A screenshot of a computer

AI-generated content may be incorrect.

CREATE DATABASE PJD-AD02;

USE PJD-AD02;

CREATE TABLE PedroCaladoDepartamento ( IdDepartamento INT(11) PRIMARY KEY AUTO\_INCREMENT, Descricao VARCHAR(20) NOT NULL );

CREATE TABLE PedroCaladoFuncionario (

IdFuncionario INT(11) PRIMARY KEY AUTO\_INCREMENT,

Departamento INT(11) NOT NULL,

Nome VARCHAR(30) NOT NULL,

DataAdmissao DATE NOT NULL,

Vencimento DECIMAL(12,2) NOT NULL,

Dependentes TINYINT(1) NOT NULL,

FOREIGN KEY (Departamento) REFERENCES PedroCaladoDepartamento(IdDepartamento)

);

INSERT INTO PedroCaladoDepartamento (IdDepartamento, Descricao) VALUES

(1, 'Produção'),

(2, 'Contabilidade'),

(3, 'Marketing'),

(4, 'Financeiro'),

(5, 'Recursos Humanos');

INSERT INTO PedroCaladoFuncionario (IdFuncionario, Departamento, Nome, DataAdmissao, Vencimento, Dependentes) VALUES

(1, 1, 'João Silva', '2024-01-10', 2500.00, 2),

(2, 2, 'Ana Costa', '2023-06-15', 3200.00, 1),

(3, 3, 'Carlos Mendes', '2022-09-20', 2800.00, 0),

(4, 4, 'Mariana Lopes', '2021-03-05', 4000.00, 2),

(5, 5, 'Pedro Nunes', '2020-07-22', 3500.00, 2);

DELETE FROM PedroCaladoFuncionario;

DELETE FROM PedroCaladoDepartamento;

SHOW CREATE TABLE PedroCaladoFuncionario;

SHOW CREATE TABLE PedroCaladoDepartamento;

SELECT

TABLE\_NAME, COLUMN\_NAME, CONSTRAINT\_NAME, REFERENCED\_TABLE\_NAME, REFERENCED\_COLUMN\_NAME

FROM

INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE

WHERE

TABLE\_NAME = 'PedroCaladoFuncionario' AND CONSTRAINT\_NAME <> 'PRIMARY';

SELECT

f.IdFuncionario, f.Nome, f.DataAdmissao, f.Vencimento, f.Dependentes,

d.Descricao AS Departamento

FROM PedroCaladoFuncionario f

JOIN PedroCaladoDepartamento d ON f.Departamento = d.IdDepartamento;

SELECT \* FROM PedroCaladoDepartamento;

SELECT \* FROM PedroCaladoDepartamento ORDER BY Descricao;

SELECT \* FROM PedroCaladoDepartamento WHERE Descricao = 'Marketing';

ALTER TABLE PedroCaladoDepartamento CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

ALTER TABLE PedroCaladoFuncionario CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

ALTER DATABASE nome\_da\_base CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

SHOW CREATE TABLE PedroCaladoDepartamento; SHOW CREATE TABLE PedroCaladoFuncionario;

SELECT schema\_name, default\_character\_set\_name, default\_collation\_name

FROM information\_schema.schemata

WHERE schema\_name = 'nome\_da\_tua\_base\_de\_dados';

-- Criar a base de dados e inserir os dados:

-- Exercício Pessoas

-- Criar a base de dados e selecioná-la

-- drop database pessoas

-- drop database if exists pessoas;

CREATE DATABASE IF NOT EXISTS pessoas;

USE pessoas;

-- Apagar as tabelas

DROP TABLE if exists Pessoa;

DROP TABLE if exists Postal;

DROP TABLE if exists Mensagem;

DROP TABLE if exists Comissao;

CREATE TABLE Postal(

CODIGO INTEGER,

LOCALIDADE CHAR(30) NOT NULL,

CONSTRAINT Postal\_PK PRIMARY KEY (CODIGO)

);

INSERT INTO Postal VALUES(1000,'LISBOA');

INSERT INTO Postal VALUES(1100,'LISBOA');

INSERT INTO Postal VALUES(1200,'LISBOA');

INSERT INTO Postal VALUES(1500,'LISBOA');

INSERT INTO Postal VALUES(2000,'SANTAREM');

INSERT INTO Postal VALUES(2300,'TOMAR');

INSERT INTO Postal VALUES(3000,'COIMBRA');

INSERT INTO Postal VALUES(4000,'PORTO');

INSERT INTO Postal VALUES(9000,'FUNCHAL');

-- ---------------------------------------------------------

-- Criação da Tabela Mensagem e Introdução dos Dados

-- ---------------------------------------------------------

CREATE TABLE Mensagem(

Id\_Msg integer PRIMARY KEY,

Mensagem CHAR(30) NOT NULL

);

CREATE UNIQUE INDEX iMsg ON Mensagem(Mensagem);

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(10,'Comissão de Vendas');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(20,'Fretes Individuais');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(30,'Fretes Empresas');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(40,'Vendas Extra');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(50,'Deslocações');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(60,'Refeições');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(70,'Combustíveis');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(80,'Transportes');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(90,'Telefonemas');

INSERT INTO Mensagem(Id\_Msg, Mensagem) VALUES(100,'Ofertas');

-- ---------------------------------------------------------

-- Criação da Tabela Comissao e Introdução dos Dados

-- ---------------------------------------------------------

CREATE TABLE Comissao(

Id INTEGER NOT NULL,

Id\_Msg integer NOT NULL,

Valor NUMERIC NOT NULL

);

CREATE INDEX iComissao ON Comissao(Id,Id\_Msg);

INSERT INTO Comissao VALUES(14,10,10500);

INSERT INTO Comissao VALUES(25,10,2500);

INSERT INTO Comissao VALUES(14,100,3750);

INSERT INTO Comissao VALUES(14,70,400);

INSERT INTO Comissao VALUES(37,40,20);

INSERT INTO Comissao VALUES(37,30,14230);

INSERT INTO Comissao VALUES(37,10,5500);

INSERT INTO Comissao VALUES(14,60,2600);

INSERT INTO Comissao VALUES(25,30,370);

INSERT INTO Comissao VALUES(42,20,20);

INSERT INTO Comissao VALUES(37,50,120);

INSERT INTO Comissao VALUES(42,30,170);

INSERT INTO Comissao VALUES(49,20,2300);

-- ---------------------------------------------------------

-- Criação da Tabela Pessoa e Introdução dos Dados

-- ---------------------------------------------------------

CREATE TABLE Pessoa(

PessoaID integer,

NOME varchar(30) NOT NULL,

IDADE integer NOT NULL,

SALARIO numeric(10,2) NOT NULL,

TELEFONE VARCHAR(25) NULL,

COD\_Postal integer,

CONSTRAINT Pessoa\_PK PRIMARY KEY (PessoaID),

CONSTRAINT Postal\_FK FOREIGN KEY (COD\_Postal) REFERENCES Postal(codigo)

);

CREATE UNIQUE INDEX iNome ON Pessoa(Nome);

CREATE INDEX iTelefone ON Pessoa(Telefone);

INSERT INTO Pessoa VALUES(42,'António Dias',43,74000,'789654',1500);

INSERT INTO Pessoa VALUES(5,'Célia Morais',26,170000,'123456',1100);

INSERT INTO Pessoa VALUES(32,'Florinda Simões',35,147000,NULL,4000);

INSERT INTO Pessoa VALUES(37,'Isabel Espada',28,86000,NULL,1100);

INSERT INTO Pessoa VALUES(49,'José António',17,210000,NULL,1500);

INSERT INTO Pessoa VALUES(14,'Nascimento Augusto',35,220000,'456123',2300);

INSERT INTO Pessoa VALUES(25,'Paulo Viegas',32,95000,NULL,1500);

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \* FROM PESSOA;

SELECT

p.PessoaID,

p.NOME,

p.IDADE,

p.SALARIO,

p.TELEFONE,

po.LOCALIDADE

FROM PESSOA p

LEFT JOIN POSTAL po ON p.COD\_POSTAL = po.CODIGO;

SELECT \* FROM PESSOA

WHERE SALARIO > 100000;

SELECT Codigo, Localidade FROM Postal

A screenshot of a computer

AI-generated content may be incorrect.

SELECT distinct Localidade FROM Postal

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

-- SELECT \* FROM PESSOA; -- todas as colunas e todas as linhas da tabela pessoa

-- SELECT \* FROM postal;

-- SELECT \* FROM mensagem;

-- SELECT \* FROM comissao;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT LOCALIDADE FROM postal;

A screenshot of a computer

AI-generated content may be incorrect.

Select PessoaID

from Pessoa

A screenshot of a computer

AI-generated content may be incorrect.

Select pessoa.pessoaID, pessoa.NOME, pessoa.SALARIO

from Pessoa

select pessoaID, NOME, IDADE, SALARIO as "Salário", TELEFONE, COD\_Postal

from Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Pessoa

WHERE idade = 35

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, NOME, SALARIO

from Pessoa

where idade=35;

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, NOME, SALARIO

from Pessoa

where idade >= 18;

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, IDADE, NOME, SALARIO

from Pessoa

where idade >= 30 AND idade <= 40;

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, IDADE, NOME, SALARIO

from Pessoa

where idade < 30 OR idade > 40;

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, NOME, IDADE

FROM Pessoa

where NOT (idade >= 30 AND idade <= 40);

A screenshot of a computer

AI-generated content may be incorrect.

select pessoaID, NOME, IDADE

FROM Pessoa

WHERE NOT idade >= 30 AND idade <= 40

A screenshot of a computer

AI-generated content may be incorrect.

**Outros Operadores**

BETWEEN – permite especificar intervalos de valores.

IN – permite especificar conjuntos de valores.

IS – permite efectuar o tratamento de nulos (NULL) – strings vazias.

LIKE – permite resolver alguns problemas naturais que existem quando se pretende comparar strings.

**Operador BETWEEN**

Este operador permite especificar intervalos de valores.

SELECT campo1, campo2, ... campon FROM tabela 1 , tabela 2, .... tabela n

WHERE valor [NOT] **BETWEEN** valor1 **AND** valor2

A screenshot of a computer

AI-generated content may be incorrect.

SELECT PessoaId, Nome, Idade, Salario

FROM Pessoa

WHERE idade BETWEEN 30 AND 40

SELECT Id, Nome, Idade, Salário

FROM Pessoa

WHERE **NOT** idade **BETWEEN** 30 **AND** 40

A screenshot of a computer

AI-generated content may be incorrect.

SELECT PessoaId, Nome, Idade, Salario

FROM Pessoa

WHERE NOT idade BETWEEN 30 AND 40

A screenshot of a computer

AI-generated content may be incorrect.

**Operador IN**

Este operador permite especificar conjuntos de valores. Exercícios SQL

P á g i n a 9 | 40

SELECT campo1, campo2, ... campon

FROM tabela 1 , tabela 2, .... tabela n

WHERE valor [NOT] **IN** (valor1, valor2, valorn)

SELECT \*

FROM Postal

WHERE Localidade IN ('LISBOA', 'RIO MAIOR')

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Postal

WHERE Localidade NOT IN ('LISBOA','RIO MAIOR')

A screenshot of a computer

AI-generated content may be incorrect.

**Operador IS**

Este operador permite efectuar o tratamento de nulos (NULL) – strings vazias. NULL não é zero, é string vazia. A comparação com NULL terá de ser efectuada sempre com o operador IS, caso contrário devolve sempre Falso.

SELECT campo1, campo2, ... campon FROM tabela 1 , tabela 2, .... tabela n WHERE valor **IS** [NOT] **NULL**

SELECT Nome

FROM Pessoa

WHERE Telefone **IS NULL**

A screenshot of a computer

AI-generated content may be incorrect.

SELECT PessoaId, Nome, Idade, Salario, Telefone, Cod\_postal

FROM Pessoa

WHERE Telefone IS NOT NULL

A screenshot of a computer

AI-generated content may be incorrect.

**Operador LIKE**

Este operador permite resolver alguns problemas naturais que existem quando se pretende comparar strings.

|  |  |
| --- | --- |
| A utilização do operador LIKE permite fazer comparações de partes da string. Para tal, utilizam-se dois WILDCARDS. WILDCARD | SIGNIFICADO |
| \* | Qualquer string de zero ou mais caracteres |
| ? | Um carácter qualquer |

SELECT \*

FROM Mensagem

WHERE Mensagem LIKE 'T%'

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Mensagem

WHERE Mensagem LIKE '%as'

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Mensagem

WHERE Mensagem LIKE '%Vendas%'

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Mensagem

WHERE Mensagem **LIKE '\_e%'**

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedência dos operadores** ORDEM | OPERADOR | | SÍMBOLO |
| 1ª | Parêntesis | | ( ) |
| 2ª | Multiplicação / Divisão | | \* / |
| 3ª | Adição /Subtracção | | + - |
| 4ª | | NOT | |
| 5ª | | AND | |
| 6ª | | OR | |

SELECT PessoaId, Nome, Idade

FROM Pessoa

WHERE Idade > 29 AND Telefone IS NULL OR Idade < 28

A screenshot of a computer

AI-generated content may be incorrect.

Não é equivalente a:

SELECT PessoaId, Nome, Idade FROM Pessoa

WHERE (Idade < 28 **OR**  Idade > 29) **AND**  Telefone **IS NULL**

A screenshot of a computer

AI-generated content may be incorrect.

Mas o anterior é equivalente a:

SELECT PessoaId, Nome, Idade

FROM Pessoa

WHERE Idade <= 27 OR (Idade >= 30 AND Telefone IS NULL)

A screenshot of a computer

AI-generated content may be incorrect.

Pretende-se seleccionar, da tabela Pessoa, o Id, Nome e Idade das pessoas com idades maiores que os 29 anos e as pessoas com idades inferiores a 28 anos, que não possuam telefone. Exercícios SQL

P á g i n a 13 | 40

SELECT PessoaId, Nome, Idade

FROM Pessoa

WHERE (Idade < 28 **OR** Idade > 29) **AND** Telefone **IS NULL**

A screenshot of a computer

AI-generated content may be incorrect.

**2. – Ordenação**

A ordenação pode ser realizada através da cláusula ORDER BY no comando SELECT. Esta cláusula, se existir, aparece sempre posicionada no final do comando SELECT.

SELECT campo1, campo2, ... campon

FROM tabela 1 , tabela 2, .... tabela n

[WHERE Condição ]

[GROUP BY ............]

[HAVING ………]

[**ORDER BY Campo [ASC | DESC], Campo [ASC | DESC]…**]

Onde **Campo** representa o nome de um Campo, uma Expressão ou a Posição pela qual se pretende ordenar o resultado do SELECT.

**ASC** indica que a ordenação é **ASC**endente e **DESC** indica que a ordenação é

**DESCendente.**

Por defeito, ou seja, caso não seja indicado um tipo de ordenação, esta é efectuada por ordem ascendente.

**1. Ordenando por uma coluna**

Considerando a tabela Pessoa, pretende-se ordenar os dados por idade.

SELECT \*

FROM Pessoa

ORDER BY Idade

A screenshot of a computer

AI-generated content may be incorrect.

**2. Ordenando por várias colunas**

Consideremos a tabela Comissões, com os atributos Id\_comissão, Id\_mensagem e valor. Pretende-se ordenar os dados da tabela comissão por Id\_comissão e por Id\_ mensagem.

SELECT \*

FROM Comissao

ORDER BY Id, Id\_Msg

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Comissao

WHERE Id < 30

ORDER BY Id ASC, Valor DESC

A screenshot of a computer

AI-generated content may be incorrect.

**3. Selecção de expressões**

Consideremos a tabela Pessoa. Pretende-se o nome e idade de todas as pessoas, seleccionando a idade que irão ter daqui a 3 anos, ordenando os dados de saída por nome.

SELECT Nome, Idade, Idade + 3

FROM Pessoa

ORDER BY Nome

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |  |
| --- | --- | --- |
| Depois de executado o comando, o resultado obtido será: Nome | Idade | Expr1002 |
| António Dias | 43 | 46 |
| Célia Morais | 36 | 39 |
| Florinda Simões | 35 | 38 |
| Isabel Espada | 28 | 31 |
| José António | 17 | 20 |

O resultado do cálculo é atribuído a uma Expr 1002, sendo que esta varia de sistema para sistema e é atribuída automaticamente.

Pretende-se o nome e idade de todas as pessoas, seleccionando a idade que irão ter daqui a 3 anos, ordenando os dados de saída por nome, e atribuíndo a idade actual a (Idade\_actual) e a idade daqui a 3 anos a (Idade\_em\_2026).

SELECT Nome, Idade AS Idade\_actual, Idade+3 AS Idade\_em\_2026

FROM Pessoa

ORDER BY Nome

A screenshot of a computer

AI-generated content may be incorrect.

**4. Ordenação e NULLs**

A forma como o NULL é colocado no resultado ordenado de um SELECT depende de sistema para sistema.

Alguns sistemas consideram o valor NULL menor que qualquer outro valor. Outros colocam o valor NULL sempre no topo dos valores, seja a ordenação ascendente ou descendente.

**5. Eliminação de repetições (DISTINCT e ALL)**

A cláusula DISTINCT colocada a seguir ao SELECT, permitir retirar os dados repetidos.

A cláusula ALL colocada a seguir ao SELECT, mostra todos os conjuntos de valores existentes, repetidos ou não.

Considerando uma tabela Postal, com os atributos Código e Local, pretende-se seleccionar todos os Locais da tabela Postal.

|  |  |
| --- | --- |
| SELECT ALL Local  FROM Postal | SELECT DISTINCT Local  FROM Postal |

SELECT ALL Localidade

FROM Postal

A screenshot of a computer

AI-generated content may be incorrect.

SELECT DISTINCT Localidade

FROM Postal

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| Depois de executados os comandos, os resultados obtidos serão: Local | Local |
| LISBOA | LISBOA |
| LISBOA | SANTARÉM |
| LISBOA | RIO MAIOR |
| SANTARÉM | TOMAR |
| RIO MAIOR | PORTO |
| TOMAR | |
| PORTO | |

Notas: As cláusulas DISTINCT e ALL só podem ser colocadas imediatamente a seguir ao SELECT.

**Sintaxes inválidas:**

SELECT Id, **DISTINCT** Id\_mensagem

A screenshot of a computer

AI-generated content may be incorrect.

SELECT DISTINCT PessoaId, ALL Id\_mensagem

A screenshot of a computer

AI-generated content may be incorrect.

**III – Funções de agregação**

Estas funções, também designadas por **Funções Estatísticas**, têm por objectivo obter informação sobre conjuntos de linhas especificadas na cláusula WHERE ou sobre grupos de linhas indicadas na cláusula GROUP BY.

|  |  |
| --- | --- |
| **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 média (**AV**era**G**e) de todos os valores da coluna. |

**FUNÇÃO COUNT**

|  |  |
| --- | --- |
| A função de agregação COUNT devolve o número de linhas que resultam num SELECT. Pode ser utilizada de três formas distintas: **Forma** | **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. |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Consideremos a seguinte relação Pessoa: **Id** | **Nome** | | **Idade** | | **Salário** | | **Telefone** | | **Cod\_postal** |
| 42 | António Dias | | 43 | | 94000 | | 7896544 | | 1000 |
| 5 | Célia Morais | | 36 | | 170000 | | 1234565 | | 1000 |
| 32 | | Florinda Simões | | 35 | | 147000 | | 4000 | |
| 37 | | Isabel Espada | | 28 | | 86000 | | 2040 | |
| 49 | | José António | | 17 | | 210000 | | 1000 | |
| 14 | Nascimento Augusto | | 35 | | 220000 | | 4561233 | | 2300 |
| 75 | | Pedro Santos | | 42 | | 235000 | | 2040 | |

3. Pretende-se saber quantas pessoas existem na tabela Pessoa.

SELECT COUNT(\*) AS Total

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

4. Pretende-se saber quantas pessoas existem na tabela Pessoa e quantas pessoas possuem telefone.

SELECT COUNT(\*) AS `Total de pessoas`,

COUNT(Telefone) AS `Total de telefones`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

5. Pretende-se saber quantos Códigos postais diferentes existem na tabela Pessoa.

SELECT COUNT(DISTINCT Cod\_postal) AS `Cód. postais diferentes`

FROM Pessoa;

A blue and black text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Slide 22

**MIN e MAX**

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

6 - Pretende-se saber qual o salário mais elevado e o salário mais baixo da tabela Pessoa.

SELECT MAX(salario) AS "Salário mais elevado",

MIN(salario) AS "Salário mais baixo"

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

7 - Pretende-se saber a idade da pessoa mais nova e a idade da pessoa mais velha da tabela Pessoa.

SELECT MAX(IDADE) AS `Maior Idade`,

MIN(IDADE) AS `Menor Idade`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

8 - Pretende-se saber o nome da primeira e da última pessoa da tabela Pessoa, se a lista fosse ordenada alfabeticamente.

SELECT MIN(nome) AS 'Primeiro',

MAX(nome) AS 'Último'

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

Slide 24

**SUM**

A função SUM devolve a soma de uma determinada coluna.

9 – Pretende-se saber o total de salários da tabela Pessoa.

SELECT SUM(salario) AS `Total de Salários`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

10 - Pretende-se saber o total das idades da tabela Pessoa.

SELECT SUM(idade) AS `Total das idades`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

11 - Pretende-se saber o total de salários da tabela Pessoa, assim como o total dos salários do próximo mês, sabendo que os salários serão aumentados em 3%.

SELECT SUM(salario) AS `Total de Salários`,

SUM(salario) \* 1.03 AS `Total de Salários com aumento`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

**AVG**

A função AVG devolve a média dos valores de uma determinada coluna.

12 - Pretende-se saber a média das idades da tabela Pessoa.

SELECT AVG(idade) AS `Média de Idades`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

Existem funções que permitem formatar o resultado do SELECT. Embora essas funções dependam de cada um dos sistemas, no Access a função FORMAT permite indicar sobre quantas casas decimais se pretende visualizar o resultado.

13 - Pretende-se saber a média das idades da tabela Pessoa, formatada a duas casas decimais.

SELECT FORMAT(AVG(idade), 2) AS `Média de Idades`

FROM Pessoa;

A screenshot of a computer

AI-generated content may be incorrect.

**NOTA FINAL:**

As funções MIN, MAX e COUNT podem ser utilizadas com qualquer tipo de dados. As funções SUM e AVG só podem ser aplicadas a campos numéricos. Se existirem valores NULL estes são ignorados por qualquer uma das funções.

**V – Agrupando a informação**

As funções de agregação são uma ferramenta útil quando usada para obter informação resumida sobre o resultado de um comando SELECT.

No entanto, estas funções podem ser particularmente úteis no tratamento de informação de forma agrupada, não como um todo, mas em grupos mais pequenos.

14 - Pretende-se mostrar as comissões e respectivos valores, ordenando o resultado por Id da tabela comissão.

SELECT Id, Valor FROM Comissao

ORDER BY Id

A screenshot of a computer

AI-generated content may be incorrect.

15 - Pretende-se saber o total de valores de comissões.

SELECT SUM(Valor) AS Total

FROM Comissao

A screenshot of a computer

AI-generated content may be incorrect.

* 1. A cláusula GROUP BY é utilizada para agrupar informação.
  2. Os registos são processados em grupos de características semelhantes.
  3. As funções de agregação podem ser utilizadas para obter informação sobre cada um dos grupos.

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

SELECT campo1, campo2, ... campon FROM tabela 1 , tabela 2, .... tabela n [WHERE Condição ]

**[GROUP BY ............] [HAVING ………]** [ORDER BY ........]

**Cláusula GROUP BY**

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

16 - Pretende-se saber, para cada Id, o total de valores de comissões.

SELECT Id, SUM(Valor) AS Total

FROM Comissao

GROUP BY Id

A screenshot of a computer

AI-generated content may be incorrect.

17 - Pretende-se saber, para cada Id, o maior valor de comissão.

SELECT `Id`, MAX(`Valor`) AS `Maior`

FROM `Comissao`

GROUP BY `Id`;

A screenshot of a computer

AI-generated content may be incorrect.

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

18 - Pretende-se saber para cada Id, o maior valor de comissão, efectuando a ordenação descendente por maior valor de comissão.

SELECT MAX(`Valor`) AS `Maior`, `Id`

FROM `Comissao`

GROUP BY `Id`

ORDER BY MAX(`Valor`) DESC;

A screenshot of a computer

AI-generated content may be incorrect.

**Cláusula HAVING**

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

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

SELECT Id, SUM(Valor) AS Total FROM Comissao

GROUP BY Id HAVING SUM(Valor) > 1000

A screenshot of a computer

AI-generated content may be incorrect.

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

SELECT Id, MAX(Valor) AS Maior FROM Comissao

GROUP BY Id HAVING MAX(Valor) < 3000

A screenshot of a computer

AI-generated content may be incorrect.

**WHERE vs HAVING**

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.

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

SELECT Id, SUM(Valor) AS Total

FROM Comissao

WHERE Valor > 1000

GROUP BY Id;

A screenshot of a computer

AI-generated content may be incorrect.

22 - Pretende-se saber o total das comissões para cada Id, considerando apenas aquelas cujo valor total seja superior a 1000.

SELECT `Id`, SUM(`Valor`) AS `Total`

FROM `Comissao`

GROUP BY `Id`

HAVING SUM(`Valor`) > 1000;

A screenshot of a computer

AI-generated content may be incorrect.

**Nota Final:**

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

Slide 31 –

**VI – Juntando várias tabelas**

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

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

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

SELECT campo1, campo2, ... campon \*

FROM tabela 1 , tabela 2, .... tabela n

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Consideremos as seguintes relações (Pessoa e Postal): Id | Nome | | Idade | | Telefone | | Cod\_postal | | Código | | Local |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 1000 | | Lisboa |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 2000 | | Santarém |
| 12 | | Isabel Silva | | 28 | | 2040 | | 2040 | | Rio Maior | |
| 49 | José António | | 17 | | 333555 | | 2000 | | 4000 | | Porto |

**Produto Cartesiano de Tabelas**

SELECT \*

FROM Pessoa, Postal

Obtém-se um resultado bastante inesperado e que consiste no produto cartesiano de dois conjuntos de elementos.

O produto cartesiano entre as tabelas Pessoa e Postal associa a cada linha da tabela Pessoa o conjunto das linhas da tabela Postal.

Dado que na tabela Pessoa existem 4 registos e na tabela Postal existem 4 registos, o resultado do produto cartesiano será 4\*4 = 16 registos.

Slide 32

**Produto Cartesiano de Tabelas**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| O resultado do produto cartesiano será: Id | Nome | | Idade | | Telefone | | Cod\_postal | | Código | | Local |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 1000 | | Lisboa |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 2000 | | Santarém |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 2040 | | Rio Maior |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 4000 | | Porto |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 1000 | | Lisboa |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 2000 | | Santarém |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 2040 | | Rio Maior |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 4000 | | Porto |
| 12 | | Isabel Silva | | 28 | | 2040 | | 1000 | | Lisboa | |
| 12 | | Isabel Silva | | 28 | | 2040 | | 2000 | | Santarém | |
| 12 | | Isabel Silva | | 28 | | 2040 | | 2040 | | Rio Maior | |
| 12 | | Isabel Silva | | 28 | | 2040 | | 4000 | | Porto | |
| 49 | José António | | 17 | | 333555 | | 2000 | | 1000 | | Lisboa |
| 49 | José António | | 17 | | 333555 | | 2000 | | 2000 | | Santarém |
| 49 | José António | | 17 | | 333555 | | 2000 | | 2040 | | Rio Maior |
| 49 | José António | | 17 | | 333555 | | 2000 | | 4000 | | Porto |

O pretendido seria a junção entre as tabelas Pessoa e Postal. Para tal é necessário, na cláusula WHERE, indicar as chaves estrangeiras de ligação.

SELECT \*

FROM Pessoa, Postal WHERE Cod\_postal = Código

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Neste caso, o resultado da selecção seria: Id | Nome | | Idade | | Telefone | | Cod\_postal | | Código | | Local |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 1000 | | Lisboa |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 1000 | | Lisboa |
| 12 | | Isabel Silva | | 28 | | 2040 | | 2040 | | Rio Maior | |
| 49 | José António | | 17 | | 333555 | | 2000 | | 2000 | | Santarém |

**Slide 33 - Equi-Join**

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

SELECT \*

FROM Pessoa, Postal

WHERE Cod\_postal = Código

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Neste caso, o resultado da selecção seria: Id | Nome | | Idade | | Telefone | | Cod\_postal | | Código | | Local |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | 1000 | | Lisboa |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | 1000 | | Lisboa |
| 12 | | Isabel Silva | | 28 | | 2040 | | 2040 | | Rio Maior | |
| 49 | José António | | 17 | | 333555 | | 2000 | | 2000 | | Santarém |

**Natural Join**

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

SELECT Pessoa.\*, Postal.Local

FROM Pessoa, Postal

WHERE Cod\_postal = Código

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Neste caso, o resultado da selecção seria: Id | Nome | | Idade | | Telefone | | Cod\_postal | | Local |
| 71 | António Dias | | 43 | | 789654 | | 1000 | | Lisboa |
| 54 | Célia Morais | | 36 | | 123456 | | 1000 | | Lisboa |
| 12 | | Isabel Silva | | 28 | | 2040 | | Rio Maior | |
| 49 | José António | | 17 | | 333555 | | 2000 | | Santarém |

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

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

**INNER JOIN**

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

INNER JOIN

1 - Pretende-se seleccionar o nome e morada completa (código e localidade) de todas as pessoas da tabela Pessoa.

SELECT Nome, Codigo, Localidade

FROM Pessoa, Postal

WHERE Cod\_postal = Postal.Codigo;

A screenshot of a computer

AI-generated content may be incorrect.

Embora ainda de uso pouco comum, é possível, em alguns sistemas, escrever o mesmo SELECT num formato em que se especifique a natureza do Join (INNER), no mesmo exemplo mas ordenando o resultado por Código Postal

SELECT Nome, Cod\_postal, Localidade

FROM Pessoa INNER JOIN Postal

ON Pessoa.Cod\_postal = Postal.Codigo

ORDER BY Pessoa.Cod\_postal

A screenshot of a computer

AI-generated content may be incorrect.

O conceito do Outer Join é obter numa ligação a totalidade das linhas de uma tabela, ainda que não exista o correspondente valor na outra tabela a que esta está ligada pela junção.

OUTER JOIN

1 - Pretende-se seleccionar o nome e morada completa (código e localidade) de todas as pessoas da tabela Pessoa, assim como todos os códigos postais existentes na tabela Postal.

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal LEFT JOIN Pessoa

ON Postal.Codigo = Pessoa.Cod\_postal

A screenshot of a computer

AI-generated content may be incorrect.

Todas as linhas da tabela Postal são apresentadas. Se existir correspondente na coluna Cod\_postal da tabela Pessoa, são mostrados os dados, senão as entradas da tabela Pessoa são preenchidas a NULL.

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal

INNER JOIN Pessoa ON Postal.Codigo = Pessoa.Cod\_postal;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal, Pessoa

WHERE Codigo = Cod\_postal;

A screenshot of a computer

AI-generated content may be incorrect.

**OUTER JOIN**

Quando o Outer Join é efectuado à direita, são considerados todos os registos da tabela da direita e apenas os registos correspondentes na tabela da esquerda.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Consideremos as seguintes relações (Pessoa e Comissão): Id | Nome | Idade | Id | Id\_msg | Valor |
| 71 | António Dias | 43 | 49 | 10 | 1250 |
| 12 | Isabel Silva | 28 | 49 | 70 | 750 |
| 49 | José António | 17 | 71 | 12 | 100 |
| 85 | João Silva | 49 | 71 | 15 | 200 |

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

SELECT `Nome`, `Valor`

FROM `Comissao`

RIGHT JOIN `Pessoa` ON `Comissao`.`Id` = `Pessoa`.`PessoaID`;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT Nome, Valor

FROM Comissao

INNER JOIN Pessoa ON Comissao.Id = PessoaID;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT Nome, Valor

FROM Comissao, Pessoa

WHERE Comissao.Id = Pessoa.PessoaID;

A screenshot of a computer

AI-generated content may be incorrect.

**SELF JOIN**

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

Pretende-se saber quais os tipos de códigos postais existentes, diferentes do local LISBOA.

SELECT p1.Codigo, p2.Localidade

FROM Postal AS p1, Postal AS p2

WHERE p1.Codigo = p2.Codigo

AND p2.Localidade <> 'Lisboa';

A screenshot of a computer

AI-generated content may be incorrect.

**UNION**

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

Pretende-se juntar o código e a descrição da tabela mensagens aos códigos postais.

SELECT Id\_msg, Mensagem

FROM Mensagem

UNION

SELECT Codigo, Localidade FROM Postal

A screenshot of a computer

AI-generated content may be incorrect.

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

Pretende-se juntar as tabelas Postal (cujos locais contenham a string “OR”) e Mensagem (cujas Id sejam inferiores a 30), ordenando o resultado por mensagem.

SELECT Id\_msg, Mensagem

FROM Mensagem

WHERE Id\_msg < 30

UNION

SELECT Codigo, Localidade

FROM Postal

WHERE Localidade LIKE '%OR%'

ORDER BY Mensagem;

A screenshot of a computer

AI-generated content may be incorrect.

**INTERSECT**

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

Pretende-se seleccionar as linhas da tabela Postal cujo Código é menor ou igual a 5000 e é maior ou igual a 3000.

SELECT \* FROM Postal

WHERE Código >= 3000 AND Código <= 5000

A screenshot of a computer

AI-generated content may be incorrect.

SELECT p1.\*

FROM Postal p1

INNER JOIN Postal p2 ON p1.Codigo = p2.Codigo

WHERE p1.Codigo >= 3000 AND p1.Codigo <= 5000;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Postal p1

WHERE p1.Codigo >= 3000 AND p1.Codigo <= 5000

AND EXISTS (

SELECT 1 FROM Postal p2

WHERE p2.Codigo = p1.Codigo AND p2.Codigo <= 5000

);

A screenshot of a computer program

AI-generated content may be incorrect.

**MINUS**

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

Pretende-se seleccionar as linhas da tabela Postal cujo Código é menor ou igual a 5000, ignorando todos os elementos cujo código está entre 2000 e 3000.

SELECT \* FROM Postal WHERE Codigo <= 5000 AND

Codigo NOT BETWEEN 2000 AND 3000

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Postal

WHERE Codigo <= 5000

AND Codigo NOT BETWEEN 2000 AND 3000;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \*

FROM Postal p1

WHERE p1.Codigo <= 5000

AND NOT EXISTS (

SELECT 1

FROM Postal p2

WHERE p2.Codigo BETWEEN 2000 AND 3000

AND p2.Codigo = p1.Codigo

);

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **RESUMO DAS JUNÇÕES** JOIN | Descrição |
| Produto Cartesiano | Juntar cada linha da tabela T1 com todas as linhas de T2. |
| INNER JOIN | Junção tradicional, em que apenas são apresentadas as  linhas comuns às duas tabelas. |
| OUTER JOIN | Extensão do Inner Join ao proporcionar todos os registos  de uma das tabelas, mesmo que sobre estes não exista qualquer ligação. |
| UNION | Todos os registos de qualquer das pesquisas (sem  duplicados). |
| UNION ALL | Todos os registos de qualquer das pesquisas (com  duplicados). |
| INTERSECT | Todos os registos comuns a ambas as pesquisas. |
| MINUS | Todos os registos da primeira pesquisa que não aparecem  na segunda. |

-- Produto cartesiano - não filtrado

SELECT \*

FROM Pessoa, Postal;

SELECT \*

FROM Pessoa;

SELECT \*

FROM Postal;

-- Produto cartesiano, filtrado com a condição de junção

SELECT \*

FROM Pessoa, Postal

WHERE Pessoa.cod\_postal = Postal.codigo;

-- Junção INNER JOIN

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal

INNER JOIN Pessoa ON Postal.Codigo = Pessoa.Cod\_postal;

-- Junção LEFT JOIN (tabela à esquerda é Postal)

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal

LEFT JOIN Pessoa ON Postal.Codigo = Pessoa.Cod\_postal;

-- Junção LEFT JOIN (tabela à esquerda é Pessoa)

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Pessoa

LEFT JOIN Postal ON Pessoa.Cod\_postal = Postal.Codigo;

-- Junção RIGHT JOIN (tabela à direita é Postal)

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Postal

RIGHT JOIN Pessoa ON Postal.Codigo = Pessoa.Cod\_postal;

-- Junção RIGHT JOIN (tabela à direita é Pessoa)

SELECT Nome, Cod\_postal, Codigo, Localidade

FROM Pessoa

RIGHT JOIN Postal ON Postal.Codigo = Pessoa.Cod\_postal;

-- Inserção de um novo registro na tabela Pessoa

INSERT INTO Pessoa (Nome, Idade, Salario)

VALUES ('Celia Duarte', 25, 100000);

-- Seleção de todos os registros da tabela Pessoa

SELECT \*

FROM Pessoa;