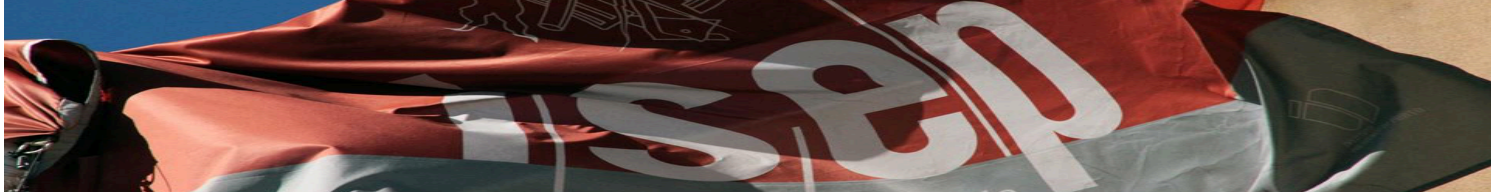


BASE DE DADOS



SQL
Structured Query
Language

Teórico-Práticas
Ano Lectivo 2016/2017



Objetivos

1. SQL – STRUCTURED QUERY LANGUAGE
2. DATA DEFINITION LANGUAGE
 - Tabelas
 - Tipo de Dados
 - Restrições de Integridade
 - Modificação de Tabelas
3. DATA MANIPULACION LAMGUAGE
 - Manipulação de dados em tabelas
 - Comando SELECT (Básico)



SQL- Structured Query Language

- ★ SQL é a linguagem mais usada nas bases dados relacionais;
- ★ Originalmente desenvolvida pela IBM;
- ★ Atualmente é um standard, o mais recente é o SQL:2011;
- ★ SQL é mais que uma linguagem de interrogação estruturada. Inclui características para a definição da estrutura de dados, para alterar os dados de uma base de dados, e para especificar esquemas de segurança. Estas características agrupam-se do seguinte modo:

DDL – Data Definition Language

DML – Data Manipulation Language

DCL – Data Control Language



SQL- Strutured Query Language

SQL

DDL

Data
Definition
Language

CREATE...
ALTER...
DROP...

DML

Data Manipulation Language

Actualização (update operations)

INSERT
UPDATE
DELETE

Consulta (retrieval operations)

SELECT



DATA DEFINITION LANGUAGE– CRIAR TABELAS

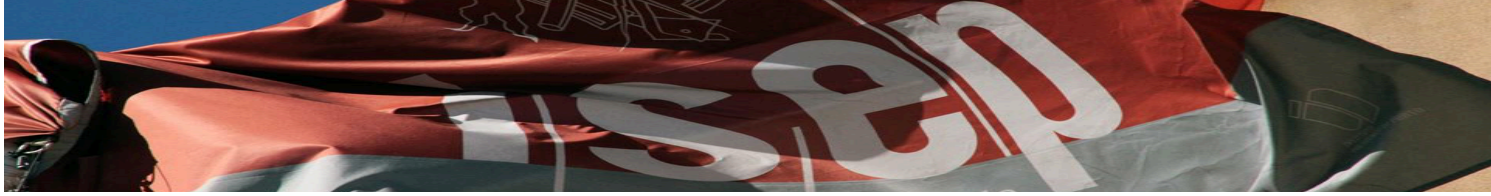
- ★ Na sua forma mais básica é preciso apenas indicar o nome da tabela, os nomes das varias colunas e o tipo de cada uma delas.

```
create table <nometabela> (  
    <nomecoluna>    <tipocoluna >,  
    <nomecoluna>    <tipocoluna>  
);
```



DATA DEFINITION LANGUAGE– TIPO DE DADOS

- ★ **char**(n) ou **character**(n) - Cadeia de caracteres de tamanho fixo n
- ★ **varchar**(n) ou **character varying**(n) - Cadeia de caracteres com tamanho máximo n
- ★ **text** - Cadeia de caracteres sem tamanho definido
- ★ **integer** ou **int** - Números inteiros (4 bytes)
- ★ **number**(precisão, escala) - Números reais sem limite de tamanho
- ★ **timestamp** - Data + hora no mesmo campo
- ★ **Date** - data entre 4712-01-01 AC e 4712-12-31 DC
 - Para forçar conversões, usar **TO_DATE**(campo,formato) e **TO_CHAR**(campo, formato)



DATA DEFINITION LANGUAGE– TIPO DE DADOS

★ Exemplo

```
create table aluno (  
    bi integer ,  
    nome varchar (56) ,  
    médiaCurso number (4 ,2) ,  
    datanascimento date  
);
```



DATA DEFINITION LANGUAGE– VALORES POR OMISSÃO

- ★ Podem ainda ser definidos valores por omissão para cada coluna usando a palavra-chave **default**.

```
create table aluno (  
    bi integer ,  
    nome varchar (56) ,  
    médiaCurso number (4 ,2) default 0,  
    datanascimento date default Sysdate  
);
```




DATA DEFINITION LANGUAGE– RESTRIÇÕES INTEGRIDADE

- ★ Em SQL podem ser definidas restrições de integridade de vários tipos.
- ★ As restrições podem ser de dois tipos:
 - de coluna (referem-se apenas a uma coluna e são descritas em frente à coluna em causa);
 - ou de tabela (referem-se a mais do que a uma coluna e ficam separadas da definição das colunas).



RESTRIÇÕES DE INTEGRIDADE– CHECK

- ★ As restrições do tipo **CHECK** permitem garantir que uma ou mais colunas seguem uma determinada regra que pode ser expressa como uma expressão matemática.
- ★ **Podemos e devemos sempre dar nomes às restrições** para que seja mais fácil identificar a razão pela qual a inserção de dados falha.

```
create table aluno (  
    bi integer ,  
    nome varchar (56) ,  
    médiaCurso number (4 ,2) default 0,  
    Constraint ck_media check (médiaCurso >= 0),  
    datanascimento date  
);
```



RESTRIÇÕES DE INTEGRIDADE– CHECK

- ★ No caso da restrição abranger mais de uma coluna temos de usar uma restrição de tabela.

```
create table aluno (  
    bi integer ,  
    nome varchar (56) ,  
    médiaCurso number (4 ,2) default 0,  
    média_notasExame number (4 ,2),  
    datanascimento date,  
    Constraint ck_mediaExames check (média_notasExame <  
    médiaCurso)  
);
```



RESTRIÇÕES INTEGRIDADE– NOT NULL

- ★ Para garantir que uma coluna não tenha valores nulos podemos usar uma restrição do tipo **not null**

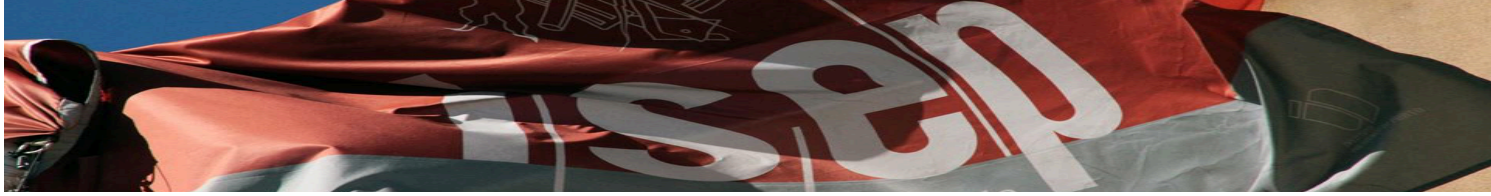
```
create table aluno (  
    bi integer ,  
    nome varchar (56) not null ,  
    médiaCurso number (4 ,2),  
    datanascimento date  
);
```



RESTRIÇÕES DE INTEGRIDADE– CHAVE PRIMÁRIA

- ★ Podemos definir uma, e só uma, chave primária para a tabela.
- ★ Uma chave primária **não pode conter valores nulos nem pode ter valores repetidos.**

```
create table aluno (  
    bi integer primary key,  
    nome varchar (56) not null ,  
    médiaCurso number (4 ,2),  
    datanascimento date  
);
```



RESTRIÇÕES DE INTEGRIDADE– CHAVE PRIMÁRIA

- ★ Uma chave primária pode ser composta por mais do que um atributo e neste caso, devemos usar uma restrição de tabela.

```
create table nota (  
    bi integer,  
    codDisc integer,  
    nota o number (4 ,2),  
    Constraint pk_nota Primary key (bi, codDisc)  
);
```




RESTRIÇÕES DE INTEGRIDADE– CHAVE CANDIDATA

- ★ Chaves candidatas (alternativas) podem ser definidas usando restrições do tipo **unique**.
- ★ Estas restrições são equivalentes às restrições de chave primária mas não obrigam os valores a ser não nulos.


```
create table aluno (  
    bi integer primary key,  
    nif integer unique ,  
    nome varchar (56) not null ,  
    médiaCurso number (4 ,2),  
    datanascimento date,  
);
```



RESTRIÇÕES DE INTEGRIDADE– CHAVE ESTRANGEIRA

- ★ Uma restrição do tipo **foreign key** permite declarar chaves estrangeiras.
- ★ Uma chave estrangeira deve sempre referenciar uma chave primária ou única.

```
create table aluno (  
  bi integer primary key,  
  nif integer unique ,  
  nome varchar (56) not null ,  
  médiaCurso number (4 ,2),  
  datanascimento date,  
  cursoid integer references curso(id)  
);
```



atributo que é
chave primária
na tabela curso



RESTRIÇÕES DE INTEGRIDADE– CHAVE ESTRANGEIRA

- ★ No caso da chave estrangeira ser **composta ou concatenada** (mais de uma coluna) usa-se uma restrição de tabela:

```
create table notasExames (  
    ano_letivo number(4),  
    epoca varchar(20),  
    cod_disc number(4),  
    bi integer,  
    Constraint fk_notasexame Foreign Key  
    (ano_letivo, epoca, cod_disc) references  
    exame_freq(ano_letivo,epoca,cod_disc)  
);
```

```
create table Exame_freq (  
    ano_letivo number(4),  
    epoca varchar(20),  
    cod_disc number(4),  
    data date,  
    Primary key  
    (ano_letivo,epoca,cod_disc)  
);
```



RESTRIÇÕES DE INTEGRIDADE– CHAVE ESTRANGEIRA

- ★ Devemos sempre definir o que acontece quando uma chave estrangeira é violada durante uma operação de remoção ou alteração.

```
create table aluno (  
    bi integer primary key,  
    nif integer unique ,  
    curso integer references curso(id) on delete set null on update  
    cascade, ...  
);
```

Neste exemplo estamos a definir que no caso de um curso ser apagado os alunos que pertençam a esse curso devem ficar com o curso a null. No caso da chave primária do curso ser modificada, essa modificação deve propagar-se e modificar também o curso da tabela aluno.



RESTRIÇÕES DE INTEGRIDADE – CHAVE ESTRANGEIRA

- ★ As alternativas para os valores de **on delete** e **on update** são:
 - **restrict** - Não deixa efetuar a operação ;
 - **cascade** - Apaga os registos associados (delete) ou altera a chave estrangeira (update);
 - **set null** - A chave estrangeira passa a null;
 - **set default** - A chave estrangeira passa a ter o valor por omissão.



MODIFICAÇÃO DE TABELAS

- ★ É possível alterar tabelas

- Acrescentar novas colunas, alterar o tipo de dados e eliminar colunas;
- Acrescentar ou retirar *constraints*.

- ★ Acrescentar uma coluna

- **ALTER TABLE** Aluno **ADD** (email varchar(100))

- ★ Eliminar uma coluna

- **ALTER TABLE** Aluno **DROP** email



Para apagar uma Tabela
DROP TABLE <nometabela>

- ★ Alterar o tipo de dados de uma coluna

- **ALTER TABLE** Aluno **MODIFY** (email varchar(75))



MODIFICAÇÃO DE TABELAS

✳ Eliminar uma constraint

- ALTER TABLE Aluno **DROP CONSTRAINT** FK_Nota_id_aluno_Aluno

✳ Acrescentar uma constraint

- ALTER TABLE Aluno **ADD CONSTRAINT** FK_Nota_id_aluno_Aluno FOREIGN KEY (id_aluno) REFERENCES Aluno(id_aluno)

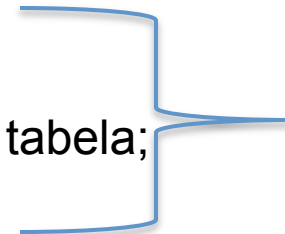
✳ Considerações:

- A sintaxe dos comandos pode variar de SGBD para SGBD;
- Nem todos os SGBD permitem todas as versões do comando ALTER TABLE;
- Quando se adiciona uma *constraint*, terá sempre de ser no formato de uma table_constraint;
- **A alteração de tipo de dados de uma coluna não pode violar regras de integridade.**



DATA MANIPULATION LANGUAGE

- ★ Data Manipulation Language (DML) é utilizada para efetuar operações de seleção, ordenação, cálculo de informação guardada em tabelas, entre outras.
- ★ COMANDOS:
 - INSERT – inserir dados numa tabela;
 - UPDATE – atualiza os dados existentes numa tabela;
 - DELETE – elimina registos numa tabela;
 - SELECT- recuperar dados da base de dados.



Manipulação
de Dados



MANIPULAÇÃO DE DADOS- **INSERT**

- ✳ O comando insert tem como função permitir inserir registo de dados em tabelas.

```
insert into <tabela> <lista de atributos>  
values <Conjunto de tuplos>
```

- ✳ Se forem indicadas o nome das colunas em que queremos inserir os dados, as restantes ficarão com o seu valor por omissão ou nulas.
- ✳ Podemos não indicar o nome das colunas. Neste caso somos obrigados a seguir a mesma ordem pela qual criamos a tabela.

```
INSERT INTO empregado VALUES (12 , ' Joao ' , 500);  
INSERT INTO empregado (id, nome) VALUES (12, 'Joao');  
INSERT INTO empregado VALUES (12 , ' Joao ' , DEFAULT) ;  
INSERT INTO empregado VALUES (12, 'Joao', NULL);
```



MANIPULAÇÃO DE DADOS- **UPDATE**

- ★ O comando update permite modificar os dados numa tabela.

determina sobre
que tabela se vão
executar as
alterações.

update <tabela>

set <Atributo> = <Expressão>, <Atributo> =
<Expressão>, ...

where <Condição>

indica pares de
colunas e valores a
atribuir a colunas.

indica sobre que registo da tabela
serão executas as alterações. A
condição pode ser uma
subconsulta).



MANIPULAÇÃO DE DADOS- **UPDATE**

★ Exemplos

```
UPDATE Departamento  
SET nome='Informática'  
WHERE nome='Civil'
```

```
UPDATE empregado  
SET salario=salario * 1.6
```

```
UPDATE empregado  
SET salario=salario * 1.6, função =“chefe”  
WHERE nome='João'
```



MANIPULAÇÃO DE DADOS- **DELETE**

- ★ Usa-se o comando delete para remover registros das tabelas.

delete from <tabela >
where <Condição>

- ★ Permite apagar apenas registros inteiros. Não é possível apagar um campo com o Delete.

```
DELETE FROM empregado WHERE salario > 2000;
```

```
DELETE FROM empregado WHERE bi = 1234;
```

```
DELETE FROM empregado;
```




ESTRUTURA BÁSICA- SELECT

- ★ O SELECT permite extrair informação de uma Base de Dados ou seja, efetuar consultas à Base de Dados.
- ★ O resultado de uma consulta é uma relação.
- ★ Na sua forma mais básica é preciso apenas indicar a cláusula SELECT e FROM, sendo as restantes opcionais.

```
SELECT [DISTINCT] < lista de atributos>  
FROM < lista de tabelas>  
[ WHERE <critério> ]  
[GROUP BY <grupo_expressão>]  
[HAVING <critério>]  
[ORDER BY <order_expressão> ASC | DESC]
```



ESTRUTURA BÁSICA- SELECT

★ Uma consulta típica da SQL tem a forma:

select A1, A2, ..., An

from r1, r2, ..., rm

where P , onde

- Ai representa um atributo;
- ri representa relação;
- P é um predicado.

Esta consulta é equivalente à expressão da álgebra relacional:

$$\Pi_{A1, A2, \dots, An} (\sigma_P (R1 \times R2 \times \dots \times Rm))$$



ESTRUTURA BÁSICA- SELECT

- ★ SQL permite duplicados nas relações, bem como nos resultados das consultas.
- ★ Para forçar a eliminação de duplicados, deve-se inserir a declaração **DISTINCT** depois do select.

```
SELECT DISTINCT < lista de atributos>
```

```
.....
```

- ★ A cláusula SELECT pode conter expressões aritméticas envolvendo os operadores +, -, *, / e operações em constantes ou atributos dos registos.
- ★ A cláusula SELECT também permite renomear os nomes da tabelas e os nomes dos atributos



ESTRUTURA BÁSICA- SELECT

★ Exemplos

Veiculo (matricula, marca modelo, kms, preço)

matricula	marca	modelo	kms	preço
23-12-QA	OPEL	ASTRA	5500	18900
12-34- QW	RENAULT	CLIO	34000	23000
23-43-TR	RENAULT	MEGANE	1000	30000
78-56-XA	OPEL	ASTRA	12000	12700

SELECT * FROM veiculo

matricula	marca	modelo	kms	preço
23-12-QA	OPEL	ASTRA	5500	18900
12-34- QW	RENAULT	CLIO	34000	23000
23-43-TR	RENAULT	MEGANE	1000	30000
78-56-XA	OPEL	ASTRA	12000	12700



ESTRUTURA BÁSICA- SELECT

★ Exemplos

SELECT marca, modelo FROM veiculo

marca	modelo
OPEL	ASTRA
RENAULT	CLIO
RENAULT	MEGANE
OPEL	ASTRA

SELECT DISTINCT marca, modelo FROM veiculo

marca	modelo
OPEL	ASTRA
RENAULT	CLIO
RENAULT	MEGANE

SELECT matricula, marca, preço*1.20 FROM veiculo

matricula	marca	preço*1.20
23-12-QA	OPEL	22680
12-34- QW	RENAULT	27600
23-43-TR	RENAULT	36000
78-56-XA	OPEL	15240



ESTRUTURA BÁSICA- SELECT

★ Exemplos

**SELECT matricula, marca, preço*1.20 AS novoPreço
FROM veiculo**

matricula	marca	novoPreço
23-12-QA	OPEL	22680
12-34- QW	RENAULT	27600
23-43-TR	RENAULT	36000
78-56-XA	OPEL	15240

**SELECT matricula, Kms AS Quilometros
FROM veiculo**

matricula	Quilometros
23-12-QA	5500
12-34- QW	34000
23-43-TR	1000
78-56-XA	12000

**Não é necessário
em ORACLE**



ESTRUTURA BÁSICA- WHERE

- ★ A cláusula **WHERE** corresponde ao predicado de seleção da álgebra relacional.
- ★ O comando **WHERE** permite selecionar um sub-conjunto de registos de uma relação que satisfazem uma determinada condição sobre alguns atributos.
- ★ A condição pode conter operadores de comparação (<, >, <=, <>, ...) e pode ser composta usando **AND**, **OR** e **NOT**.
- ★ Permite também o uso de **IS NULL** ou **IS NOT NULL**

```
SELECT [DISTINCT] lista-atributos  
FROM lista-tabelas  
WHERE condições
```



ESTRUTURA BÁSICA- WHERE

★ Exemplos

SELECT * FROM veiculo WHERE marca = "OPEL" OR kms > 12000;

matricula	marca	modelo	kms	preço
23-12-QA	OPEL	ASTRA	5500	18900
12-34-QW	RENAULT	CLIO	34000	23000
78-56-XA	OPEL	ASTRA	12000	12700

SELECT * FROM VEICULO WHERE marca IS NULL;

matricula	marca	modelo	kms	preço
-----------	-------	--------	-----	-------

SELECT * FROM empregado WHERE preço/kms > 2;

matricula	marca	modelo	kms	preço
23-12-QA	OPEL	ASTRA	5500	18900



ESTRUTURA BÁSICA- WHERE

- ★ Permite usar o operador **LIKE** em operações com strings.
- ★ O operador LIKE é usado para pesquisas não exatas.
- ★ Faz uso de dois caracteres especiais:

_ (substitui 1 caracter)

% (substitui qualquer sequência de caracteres)

Select * From Carro where marca like '__NA%'

matricula	marca	preço
12-34- QW	RENAULT	23000
23-43-TR	RENAULT	30000



Carro		
matricula	marca	preço
23-12-QA	OPEL	18900
12-34- QW	RENAULT	23000
23-43-TR	RENAULT	30000
78-56-XA	OPEL	12700



ESTRUTURA BÁSICA - GROUP BY E HAVING

- ★ O resultado da cláusula GROUP BY é uma tabela em que as linhas com valores iguais para a lista-atributos-do-grupo são agrupadas.
- ★ A lista-atributos da cláusula SELECT consiste(1) numa lista de nomes e (2) uma lista de termos de agrupadores.
- ★ Todas as colunas da cláusula SELECT têm de estar incluídas na lista-atributos-do-grupo.

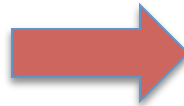
```
SELECT [DISTINCT] lista-atributos  
FROM lista-tabelas  
WHERE condições  
GROUP BY lista-atributos-do-grupo  
HAVING condições-do-grupo
```



ESTRUTURA BÁSICA - GROUP BY E HAVING

- ★ A cláusula HAVING serve para filtrar os valores dos grupos.
- ★ Elimina do resultado os grupos que não satisfazem a condição.
- ★ As condições na cláusula HAVING são aplicadas após a formação dos grupos.

```
SELECT marca, SUM(Kms)  
FROM VEICULO  
GROUP BY marca  
HAVING SUM(kms) < 9000
```



marca	kms
OPEL	5500
RENAULT	1000



ESTRUTURA BÁSICA- ORDER BY

- ★ A resposta a uma pergunta pode ser ordenada segundo uma ou mais colunas.
- ★ Para proceder à ordenação usa-se o comando **ORDER BY** seguido das colunas que queremos ordenar.
- ★ Por omissão as colunas são ordenadas ascendentemente.
- ★ Podemos mudar a direção da ordenação usando as palavras **ASC** e **DESC**.

```
SELECT [DISTINCT] lista-atributos  
FROM lista-tabelas  
WHERE condições  
GROUP BY lista-atributos-do-grupo  
HAVING condições-do-grupo  
ORDER BY <order_expressão> ASC | DESC]
```




ESTRUTURA BÁSICA- ORDER BY

★ Exemplos

**SELECT marca, kms FROM VEICULO
ORDER BY marca DESC, kms ;**

marca	kms
RENAULT	34000
OPEL	5500
OPEL	12000

**SELECT marca, kms
FROM VEICULO
WHERE marca="OPEL"
ORDER BY kms DESC ;**

marca	kms
OPEL	12000
OPEL	5500