições seguintes são equivalentes:

valor >= valor1 AND valor <= valor2;

(valor >= valor1) AND (valor <= valor2);

valor BETWEEN valor1 and valor2

Da mesma forma também são equivalentes:

valor NOT BETWEEN valor1 AND valor2

NOT (valor BETWEEN valor1 AND valor2)

NOT (valor >= valor1 AND valor <= valor2)

NOT (valor >= valor1) OR NOT (valor <= valor2)

SQL - BETWEEN

Select id, nome, idade, salario
From pessoa
Where idade >= 30 AND idade <= 40;

Ou

Select id, nome, idade, salario
From pessoa
Where idade BETWEEN 30 AND 40;

ID	NOME	IDADE	SALARIO
32	Florinda Simões	35	147000
14	Nascimento Augusto	35	220000
25	Paulo Viegas	32	95000

SQL - BETWEEN

Select id, nome, idade, salario
From pessoa
Where idade<30 OR idade>40;

Ou

Select id, nome, idade, salario
From pessoa
Where NOT (idade>=30 AND idade<=40);

ID	NOME	IDADE	SALARIO
42	António Dias	43	74000
5	Célia Morais	26	170000
37	Isabel Espada	28	86000
49	José António	17	210000

SQL - BETWEEN

Select id, nome, idade, salario
From pessoa
Where idade NOT BETWEEN 30 AND 40;
Ou
Select id, nome, idade, salario
From pessoa
Where NOT (idade BETWEEN 30 AND 40);

ID	NOME	IDADE	SALARIO
42	António Dias	43	74000
5	Célia Morais	26	170000
37	Isabel Espada	28	86000
49	José António	17	210000

SQL - IN

O operador IN permite identificar se um dado elemento faz parte ou não de um conjunto de valores específico.

Sintaxe:

Select ...

from ...

where valor [NOT] IN (valor1,..., valork)

SQL - IN

Select *
from postal
where localidade='LISBOA' OR localidade='TOMAR'
OU
Select *
from postal
where localidade IN ('LISBOA','TOMAR');

CODIGO	LOCALIDADE
1000	LISBOA
1100	LISBOA
1200	LISBOA
1500	LISBOA
2300	TOMAR

SQL - IN

Select *
from postal
where localidade<>'LISBOA' AND localidade <>
'TOMAR'

OU
Select *
from postal
where localidade NOT IN ('LISBOA','TOMAR');

CODIGO	LOCALIDADE
2000	SANTAREM
3000	COIMBRA
4000	PORTO
9000	FUNCHAL

SQL - IS

Operador para tratamento de valores nulos (NULL –
Campos não preenchidos)

Sintaxe

Select ...

From ...

where campo IS [NOT] NULL

SQL - IS

Select nome, telefone
From pessoa

OU

NOME	TELEFONE
António Dias	789654
Célia Morais	123456
Florinda Simões	
Isabel Espada	
José António	
Nascimento Augusto	456123
Paulo Viegas	

NOME	TELEFONE
António Dias	789654
Célia Morais	123456
Florinda Simões	(NULL)
Isabel Espada	(NULL)
José António	(NULL)
Nascimento Augusto	456123
Paulo Viegas	(NULL)

SQL - IS

Select nome, telefone
From pessoa
Where telefone IS NULL

NOME	TELEFONE
Florinda Simões	
Isabel Espada	
José António	
Paulo Viegas	

SQL - IS

A possibilidade que os SGBD's têm de utilizar um valor que indique a própria ausência de valor é algo de enorme importância e que tem prós e contras.

Resumindo:

- As bases de relacionais possuem um indicador de inexistência de valor: NULL;

- O NULL não é zero, nem a string vazia;

- A comparação com NULL tem de ser sempre feita usando o operador IS;

- A comparação com valores NULL, sem o operador IS, devolve sempre FALSE.

SQL - LIKE

O operador LIKE permite resolver alguns problemas naturais que existem quando se pretendem comparar *strings*.

A utilização do operador LIKE, permite evitar comparações de partes da *string*. Para tal, utilizam-se dois *wildcards*.

Wildcard	Significado
%	Qualquer conjunto de zero ou mais caracteres
— (underscore)	Um caracter qualquer

SQL - LIKE

**Seleccionar todas as mensagens que
comecem pela letra “T”**

Select *

From mensagem

Where mensagem like 'T% '

ID_MSG	MENSAGEM
90	Telefonemas
80	Transportes

não foram seleccionadas linhas

**Seleccionar todas as mensagens que
terminem por “as”**

Select *

From mensagem

Where rtrim(mensagem) like '%as‘

ID_MSG	MENSAGEM
10	Comissão de Vendas
30	Fretes Empresas
90	Telefonemas
100	Ofertas

SQL - LIKE

Seleccionar todas os nomes que não tenham a *string* “da”

Select nome

From pessoa

Where nome NOT like ‘%da% ‘

ou

Select nome

From pessoa

Where NOT (nome like ‘%da% ‘)

NOME
António Dias
Célia Morais
José António
Nascimento Augusto
Paulo Viegas

Seleccionar todos os nome cujo segundo caracter é um “a”

Select nome

From pessoa

Where nome like ‘_a%‘

NOME
Nascimento Augusto
Paulo Viegas

SQL - LIKE

Procurar wildcards:

Select mensagem

From mensagem

Where mensagem like '%@%%' escape '@'

Insert into mensagem (id_msg, mensagem) values (200,'teste%teste')

Delete from mensagem where id_msg=200;

SQL - WildCards

Pesquisa	Descrição	Exemplo
'[abc]%'	Qualquer string começada obrigatoriamente por a, b ou c	Altivo, bola, cão
'[!abc]%'	Qualquer string não começada por a, b ou c	Dália, floral
'[0-9]%[abc]'	String começada por um dígito e terminada pelo carácter a, b ou c	999778a, 8b
'[a-zA-z]_'	String formada apenas por dois caracteres. O primeiro um carácter maiúsculo ou minúsculo. O segundo é um carácter qualquer.	Aa, a2, a_
'[!0-9]%'	O primeiro carácter não pode ser um dígito.	Abc, a778
'Al[m-r]%'	String começada por 'Al'. O terceiro entre os caracteres 'm' e 'r' (m,n,o,p,q,r). Depois pode ser qualquer. (Dimensão mínima 3)	Almodovar, Alma,

SQL – Precedências

Se numa operação aparecem vários operadores, alguns deles são executados antes de outros.

Ordem
de
execução



Parêntesis	()
Multiplicação/Divisão	* /
Adição/Subtracção	+ -
NOT	
AND	
OR	

SQL – Ordenação

Como ordenar a solicitação dos resultados fornecidos por um SELECT:

Sinaxe:

```
Select Campo1, Campo2, ... Campok, *  
From Tabela1, ..., tabelak  
[Where Condição]  
[Group by ...]  
[Having ....]  
[Order by Campo [ASC|DESC], Campo [ASC|DESC], ...]
```

ORDER BY:

CAMPO – representa o nome do “campo”, uma expressão ou a posição;

ASC – significa **ASC**ENDER;

DESC – significa **DESC**ENDER

SQL – Ordenação

Selecionar todas as pessoas, ordenando o resultado pela idade:

Select *
From pessoa
Order by Idade

OU

Select *
From pessoa
Order by 3

OU

Select *
From pessoa
Order by Idade ASC

ID	NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL
49	José António	17	210000		1500
5	Célia Morais	26	170000	123456	1100
37	Isabel Espada	28	86000		1100
25	Paulo Viegas	32	95000		1500
32	Florinda Simões	35	147000		4000
14	Nascimento Augusto	35	220000	456123	2300
42	António Dias	43	74000	789654	1500

SQL – Ordenação

Seleccionar o NOME e o SALARIO de todas as pessoas que têm telefone, ordenando o resultado pelo SALARIO, de tal forma que os maiores ordenados fiquem no topo da lista:

```
Select Nome, Salario  
From pessoa  
Where telefone is not NULL  
Order by Salario DESC
```

NOME	SALARIO
Nascimento Augusto	220000
Célia Morais	170000
António Dias	74000

SQL – Ordenação

Seleccionar o NOME e o SALARIO de todas as pessoas que têm telefone, ordenando o resultado pelo SALARIO, de tal forma que os maiores ordenados fiquem no topo da lista:

```
Select Nome, Salario  
From pessoa  
Where telefone is not NULL  
Order by Salario DESC
```

NOME	SALARIO
Nascimento Augusto	220000
Célia Morais	170000
António Dias	74000

SQL – Ordenação

Seelcionar todo o conteúdo da tabela COMISSAO,
ordenado por id e ID_MSG:

```
Select *  
from comissao  
Order by id, id_msg
```

ID	ID_MSG	VALOR
14	10	10500
14	60	2600
14	70	400
14	100	3750
25	10	2500
25	30	370
37	10	5500
37	30	14230
37	40	20
37	50	120
42	20	20
42	30	170
49	20	2300

SQL – Ordenação

Ordenações diferentes para diferentes campos:

Select *

From Comissao

Where id < 40

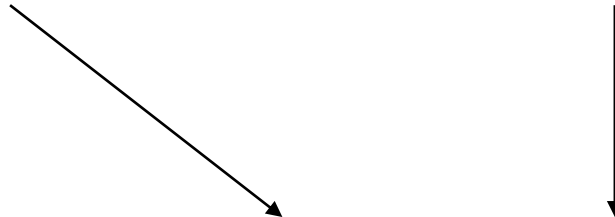
Order by id ASC, id_msg DESC

ID	ID_MSG	VALOR
14	100	3750
14	70	400
14	60	2600
14	10	10500
25	30	370
25	10	2500
37	50	120
37	40	20
37	30	14230
37	10	5500

SQL – Ordenação

Seleccção de expressões:

Select Nome, **idade** as **idade_agora**, **idade+1** as **Idade_mais_1**
From Pessoa
Order by nome



NOME	IDADE_AGORA	IDADE MAIS 1
António Dias	43	44
Célia Morais	26	27
Florinda Simões	35	36
Isabel Espada	28	29
José António	17	18
Nascimento Augusto	35	36
Paulo Viegas	32	33

SQL – Ordenação

Ordenação por posição:

Selecione, na tabela Comissão, o campo Valor a receber, o montante do imposto (21%) e o valor líquido para os indivíduos cujo id é igual a 14 e 25.

```
Select id, valor as Liquido, valor*0.21 as imposto,  
       valor+valor*0.21 as bruto
```

```
From comissao
```

```
where id in (14,25)
```

```
Order by id
```

ID	LIQUIDO	IMPOSTO	BRUTO
14	10500	2205	12705
14	2600	546	3146
14	400	84	484
14	3750	787,5	4537,5
25	2500	525	3025
25	370	77,7	447,7

SQL – Ordenação

Ordenar por outras colunas:

Select id, valor as Liquido, valor*0.21 as
imposto, valor+valor*0.21 as bruto

From comissao

where id in (14,25)

Order by id, valor*0.21

OU

Select id, valor as Liquido, valor*0.21 as
imposto, valor+valor*0.21 as bruto

From comissao

where id in (14,25)

Order by id, 3

OU

Select id, valor as Liquido, valor*0.21 as
imposto, valor+valor*0.21 as bruto

From comissao

where id in (14,25)

Order by 1, 3

ID	LIQUIDO	IMPOSTO	BRUTO
14	10500	2205	12705
14	2600	546	3146
14	400	84	484
14	3750	787,5	4537,5
25	2500	525	3025
25	370	77,7	447,7

SQL – Ordenação

Ordenação com NULL

A forma como o valor NULL é colocado no resultado depende de sistema para sistema.

Alguns consideram o valor NULL como o menor valor outros como o maior.

```
Select id, nome, telefone  
From pessoa  
Order by telefone
```

ID	NOME	TELEFONE
5	Célia Morais	123456
14	Nascimento Augusto	456123
42	António Dias	789654
32	Florinda Simões	
25	Paulo Viegas	
37	Isabel Espada	
49	José António	

SQL – Ordenação

Eliminação de repetições, **DISTINCT/ALL** (A clausula **ALL** está por defeito associada ao **SELECT**):

Select localidade
From postal

LOCALIDADE
LISBOA
LISBOA
LISBOA
LISBOA
SANTAREM
TOMAR
COIMBRA
PORTO
FUNCHAL

Select **DISTINCT** localidade
From postal

LOCALIDADE
COIMBRA
FUNCHAL
LISBOA
PORTO
SANTAREM
TOMAR

SQL – Juntando tabelas

Select *
From postal

CODIGO	LOCALIDADE
1000	LISBOA
1100	LISBOA
1200	LISBOA
1500	LISBOA
2000	SANTAREM
2300	TOMAR
3000	COIMBRA
4000	PORTO
9000	FUNCHAL

Select *
From pessoa

ID	NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL
42	António Dias	43	74000	789654	1500
5	Célia Morais	26	170000	123456	1100
32	Florinda Simões	35	147000		4000
37	Isabel Espada	28	86000		1100
49	José António	17	210000		1500
14	Nascimento Augusto	35	220000	456123	2300
25	Paulo Viegas	32	95000		1500

O que será:
Select *
From pessoa, postal ?

SQL – Juntando tabelas

!”#\$%#&/

Produto
Cruzado:

Select *
from postal
cross join
pessoa

NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL	CODIGO	LOCALIDADE
António Dias	43	74000	789654	1500	1000	LISBOA
Célia Morais	26	170000	123456	1100	1000	LISBOA
Florinda Simões	35	147000		4000	1000	LISBOA
Isabel Espada	28	86000		1100	1000	LISBOA
José António	17	210000		1500	1000	LISBOA
Nascimento Augusto	35	220000	456123	2300	1000	LISBOA
Paulo Viegas	32	95000		1500	1000	LISBOA
António Dias	43	74000	789654	1500	1100	LISBOA
Célia Morais	26	170000	123456	1100	1100	LISBOA
Florinda Simões	35	147000		4000	1100	LISBOA
Isabel Espada	28	86000		1100	1100	LISBOA
José António	17	210000		1500	1100	LISBOA
Nascimento Augusto	35	220000	456123	2300	1100	LISBOA
Paulo Viegas	32	95000		1500	1100	LISBOA
NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL	CODIGO	LOCALIDADE
António Dias	43	74000	789654	1500	1200	LISBOA
Célia Morais	26	170000	123456	1100	1200	LISBOA
...

SQL – Juntando tabelas

Devemos ter cuidado para não provocar excesso de carga de processamento.

O mais usual é o INNER JOIN (em particular NATURAL JOIN), em que é efectuada a comparação entre a chave primária de uma relação com a chave estrangeira de outra.

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

SQL – Juntando tabelas

Selecionar as localidades unicamente das pessoas que não têm telefone.

```
Select localidade  
from pessoa, postal  
where cod_postal = codigo  
AND telefone is NULL
```

LOCALIDADE
PORTO
LISBOA
LISBOA
LISBOA

SEM REPETIÇÕES

```
Select DISTINCT localidade  
from pessoa, postal  
where cod_postal = codigo  
AND telefone is NULL
```

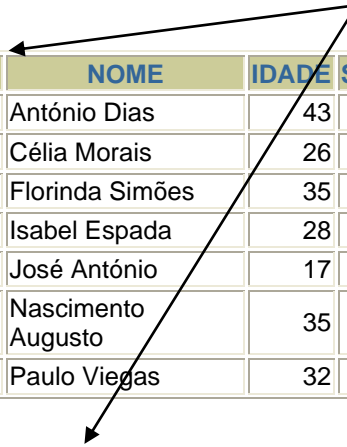
LOCALIDADE
LISBOA
PORTO

SQL – Juntando tabelas

Devemos ter atenção aos casos em que duas tabelas têm campos com o mesmo nome

Select *
From pessoa

Select *
From comissao



ID	NOME	IDADE	SALARIO	TELEFONE	COD_POSTAL
42	António Dias	43	74000	789654	1500
5	Célia Morais	26	170000	123456	1100
32	Florinda Simões	35	147000		4000
37	Isabel Espada	28	86000		1100
49	José António	17	210000		1500
14	Nascimento Augusto	35	220000	456123	2300
25	Paulo Viegas	32	95000		1500

ID	ID_MSG	VALOR
14	10	10500
25	10	2500
14	100	3750
14	70	400
37	40	20
37	30	14230
37	10	5500
14	60	2600
25	30	370
42	20	20
37	50	120
42	30	170
49	20	2300

SQL – Juntando tabelas

Se os campos de selecção ou junção de duas tabelas tiverem o mesmo nome, então para evitar ambiguidades, cada um dos campos deve ser precedido pelo **Nome da Tabela**, seguido de um **Ponto**.

```
Select pessoa.id, nome, valor  
from pessoa, comissao  
where pessoa.id = comissao.id
```

ID	NOME	VALOR
14	Nascimento Augusto	10500
25	Paulo Viegas	2500
14	Nascimento Augusto	3750
14	Nascimento Augusto	400
37	Isabel Espada	20
37	Isabel Espada	14230
37	Isabel Espada	5500
14	Nascimento Augusto	2600
25	Paulo Viegas	370
42	António Dias	20
37	Isabel Espada	120
42	António Dias	170
49	José António	2300

SQL – Juntando tabelas

O nome de uma tabela pode ser reduzido ou alterado num **SELECT**, através da utilização de um **ALIAS** que deve ser colocado à frente da designação real da tabela e que passará a ser um outra forma de identificar a tabela.

Select **P.id**, nome, valor
from pessoa P, comissao C
where **P.id = C.id**
Order by nome

ID	NOME	VALOR
42	António Dias	20
42	António Dias	170
37	Isabel Espada	20
37	Isabel Espada	14230
37	Isabel Espada	120
37	Isabel Espada	5500
49	José António	2300
14	Nascimento Augusto	10500
14	Nascimento Augusto	2600
14	Nascimento Augusto	3750
14	Nascimento Augusto	400
25	Paulo Viegas	2500
25	Paulo Viegas	370

SQL – Juntando tabelas

UNION – permite juntar o conteúdo de dois ou mais comandos **SELECT**

Numa **UNION**, o número de campos a seleccionar em cada um dos comandos do **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.

```
select id_msg, mensagem from mensagem  
UNION
```

```
select codigo, localidade from postal
```

ID_MSG	MENSAGEM
10	Comissão de Vendas
20	Fretes Individuais
30	Fretes Empresas
40	Vendas Extra
50	Deslocações
60	Refeições
70	Combustíveis
80	Transportes
90	Telefonemas
100	Ofertas
1000	LISBOA
1100	LISBOA
1200	LISBOA
1500	LISBOA
ID_MSG	MENSAGEM
2000	SANTAREM
2300	TOMAR
3000	COIMBRA
4000	PORTO
9000	FUNCHAL

SQL – Juntando tabelas

Numa UNION, o nome das colunas apresentado no resultado é o nome das colunas seleccionadas na primeira instrução SELECT.

Cada comando SELECT pode conter a sua própria cláusula WHERE. No entanto, só poderá existir uma única cláusula ORDER BY no ultimo SELECT, sendo a ordenação aplicada a todo o resultado.

SQL – Juntando tabelas

```
select id_msg, mensagem from mensagem  
Where id_msg <= 50  
UNION  
select codigo, localidade from postal  
Where localidade like '%AR%'  
Order by Mensagem
```

ID_MSG	MENSAGEM
10	Comissão de Vendas
50	Deslocações
30	Fretes Empresas
20	Fretes Individuais
2000	SANTAREM
2300	TOMAR
40	Vendas Extra

SQL – Juntando tabelas

Por defeito uma UNION remove sempre linhas duplicadas
Select id from pessoa select id from comissao

ID
42
5
32
37
49
14
25

ID
14
25
14
14
37
37
37
14
25
42
37
42
49

Select id from pessoa
UNION
Select id from comissao

ID
5
14
25
32
37
42
49

SQL – Juntando tabelas

Se em vez do UNION, usarmos o UNION ALL, os duplicados não são retirados da selecção.

Select id from pessoa
UNION ALL
Select id from comissao

ID
42
5
32
37
49
14
25
14
25
14
14
37
37
37
ID
14
25
42
37
42
49

SQL – Juntando tabelas

O operador **INTERSECT** permite juntar o resultado de dois comandos **SELECT** apresentando apenas as linhas que resultam de ambos os comandos.

```
Select * from postal where codigo <= 5000  
INTERSECT  
Select * from postal where codigo >= 3000
```

CODIGO	LOCALIDADE
3000	COIMBRA
4000	PORTO

SQL – Juntando tabelas

O operador MINUS devolve os registos que resultam do primeiro SELECT e que não aparecem no segundo.

Select * from postal where codigo <= 5000

MINUS

Select * from postal where codigo BETWEEN 2000 and 3000

CODIGO	LOCALIDADE
1000	LISBOA
1100	LISBOA
1200	LISBOA
1500	LISBOA
4000	PORTO

NOTA: Os operadores MINUS e EXCEPT são equivalentes em ORACLE.

SQL – Funções de Agregação

As funções de agragação são:

Função	Descrição
COUNT	Devolve o número de linhas
MAX	Devolve o maior valor da coluna
MIN	Devolve o menor valor da coluna
SUM	Devolve a soma de todos os valores da coluna
AVG	Devolve a media de todos os valores da coluna

SQL – Funções de Agregação

A função **COUNT**, devolve o número de linhas que resultam de um **SELECT**. Pode ser usada de três formas distintas

Função	Descrição
COUNT (*)	Devolve o número de linhas que resulta de um SELECT
COUNT (coluna)	Devolve o número de ocorrências na coluna diferentes de NULL
COUNT (DISTINCT coluna)	Devolve o número de ocorrências (sem repetições) na coluna.

SQL – Funções de Agregação

A contagem do número total de linhas de uma tabela faz-se utilizando COUNT(*)

```
select COUNT(*) as TOTAL  
from pessoa
```

TOTAL
7

Qual o número de pessoas e quantas é que têm telefone:

```
Select COUNT(*) as Tot1, COUNT(telefone) as Tot2  
from pessoa
```

TOT1	TOT2
7	3

SQL – Funções de Agregação

Quantas pessoas não têm telefone?

Select Count(*)

From Pessoa

Where telefone is NULL

COUNT(*)
4

Quantas comissões têm valor superior a 1000?

Select Count(*) as Quantas

From comissao

Where valor > 1000

QUANTAS
7

SQL – Funções de Agregação

Quantos id's existem na tabela comissão?

```
Select Count(id) as Total  
From comissao
```

TOTAL
13

Quantos id's distintos existem na tabela comissão?

```
Select Count(DISTINCT id) as Total  
From comissao
```

TOTAL
5

NOTA: A palavra reservada DISTINCT, dentro dos parêntesis de uma função de agregação, aplica essa função ignorando repetições.

SQL – Funções de Agregação

As funções MIN e MAX permitem obter o menor e o maior valor de uma determinada coluna.

Qual o valor do maior salário?

```
Select MAX(salario) as BIG_ONE  
From pessoa
```

BIG_ONE	
	220000

Qual a idade do empregado mais novo?

```
Select MIN(idade) as MIN_IDADE  
From pessoa
```

MIN_IDADE	
	17

SQL – Funções de Agregação

As funções MIN e MAX podem ser aplicadas a colunas que não contenham valores numéricos. No caso de serem aplicadas a campos do tipo *string* são devolvidos os Menores e Maior valores usando uma comparação alfabética.

```
select min(NOME) as PRIMEIRO  
from pessoa
```

PRIMEIRO
António Dias

SQL – Funções de Agregação

A função SUM devolve a soma de uma determinada coluna.

Qual o valor total de comissões a pagar?

```
select SUM(valor) as TOTAL_COMISSOES  
from comissao
```

TOTAL_COMISSOES
42480

SQL – Funções de Agregação

A função AVG devolve a media de uma determinada coluna.

Qual salario medio das pessoas com mais de 30 anos?

```
select AVG(salario) as Salario_Medio  
from pessoa  
Where idade > 30
```

SALARIO_MEDIO	
	134000

SQL – Funções de Agregação

As funções MIN, MAX, COUNT(...) e COUNT(*) podem ser utilizadas com qualquer tipo de dados, enquanto que as funções SUM e AVG apenas podem ser aplicadas a campos numéricos.

Se existir o valor NULL na coluna em que a função de agregação é aplicada, estes valores são ignorados.

SQL – Agrupando a informação

As funções de agregação são muito uteis 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, mas não como um todo, tal como anteriormente, mas em grupos mais pequenos

SQL – Agrupando a informação

Mostrar o valor das comissões existentes no sistema.

```
select id, valor  
from comissao  
order by id
```

ID	VALOR
14	10500
14	2600
14	400
14	3750
25	2500
25	370
37	5500
37	14230
37	20
37	120
42	20
42	170
49	2300

SQL – Agrupando a informação

Podemos saber o total geral da tabela:

```
Select SUM(valor) as TOTAL  
From comissao
```

TOTAL
42480

Mas o nosso objectivo consiste em obter o valor, não só total, mas a soma total das comissões de cada id. É aí que entra a clausula **GROUP BY**.

SQL – Agrupando a informação

As cláusulas GROUP BY e HAVING fazem parte do comando SELECT e aparecem na seguinte ordem:

```
select Campos  
from tabelas  
[where Condição]  
[GROUP BY ...]  
[HAVING ...]  
[ORDER BY ...]
```

SQL – Agrupando a informação

Vejamos a situação:

Mostrar o Valor das Comissões existentes?

```
Select id, valor  
From comissao  
Order by id
```

ID	VALOR
14	10500
14	2600
14	400
14	3750
25	2500
25	370
37	5500
37	14230
37	20
37	120
42	20
42	170
49	2300

Mostrar o total Valor das Comis

```
Select sum(valor) as Total  
From comissao
```

Mas e se quisermos saber o total para o **id:14**; **id: 25**; **id: 37**...???

TOTAL
42480

SQL – Agrupando a informação

Mostrar para cada id o total das comissões:

```
Select id, sum(Valor) as Total  
From Comissao  
group by id
```

ID	TOTAL
14	17250
25	2870
37	19870
42	190
49	2300

Repare-se que ao contrário dos casos anteriores em que, da utilização de funções de agregação resultava sempre uma única linha, através da cláusula **GROUP BY**, resulta o valor da agregação para cada um dos grupos.

SQL – Agrupando a informação

Façamos uma pequena variação ao exercício anterior, mostrando o nome em vez do id.

Neste caso, necessitamos da tabela de Comissões e da tabela das Pessoas. Nesta ultima, iremos obter o nome de cada uma das pessoas envolvidas.

```
Select Nome, Sum(valor) as total  
From pessoa p, comissao c  
Where p.id=c.id  
Group by nome
```

NOME	TOTAL
António Dias	190
Isabel Espada	19870
José António	2300
Nascimento Augusto	17250
Paulo Viegas	2870

SQL – Agrupando a informação

Mostrar o valor total (Salário + Comissões) a receber por cada pessoa.

```
Select Nome, sum(valor) + salario as total  
From pessoa p, comissao c  
Where p.id = c.id  
Group by nome, salario
```

NOME	TOTAL
António Dias	74190
Isabel Espada	105870
José António	212300
Nascimento Augusto	237250
Paulo Viegas	97870

No exemplo acima apresentado e Salário, uma vez que o Nome e Salário são as colunas seleccionadas e o Salário é utilizado no cálculo da expressão a apresentar.

SQL – Agrupando a informação

Seleccionar a maior comissão da cada id.

```
Select id, MAX(valor) as Maior  
From comissao  
Group by id
```

ID	MAIOR
14	10500
25	2500
37	14230
42	170
49	2300

Seleccionar o número de comissões para cada id.

```
Select id, COUNT(valor) as Num_Com  
From comissao  
Group by id
```

ou

```
Select id, COUNT(*) as Num_Com  
From comissao  
Group by id
```

ID	NUM_COM
14	4
25	2
37	4
42	2
49	1

NOTA: A função de agregação COUNT(*), se utilizada em conjunto com a clausula GROUP BY, devolve o número de ocorrências (linhas) dentro de cada grupo.

SQL – Agrupando a informação

Mostrar o valor e o número das comissões sobre ‘Vendas’ a receber por cada Pessoa, indicando também a Localidade onde reside.

Este é um exemplo interessante em que vamos utilizar a totalidade das tabelas envolvidas!!!!

- A tabela Pessoa é utilizada para obter o Nome;
- A tabela Postal é utilizada para obter a localidade onde reside;
- A tabela Comissão é utilizada para obter o cálculo das comissões;
- A tabela Mensagem é utilizada para pesquisar as Mensagens de Comissão que contenham a palavra ‘Vendas’;

SQL – Agrupando a informação

Mostrar o valor e o número das comissões sobre ‘Vendas’ a receber por cada Pessoa, indicando também a Localidade onde reside.

```
Select Nome, Localidade, sum(valor) as TOT, count(*) as NUM  
From pessoa p, comissao c, postal p, mensagem m  
Where p.id = c.id And p.cod_postal = p.codigo  
And c.id_msg = m.id_msg And m.mensagem like '%Vendas%'  
Group by nome, localidade
```

NOME	LOCALIDADE	TOT	NUM
Isabel Espada	LISBOA	5520	2
Nascimento Augusto	TOMAR	10500	1
Paulo Viegas	LISBOA	2500	1

SQL – Agrupando a informação

Mostrar o resultado a pagar para cada uma das Comissões.
O resultado deverá ser ordenado de forma descendente.

```
Select Mensagem, sum(valor) as TOT_COM  
From comissao c, mensagem m  
Where c.id_msg = m.id_msg  
Group by mensagem  
Order by sum(valor) desc
```

MENSAGEM	TOT_COM
Comissão de Vendas	18500
Fretes Empresas	14770
Ofertas	3750
Refeições	2600
Fretes Individuais	2320
Combustíveis	400
Deslocações	120
Vendas Extra	20

SQL – Agrupando a informação

Se um comando SELECT contiver a clausula GROUP BY, então todas as colunas seleccionadas (no SELECT) têm de estar presentes na clausula GROUP BY.

Características da clausula GROUP BY:

- A clausula GROUP BY é utilizada para agrupar informação;
- Os registos são processados em grupos de características semelhantes;
- As funções de Agregação (COUNT, MIN, MAX, etc,...) podem ser utilizadas para obter informações sobre cada grupo.

SQL – Agrupando a informação

A clausula **HAVING** serve para fazer restrições ao nível dos grupos que são processados

Mostrar o valor total das comissões agrupadas por id.

```
Select id, sum(valor) as TOT  
from comissao  
Group by id
```

ID	TOT
14	17250
25	2870
37	19870
42	190
49	2300

E se destes, só nos interessassem
2000?

es a

SQL – Agrupando a informação

Mostrar o valor total das comissões agrupadas por id, cujos totais sejam superiores a 2000.

```
Select id, sum(valor) as TOT  
from comissao  
Group by id  
having sum(valor) > 2000;
```

ID	TOT
14	17250
25	2870
37	19870
49	2300

A clausula HAVING, actua sobre o resultado dos grupos.

SQL – Agrupando a informação

WHERE vs. HAVING

Por vezes surge a dúvida sobre quando utilizar a cláusula Where ou a cláusula Having para restringir o conjunto de registos a apresentar.

Esta dúvida é simples de retirar.

Utiliza-se a cláusula Where sempre que se pretende restringir os registos a considerar na selecção. A cláusula Having serve para restringir os grupos que foram formados depois de aplicada a restrição da cláusula where.

SQL – Agrupando a informação

Se utilizar a cláusula Where num Select contendo Group By, o conjunto dos registos agrupados é apenas aquele que resulta da restrição imposta pela cláusula Where.

Saber qual o total das comissões (agrupado por id), considerando apenas aquelas de **valor** superior a 1000.

```
Select id, sum(valor)  
From comissao  
Where valor > 1000  
Group by id
```

ID	SUM(VALOR)
14	16850
25	2500
37	19730
49	2300

Não se podem utilizar funções de agregação na cláusula Where.
Utiliza-se a cláusula Having sempre que se pretende resumir o conjunto dos Grupos de Registos a considerar.

SQL – Agrupando a informação

Saber qual o total das comissões (agrupadas por id), considerando apenas aqueles cujo **valor total** seja superior a 1000

```
Select id, sum(valor)  
From comissao  
Group by id  
Having sum(valor) > 1000
```

ID	SUM(VALOR)
14	17250
25	2870
37	19870
49	2300

A clausula Where só pode ser aplicada a registos individuais, enquanto a clausula Having só pode ser aplicada a funções de grupo.

SQL – Agrupando a informação

Num select com a clausula Group By, se existirem Null's na coluna ou colunas de agrupamento, estes são também agrupados.

Seleccionar os vários grupos de telefone

```
select telefone, count(*)  
from pessoa  
group by telefone
```

TELEFONE	COUNT(*)
123456	1
456123	1
789654	1
	4

SQL – Agrupando a informação

E ainda a Ordenação:

Ordenação por uma Expressão ou função de agregação:

```
Select id, count(*)  
From comissao  
Group by id  
Order by count(id)
```

ID	COUNT(*)
49	1
25	2
42	2
14	4
37	4

Ordenação pela ordem

```
Select id, count(id)  
From comissao  
Group by id  
Order by 2 desc
```

ID	COUNT(ID)
14	4
37	4
25	2
42	2
49	1

Transacções e Recuperação

Unidade lógica de trabalho

- contém um ou mais comandos SQL para manipulação dos dados executados por um único utilizador

```
SET TRANSACTION
{ { READ ONLY | READ WRITE }
| ISOLATION LEVEL { SERIALIZABLE |
READ COMMITTED }
| USE ROLLBACK SEGMENT
  rollback_segment
}
| NAME 'nome_transacção'
};
```

Transacções e Recuperação

READ ONLY

- identifica uma transacção de leitura
- permite somente consultas (i.e., SELECT)
- especifica que as consultas da transacção somente acedem a alterações realizadas com sucesso (i.e., committed) antes do início da transacção

indicada para transacções que executem consultas em várias tabelas do BD ao mesmo tempo que estas tabelas estão sendo alteradas por outros utilizadores

Transacções e Recuperação

Exemplo

COMMIT;

SET TRANSACTION READ ONLY;

SELECT

COMMIT;

Transacções e Recuperação

READ WRITE

– identifica uma transacção de leitura e escrita

permite a especificação de

- consultas (i.e., SELECT)
- operações (i.e., INSERT, UPDATE, DELETE)

Commit, Rollback e Savepoint

COMMIT – torna os eventos de uma transacção permanentes;
SAVEPOINT – cria uma “marca” numa transacção;
ROLLBACK – apaga/recua eventos numa transacção.

Commit, Rollback e Savepoint

BEGIN

SAVEPOINT A1;

INSERT INTO marinheiro_fff (mid, mnome ,grau,idade) VALUES
(marinheiro_seq.nextval, 'Filipe', 10,22);

SAVEPOINT A2;

INSERT INTO marinheiro_fff (mid, mnome ,grau,idade) VALUES
(marinheiro_seq.nextval, 'Miguel', 10,22);

SAVEPOINT A3;

INSERT INTO marinheiro_fff (mid, mnome ,grau,idade) VALUES
(marinheiro_seq.nextval, 'Fidalgo', 10,22);

SAVEPOINT A4;

ROLLBACK TO A3;

END;

Commit, Rollback e Savepoint

A) Se executarmos o comando seguinte o que acontecerá? Porquê?

```
SELECT *  
FROM marinheiro_fff  
WHERE last_name = 'Fidalgo';
```

B) Executemos agora o seguinte comando:

```
ROLLBACK to SAVEPOINT A2;
```

O que acontece?

C) Nas presentes condições, se executarmos o comando seguinte o que acontecerá? Porquê?

```
SELECT mname  
FROM marinheiro_fff  
WHERE mname = 'Filipe';
```