Tutorial SQL

1. CRIAR BANCO DE DADOS

```
CREATE DATABASE dbDanielle;
```

2. EXCLUIR BANCO DE DADOS

```
DROP DATABASE dbdanielle;
```

3. CRIAR TABELA

```
CREATE TABLE nome_da_tabela (
  coluna1 tipo (tamanho) restrição,
  coluna2 tipo (tamanho) restrição,
  coluna3 tipo (tamanho) restrição,
  table_constraints
);
```

- Especificamos o nome da tabela depois de create table
- Cada coluna da tabela deve ser separada por vírgula e a última linha não pode terminar com vírgula.
- A definição de cada coluna deve seguir o padrão:

```
nome_da_coluna tipo_do_dado(comprimento_do_dado) restrição
...
nome varchar(100) not null
```

- Por fim, se especifica as constraints da tabela (chave primaria, chave estrangeira, …)

RESTRIÇÕES (CONSTRAINTS):

- NOT NULL – coluna não pode ser nula
- UNIQUE – garante que os valores daquela coluna é único em todas as linhas da tabela
- PRIMARY KEY – Uma chave primária define unicamente uma linha da tabela. Cada tabela pode ter apenas 1 chave primária, porém essa chave primária pode ser composta (formada por mais de uma coluna).
- CHECK – garante que o valor de uma coluna deve satisfazer uma expressão booleana.
- FOREIGN KEY – garante que os valores de uma coluna de uma tabela existem em outra coluna de outra tabela. Uma tabela pode ter mais de uma chave estrangeira.

Exemplos:
Digamos que devemos ter uma tabela para guardar os empregados de uma empresa:

```
CREATE TABLE empregados (
   matrEmp int primary key,
   nomeEmp varchar(90) not null,
   cpfEmp char(11) unique not null,
   emailEmp varchar(50) unique not null,
   datanascEmp date
);
```

Para ver a tabela criada, digite: select * from empregados;

Agora vamos criar a tabela dependentes:

```
CREATE TABLE dependentes (
   cpfDep char(11) primary key,
   nomeDep varchar(90) not null,
   matriculaEmp int not null,
   foreign key (matriculaEmp) references empregados (matrEmp)
);
```

Então, criamos uma tabela dependentes e informamos que sua chave estrangeira apontava para a tabela de empregados na coluna matrEmp.

4. EXCLUIR TABELA

Use o comando DROP TABLE para descartar uma tabela existente. Não será possível descartar uma tabela se ela estiver referenciada por outros objetos, ou se contiver quaisquer dados. Somente o dono pode remover a tabela. Para deixar uma tabela sem linhas, sem removê-la, deve ser usado o comando DELETE ou TRUNCATE. O comando DROP TABLE sempre remove todos os índices, regras, gatilhos e restrições existentes na tabela de destino.

```
DROP TABLE nome_tabela;
```

5. ALTERAR TABELA

O comando ALTER TABLE é usado para alterar a definição de uma tabela. Dentre as várias ações, com esse comando é possível adicionar coluna, remover colunas, mudar o tipo de dado de uma coluna da tabela, dentre outras ações.

EXEMPLOS:

- Para adicionar uma coluna do tipo varchar a uma tabela:

```
ALTER TABLE empregados ADD COLUMN endereco varchar(30);
```

- Para remover uma coluna da tabela:

```
ALTER TABLE empregados DROP COLUMN endereco RESTRICT;
```

- Para mudar o tipo de dado de uma coluna existente:

```
ALTER TABLE empregados
   ALTER COLUMN endereco TYPE varchar(80);
```

- Para mudar o tipo de duas colunas existentes em uma única operação:

```
ALTER TABLE empregados
   ALTER COLUMN endereco TYPE varchar(80),
   ALTER COLUMN nome TYPE varchar(100);
```

- Para mudar o nome da tabela existente:

```
ALTER TABLE nome_da_tabela RENAME TO novo_nome_da_tabela;
```

- Para mudar o nome de uma coluna existente:

```
ALTER TABLE empregados RENAME COLUMN endereco TO cidade;
```

- Para adicionar uma restrição de NOT NULL na coluna:

```
ALTER TABLE produtos ALTER COLUMN cod_prod SET NOT NULL;
```

- Para adicionar uma restrição de UNIQUE na coluna:

```
ALTER TABLE produtos
    ADD CONSTRAINT unq_cod_prod UNIQUE (cod_prod);
```

- Para adicionar uma restrição de chave estrangeira a uma tabela:

```
ALTER TABLE dependentes
   ADD CONSTRAINT fk_depend,
        foreign key (matriculaEmp) references empregados (matrEmp);
```

- Para adicionar uma restrição de chave primária a uma tabela, levando em conta que a tabela pode possuir somente uma única chave primária:

```
ALTER TABLE dependentes ADD PRIMARY KEY (cpfDep);
```

- Para adicionar um valor DEFAUL numa coluna:

```
ALTER TABLE aluno ALTER cidade SET DEFAULT 'Fortaleza';
```

6. INCLUSÃO DE VALORES

O comando para inclusão no banco de dados é o INSERT, que possui a seguinte estrutura:

```
INSERT INTO NOMETABELA(COLUNA1, COLUNA2, COLUNA3....)
VALUES(VALOR1, VALOR2, VALOR3...);
```

OU

```
INSERT INTO NOMETABELA
VALUES(VALOR1, VALOR2, VALOR3...);
```

Exemplo:

```
INSERT INTO curso (codCurso, nomeCur)
VALUES (1, 'ADS'),
(2, 'RC'),
(3, 'CC');
```

OU

```
INSERT INTO curso
VALUES (1, 'ADS'),
(2, 'RC'),
(3, 'CC');
```

Onde:

- Nome_tabela: nome da tabela no qual será inserido os dados.
- Lista-de-campos: nome das colunas que receberão os valores.
- Lista-dados: valores que serão inseridos na tabela. Estes campos devem estar na mesma ordem descrita em lista-de-campos, todos separados por vírgula. Se for utilizado um comando SELECT o mesmo deve retornar a mesma quantidade de colunas com os mesmos tipos de dados especificados em lista-de-campos.

Exemplo:

```
INSERT INTO NOMETABELA(COLUNA1, COLUNA2, COLUNA3....)
VALUES(VALOR1, VALOR2, VALOR3...);

INSERT INTO EMPREGADOS(CODIGO, NOME, SALARIO, SECAO)
VALUES(1, 'HELBERT CARVALHO', 1.500, 1);
```

OU

```
INSERT INTO NOMETABELA
VALUES(VALOR1, VALOR2, VALOR3...);

INSERT INTO EMPREGADOS
VALUES(1,'HELBERT CARVALHO',1500,1);
```

Na segunda opção foi omitida a declaração dos campos. Essa sintaxe funciona somente se for repassado valores para todas as colunas.
Para INSERIR VÁRIOS VALORES ao mesmo tempo, seguir:

```
INSERT INTO tblAutores
VALUES
(1, 'Daniel', 'Barret'),
(2, 'Gerald', 'Carter'),
(3, 'Mark', 'Sobell'),
```

(4, 'William', 'Stanek');

7. ATUALIZANDO DADOS

O comando para atualizar registros é UPDATE, que tem a seguinte sintaxe

```
UPDATE nome_tabela
SET COLUNA = 'novo_valor'
WHERE CONDIÇÃO
```

Onde:

- Nome_tabela: nome da tabela que será modificada
- Campo: campo que terá seu valor alterado
- Novo_valor: valor que substituirá o antigo dado cadastrado em campo
- Where: Se não for informado, a tabela inteira será atualizada
- Condição: regra que impõe condição para execução do comando

Exemplo:

```
UPDATE DEPARTAMENTO
SET SALARIO = 2000
WHERE CODIGODEP = 1
```

No trecho acima, todos os colaboradores que fazem parte do departamento 1 terão o salário alterado para 2000.

```
UPDATE DEPARTAMENTO
SET NOME = 'HEBERT CARVALHO', SALARIO = 2000
WHERE CODIGO = 1
```

Neste exemplo alteramos mais de um campo de uma vez.

8. REMOVENDO DADOS

O comando utilizado para apagar dados é o DELETE.

```
DELETE FROM nome_tabela
WHERE condição
```

Onde:
Nome_tabela: nome da tabela que será modificada
Where: cláusula que impõe uma condição sobre a execução do comando

Exemplo:

```
DELETE FROM EMPREGADOS
WHERE CODIGO = 125
```

9. CONSULTAS

O comando SELECT permite recuperar os dados de um objeto do banco de dados, como uma tabela.
A sintaxe mais básica do comando é:

```
SELECT <lista_de_campos>
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>
```

Exemplo Select Simples:

```
SELECT *
FROM CURSOS

SELECT COD, NOME
FROM CURSOS
```

A cláusula Where permite ao comando SQL passar por condições de filtragem.

Exemplos:

```
SELECT cod, nome
FROM CURSOS
WHERE cod = 1


SELECT codigo, nome
FROM empregado
WHERE salario > 5000


SELECT *
FROM empregado
WHERE sexo = 'F'
```

LIKE

O comando LIKE serve para partes de palavras ou números. O like é case-sensitive, então 'maria' é diferente de 'MARIA'.

O like utiliza muito o coringa % para substituir qualquer quantidade de caracteres:

Para buscar todas as palavras que começam com a letra E.

```
select *
from NOME_TABELA
where nome like 'E%'
```

Buscar todas as palavras que terminam com a letra ‘a’;
select *
from NOME_TABELA
where nomes like '%a';

Buscar todas as palavras que contenham a sílaba ‘li’.
select *
from NOME_TABELA
where nomes like '%li%';

Buscar todos os cliente que o nome inicia com João.

SELECT *
FROM empregado
WHERE nome LIKE ‘João%’

Também podemos usar operadores lógicos para usar mais de uma condição dentro do WHERE.

Neste comando, todos os clientes do gênero feminino com nomes que iniciam com R serão retornados:

SELECT *
FROM empregado
WHERE sexo = ‘F’ AND nome LIKE ‘R%’

Operadores Booleanos
A linguagem SQL utiliza operadores booleanos AND, OR e NOT para especificar uma condição/restrição na cláusula WHERE.

Operadores IN, BETWEEN e NOT BETWEEN.

IN: é utilizado para fazer a filtragem a partir de uma lista de buscas.

SELECT *
FROM empregado
WHERE cidade IN (‘Fortaleza’, ‘Rio de Janeiro’, ‘São Paulo’)

SELECT *
FROM empregado
WHERE matricula IN (1, 2, 3)

BETWEEN: é utilizado para fazer buscas entre intervalos. É mais utilizado para filtrar intervalos de datas e valores

Listar todos os clientes que foram cadastrados entre as datas 12/10/2019 e 31/12/2019’.

```
SELECT *
FROM clientes
WHERE data_cadastro
BETWEEN '2019-10-12' AND '2019-12-31'
```

Listar todos os empregados cujo salário esteja entre 5000 e 8000:

```
SELECT *
FROM Empregado
WHERE Salario BETWEEN 5000 AND 8000
```

Listar todos os alunos cujo nome esteja entre as letras "M" e "P"

```
SELECT nome_aluno
FROM alunos
WHERE nome_aluno BETWEEN 'M' and 'P'
```

Obs: Ao usar o Between pra texto, significa dizer "traga todos os alunos que é maior ou igual a M e menor ou igual a P. Portanto, o nome Paulo é maior que somente a letra P, por isso, ele não retorna esse registro na consulta.

Listar todos os funcionários que não possuam salário entre 2000 e 3000

```
SELECT *
FROM funcionario
WHERE salario NOT BETWEEN 2000 AND 3000
```

O **ORDER BY** é utilizado para ordenação.
Podemos ordenar em ordem crescente (ASC) ou em ordem decrescente (DESC).

Listar todos os clientes ordenados pelo nome em ordem decrescente.

```
SELECT *
FROM clientes
ORDER BY nome DESC
```

```
SELECT *
FROM Produtos
ORDER BY nome_produto;
```

FUNÇÕES DE AGRUPAMENTO

São cinco as funções básicas de agrupamento:

- AVG: Retorna a média do campo especificado

  ```
  SELECT AVG(VALOR)
  FROM PEDIDOS
  ```

- MIN/MAX/SUM: Respectivamente retorna o

- menor valor, o maior e o somatório de um grupo de registros:

```
SELECT MIN(VALOR) FROM PEDIDOS
SELECT MAX(VALOR) FROM PEDIDOS
SELECT SUM(VALOR) FROM PEDIDOS
```

- COUNT: Retorna a quantidade de itens da seleção

```
SELECT COUNT(CODIGO)
FROM CLIENTES
```