SQL CHEAT SHEET

DDL

- Criar (CREATE)
- Destruir (DROP)
- Modificar (ALTER)

DML

- Consultar (SELECT)
- Inserir (INSERT)
- Remover (DELETE)
- Atualizar (UPDATE)

TIPOS DE DADOS

NÚMEROS EXATOS

- INT
- NUMERIC (P,S)
- DECIMAL (P,S)
- BIGINT
- BIT
- SMALLINT

NÚMEROS APROXIMADOS

- FLOAT
- REAL

BINÁRIOS

- BINARY
- VARBINARY
- IMAGE

CARACTERES

- CHAR
- VARCHAR
- TEXT

DATA E HORA

- DATE
- DATETIME
- DATETIME2
- TIME
- SMALLDATETIME

CARACTERES UNICODE

- NCHAR
- NVARCHAR
- NTEXT

DADOS ESPECIAIS

- UNIQUEIDENTIFIER
- TIMESTAMP
- XML
- ROW

OPERADORES ARITMÉTICOS

+ (Somar)	Adição
- (Subtrair)	Subtração
* (Multiplicar)	Multiplicação
/ (dividir)	Divisão
% (módulo)	Retorna o resto inteiro de uma divisão

OPERADORES RELACIONAIS

=	Igual a
<	Menor que
>	Maior que
>=	Maior ou igual a
<=	Menor ou igual a
<> ou !=	Diferente

OPERADORES LÓGICOS

AND	E
OR	Ou
NOT	Não

OPERADORES COMPOSTOS

+=	Adiciona e define
-=	Subtrai e define
*=	Multiplica e define
/=	Divide e define
%=	Cálculo Módulo e define

FUNÇÕES

AVG (expr)	Calcula a média aritmética dos valores dentro do grupo .
COUNT (expr)	Conta quantos valores existem da expressão dada dentro do grupo.
COUNT (*)	Conta quantas linhas existem dentro do grupo.
MAX (expr)	Retorna o máximo valor de expressão dentro do grupo.
MIN (expr)	Retorna o mínimo valor de expressão dentro do grupo.

PALAVRAS-CHAVE

ADD	Adiciona uma coluna em uma tabela existente.
ADD CONSTRAINT	Especifica uma chave primária, chave estrangeira, referencial, exclusiva ou restrição de verificação em uma coluna nova, existente ou em um conjunto de colunas.
AS	Renomear um campo (coluna) ou uma tabela temporariamente.
LIKE e NOT LIKE	Pesquisa um padrão especificado em uma coluna
JOIN	Combina colunas de uma ou mais tabelas.
SET	Especifica quais colunas e valores devem ser atualizados em uma tabela.
UNIQUE	Garante que todos os valores em uma coluna sejam diferentes.
UNION	Combina o conjunto de resultados de duas ou mais instruções.

PET-SI
Universidade Federal do Ceará

SQL CHEAT SHEET

CLÁUSULAS/COMANDOS

SELECT
- Lista os campos que contêm dados de interesse

FROM
- Lista as tabelas que contêm os campos listados na cláusula SELECT.

WHERE
- Especifica critérios de campo que devem ser atendidos por cada registro a ser incluído nos resultados.

ORDER BY
- Especifica como classificar os resultados.

GROUP BY
- Em uma SQL que contém funções agregadas, agrupa diversos registros com base em uma ou mais colunas de uma tabela

HAVING
- Em uma SQL que contém funções agregadas, especifica condições que se aplicam aos campos resumidos na instrução SELECT.

DISTINCT
- Elimina tuplas duplicadas do resultado de uma consulta.

INSERT
- Adiciona registros a uma tabela.

UPDATE
- Atualiza os registros já inseridos.

DELETE
- Exclui registros de uma tabela.

CREATE
- Cria novas tabelas .

ALTER
- Utilizado para alterar uma tabela já existente.

DROP
- Remove uma tabela ou o banco de dados.

RENAME
- Renomeia uma tabela criada, uma coluna em uma tabela ou um banco de dados.

LIMIT
- Especifica o número de linhas que devem ser retornadas no resultado de uma consulta..

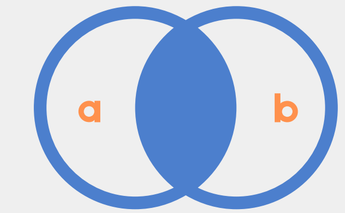
BETWEEN e NOT BETWEEN
- Especifica um intervalo a ser testado, substituem o uso dos operadores <= e =>.

IN e NOT IN
- Procuram dados que estão ou não contidos em um dado conjunto de valores.

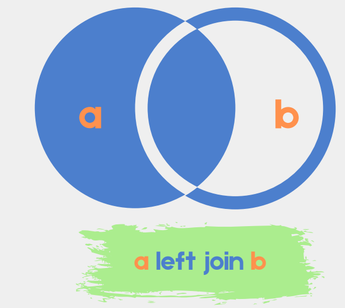
IS NULL e NOT NULL
- Identificam se o atributo tem o valor nulo (não informado) ou não.

EXISTS E NOT EXISTS
- Especifica uma subconsulta a ser testada quanto à existência(ou não) de linhas.

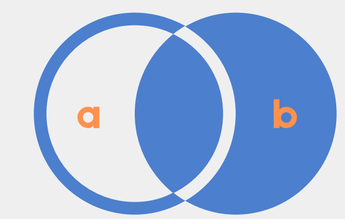
JOINS



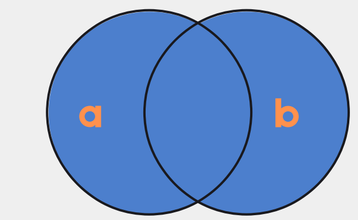
a inner join b



a left join b



a right join b



a full outer join b

SQL CHEAT SHEET

CREATE

- **CRIAR UMA TABELA**

```
CREATE TABLE < nome_da_tabela>(
            nome_atributo_1     tipo_01,
            nome_atributo_2     tipo_02,
            ...
            nome_atributo_n      tipo_n );
```

- **CRIAR UMA TABELA (COM CHAVE PRIMÁRIA)**

```
CREATE TABLE < nome_da_tabela>(
            nome_atributo_1     tipo_01,
            nome_atributo_2     tipo_02,
            ...
            nome_atributo_n      tipo_n,
           CONSTRAINT <nomedatabela_pkey>
           PRIMARY KEY (<nome dos atributos>) );
```

- **CRIAR UMA TABELA (COM CHAVE SECUNDÁRIA)**

```
CREATE TABLE < nome_da_tabela>(
            nome_atributo_1     tipo_01,
            nome_atributo_2     tipo_02,
            ...
            nome_atributo_n      tipo_n,
           CONSTRAINT <nomedatabela_pkey>
           PRIMARY KEY (<nome dos atributos>),
           CONSTRAINT <nomedatabela_fkey>
           FOREIGN KEY (< atributo>) REFERENCES
           < nome_da_outra_tabela>
          (< chave_primaria_da_outra_tabela>));
```

- **CRIAÇÃO DE ÍNDICES EM UMA TABELA EXISTENTE (INDEX)**

```
CREATE [UNIQUE] INDEX  < nome>  ON <tabela> ( <atributo1> ,
<atributo2>...);
```

- **CRIAR VISÕES (VIEW)**

```
CREATE VIEW  < nome da visão>   AS SELECT  coluna1, coluna2, ...
coluna_n FROM <nome_tabela> WHERE <condição>;
```

NOT EXISTS (PÁG. 2)

- **O comando CREATE TABLE IF NOT EXISTS <nome_tabela > ... somente irá criar a tabela caso ela não exista no banco de dados.**

UNIQUE (PÁG.1)

- **Coluna (só é abrangida uma coluna)**
  Ex: CONSTRAINT <nome> UNIQUE
- **Tabela (são abrangidas várias colunas)**
  Ex: CONSTRAINT <nome> UNIQUE (coluna1,coluna2,...)

```
CREATE TABLE  Pessoas(
            ID  int NOT NULL ,
            Nome     varchar (100) NOT NULL,
            Sobrenome varchar (100),
            Data_nascimento  date,
            CONSTRAINT  UC_PESSOAS UNIQUE (ID , Nome));
```

SERIAL

- **A chave primária pode ser definida por auto-numeração. Assim, a chave inteira é incrementada de 1 a cada nova inserção.**

```
CREATE TABLE  Pessoas(
            ID   SERIAL ,
            Nome     varchar (100) NOT NULL,
            Sobrenome varchar (100),
            Data_nascimento  date,
            CONSTRAINT  pessoas_pkey  PRIMARY KEY (ID));
```

DICA

OR REPLACE :

A instrução CREATE OR REPLACE VIEW... verifica se já existe uma view com o mesmo nome , e caso exista, fará a substituição pela mais recente.

OBS: OR REPLACE não remove colunas de uma view.

SQL CHEAT SHEET

ALTER

- **ALTERAR UMA TABELA EXISTENTE**

```
ALTER TABLE < ação>;
```

- **<ação> pode ser:**

  Adicionar/remover uma coluna na tabela ou uma restrição de integridade.

- **ACRESCENTAR UMA COLUNA NA TABELA**

```
ALTER TABLE <nome_da_tabela> ADD
<nome_da_coluna> <tipo_do_dado>;
```

- **REMOVER COLUNA DA TABELA**

```
ALTER TABLE < nome_da_tabela> DROP
<nome_da_coluna> ;
```

- **ALTERAR O TIPO DE UMA COLUNA DA TABELA**

```
ALTER TABLE < nome_da_tabela> ALTER COLUMN
<nome_da_coluna> TYPE <novo_tipo> ;
```

- **RENOMEAR UMA COLUNA**

```
ALTER TABLE < nome_da_tabela> RENAME COLUMN
<nome_da_coluna> TO <novo_nome> ;
```

- **RENOMEAR UMA TABELA**

```
ALTER TABLE < nome_da_tabela> RENAME TO <novo_nome> ;
```

- **SET/DROP DEFAULT**
  - **Estas formas definem ou removem o valor padrão para a coluna.**
  - **As linhas existentes na tabela não são modificadas. O valor padrão somente é aplicado aos comandos INSERT subsequentes.**

- **SETANDO UM "VALOR" PADRÃO AO ATRIBUTO**

```
ALTER TABLE <nome_da_tabela> ALTER COLUMN <nome_da_coluna>
SET DEFAULT <valor_padrão>;
```

- **REMOVENDO UM "VALOR" PADRÃO DO ATRIBUTO**

```
ALTER TABLE<nome_da_tabela> ALTER COLUMN <nome_da_coluna>
DROP DEFAULT ;
```

EXEMPLO

```
CREATE TABLE Contatos (

Id INTEGER,
Nome VARCHAR(50),
Endereco VARCHAR(50),
Email VARCHAR(50));
```

- **RENOMEAR A TABELA**

```
ALTER TABLE contatos RENAME TO CLIENTES ;
```

- **ALTERAR O TIPO DE UMA COLUNA DA TABELA**

```
ALTER TABLE CLIENTES ALTER COLUMN ID TYPE
VARCHAR ;
```

- **ACRESCENTAR UMA COLUNA NA TABELA**

```
ALTER TABLE CLIENTES ADD TELEFONE INTEGER;
```

SQL CHEAT SHEET

INSERT

O COMANDO INSERT

É essencial para inserirmos dados em uma tabela. Com ele, podemos adicionar um ou vários registros ao mesmo tempo, além disso podemos indicar em quais campos os dados serão inseridos.

INSERT INTO

O INSERT INTO é utilizado para inserir dados em uma tabela. Sendo necessário indicar quais campos serão inseridos e seus valores correspondentes.

- **INSERIR VALOR NO ATRIBUTO**

```
INSERT INTO <nome_da_tabela>
(coluna1, coluna2, ..., colunaN) VALUES
(valor_da_coluna1,valor_da_coluna2,...
valor_da_colunaN);
```

SQL INSERT INTO + SELECT

É utilizado para inserir os dados de uma tabela em outra. Na prática é: selecione determinados campos da tabela "X" e insira esses valores na tabela "Y".

- **INSERIR DADOS DE UMA TABELA PARA OUTRA**

```
INSERT INTO <nome_tabela_destino> SELECT (coluna1, coluna2, ..., colunaN) FROM <nome_tabela_origem> WHERE <condição>;
```

<CONDIÇÃO>

- Caso exista algum critério de seleção dos dados.
- <nome_do_atributo> <operador> <valor>

ATENÇÃO !

O comando VALUES são os valores que correspondem aos campos que você informou no primeiro parêntese (coluna1, coluna2, ..., colunaN) , ou seja, os dados que serão inseridos nesses campos. Logo, é preciso seguir a ordem que você especificou os campos .

EXEMPLO

```
INSERT INTO ALUNO (nome_aluno,
idade) VALUES
  ('Ana Maria', 30),
  ('João Pedro', 25),
  ('Bruno Sales', 22 );
```

OBSERVE !

Após o VALUES nos valores dos campos, foram utilizadas aspas simples já que os valores são STRINGS, isso é necessário porque os campos do tipo VARCHAR ou CHAR são considerados STRINGS nos banco de dados.

SQL CHEAT SHEET

UPDATE

- **SINTAXE DO COMANDO UPDATE**

```
UPDATE <nome_da_tabela> SET
<nome_da_coluna1> = < valor_da_coluna1>,
< nome_da_coluna2> = <valor_da_coluna2>
WHERE <condição>;
```

- **ATUALIZAR VÁRIAS LINHAS DA TABELA**

```
UPDATE <nome_da_tabela SET <nome_da_coluna> =
<novo_valor_da_coluna> WHERE
<nome_da_coluna> = <valor_da_coluna>;
```

- **ATUALIZAR UMA LINHA DA TABELA**

```
UPDATE <nome_da_tabela SET
<nome_da_coluna> = <novo_valor_da_coluna>
WHERE <condição>;
```

EXEMPLIFICANDO:

```
UPDATE FUNCIONARIO SET CARGO =
'PROGRAMADOR' WHERE ID = 4 ;
```

OPERADORES OR, AND e NOT + WHERE

- Possibilita combinar várias condições e realizar uma filtragem mais apurada na tabela .

EXEMPLIFICANDO:

```
UPDATE < nome_da_tabela > SET
 < nome_da_coluna > = < valor_da_coluna >
WHERE < condição > AND < condição > ;
```

EXEMPLIFICANDO:

```
UPDATE FUNCIONARIO SET NOME = 'DAYANA
PRISCILLA COSTA' WHERE NOME = 'DAYANA' AND
CARGO = 'PROGRAMADOR' ;
```

BOA PRÁTICA!

Ao utilizar o comando UPDATE é recomendável a especificação da cláusula WHERE. Ainda que a cláusula seja opcional e que possa ser omitida no código, é sempre bom utilizá-la.

- A não ser em casos específicos em que você realmente precise atualizar todas as linhas da coluna de uma tabela.

EXEMPLO

TABELA: FUNCIONARIO

ID	NOME	CARGO	SALARIO
1	VIVIANA MARIA	PROFESSORA	3.350.00
2	CAMILA VITÓRIA	MÉDICA	13.350.00
3	FELIPE SANTOS	DESENVOLVEDOR	5.550.00
4	DAYANA PRISCILLA	DBA	3.150.00

```
UPDATE Funcionario SET cargo = 'PROGRAMADOR' WHERE id = 4 ;
```

ID	NOME	CARGO	SALARIO
1	VIVIANA MARIA	PROFESSORA	3.350.00
2	CAMILA VITÓRIA	MÉDICA	13.350.00
3	FELIPE SANTOS	DESENVOLVEDOR	5.550.00
4	DAYANA PRISCILLA	PROGRAMADOR	3.150.00

```
UPDATE Funcionario SET cargo = 'PROGRAMADOR' ;
```

ID	NOME	CARGO	SALARIO
1	VIVIANA MARIA	PROGRAMADOR	3.350.00
2	CAMILA VITÓRIA	PROGRAMADOR	13.350.00
3	FELIPE SANTOS	PROGRAMADOR	5.550.00
4	DAYANA PRISCILLA	PROGRAMADOR	3.150.00

OBSERVE !

Sem a cláusula WHERE todas as linhas da coluna CARGO foram atualizadas. Para que isso não aconteça, quando essa não é a intenção, não esqueça de especificar a cláusula WHERE.

SQL CHEAT SHEET

SELECT

- **RETORNAR DADOS DE UMA TABELA**

```
SELECT <lista_de_atributos> FROM <nome_da_tabela>;
```

- **RETORNAR TODOS OS DADOS DE UMA TABELA**

```
SELECT * FROM <nome_da_tabela>;
```

- **SELECT + WHERE**

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <condição>;
```

- **SELECT + LIKE** (FILTRO DE TEXTO)

1.BUSCAR DADOS QUE COMEÇAM COM UMA DETERMINADA LETRA

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE 'LETRA%';
```

2.BUSCAR DADOS QUE TERMINAM COM UMA DETERMINADA LETRA

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE '%LETRA';
```

3.BUSCAR DADOS POR UM TERMO EM QUALQUER POSIÇÃO DO CAMPO

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE '%TERMO%';
```

4.BUSCAR DADOS PELA SEGUNDA LETRA DE UM DETERMINADO CAMPO

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE '_LETRA%';
```

5.BUSCAR DADOS COM CARACTERES ENTRE OUTRAS LETRAS

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE '% LETRA__LETRA%';
```

6.BUSCAR DADOS COM PALAVRAS QUE COMEÇAM E TERMINAM COM DETERMINADO CARACTERE

```
SELECT <lista_de_atributos> FROM <nome_da_tabela> WHERE <nome_do_atributo> LIKE 'LETRA%LETRA';
```

OBSERVE !

O caractere sublinhado "_" pode ser utilizado para substituir um caractere como também mais de um.

EXEMPLIFICANDO:

TABELA : FUNCIONARIO (PÁG 6)

```
SELECT ID,NOME,SALARIO FROM FUNCIONARIO WHERE NOME LIKE '%V_____A%';
```

O comando acima, foi realizado com cinco caracteres sublinhado "_". Indicando que entre a letra "V" e a letra "A" deve ter rigorosamente cinco caracteres entre elas.

RESULTADO:

ID	NOME	SALARIO
1	VIVIANA MARIA	3.350.00
2	CAMILA VITÓRIA	13.350.00

- **RETORNAR QUANTIDADE DE ITENS DA SELEÇÃO**

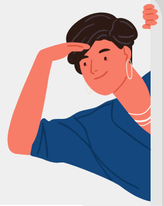
```
SELECT COUNT(<nome_da_coluna>) FROM <nome_da_tabela> WHERE <condição>;
```

- **RETORNAR VALORES DISTINTOS** (NÃO RETORNA VALORES DUPLICADOS NO RESULTADO DA CONSULTA)

```
SELECT DISTINCT(<nome_da_coluna>) FROM <nome_da_tabela> ;
```

SQL CHEAT SHEET

SELECT - ORDEM LÓGICA DE EXECUÇÃO



VOCÊ SABIA?

O comando SELECT é o comando mais usado em SQL, com ele é possível realizar consultas aos dados que pertencem a uma determinada tabela.

Por isso, é muito importante conhecer bem o comando SELECT e entender a ordem lógica que este comando é executado internamente.

Um dos motivos no qual tenho certeza que você vai gostar de aprender sobre isso, é que você vai querer criar queries fenomenais.

Vamos lá?

Ter em mente como funciona o comando SELECT internamente vai te ajudar a evitar erros nas suas consultas.

Você deve estar se perguntando: "Como assim?"

É que por causa desta ordem lógica não podemos dar um ALIAS para uma coluna na cláusula SELECT e fazer referência ao ALIAS na cláusula WHERE. Pois, na ordem de execução da query, o WHERE é executado antes do SELECT, ou seja, quando o WHERE está sendo executado o ALIAS não existe ainda.

Observou como é bom entender bem as coisas ? Veja que não pular esse passo vai te poupar de erros .

Você verá agora duas ordens corretas por óticas diferentes.

São elas:

- Ordem da Sintaxe = ordem da escrita do comando.
- Ordem de Execução = ordem do processamento da query.

ORDEM DE SINTAXE

1 - SELECT
2 - FROM
3 - WHERE
4 - GROUP BY
5 - HAVING
6 - ORDER BY

ORDEM DE EXECUÇÃO

1 - FROM
2 - WHERE
3 - GROUP BY
4 - HAVING
5 - SELECT
6 - ORDER BY

ALIAS

Um ALIAS pode ser uma boa solução na hora de lidar com tabelas ou colunas com nomes muitos longos ou complexos.

Como o ALIAS serve para simplificar, renomeando um campo(coluna) ou uma tabela temporariamente, geralmente ele é curto e simples.

- Substituindo nomes de tabelas:

```
SELECT ALIAS.<nome_da_coluna>
FROM<nome_da_tabela> ALIAS
WHERE <condição>;
```

EXEMPLIFICANDO:

```
SELECT F.nome FROM Funcionario
F WHERE F.salario > 3000;
```

OBSERVE !

O ALIAS para substituir nomes de tabelas são definidos na cláusula FROM .

SQL CHEAT SHEET

DELETE

O COMANDO DELETE

É o comando de linguagem de manipulação de dados (DML). Ele é usado para excluir alguns ou todos os registros armazenados em uma tabela do banco de dados.

- **EXCLUINDO DADOS DE UMA TABELA**

```
DELETE FROM <nome_da_tabela> WHERE
<condição>;
```

EXEMPLIFICANDO:

```
DELETE FROM FUNCIONARIO WHERE SALARIO >
5000.00;
```

RESULTADO

Todos os funcionários com o salário maior que R$ 5000,00 serão excluídos.

ATENÇÃO !

Tenha muito cuidado ao utilizar o comando DELETE sem a cláusula WHERE, porque TODOS os registros armazenados em uma tabela serão excluídos.

DROP

O COMANDO DROP

É o comando de linguagem de definição de dados (DDL). Ele é usado para remover todo o esquema, tabela ou restrição do banco de dados.

- **REMOVENDO UMA TABELA**

```
DROP TABLE <nome_da_tabela> ;
```

EXEMPLIFICANDO:

```
DROP TABLE FUNCIONARIO;
```

RESULTADO

A tabela <Funcionario> será removida juntamente com todas as suas informações .

ATENÇÃO !

Tenha muito cuidado ao utilizar o comando DROP TABLE, porque uma vez que uma tabela é excluída, todas as suas informações também serão perdidas PARA SEMPRE .