

Fundamentos de Base de Dados

SQL- Structured Query Language

Docente: Fátima Leal

DCT DEPARTAMENTO CIÊNCIA
E TECNOLOGIA

Até agora...

- Operações elementares da Álgebra Relacional

SQL

- **Tomemos novamente como exemplo a base de dados EMPRESA que temos vindo a trabalhar.**

FUNCIONARIO(F_ident, nome, sobrenome, morada, dt_nasc, salario, sexo, Super_ID, Dnum)

DEPENDENTE(DID, F_ident, nome, dt_nasc, sexo, relacionamento)

DEPARTAMENTO(Dnum, nome, F_ident, dt_inicio)

LOCALIZACOES(LID, Dnum, localização)

PROJETO(Pnum, nome, localização, Dnum)

TRABALHA_EM(TID, Pnum, F_ident, horas)

SQL



- SQL – *Structured Query Language*
- **Linguagem de consulta** que permite **definir, questionar e manipular** bases de dados.
- **Baseia-se** nas linguagens de **consulta formais** – Álgebra Relacional e Cálculo Relacional
- Permite **especificar restrições** que devem ser impostas aos dados possibilitando a **implementação a integridade e segurança** da informação armazenada
- Permite uma **ligação** a outras linguagens de programação

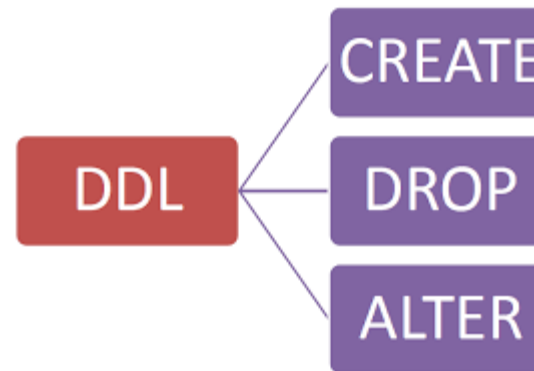
SQL

- É uma **linguagem standard** para todos os **sistemas comerciais de gestão de bases de dados** relacionais
- A linguagem SQL tem vários componentes:
 - **DDL** (Data Definition Language) – Linguagem de Definição de Dados
 - CREATE, ALTER, DROP
 - **DML** (Data Manipulation Language) – Linguagem de Manipulação de Dados
 - INSERT, UPDATE, DELETE
 - **DCL** (Data Control Language) – Linguagem de Controlo de Dados



SQL – Data Definition Language (DDL)

- CREATE
- DROP
- ALTER



SQL – Domínio dos atributos

- **Valores numéricos**

- TINYINT: 1 byte
- SMALLINT: 2 bytes
- INT: 4 bytes
- BIGINT: 8 bytes
- FLOAT: 4 bytes
- NUMERIC: 5-17 bytes
- DECIMAL(N, D): N dígitos com D dígitos depois do ponto decimal

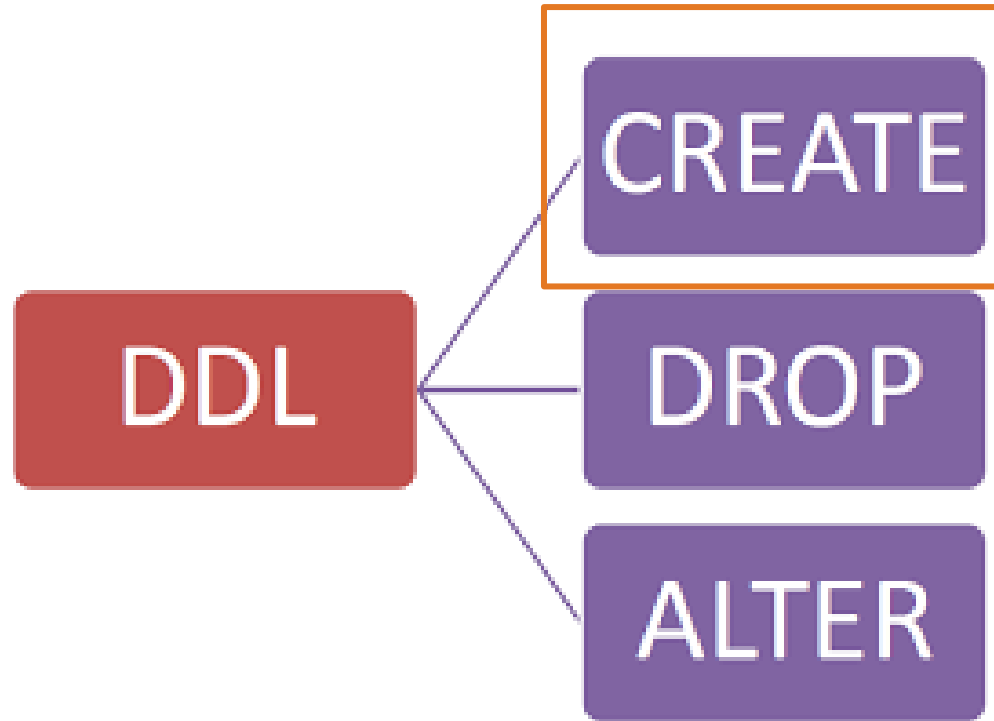
- **Valores temporais**

- DATE: formato 'YYYY-MM-DD' ('2004-01-30')
- TIME: formato 'HH:MM:SS' ('09:12:47')
- DATETIME: formato 'YYYY-MM-DD HH:MM:SS' ('2004-01-30 09:12:47')
- TIMESTAMP: formato YYYYMMDDHHMMSS (20040130091247)

SQL – Domínio dos atributos

- **Valores lógicos**
 - BOOLEAN: TRUE ou FALSE
- **Sequências de texto** (strings)
 - CHAR(N): string de comprimento fixo de N caracteres, $0 \leq N \leq 255$
 - VARCHAR(N): string de comprimento variável até N caracteres, $0 \leq N \leq 255$
 - TEXT: string de comprimento variável até 65 Kbytes
 - LONGTEXT: string de comprimento variável até 4.3 Gbytes

SQL – DDL

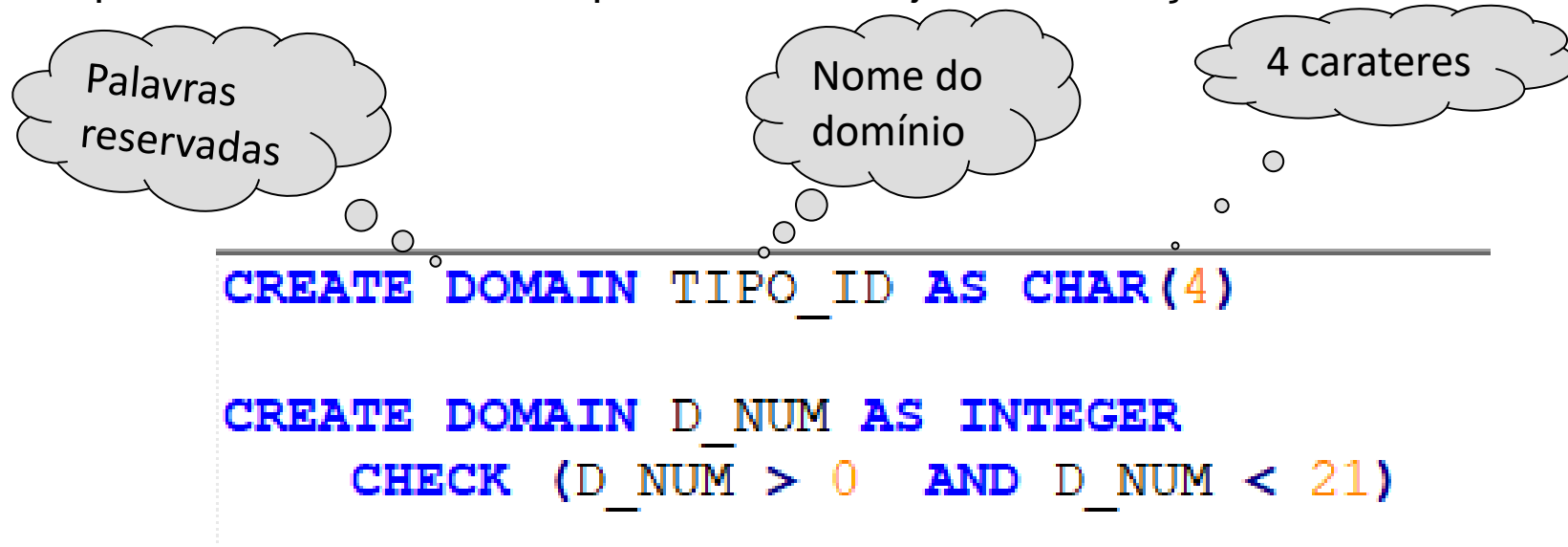


SQL – CREATE

- O comando CREATE permite criar objetos na base de dados
 - CREATE SCHEMA
 - CREATE DOMAIN
 - CREATE TABLE
 - CREATE INDEX
 - CREATE VIEW
 - CREATE TRIGGER
 - ...

SQL – CREATE DOMAIN

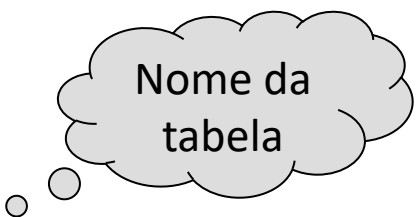
- O comando **CREATE DOMAIN** permite **declarar um domínio** (tipo de dados) especificando o seu nome para ser usado junto da criação de atributos.



SQL – CREATE TABLE

- O comando CREATE TABLE é usado para criar tabelas na base de dados, atribuindo-lhe um nome e especificando os seus atributos e restrições iniciais

```
CREATE TABLE FUNCIONARIO  
(  
    atributos ...  
    restrições ...  
);
```



Nome da
tabela

SQL – CREATE TABLE

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo, F_supervisor, F_Dnum)

```
CREATE TABLE FUNCIONARIO
( F_ident          Tipo_ID,
  F_nome           VARCHAR(10) NOT NULL,
  F_sobrenome       VARCHAR(20) NOT NULL,
  F_morada          VARCHAR(30),
  F_dt_nasc         DATE,
  F_salario         NUMERIC,
  F_sexo            CHAR(1),
  F_supervisor      CHAR(9),
  F_Dnum            INT NOT NULL,
  PRIMARY KEY      (F_ident));
```

Nome da tabela

Tipo de atributo

Restrição para valores nulos

Chave primária

SQL – CREATE TABLE

Restrições Atributos

- Podemos definir **valores por omissão** e **restrições sobre os atributos**
- Definir o valor por omissão para um atributo.
 - **<ATRIB> <DOMÍNIO> DEFAULT <VAL>** `salario DECIMAL(10,2) DEFAULT 635.00`
- Não permitir que um atributo possua valores NULL.
 - **<ATRIB> <DOMÍNIO> NOT NULL** `salario DECIMAL(10,2) NOT NULL`
- Restringir os valores.
 - **<ATRIB> <DOMÍNIO> CHECK (<COND>)** `salario DECIMAL(10,2) CHECK (salario > 0)`
- Configuração das Restrições de Integridade da Chave
 - **PRIMARY KEY (<ATRIB_1>, ..., <ATRIB_N>)** `PRIMARY KEY (ident)`
 - **UNIQUE (<ATRIB_1>, ..., <ATRIB_N>)** `UNIQUE (nome)`

SQL – CREATE TABLE

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc,
F_salario, F_sexo, F_supervisor, F_Dnum)

```
CREATE TABLE FUNCIONARIO
( F_ident      Tipo_ID,
  F_nome       VARCHAR(10) NOT NULL,
  F_sobrenome  VARCHAR(20) NOT NULL,
  F_morada     VARCHAR(30) ,
  F_dt_nasc    DATE,
  F_salario    NUMERIC,
  F_sexo       CHAR(1),
  F_supervisor CHAR(9),
  F_Dnum       INT NOT NULL,
  PRIMARY KEY  (F_ident));
```

Chaves
estrangeiras?

SQL – CREATE TABLE

Restrições Integridade Referencial

- Configuração das **Restrições de Integridade Referencial**
- Definir uma chave externa da tabela
 - **FOREIGN KEY** (<ATRIB_1>, ..., <ATRIB_N>)
REFERENCES <TABELA>(<CHAVE_1>, ..., <CHAVE_N>)

- Retomando o exemplo:

```
foreign key (F_supervisor) references Funcionario(F_ident)
foreign key (F_Dnum) references DEPARTAMENTO (D_num)
```


SQL – CREATE TABLE

Restrições Integridade Referencial

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc,
F_salario, F_sexo, F_supervisor, F_Dnum)

```
CREATE TABLE FUNCIONARIO
( F_ident          Tipo_ID,
  F_nome           VARCHAR(10) NOT NULL,
  F_sobrenome       VARCHAR(20) NOT NULL,
  F_morada          VARCHAR(30) ,
  F_dt_nasc         DATE,
  F_salario         NUMERIC,
  F_sexo            CHAR(1) ,
  F_supervisor      CHAR(9) ,
  F_Dnum            INT NOT NULL,
  PRIMARY KEY      (F_ident)
  FOREIGN KEY (F_supervisor) REFERENCES Funcionario(F_ident)
  FOREIGN KEY (F_Dnum) REFERENCES DEPARTAMENTO (D_num)
)
```

SQL – CREATE TABLE

Restrições Integridade Referencial

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc,
F_salario, F_sexo, F_supervisor, F_Dnum)

```
CREATE TABLE FUNCIONARIO
( F_ident          Tipo_ID,
  F_nome           VARCHAR(10) NOT NULL,
  F_sobrenome       VARCHAR(20) NOT NULL,
  F_morada          VARCHAR(30) ,
  F_dt_nasc         DATE,
  F_salario         NUMERIC,
  F_sexo            CHAR(1) ,
  F_supervisor      CHAR(9) ,
  F_Dnum            INT NOT NULL,
  PRIMARY KEY      (F_ident)
  FOREIGN KEY (F_supervisor) REFERENCES Funcionario(F_ident)
  FOREIGN KEY (F_Dnum) REFERENCES DEPARTAMENTO (D_num)
)
```

A tabela departamento já está criada??

SQL – CREATE TABLE

Manutenção Integridade Referencial

- As restrições de **integridade referencial** podem ser violadas quando **inserimos, removemos ou quando alteramos** um valor de uma chave primária ou chave externa.
- O SQL por omissão rejeita essas operações. É **possível modificar esse comportamento** para as operações de remoção (**ON DELETE**) e alteração (**ON UPDATE**) que violem a integridade referencial
 - **ON DELETE SET NULL / ON UPDATE SET NULL**: coloca a NULL
 - **ON DELETE SET DEFAULT / ON UPDATE SET DEFAULT**: coloca o de omissão
 - **ON DELETE CASCADE**: remove todas as linhas que referenciam a removida.
 - **ON UPDATE CASCADE**: atualiza com o novo valor a chave externa das linhas que referenciam a alterada.

SQL – CREATE TABLE

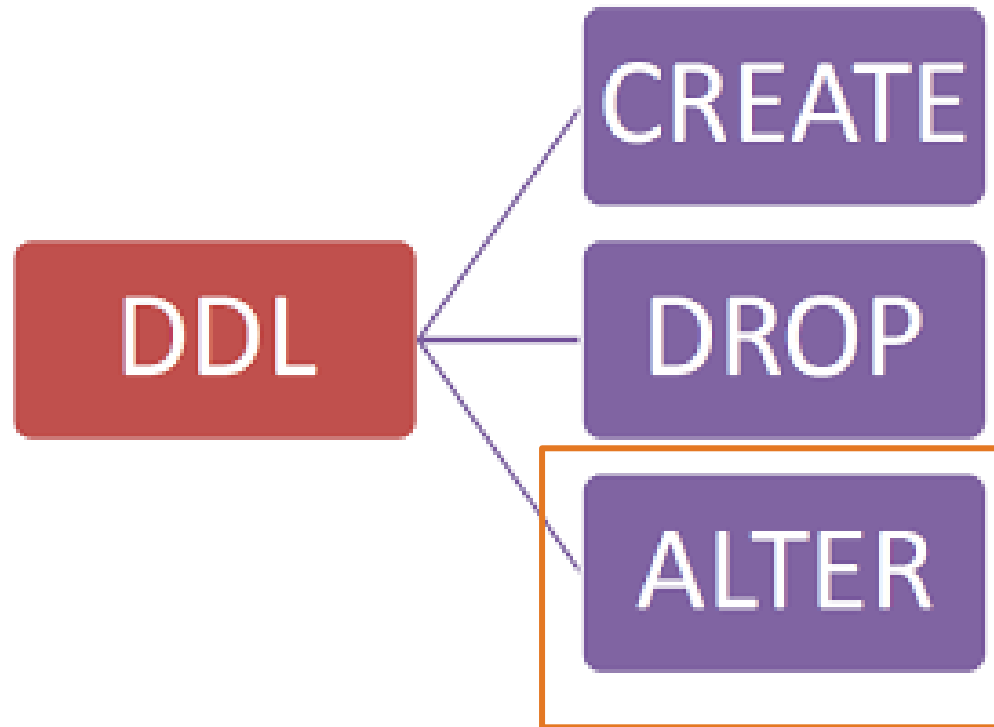
Manutenção Integridade Referencial

- Exemplos:

```
FOREIGN KEY (F_Dnum) REFERENCES DEPARTAMENTO (D_num)  
ON DELETE SET DEFAULT ON UPDATE CASCADE
```

```
FOREIGN KEY (F_Dnum) REFERENCES DEPARTAMENTO (D_num)  
ON DELETE CASCADE ON UPDATE CASCADE
```

SQL – DDL



SQL – ALTER

- O comando ALTER TABLE é usado para alterações nas definições dos objetos criados na base de dados

```
ALTER TABLE FUNCIONARIO  
    especificações de alterações ...;
```

- **ADD** <ATRIB> <DOMÍNIO> - adiciona um novo atributo à tabela adicionando valores NULL em todas as linhas.
- **DROP** <ATRIB> - remove um atributo da tabela.
- **ALTER** <ATRIB> [SET | DROP] <OPÇÕES> - altera as restrições de um atributo da tabela.

SQL – ALTER

```
CREATE TABLE FUNCIONARIO
( F_ident          Tipo_ID,
  F_nome           VARCHAR(10) NOT NULL,
  F_sobrenome      VARCHAR(20) NOT NULL,
  F_morada         VARCHAR(30) ,
  F_dt_nasc        DATE,
  F_salario        NUMERIC,
  F_sexo           CHAR(1) ,
  F_supervisor     CHAR(9) ,
  F_Dnum           INT NOT NULL,
  PRIMARY KEY      (F_ident)
  FOREIGN KEY (F_supervisor) REFERENCES Funcionario(F_ident)
);
```

```
CREATE TABLE DEPARTAMENTO
( D_num           INT           NOT NULL,
  D_nome          VARCHAR(15)   NOT NULL,
  D_diretor       char(9)       NOT NULL,
  D_dt_inicio     DATE,
  PRIMARY KEY (D_num),
  UNIQUE         (D_nome),
  FOREIGN KEY (D_diretor) REFERENCES FUNCIONARIO(F_ident) );
```

Retomando o exemplo com as tabelas FUNCIONARIO e DEPARTAMENTO já criadas

SQL – ALTER

```
ALTER TABLE FUNCIONARIO  
ADD FOREIGN KEY (F_Dnum) REFERENCES DEPARTAMENTO (D_num)
```

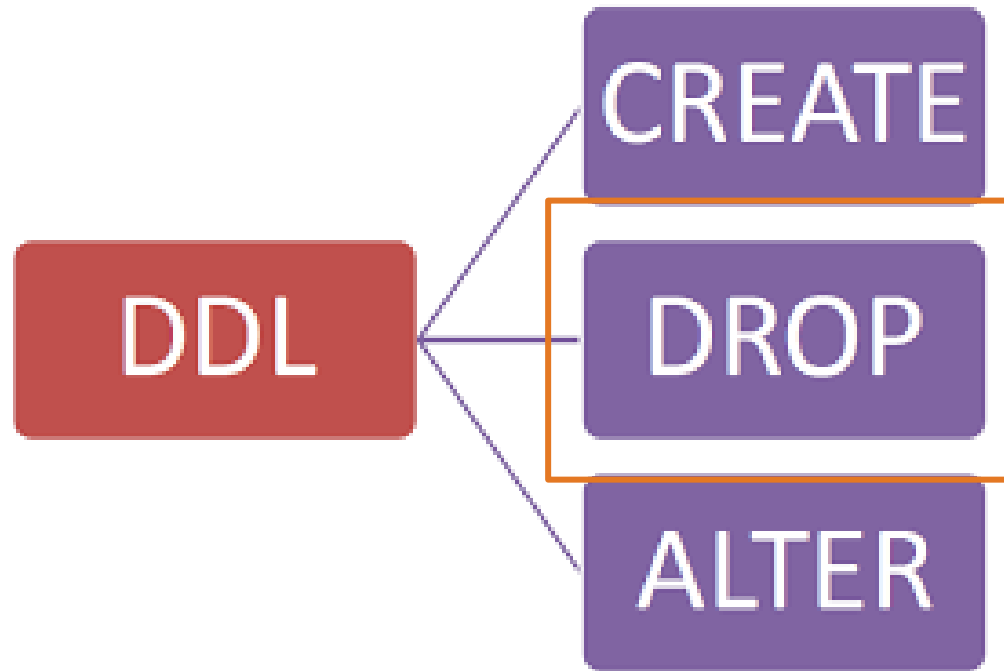
Adiciona uma chave externa à
tabela FUNCIONARIO

Outros exemplos:

```
ALTER TABLE FUNCIONARIO  
ALTER F_supervisor SET DEFAULT '1163'
```

Adiciona valor *default* para o
atributo F_supervisor

SQL – DDL



SQL – DROP

- O comando DROP é usado para alterações nas definições dos objetos criados na base de dados

DROP TABLE FUNCIONARIO

Excluir



SQL – Data Manipulation Language (DML)

- INSERT
- UPDATE
- DELETE

DML

INSERT
UPDATE
DELETE
SELECT

SQL – INSERT

- O comando INSERT **permite inserir uma linha na tabela.**

```
INSERT INTO <TABELA>[(<ATRIB_1>, ..., <ATRIB_N>)]  
VALUES (<VAL_A1>, ..., <VAL_AN>), ..., (<VAL_M1>, ..., <VAL_MN>);
```

- Quando **não se indica os atributos** da tabela assume-se **que são todos e pela ordem definida** quando da sua criação.
- Quando se **explicita o nome dos atributos** é possível **definir a ordem** e indicar apenas parte dos atributos.
- Nos **atributos não indicados** é inserido o **valor por omissão** (se definido) ou o **valor NULL**. Os **atributos** com declarações **NOT NULL** e **sem declarações DEFAULT** **devem** ser sempre indicados.

SQL – INSERT

- Exemplos

```
INSERT INTO FUNCIONARIO  
VALUES ('1163', 'Carlos', 'Martins', 'Porto', '12/08/1974',  
       '4560.00', 'M', NULL, '4');
```

```
INSERT INTO Funcionario (F_nome, F_morada, F_salario) VALUES ('Luis', 'Porto', 1000)
```

SQL - UPDATE

- O comando UPDATE permite modificar os valores dos atributos de uma ou mais linhas de uma relação.

PROJETO(P_num, P_nome, P_loc, P_Dnum)

```
UPDATE PROJETO  
SET P_loc = 'Porto', P_Dnum = '4'  
WHERE P_num = '1'
```

Novos valores

Onde é que os valores devem ser alterados?

condição

SQL – DELETE

- O comando DELETE **remove linhas** de uma tabela.

```
DELETE FROM <TABELA>  
[WHERE <COND>];
```

- Quando **não se indica a condição WHERE** **todas as linhas são removidas** mas a tabela mantém-se na BD embora vazia.
- Esta operação **apenas remove diretamente linhas de uma tabela**. No entanto, esta operação **pode propagar-se** a outras tabelas para manter as restrições de integridade referencial.
 - **ON DELETE SET NULL**: coloca o valor NULL na chave externa das linhas que referenciam as removidas.
 - **ON DELETE SET DEFAULT**: coloca o valor por omissão na chave externa das linhas que referenciam as removidas.
 - **ON DELETE CASCADE**: remove todas as linhas que referenciam as removidas.

SQL – DELETE

- Exemplos

```
DELETE FROM FUNCIONARIO
```

Apaga todos as linhas da tabela FUNCIONARIO

```
DELETE FROM FUNCIONARIO  
WHERE F_ident = '1167'
```

Exclui apenas o funcionário com ident = 1167. Apenas exclui uma linha porque ident é a chave primária.

```
DELETE FROM FUNCIONARIO  
WHERE F_nome = 'Maria'
```

Exclui funcionários com nome Maria. Neste exemplo, várias linhas podem ser apagadas.

SQL- Estrutura de consultas

- Estrutura básica de uma consulta para extração de informação SQL.

```
SELECT < lista de atributos>  
FROM < lista de tabelas >  
WHERE < condição >
```

- A lista de atributos contém os **nomes dos atributos que serão recuperados** pela consulta.
- A lista de tabelas contém as **tabelas necessárias para encontrar a informação** desejada na consulta.
- A condição é a **expressão condicional** que identifica quais as linhas das tabelas mencionadas devem ser considerados na consulta.

SQL- SELECT FROM

Consulta 1: Obtenha os identificadores de todos os funcionários.

```
SELECT F_ident  
FROM FUNCIONARIO
```

- Na álgebra relacional seria:

$$\pi_{F_ident} (FUNCIONARIO)$$

- No cálculo relacional

$$\{f.F_ident \mid FUNCIONARIO(f)\}$$

SQL- SELECT FROM WHERE

Consulta 2: Obtenha as datas de nascimento e as moradas dos funcionários que possuem sobrenome English.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc,
F_salario, F_sexo, F_supervisor, F_Dnum)

```
SELECT F_dt_nasc, F_morada  
FROM FUNCIONARIO  
WHERE F_sobrenome = 'English'
```

- Na álgebra relacional seria:

$$\pi_{F_dt_nasc, F_morada} (\sigma_{F_sobrenome="English"} (FUNCIONARIO))$$

- No cálculo relacional

$$\{f.F_dt_nasc, f.F_morada \mid FUNCIONARIO(f) \text{ AND } f.F_sobrenome="English"\}$$

SQL- SELECT FROM WHERE

Consulta 3: Obtenha os nomes e as moradas de todos os funcionários que trabalham no departamento de nome Research.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo, F_supervisor, F_Dnum)
DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)

```
SELECT F_nome, F_morada  
FROM FUNCIONARIO, DEPARTAMENTO  
WHERE D_nome = 'Research' AND F_Dnum=D_num
```

SQL- SELECT FROM WHERE

```
SELECT F_nome, F_morada  
FROM FUNCIONARIO, DEPARTAMENTO  
WHERE D_nome = 'Research' AND F_Dnum=D_num
```

- Na álgebra relacional:

$$FDEP < -FUNCIONARIO \bowtie_{F_Dnum=D_num} (DEPARTAMENTO)$$
$$RES < -\pi_{F_nome, F_morada}(\sigma_{D_nome="Research"}(FDEP))$$

- No cálculo relacional:

$$\{f.F_nome, f.F_morada \mid FUNCIONARIO(f) \text{ AND } (\exists d)(DEPARTAMENTO(d) \text{ AND } d.D_nome = 'Research' \text{ AND } d.D_num = f.F_Dnum)\}$$

SQL- SELECT FROM WHERE

FUNCIONARIO				
<u>F_ident</u>	F_nome	F_morada	...	F_Dnum
1163	Carlos	Porto	...	4
1164	Maria	Porto	...	1
1165	Pedro	Lisboa	...	1
1166	Joana	Lisboa	...	4
1167	Luís	Lisboa	...	3

DEPARTAMENTO		
<u>D_num</u>	D_nome	...
1	Informatica	...
3	Research	...
4	Recursos Humanos	...

Produto Cartesiano

FROM FUNCIONARIO, DEPARTAMENTO							
F_ident	F_nome	F_morada	...	F_Dnum	D_num	D_nome	...
1163	Carlos	Porto	...	4	1	Informatica	...
1163	Carlos	Porto	...	4	3	Engenharia	...
1163	Carlos	Porto	...	4	4	Recursos Humanos	...
	
	

SQL- SELECT FROM WHERE

```
WHERE D_nome = 'Research' AND F_Dnum=D_num
```

<u>F_ident</u>	F_nome	F_morada	...	F_Dnum	<u>D_num</u>	D_nome
1163	Carlos	Porto	...	4	4	Recursos Humanos
1164	Maria	Porto	...	1	1	Informatica
1165	Pedro	Lisboa	...	1	1	Informatica
1166	Joana	Lisboa	...	4	4	Recursos Humanos
1167	Luís	Lisboa	...	3	3	Research

```
SELECT F_nome, F_morada
```

nome	morada
Luís	Lisboa

SQL- SELECT FROM WHERE

Consulta 4: Para cada projeto localizado em Houston, lista o número do projeto, o número do departamento que controla o projeto, e o nome, morada e data de nascimento do diretor desse departamento.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)

DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)

PROJETO(P_num, P_nome, P_loc, P_Dnum)

```
SELECT P_num, P_Dnum, F_nome, F_morada, F_dt_nasc  
FROM FUNCIONARIO, DEPARTAMENTO, PROJETO  
WHERE P_Dnum=D_num AND D_diretor=F_ident AND P_loc = 'Houston'
```


SQL- SELECT*

- **SELECT ***
- Significa que se pretende **selecionar todos os atributos** sem os mencionar explicitamente.

```
SELECT *  
FROM <TABELA_1>, ..., <TABELA_M>  
[WHERE <COND>];
```

- Analisemos um exemplo

SQL- SELECT*

Consulta 5: Obtenha os funcionários que trabalham no departamento 4 e cujo salário é superior a 2000 euros.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)

```
SELECT *  
FROM FUNCIONARIO  
WHERE F_Dnum = 4 AND F_salario > 2000
```

SQL- SELECT DISTINCT/ALL

- **DISTINCT remove as linhas em duplicado** do resultado da consulta.
- **ALL não remove as linhas em duplicado** do resultado da consulta.

```
SELECT DISTINCT/ALL <ATRIB_1>, ..., <ATRIB_N>  
FROM <TABELA_1>, ..., <TABELA_M>  
[WHERE <COND>];
```

- Ao contrário da álgebra relacional o **SQL não remove automaticamente as linhas em duplicado**
- Analisemos um exemplo...

SQL- SELECT DISTINCT

Consulta 6: Obtenha os salários de todos os funcionários.

```
SELECT ALL F_salario  
FROM FUNCIONARIO
```

Não remove valores duplicados

Consulta 7: Obtenha os valores distintos de salários praticados na empresa.

```
SELECT DISTINCT F_salario  
FROM FUNCIONARIO
```

Remove valores duplicados

SQL- Renomeação de atributos

- Por vezes, pode acontecer tabelas diferentes terem **atributos com o mesmo nome. Nesta situação, existe ambiguidade de atributos.**
- Uma solução passa por **anteceder o atributo com o nome da relação** a que pertencem desta forma:
 - DEPARTAMENTO.nome
 - FUNCIONARIO.nome
 - PROJETO.nome
- No entanto, é possível fazer **renomeação de atributos** de modo a que façam mais sentido

<ATRIB> **AS** <NOVO_NOME>

```
SELECT FUNCIONARIO.nome AS func_nome, salario  
FROM ...  
WHERE ...;
```

SQL- Renomeação de tabelas

Consulta 8: Lista os nomes dos funcionários e dos seus supervisores.

```
SELECT F.F_nome, S.F_nome  
FROM FUNCIONARIO AS F, FUNCIONARIO AS S  
WHERE F.F_supervisor = S.F_ident
```

Como estamos a utilizar a mesma tabela duas vezes, é útil fazer uma renomeação dessas tabelas utilizando o AS.

Mas, uma relação pode ser renomeada sempre que seja útil, mesmo que não seja referenciada mais do que uma vez na consulta.

SQL- BETWEEN

- Permite definir um intervalo numérico de comparação.

<ATRIB> BETWEEN <VAL_1> AND <VAL_2>

Consulta 9: Obtenha os funcionários que trabalham no departamento 4 e cujo salário é superior a 20 000 euros e inferior a 40 000 euros.

```
SELECT *  
FROM FUNCIONARIO  
WHERE F_Dnum = 4 AND (F_salario BETWEEN 20000 AND 40000)
```

SQL- LIKE

- Permite **comparar atributos do tipo string** com sequências de texto padrão.
 <ATRIB> [NOT] **LIKE** <PADRÃO>
- Existem **dois caracteres** com significado especial:
 - % representa um número arbitrário de caracteres
 - _ representa um qualquer caracter

Consulta 10: Obtenha os funcionários cujas moradas indiquem a cidade de Houston.

SQL- LIKE

Consulta 10: Obtenha os funcionários cujas moradas indiquem a cidade do Houston.

```
SELECT *  
FROM FUNCIONARIO  
WHERE F_morada LIKE '%Houston%'
```

Nota que morada é uma string que pode conter diversas informações. Com o LIKE podemos verificar se Houston existe dentro da string utilizando uma expressão regular.

SQL- LIKE

Consulta 11: Obtenha os funcionários que nasceram na década de 60.

```
SELECT *  
FROM FUNCIONARIO  
WHERE F_dt_nasc LIKE '196_ - _ - _'
```

SQL- IS NULL

- Permite verificar se os **valores de um atributo são NULL** (não conhecido, em falta ou não aplicável).
 <ATRIB> IS [NOT] NULL

Consulta 12: Obtenha o nome dos funcionários que não têm supervisor.

```
SELECT F_nome  
FROM FUNCIONARIO  
WHERE F_supervisor IS NULL
```

SQL- ORDER BY

- Permite **definir a ordem dos tuplos** do resultado.

ORDER BY <ATRIB_1> [**ASC | DESC**], ..., <ATRIB_N> [**ASC | DESC**]

Consulta 13: Obtenha a lista de nomes dos funcionários e dos projetos nos quais eles trabalham. Ordene o resultado por ordem alfabética.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)

DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)

PROJETO(P_num, P_nome, P_loc, P_Dnum)

TRABALHA_EM(TID, Pnum, F_ident, horas)

SQL- ORDER BY

Consulta 13: Obtenha a lista de nomes dos funcionários e dos projetos nos quais eles trabalham. Ordene o resultado por ordem alfabética.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)

DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)

PROJETO(P_num, P_nome, P_loc, P_Dnum)

TRABALHA_EM(TID, Pnum, F_ident, horas)

```
SELECT F_nome, P_nome
FROM FUNCIONARIO, TRABALHA_EM, PROJETO
WHERE F_ident = T_F_ident AND T_Pnum = P_num
ORDER BY F_nome ASC
```

SQL- UNION, INTERSECT, EXCEPT

- As linhas em duplicado **são removidas**:
 - Reunião: (<CONSULTA_1>) **UNION** (<CONSULTA_2>)
 - Intersecção: (<CONSULTA_1>) **INTERSECT** (<CONSULTA_2>)
 - Diferença: (<CONSULTA_1>) **EXCEPT** (<CONSULTA_2>)
- Existem também versões destes operadores que **não removem as linhas em duplicado**:
 - Reunião: (<CONSULTA_1>) **UNION ALL** (<CONSULTA_2>)
 - Intersecção: (<CONSULTA_1>) **INTERSECT ALL** (<CONSULTA_2>)
 - Diferença: (<CONSULTA_1>) **EXCEPT ALL** (<CONSULTA_2>)
- Estas operações só se aplicam a relações compatíveis para a reunião, ou seja, ambas as relações devem ter os mesmos atributos e na mesma ordem

SQL- UNION, INTERSECT, EXCEPT

Consulta 14: Obtenha a identificação dos funcionários que trabalham no departamento 4 ou que supervisionam um funcionário que trabalha no departamento 4.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo, F_supervisor, F_Dnum)

```
SELECT F_ident
FROM FUNCIONARIO
WHERE F_Dnum = 4
UNION
SELECT F_supervisor
FROM FUNCIONARIO
WHERE F_Dnum = 4
```

Nota que o conceito é o mesmo que na álgebra relacional.

SQL- JOIN

- O SQL permite a junção de tabelas. Os conceitos são os mesmos que na álgebra relacional apenas muda a sintaxe.

<TABELA_1> [INNER] JOIN <TABELA_2> ON <COND>

<TABELA_1> LEFT [OUTER] JOIN <TABELA_2> ON <COND>

<TABELA_1> RIGHT [OUTER] JOIN <TABELA_2> ON <COND>

<TABELA_1> FULL [OUTER] JOIN <TABELA_2> ON <COND>


SQL- JOIN

- Exemplos do uso JOIN

Consulta 3: Obtenha os nomes e as moradas de todos os funcionários que trabalham no departamento Research.

```
SELECT F_nome, F_morada  
FROM FUNCIONARIO INNER JOIN DEPARTAMENTO ON F_Dnum = D_num  
Where D_nome = 'Research'
```

Condição de junção vem
após do ON



SQL- JOIN

Consulta 8: Para cada funcionário, recupere o seu nome e o seu supervisor.

```
SELECT F.F_nome, S.F_nome  
FROM FUNCIONARIO AS F, FUNCIONARIO AS S  
WHERE F.F_supervisor = S.F_ident
```

A solução que vimos
sem JOIN

```
SELECT F.F_nome, S.F_nome  
FROM FUNCIONARIO AS F JOIN FUNCIONARIO AS S  
ON F.F_supervisor = S.F_ident
```

Solução com JOIN

SQL- JOIN

Consulta 15: Obtenha a identificação dos funcionários que não têm dependentes.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)
DEPENDENTE(DEPEND_F_ident, DEPEND_nome, DEPEND_dt_nasc, DEPEND_sexo,
DEPEND_relacionamento)

```
SELECT F_ident  
FROM FUNCIONARIO LEFT OUTER JOIN DEPENDENTE ON F_ident = Depend_F_ident  
WHERE Depend_F_ident IS NULL
```

Relembra que a junção externa à esquerda preenche a NULL os atributos da parte direita que não satisfazem a condição de junção.

SQL- JOIN

Consulta 16: Para todos os projetos localizados em Houston, obtenha o nome do projeto e o nome do respectivo gestor do departamento ao qual o projeto está associado.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)
DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)
PROJETO(P_num, P_nome, P_loc, P_Dnum)

```
SELECT P_nome, D_diretor
FROM FUNCIONARIO JOIN DEPARTAMENTO ON F_ident = D_diretor JOIN PROJETO ON P_Dnum = D_num
WHERE P_loc = 'Houston'
```

O uso do join permite queries mais rápidas

```
SELECT P_nome, D_diretor
FROM FUNCIONARIO, DEPARTAMENTO, PROJETO
WHERE F_ident = D_diretor AND P_Dnum = D_num AND P_loc = 'Houston'
```

SQL- Funções de Agregação

- O SQL permite utilizar **funções de agregação** tal como vimos na álgebra.
- Os **valores NULL** existentes nos atributos **não são considerados** pelas funções de agregação.

COUNT(<ATRIB>)

SUM(<ATRIB>)

MAX(<ATRIB>)

MIN(<ATRIB>)

AVG(<ATRIB>)

```
SELECT SUM(F_salario), MAX(F_salario), MIN(F_salario), AVG(F_salario)
FROM FUNCIONARIO
```

SQL- Funções de Agregação

Consulta 17: Obtenha o número de funcionários que trabalham no departamento Research e a respetiva média salarial.

Conta todas as linhas

```
SELECT COUNT(*) AS num_func, AVG(F_salario) as Msalario
FROM FUNCIONARIO JOIN DEPARTAMENTO ON F_Dnum = D_num
WHERE D_nome = 'Research'
```

Consulta 18: Obtenha o número de valores diferentes de salário.

```
SELECT COUNT(DISTINCT F_salario)
FROM FUNCIONARIO
```

Mesmo conceito que vimos anteriormente. Elimina repetições.

SQL- GROUP BY

- Utilizado com as funções de agregação para criação de grupos.

`GROUP BY <ATRIB_1>, ..., <ATRIB_N>`

- Exemplo

Consulta 19: Obtenha o número de funcionários por departamento e a respetiva média salarial.

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo, F_supervisor, F_Dnum)

```
SELECT F_Dnum, COUNT(*) AS funcDep, AVG(F_salario) As MEDsal
FROM FUNCIONARIO
GROUP BY F_Dnum
```

SQL- GROUP BY - HAVING

- Permite **especificar uma condição de seleção** sobre os grupos criados com GROUP BY

GROUP BY <ATRIB_1>, ..., <ATRIB_N>
HAVING <GROUP_COND>

- Equivale à **cláusula WHERE** mas no **contexto do GROUP BY**.
- Os atributos do corpo da cláusula HAVING devem estar em funções de agregação ou devem fazer parte dos atributos de agrupamento.
- Exemplo

SQL- GROUP BY - HAVING

Consulta 20: Qual o nome dos departamento que têm mais que 3 funcionários?

FUNCIONARIO(F_ident, F_nome, F_sobrenome, F_morada, F_dt_nasc, F_salario, F_sexo,
F_supervisor, F_Dnum)

DEPARTAMENTO(D_num, D_nome, D_diretor, D_dt_inicio)

```
SELECT D_nome, COUNT(*) AS funcDep
FROM FUNCIONARIO, DEPARTAMENTO
WHERE F_Dnum = D_num
GROUP BY D_nome
HAVING COUNT(*) > 3
```

SQL- GROUP BY - HAVING

Consulta 21: Para os projetos com mais do que dois funcionários, obtenha o número do projeto, o nome do projeto e o número de funcionários que nele trabalham.

PROJETO(P_num, P_nome, P_loc, P_Dnum)
TRABALHA_EM(TID, Pnum, F_ident, horas)

```
SELECT P_num, P_nome, COUNT(*) numfun
FROM PROJETO, TRABALHA_EM
WHERE P_num = T_Pnum
GROUP BY P_num, P_nome
HAVING COUNT(*) > 2
```

SQL- Consultas Encadeadas

- Existem **consultas** que envolvem dados obtidos por **outras consultas**. Estes casos, são resolvidos pelo uso de **subconsultas** ou **consultas encadeadas**
- Subconsultas são blocos SELECT-FROM-WHERE especificados **dentro de uma clausula WHERE** de forma adequada

```
SELECT ...  
FROM ...  
WHERE ... .. (SELECT ...  
               FROM ...  
               WHERE ... ..);
```

Conectar as duas consultas

SQL- Consultas Encadeadas

- A conexão entre duas consultas pode ser feita com:

[<ATRIB> | <VAL>] [NOT] IN <CONSULTA>

[<ATRIB> | <VAL>] [= | < | <= | > | >= | <>] <CONSULTA>

[<ATRIB> | <VAL>] [NOT] [= | < | <= | > | >= | <>] ALL <CONSULTA>

[<ATRIB> | <VAL>] [NOT] [= | < | <= | > | >= | <>] ANY <CONSULTA>

Se a clausula WHERE for verdadeira, a informação em questão entra na resposta.
Sendo falsa, a informação não é incluída.

SQL- Consultas Encadeadas

Consulta 22: Obtenha os nomes de todos os funcionários que possuem dois ou mais dependentes.

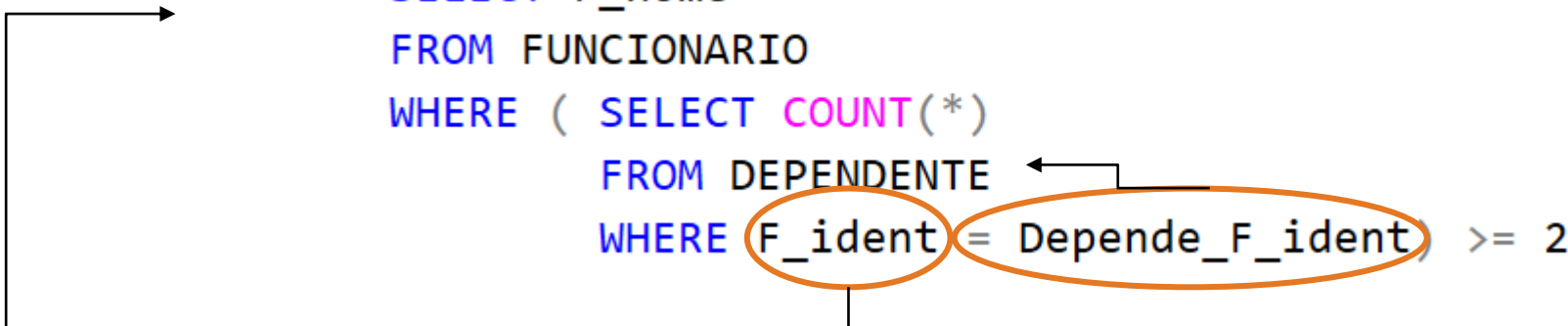
```
SELECT F_nome  
FROM FUNCIONARIO  
WHERE ( SELECT COUNT(*)  
        FROM DEPENDENTE  
        WHERE F_ident = Dependente_F_ident) >= 2
```

Retorna uma linha única com um valor inteiro referente à contagem.

SQL- Consultas Encadeadas

Consulta 22: Obtenha os nomes de todos os funcionários que possuem dois ou mais dependentes.

```
SELECT F_nome  
FROM FUNCIONARIO  
WHERE ( SELECT COUNT(*)  
        FROM DEPENDENTE  
        WHERE F_ident = Dependente_F_ident ) >= 2
```



Repara que como a tabela FUNCIONARIO está a ser chamada na consulta anterior, a subconsulta consegue ir buscar atributos definidos na anterior.

SQL- Consultas Encadeadas

FUNCIONARIO		
<u>F_ident</u>	<u>F_nome</u>	...
1163	Carlos
1164	Maria	...
1165	Pedro	...
1166	Joana	...

DEPENDENTE		
<u>DEPENDE_F_ident</u>	<u>DEPENDE_nome</u>	...
1163	Amanda	...
1164	Fábio	...
1165	João	...
1163	Carlos	...
1164	Joana	...
1163	Teresa	

COUNT para 1163

3

Verdadeiro

COUNT para 1164

2

Verdadeiro

COUNT para 1165

1

Falso

COUNT para 1166

0

Falso

Nota que para cada funcionário, ou seja, para cada linha, é verificado o resultado da subconsulta.

Serão incluídos os dados na resposta **quando a clausula WHERE for verdadeira.**

SQL- Consultas Encadeadas

Consulta 23: Liste os nomes dos funcionários que possuem salários maiores do que os salários de todos os funcionários do departamento 5.

```
SELECT F_nome
FROM FUNCIONARIO
WHERE F_salario > ALL ( SELECT F_salario
                        FROM FUNCIONARIO
                        WHERE F_Dnum = 5)
```

Para cada funcionário a clausula WHERE irá verificar se o salário dele é maior do que todos aqueles que resultaram da consulta seguinte.

Todos os salários dos funcionários do departamento 5.

SQL- Consultas Encadeadas

Consulta 24: Encontre os nomes de cada funcionário que possui um dependente do mesmo sexo que ele.

```
SELECT F_nome  
FROM FUNCIONARIO  
WHERE F_ident IN ( SELECT DEPENDENTE_F_ident  
FROM DEPENDENTE  
WHERE F_sexo = Dependente_Sexo)
```

O IN vai testar se F_ident pertence
ao conjunto obtido na subconsulta

SQL- Consultas Encadeadas

```
]SELECT F_nome  
FROM FUNCIONARIO  
WHERE F_ident IN ( SELECT DEPENDENTE_F_ident  
FROM DEPENDENTE  
WHERE F_sexo = Dependente_Sexo)
```

FUNCIONARIO		
<u>F_ident</u>	F_nome	F_sexo
1163	Carlos	M
1164	Maria	F
1165	Pedro	M
1166	Joana	F

DEPENDENTE		
<u>DEPENDENTE_F_ident</u>	DEPENDENTE_nome	DEPENDENTE_sexo
1163	Amanda	F
1164	Fábio	M
1165	João	M
1163	Carlos	M
1164	Joana	F
1163	Teresa	F

SQL- Consultas Encadeadas

Consulta 25: Obtenha a identificação dos funcionários que trabalham nos projetos 1, 2 ou 3.

TRABALHA_EM(TID, Pnum, F_ident, horas)

```
SELECT DISTINCT T_F_ident  
FROM TRABALHA_EM  
WHERE T_Pnum IN (1,2,3)
```

SQL - EXISTS

- Permite verificar se **o resultado** de uma consulta encadeada e correlacionada **é vazio (não possui nenhuma linha) ou não**.
- Como vimos, uma consulta encadeada é avaliada repetidamente contra cada linha da consulta externa.

[NOT] EXISTS <CONSULTA>

- EXISTS CONSULTA é verdadeiro se CONSULTA não for vazio. Caso contrário é falso.
- Analisemos novamente a consulta 27..

SQL - EXISTS

Consulta 26: Encontre os nomes de cada funcionário que possui um dependente do mesmo sexo que ele.

```
SELECT F_nome
FROM FUNCIONARIO
WHERE EXISTS ( SELECT DEPENDENTE_F_ident
                FROM DEPENDENTE
                WHERE Dependente_Sexo = F_sexo AND F_ident = Dependente_F_ident)
```

A clausula WHERE será verdadeira se o EXISTS for verdadeiro.

Quando é que o EXISTS é verdadeiro?

SQL - EXISTS

FUNCIONARIO		
<u>F_ident</u>	F_nome	F_sexo
1163	Carlos	M
1164	Maria	F
1165	Pedro	M
1166	Joana	F

DEPENDENTE		
DEPENDENTE_F_ident	DEPENDENTE_nome	DEPENDENTE_sexo
1163	Amanda	F
1164	Fábio	M
1165	João	M
1163	Carlos	M
1164	Joana	F
1163	Teresa	F

EXISTS para 1163

1163	Carlos	M
------	--------	---

Verdadeiro

EXISTS para 1164

1164	Joana	F
------	-------	---

Verdadeiro

EXISTS para 1165

1165	João	M
------	------	---

Verdadeiro

EXISTS para 1166



Falso

SQL - EXISTS

Consulta 27: Obtenha o nome dos funcionários que não têm dependentes.

```
SELECT F_nome
FROM FUNCIONARIO
WHERE NOT EXISTS ( SELECT *
                    FROM DEPENDENTE
                    WHERE F_ident = Dependente_F_ident)
```

Quando é que o NOT
EXISTS é verdadeiro?

SQL - EXISTS

FUNCIONARIO		
<u>F_ident</u>	<u>F_nome</u>	<u>F_sexo</u>
1163	Carlos	M
1164	Maria	F
1165	Pedro	M
1166	Joana	F

DEPENDENTE		
<u>DEPENDENTE_F_ident</u>	<u>DEPENDENTE_nome</u>	<u>DEPENDENTE_sexo</u>
1163	Amanda	F
1164	Fábio	M
1165	João	M
1163	Carlos	M
1164	Joana	F
1163	Teresa	F

NOT EXISTS para 1163

1163	Amanda	F
1163	Carlos	M
1163	Teresa	F

FALSO

NOT EXISTS para 1164

1164	Fábio	M
1164	Joana	F

FALSO

NOT EXISTS para 1165

1165	João	M
------	------	---

FALSO

NOT EXISTS para 1166

VERDADEIRO

SQL – Subconsultas VS UPDATE

- As subconsultas podem ser utilizadas em outros contextos utilizando as mesmas regras. Vejamos o exemplo do UPDATE

```
UPDATE FUNCIONARIO
SET F_salario = F_salario *1.1
WHERE F_Dnum IN ( SELECT D_num
                  FROM DEPARTAMENTO
                  WHERE D_diretor = '1163' )
```

Aumento de salário para os funcionários que trabalham para um departamento cujo diretor é o funcionário 1163

SQL – Subconsultas VS DELETE

```
DELETE FUNCIONARIO  
WHERE F_Dnum IN ( SELECT D_num  
                  FROM DEPARTAMENTO  
                  WHERE D_diretor = '1163')
```

Apagar os funcionários que trabalham para um departamento cujo diretor é o funcionário 1163

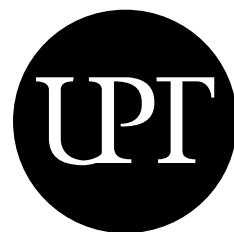
SQL – Data Control Language (DCL)

- Conjunto de instruções que controlam a segurança e o acesso aos dados da tabela.
- **Conceder permissões** (INSERT, UPDATE, DELETE) ao utilizador

```
GRANT privilegio ON nomeRelacao TO utilizador
```

- **Remover permissões**

```
REVOKE privilegio ON nomeRelacao FROM utilizador
```



UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.