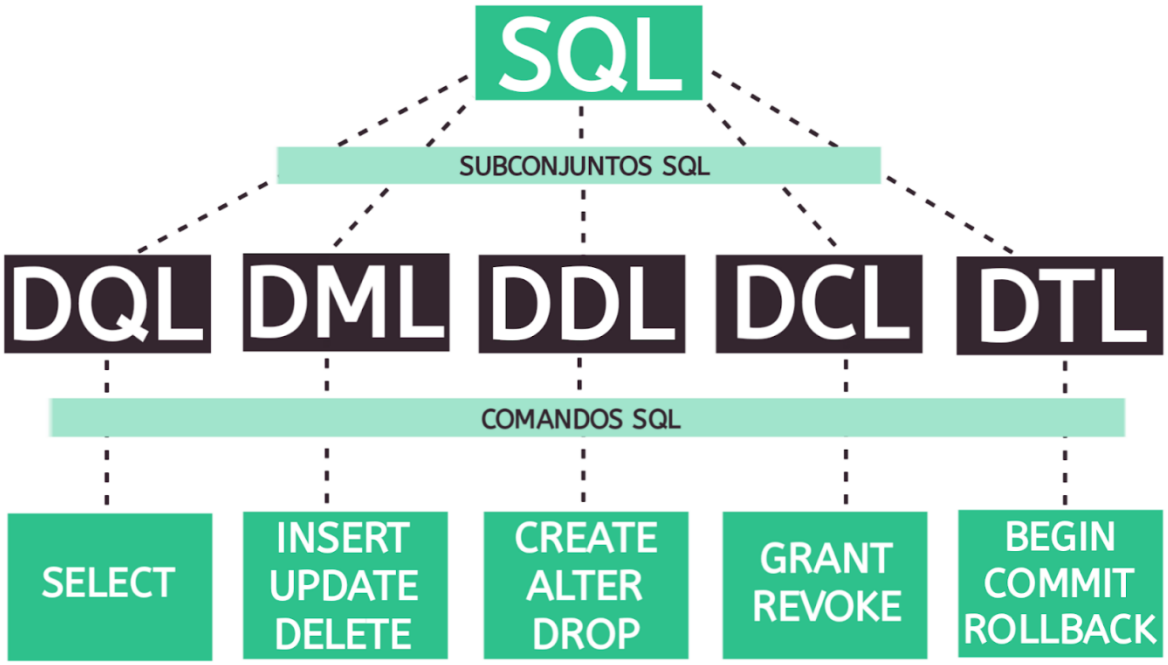
RESUMÃO SQL

Feita por: Guilherme Henrique Almeida da Silva

1 O QUE É O SQL?

SQL – STRUCTURED QUERY LENGUAGE ou Linguagem de consulta estruturada é a linguagem de programação padrão para trabalhar com banco de dados relacionais, permitindo gerenciar e manipular dados e possui grande poder de consulta!

1.1 Imagem do SQL e seus subconjuntos de linguagem



- **DQL -> DATA QUERY LANGUAGE**
- **DML -> DATA MANIPULATION LANGUAGE**
- **DDL -> DATA DEFINATION LANGUAGE**
- **DCL -> DATA CONTROL LANGUAGE**
- **DTL -> DATA TRANSACTION LANGUAGE**

Obs.: Geralmente só trabalhamos com DDL e DML, o SELECT fica dentro de DML.

2 O QUE É O DDL?

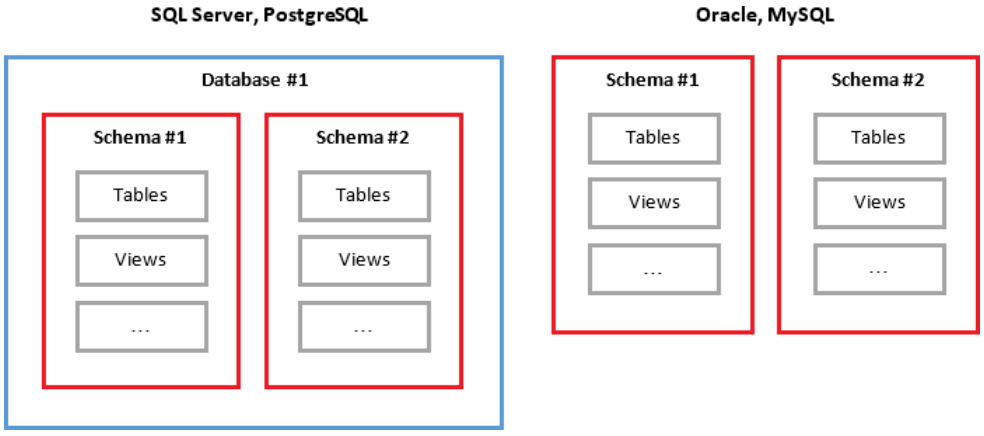
DDL –> DATA DEFINATION LANGUAGE - É a maneira como o banco de dados será definido, ou seja, é a definição da forma como os dados são estruturados e contém os comandos que interagem com os objetos do banco, isto corresponde a criação de DATABASE, SCHEMA e TABELAS através das operações do DDL.

2.1 OPERAÇÕES DDL

Antes de falarmos das operações do DDL, é necessário entender a diferença entre o DATABASE e o SCHEMA.

DATABASE -> é o contêiner geral para todos os dados, ou seja, é o contêiner maior que contém todos os dados e objetos necessários para a aplicação ou conjuntos de aplicações.

SCHEMA -> é a organização lógica dentro de um banco de dados para agrupar objetos relacionados, utilizado para agrupar tabelas, funções etc. Permitindo que diferentes grupos de objetos sejam mantidos separados dentro do mesmo banco de dados. Dependendo do SGBD, faz função de DATABASE.



2.2 CREATE

COM CREATE podemos criar o DATABASE (BANCO DE DADOS), SCHEMAS, TABELAS.

- **Sintaxe para a criação de um DATABASE (base de dados):**

1 - CREATE DATABASE <nome_do banco>;

EXEMPLO:

1- CREATE DATABASE mercado**;**

- **Sintaxe para a criação de relações(tabelas):**

1- CREATE TABLE <nome_da_tabela> (

2- Nome_coluna tipo de dado(domínio) constraints(restrição),

3- Nome_coluna tipo de dado(domínio) constraints (restrição),

4- Nome_coluna tipo de dado(domínio) constraints (restrição)

5-);

Obs.: A criação de colunas(campos) e seu devidos dados (domínios e restrições) devem estar dentro dos parênteses (...) e o ponto e vírgula " ; " no final é obrigatório.

EXEMPLO:

1 – CREATE TABLE produto (

2 - id_produto INT PRIMARY KEY,

3 - nome VARCHAR(45) NOT NULL,

4 - preco FLOAT NOT NULL

5 -);

- **Sintaxe para a criação de SCHEMA:**

1- CREAT SCHEMA <nome_do_SCHEMA>;

EXEMPLO:

1- CREATE SCHEMA setor_financeiro;

2.3 ALTER

O ALTER permite alterar permite alterar DATABASE, SCHEMA, TABELAS. Geralmente usamos para a alteração de relações(tabelas).

- **Sintaxe básica para a alteração de tabelas:**

1 – ALTER TABLE <nome_da_tabela><comandos> <ação específicas>;

Comandos:

ADD COLUMN novo_atributo tipo de dado constraints(restrição de coluna)

ADD CONSTRAINTS nome da restrição constraints (restrição de tabela ou coluna)

DROP COLUMN nome da coluna contrainsts(restrições de chave se houver [RESTRICT, CASCADE])

OBS 1.: RESTRIÇÕES DEVEM SER ANALISADAS SE É NECESSÁRIA COLOCAR OU NÃO.

Ações específicas:

CASCADE: Atualiza ou exclui os registros da tabela filha automaticamente, ao atualizar ou excluir um registro da tabela pai, ou seja, A opção CASCADE permite excluir ou atualizar os registros relacionados presentes na tabela filha automaticamente, quando um registro da tabela pai for atualizado (**ON UPDATE**) ou excluído (**ON DELETE**). É a opção mais comum aplicada;

RESTRICT: Rejeita a atualização ou exclusão de um registro da tabela pai, se houver registros na tabela filha, ou seja, impede que ocorra a exclusão ou a atualização de um registro da tabela pai, caso ainda haja registros na tabela filha. Uma exceção de violação de chave estrangeira é retornada. A verificação de integridade referencial é realizada antes de tentar executar a instrução UPDATE ou DELETE;

SET NULL: Define como NULL o valor do campo na tabela filha, ao atualizar ou excluir o registro da tabela pai;

SET DEFAULT: Define o valor da coluna na tabela filha, como o valor definido como default(padrão) para ela, ao excluir ou atualizar um registro na tabela pai.

NO ACTION: Essa opção equivale à RESTRICT, porém a verificação de integridade referencial é executada após a tentativa de alterar a tabela. É a opção padrão, aplicada caso nenhuma das opções seja definida na criação da chave estrangeira.

OBS.2: AVALIE SE É NECESSÁRIO COLOCAR AS AÇÕES ESPECÍFICAS.

EXEMPLO:

1 – ALTER TABLE produto **ADD COLUMN** quantidade INT NOT NULL;

- Também tem como modificar e alterar as colunas da tabela, veja a sintaxe básica:

1- ALTER TABLE <nome_da_tabela>**MODIFY COLUMN** <nome_da_coluna> <novo tipo de dado> <restrição><ações específicas>;

EXEMPLO:

2 – ALTER TABLE produto MODIFY COLUMN preco DECIMAL NOT NULL NO ACTION**;**

2.4 DROP

Usamos o DROP para a deleção de DATABASE (Banco de dados), SCHEMA, TABELAS, COLUNAS E RESTRIÇÕES.

- Sintaxe para a deleção de SCHEMA (base de dados):

1 - DROP DATABASE <nome_do_banco> <ação>;

Ações:

CASCADE: Apaga o (DATABASE) e todos os seus elementos relacionados;

RESTRICT: Só apaga o (DATABASE) se ele não contiver nenhum elemento ou elementos relacionados a ele;

EXEMPLO:

1- DROP DATABASE mercado CASCADE;

OBS.1: QUANDO O BANCO DE DADOS É DELETADO, TODOS OS DADOS QUE ESTÃO CONTIDO NELE TAMBÉM É DELETADO.

OBS.2: AVALIE SE É NECESSÁRIO COLOCAR AS AÇÕES, CASCADE OU RESTRICT.

- **Sintaxe para a deleção de TABELAS (RELAÇÕES):**

1 – DROP TABLE <nome_da_tabela> <ações>;

Ações:

CASCADE: Atualiza ou exclui os registros da tabela filha automaticamente, ao atualizar ou excluir um registro da tabela pai, ou seja, A opção CASCADE permite

excluir ou atualizar os registros relacionados presentes na tabela filha automaticamente, quando um registro da tabela pai for atualizado (**ON UPDATE**) ou excluído (**ON DELETE**). É a opção mais comum aplicada;

RESTRICT: Rejeita a atualização ou exclusão de um registro da tabela pai, se houver registros na tabela filha, ou seja, impede que ocorra a exclusão ou a atualização de um registro da tabela pai, caso ainda haja registros na tabela filha. Uma exceção de violação de chave estrangeira é retornada. A verificação de integridade referencial é realizada antes de tentar executar a instrução UPDATE ou DELETE;

EXEMPLO:

1- DROP TABLE produto CASCADE ON DELETE;

OBS 1.: TODOS OS DADOS DA TABELA E A TABELA SERÃO EXCLUÍDOS, DELETADOS, MAS O BANCO DE DADOS AINDA EXISTIRÁ.

OBS 2.: AVALIE SE É NECESSÁRIO COLOCAR AS AÇÕES!

- **Sintaxe de como deletar um SCHEMA:**

1- DROP SCHEMA <nome_do_SCHEMA> <ações>;

Ações:

CASCADE: Vai apagar o SCHEMA e todos os elementos relacionados a ele.

RESTRICT: Vai apagar o SCHEMA se ele não estiver nem um elemento ou outros elementos relacionados.

EXEMPLO:

1-DROP SCHEMA setor_financeiro;

2.5 TRUNCATE

Com o TRUNCATE podemos excluir todos os registros(linha) de uma TABELA, em uma única instrução.

Serve como a query **DELETE** do DML sem a **CLÁSULA WHERE**. Ou seja, não é realizado um registro de log de cada linha excluída. Por isso, é preciso tomar muito

cuidado ao decidir usar esse comando. Dependendo do SGBD usado, não é possível realizar **ROLLBACK** após executar o TRUNCATE.

- **Sintaxe básica do comando TRUNCATE:**

  1- TRUNCATE TABLE <nome_da_tablea>;

EXEMPLO:

1- TRUNCATE TABLE produtos;

3 O QUE É CONSTRAINTS?

São restrições aplicadas em uma coluna(campo/atributo) da tabela, visando limitar os tipos de dados (domínio) que serão inseridos, podem ser especificados no momento da criação da tabela (CREAT) ou depois dela ter sido criada (ALTER).

Nos tópicos abaixo temos as principais constrainsts (restrições.)

3.1 UNIQUE

UNIQUE é uma restrição, cujo aquela coluna(campo/domínio) terá dados com apenas um único valor, ou seja, os valores não podem ser repetidos.

EXEMPLO:

```
1- ALTER TABLE funcionario ADD COLUMN especialidade VARCHAR(45)
   UNIQUE;
```

3.2 NOT NULL

NOT NULL é uma restrição, cujo aquele campo(coluna/domínio) não aceita valores vazios ou nulos.

EXEMPLO:

```
1- ALTER TABLE produto ADD COLUMN preco FLOAT NOT NULL;
```

3.3 PRIMARY KEY

PRIMARY KEY a famosa chave primária, é uma restrição que por padrão já é UNIQUE e NOT NULL devido a INTEGRIDADE DE VAZIO E INTEGRIDADE DE CHAVE do modelo relacional, ela serva para identificar exclusivamente uma linha (registro/tupla) de uma tabela.

EXEMPLO:

```
1- CREATE TABLE cliente(
2- Id_cliente INT PRIMARY KEY,
3- nome VARCHAR(45) NOT NULL
4- );
```

3.4 FOREIGN KEY

FOREIGN KEY ou chave estrangeira, é uma restrição que estabelece uma ligação (relacionamento) com a chave primária de outra tabela, serve para sabermos qual registros de tabelas relacionadas estão interagindo, detalhe a chave estrangeira deve referenciar a chave primária, conter os mesmo atributos e restrições.

EXEMPLO:

```
1- CREATE TABLE pedido(
2- Id_pedido INT PRIMARY KEY,
3- fk_id_cliente INT,
4- data_pedido DATE NOT NULL,
5- FOREIGN KEY (fk_id_cliente ) REFERENCES cliente (id_cliente )
6- );
```

3.5 CHECK

É usado para limitar uma faixa de valores que podem ser colocados em uma coluna, se definida em uma única coluna ela permitirá apenas determinados valores para a coluna. Caso a constraint CHECK for definida para a tabela, ela poderá limitar os valores em algumas colunas com base nos valores de outras colunas do registro.

EXEMPLO:

```
1- ALTER TABLE cliente ADD COLUMN idade INT NOT NULL CHECK(idade > 0);
```

3.6 DEFAULT

É usado para definir um valor padrão especificado em uma coluna, o valor padrão será adicionado a todos os novos registros caso nenhum outra valor seja especificado na hora de inseri-lo.

EXEMPLO:

```
1- ALTER TABLE produto ADD COLUMN preco FLOAT NOT NULL DEFAULT(10);
```

4 O QUE É O DML?

O DML -> DATA MANIPULATION LANGUAGE - basicamente são as querys ou comandos que vamos utilizar para manipular os dados que serão inseridos, armazenados no banco de dados.

4.1 OPERAÇÕES DML

Antes de sabermos quais são os Operadores é extremamente necessário entender os principais tipos de dados(domínio) para conseguirmos fazer a manipulação correta e respeitar as quatros restrições do modelo relacional.

4.2 PRINCIPAIS TIPOS DE DADOS

DADOS DO TIPO TEXTO -> TEXT – usado para armazenar grandes quantidades de texto, como um parágrafo ou um documento. EXEMPLOS: TINYTEXT, MEDIUMTEXT, LONGTEXT, CHAR E VARCHAR.

Lembrando CHAR E VARCHAR são considerados do tipo texto, mas eles armazenam grandes quantidades de caracteres (strings), o CHAR é para tamanho fixo e o VARCHAR é para tamanho flexível, variável.

DADOS DO TIPO NUMÉRICO –> Armazenam números, dependendo do nome do tipo de dado, vai armazenar números quebrados (pontos flutuantes) ou números inteiros do negativo ao positivo. EXEMPLOS: INTEGER (INT), TYNYINT, BIGINT, FLOAT, DOUBLE, DECIMAL.

INT - Armazena números inteiros.

FLOAT, DOUBLE, DECIMAL – Armazenam números de ponto flutuante.

DADOS DO TIPO DATA -> São dados que armazenam datas e horas. EXEMPLO: DATE, TIME, DATE TIME. TIMESTAMP.

DATE -> Armazena apenas a data.

TIME -> Armazena apenas o horário

DADOS DO TIPO BOLEANO -> Também conhecido como tipo de dado lógico, possui apenas dois valores, 0 ou 1, TRUE or FALSE. Extremamente importante em tomadas de decisão.

0 -> é o falso (false) e o 1 -> é o verdadeiro (true).

4.3 SELECT

É a primeira query ou comando do DML, com o SELECT podemos realizar consultas, essas consultas iram retornar linhas(tuplas) de uma tabela ou mais tabelas, cujo contém os dados especificados na consulta. 90% da ação do banco de dados é SELECT.

- **Sintaxe básica de um comando SELECT:**

```
1- SELECT <lista de atributos>
2- FROM <lista de tabelas>
3- WHERE <condição>;
```

Lista de atributos -> É uma lista com nome das tabelas com os atributos ou o nome dos atributos de uma tabela específica cujo valores serão retornados.

EXEMPLO:

```
1- SELECT nome.departamento  FROM departamento WHERE d_num = 1;
2- /*consulta com o atributo específico de uma tabela*/
3- SELECT nome FROM departamento WHERE d_num = 1;
```

Lista de tabelas -> É uma lista com os nomes das tabelas necessárias para processar as consultas.

EXEMPLO:

```
1- SELECT * FROM departamento, funcionário;
```

OBS.: O (*) significa todos os atributos da tabela.

Condição -> É uma expressão booleana que identifica as tuplas que devem ser recuperadas por uma consulta.

EXEMPLO:

```
SELECT * FROM departamento WHERE dep_num = 103;
```

Obs.1: Geralmente o WHERE para informar qual é a condição, o WHERE é muito utilizado para filtrar tuplas e trazer informações mais precisas.

Obs.2: Quando fazemos alguma seleção sem o WHERE vai ser retornado todas as linhas daquela tabela.

IMPORTANTE !!!!!!!!!!

Obs.3: Quando fazemos uma consulta entre duas tabelas, retorna o produto cartesiano daquelas duas tabelas, dando origem as "linhas suja", ou seja, vai retornar todas as combinações possíveis entre as tuplas de cada tabela, gerando inconsistência de dados porque os dados ficam com os relacionamentos errados, ISSO OCORRE QUANDO NÃO USAMOS A CLÁSULA WHERE.

4.4 QUALIFICANDO UM CAMPO

Podemos usar mais de um campo (coluna, atributo), desde que estes estejam em tabelas diferentes. Uma consulta que referência dois ou mais campos com o mesmo nome, para não dá redundância ou duplicação de atributos, devemos qualificar o atributo com o nome das tabelas.

EXEMPLO:

1- SELECT nome.pessoa, nome.funcionario FROM pessoa, funcionário;

Obs.: Veja que o ponto final, é uma concatenação para a tabela pessoa, o mesmo serve para o funcionário, é assim que qualificamos campos no sql.

4.5 OPERADORES ARITIMÉTICOS

São operações ou cálculos matemáticos que podem ser realizados, como isso é uma linguagem de programação, é possível realizar cálculos matemáticos.

Abaixo está a tabela dos operadores aritméticos, seus símbolos e significados.

OPERADORES ARITIMÉTICOS E SEUS SÍMBOLOS NO SQL	
ADIÇÃO	+
SUBTRAÇÃO	-
MULTIPLICAÇÃO	*
DIVISÃO	/

4.6 OPERADORES LÓGICOS

Os operadores lógicos, são operadores ou comandos que seguem a tabela verdade da lógica matemática, esses comandos, servem para filtrar tuplas e linhas específicas no banco de dados, e o seu comportamento vai depender de uma condição que você determinar, geralmente usamos mais em consultas!

Abaixo temos uma tabela com os operadores lógicos e seus conceitos!

OPERADORES LÓGICOS E SEUS CONCEITOS	
AND (E)	**Só retorna os resultados da consulta se tivermos duas condições verdadeiras. Ou seja, somente se tiver valor lógico verdadeiro.**
OR (OU)	**Só retorna os resultados das consultas se tivermos ao menos uma condição que tiver valor lógico verdadeira.**
NOT (NÃO)	**Nega o valor lógico de uma condição, se uma for verdadeira, com NOT ela se torna falsa, ou seja, invertemos o valor lógico.**

- Geralmente são utilizados junto a cláusula WHERE para maior estabilidade da consulta!

EXEMPLOS:

1- SELECT * FROM produto WHERE preco = 100 AND nome = 'panela';
<condição1> e <condição2>

2- SELECT * FROM produtos WHERE preco = 100 OR nome = 'panela';
<condição 1> ou <condição 2>

3- SELECT * FROM produtos WHERE preco = 100 NOT nome = 'panela';

4.7 OPERADORES DE COMPARAÇÃO

Os operadores de comparação são bastante utilizados em consultas! Eles vão comparar consultas com uma condição que você determinar! Geralmente utilizada junto com a cláusula WHERE.

Abaixo vemos uma tabela com os operadores de comparação junto com seus símbolos e conceitos.

OPERADORES DE COMPARAÇÃO SEUS SÍMBOLOS E COMCEITO		
OPERADOR	SIMBOLO	CONCEITO
Maior que	>	**Filtra linhas maior que uma condição.**
Menor que	<	**Filtra linhas menor que uma condição.**
Maior igual	=>	**Filtra linhas maiores ou igual a uma condição.**
Menor igual	<=	**Filtra linhas menores ou igual a uma condição.**
Igual a	=	**Filtra linhas iguais a uma condição.**
Diferente de	<>	**Filtra linhas diferentes de uma condição.**

4.8 OPERADORES AVANÇADOS DE CONSULTA SQL

São operadores ou comandos nativos do SQL que nos ajuda fazer consultas mais precisas e complexas, podem ser realizadas com a cláusula WHERE o não, mas geralmente é mais útil utilizar com WHERE devido a segurança e consistência dos dados.

Abaixo veremos uma tabela com os operadores avançados de consulta SQL e seus conceitos!

OPERADORES AVANÇADOS DE CONSULTA!	
IN	Seleciona especificamente um conjunto de valores (linhas) em uma lista que definimos.

BETWEEN	Seleciona uma faixa de valores(linhas), um intervalo de valores.
LIKE	Compara um padrão de strings com caracteres coringa '%' ou '_'
ILEKE	É a mesma coisa que o LIKE mas sem o CASE SENSITIVE, compara strings com um padrão específico usando caracteres coringas '%' porcentagem para muitos caracteres ou '_' underline para um único caractere, serve também para o LIKE
NOT IN	Não vai retornar aquela seleção de valores na consulta
NOT BETWEEN	Não vai retornar aquela faixa de valores na consulta
NOT LIKE	Não vai retornar aquele padrão de string na consulta
NOT ILIKE	Não vai retornar aquece padrão de stringa na consulta

EXEMPLOS:

1- SELECT * FROM produto WHERE preco IN (50, 75, 100);

2- SELECT * FROM produto WHERE preco BETWEEN 50 AND 100;

3- SELECT * FROM produto WHERE nome LIKE('la%');

4- SELECT * FROM produto WHERE nome ILIKE ('%La');

Obs.1: AND no BETWEEN é utilizado para determinar a faixa de valores.

Obs.2: O caractere CORINGA de LIKE ou ILIKE possui uma lógica. Se a % estiver no final, irá retornar todos os caracteres que começa com aquele caractere coringa. Se

a % estiver no início, irá retornar todos os caracteres que terminam com aquele caractere coringa.

Obs.3: Em consultas com JOIN utilizamos muito o LIKE e o ILIKE

4.9 INSERT

Usamos este comando do DML para inserir dados no SCHEMA ou DATABASE.

- **Sintaxe básica do comando INSERT**

  1- INSERT INTO<nome_tabela>(dentro de parêntese os atributos, campos a serem inseridos.) VALUES(dentro dos parêntese , os dados que correspondem a cada campo)

Exemplo:

1- INSERT INTO produtos (nome, preco, quantidade) VALUES(”panela”, 100, 10);

Obs.1: Os campos devem respeitar a ordem em que foram definidos.

Obs.2: Os dados devem respeitar o domínio de cada campo.

4.10 UPDATE

Usamos esse comando para atualizar uma tupla(linha) específica.

- **Sintaxe básica para este comando:**

  1- UPDATE <nome_tabela> SET <nome_coluna> = <novo_atributo> WHERE <condição>;

Exemplo:

1- UPDATE produto SET preço = 200 WHERE cod_produto = 2;

➢ O SET **é obrigatório no comando** UPDATE, pois é ele que vai dizer ao banco de dados qual campo deve ter seu atributo atualizado, ou lista de campos com sua lista de dados atualizados.

Obs.: Não se faz atualização sem a cláusula WHERE pois se fizer, todas as tuplas daquela tabela irão ser atualizados, o mesmo vale para o comando DELETE.

4.11 DELETE

Usamos o DELETE para deletar tuplas, linhas específicas de uma tabela, podemos deletar essas tuplas com o DELETE, mas a tabela ainda existirá, como vimos, para deletar uma tabela, usamos o DROP TABLE do DDL. Já linhas da tabela, usamos DELETE do DML.

- **Sintaxe básica para o comando DELETE:**

1- DELETE FROM <nome_tabela> WHERE <condição>;

EXEMPLO:

1- DELETE FROM produto WHERE nome = “panela”;

Obs.1: Ao deletar lembre-se da restrição de INTEGRIDADE REFERENCIAL do MODELO RELACIONAL, pois o banco de dados por motivos de segurança não permite a deleção de um Primary Key de uma tabela que possui Foreign Key em outras tabelas.

Obs.2: SEMPRE UTILIZE A CLÁUSULA WHERE PARA DELEÇÃO DE LINHAS, POIS SE NÃO UTILIZAR, TODAS AS LINHAS DAQUELA TABELA SERÁ DELETADA.

5 O QUE SIGNIFICA CLÁSULAS EM BD

É uma parte da instrução específica que iremos utilizar naquela query para filtrarmos os dados e obter consultas mais rapidamente e especificamente!

5.1 WHERE

É importante para filtrarmos os resultados e impor condições as consultas que iremos realizar! Na maioria das consultas **é OBRIGATÓRIO** usar o WHERE.

EXEMPLO:

```
1– UPDATE produto SET nome_produto = 'Feijão' WHERE id_produto = '2';
```

5.2 GROUP BY

Muito utilizado em consultas com a cláusula SELECT e quando queremos selecionar várias tabelas e agrupá-los em uma ou mais campos(colunas), agrega um conjunto de registros(linhas) semelhantes.

A seleção fica ainda mais poderosa quando usamos funções como: SUM, COUNT, MIN, MAX e AVG.

EXEMPLO:

```
1- SELECT a.nome_vendedor, COUNT(b.venda_id) AS nr_vendas, b.estado
2- FROM vendedores AS a INNER JOIN vendas AS b
3- ON a.vendedor_id = b.vendedor_id
4- GROUP BY b.vendedor_id, b.estado;
```

RESULTADO: Vamos ser uma tabela temporária agrupada pelo vedendor e o estado que ele pertence, contendo os dados do nome do vendedor, quantidade de produtos vendidos e o estado que ele pertence.

5.3 ORDER BY

Utilizando essa instrução, **podemos Ordenar os dados numérico de maneira crescente ASC ou decrescente DESC, já o alfabeto, ASC = ordem normal de 'a' até 'z' DESC para inverter a ordem do alfabeto, de 'z' até 'a'.**

EXEMPLO:

```
1- SELECT DISTINCT p.emp_no, p.first_name, p.last_name, p.gender,
   MAX(s.salary) AS salário_máximo
```

```
2- FROM salaries AS s INNER JOIN employees AS p ON(s.emp_no = p.emp_no)
3- WHERE s.salary >= 45000 AND s.salary <= 50000 GROUP BY p.emp_no ORDER
   BY p.emp_no ASC;
```

RESULTADO: A consulta retorna uma lista de empregados com seus respectivos números (emp_no), nomes (first_name, last_name), e gêneros (gender), juntamente com o maior salário (salário_máximo) que cada empregado recebeu dentro do intervalo de 45.000 a 50.000.

Os resultados são agrupados por número de empregado (emp_no), garantindo que cada empregado apareça apenas uma vez, mostrando o salário máximo que ele recebeu nesse intervalo. A lista é ordenada pelo número do empregado em ordem crescente.

5.4 DISTINCT

Essa cláusula assegura os dados retornados não contenham registros duplicados, como no exemplo acima.

5.5 UNION

Union serve para unir o resultado de duas ou mais consultas distintas em uma tabela temporária, removendo os resultados duplicado, as famosas linhas sujas ou duplicatas.

EXEMPLO:

```
1- SELECT * FROM produtos
2- UNION
3- SELECT * FROM funcionários;
```

5.6 UNION ALL

Tem o mesmo funcionamento do UNION, mas este retornará dados duplicados, ocorrendo um risco de haver linhas suja.

EXEMPLO:

```
4- SELECT * FROM produtos
5- UNION ALL
6- SELECT * FROM funcionários;
```

5.7 HAVING

Having é uma cláusula usada para filtrar/fazer uma triagem dos dados dos dados retornados pela cláusula Group By (usada para **agrupar linhas** que possuem valores iguais em colunas especificadas. É usada **junto com funções de agregação** (SUM(), COUNT (), AVG(), etc.) para realizar **cálculos por grupo**.)

Exemplo:

```
1-  SELECT a.nome_vendedor, COUNT(b.venda_id) AS nr_vendas, b.estado
2-  FROM vendedores AS a INNER JOIN vendas AS b
3-  ON a.vendedor_id = b.vendedor_id
4-  GROUP BY b.vendedor_id, b.estado
5 – HEVING COUNT(b.venda_id) > 10000;
```

OBSERVAÇÃO: WHERE é usado para filtrar linhas da tabela antes de qualquer agrupamento (GROUP BY) acontecer. GROUP BY agrupa linhas com valores iguais nas colunas indicadas. HAVING filtra os grupos formados pelo GROUP BY com base em funções de agregação. Então se formos fazer um SELECT contendo as três cláusulas, o WHERE vem antes do GROUP BY e o HAVING depois do GROUP BY

Exemplo:

```
1-  SELECT a.nome_vendedor, COUNT(b.venda_id) AS nr_vendas, b.estado
2-  FROM vendedores AS a INNER JOIN vendas AS b
3-  ON a.vendedor_id = b.vendedor_id
4   -  WHERE a.vendedor ILIKE “Guilherme%”
5-  GROUP BY b.vendedor_id, b.estado
6- HEVING COUNT(b.venda_id) > 10000;
```

6 FUNÇÃO DE AGREGAÇÃO

6.1 O que é e para que serve?

Funções de **agregação** no SQL servem para **resumir** ou **consolidar** dados de múltiplas linhas em um único valor. Elas são amplamente utilizadas em **consultas que envolvem agrupamentos (GROUP BY)** ou **análises estatísticas**.

6.2 Max

Retorna a maior quantidade dos registros. Ou seja, o valor máximo.

Exemplo:

```
1- SELECT a.nome_vendedor, MAX(b.venda) AS vl_max_vendas, b.estado
2- FROM vendedores AS a INNER JOIN vendas AS b
3- ON a.vendedor_id = b.vendedor_id
4- - WHERE a.vendedor ILIKE “Guilherme%”
5- GROUP BY b.vendedor_id, b.estado;
```

6.3 Min

Retorna a menor quantidade dos registros. Ou seja, o valor mínimo.

Exemplo:

```
1- SELECT a.nome_vendedor, MIN(b.venda) AS vl_min_vendas, b.estado
2- FROM vendedores AS a INNER JOIN vendas AS b
3- ON a.vendedor_id = b.vendedor_id
4- - WHERE a.vendedor ILIKE “Guilherme%”
5- GROUP BY b.vendedor_id, b.estado;
```

6.4 Sum

Retorna o somatório dos registros. Ou seja, o valor de uma linha somada ao valor de outra linha e posteriormente até retorna o valor total deste somatório.

Exemplo:

```
1- SELECT a.nome_vendedor, SUM(b.vendas) AS soma_vendas, b.estado
2- FROM vendedores AS a INNER JOIN vendas AS b
```

```
3- ON a.vendedor_id = b.vendedor_id
4- - WHERE a.vendedor ILIKE “Guilherme%”
5- GROUP BY b.vendedor_id, b.estado;
```

6.5 Count

Retorna a quantidade de registros presentes naquela tabela. Ou seja, se temos 30 registros, ele irá retornar os 30 registros.

Exemplo:

```
1- SELECT a.nome_vendedor, COUNT(b.venda_id) AS nr_vendas, b.estado
2- FROM vendedores AS a INNER JOIN vendas AS b
3- ON a.vendedor_id = b.vendedor_id
4- - WHERE a.vendedor ILIKE “Guilherme%”
5- GROUP BY b.vendedor_id, b.estado;
```

6.6 AVG

Retorna a média aritmética dos registros presentes naquela tabela.

Exemplo:

```
1- SELECT a.nome_vendedor, AVG(total_vendas) AS md_vendas, b.estado
2- FROM (
3- SELECT a.vendedor_id, b.estado, SUM(b.venda) AS total_vendas
4- FROM vendedores AS a
5- INNER JOIN vendas AS b ON a.vendedor_id = b.vendedor_id
6- WHERE a.vendedor ILIKE 'Guilherme%'
7- GROUP BY a.vendedor_id, b.estado
8- ) AS subquery
9- GROUP BY nome_vendedor, estado;
```

7 JOINS

7.1 O que é o Join e para que serve os Joins?

JOINS são comandos SQL utilizados para **combinar dados de duas ou mais tabelas** com base em uma **condição lógica de relacionamento**.

Na prática, os JOINS **unem registros que possuem correspondência entre tabelas**, geralmente utilizando **chaves primárias (Primary Key)** e **chaves estrangeiras (Foreign Key)** — ou ainda por **valores em comum entre colunas que não sejam nulos**.

7.2 A importância do produto cartesiano para ser entender Join?

O **produto cartesiano** entre duas tabelas significa **combinar todas as linhas da primeira com todas as linhas da segunda, sem considerar nenhuma condição de junção**. Isso não é muito interessante porque causa muita duplicação de registros, e retorna dados conhecidos como registros fantasmas ou registros falsos.

Matematicamente:

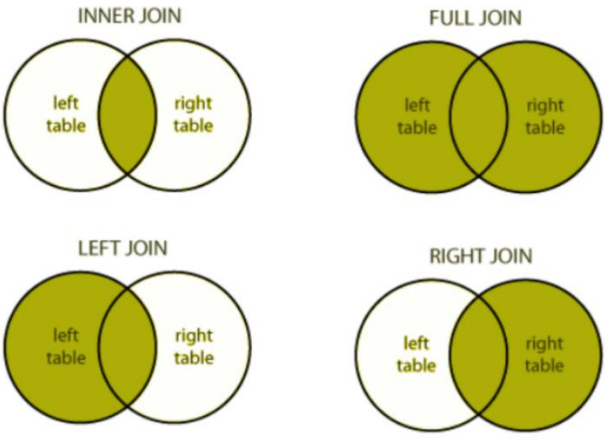
- Se a **Tabela A** tem m linhas e a **Tabela B** tem n linhas, o produto cartesiano terá **m × n linhas**

7.3 LEFT JOIN

LEFT JOIN basicamente pega todos as linhas/registros da tabela esquerda e a interseção (registros em comum com as outras tabelas).

7.4 RIGHT JOIN

RIGHT JOIN basicamente pega todos as linhas/registros da tabela direita e a interseção (registros em comum com as outras tabelas). **É o oposto do LEFT JOIN.**



7.5 INNER JOIN

INNER JOIN é basicamente os registros(linhas/instâncias) em comum (interseção) de todas as tabelas naquele SELECT! É proibido usar **WHERE em id** quando estamos usando **JOINS**, **PORQUE: Todo Select usando WHERE com id pk e id fk é um INNER JOIN.**

7.6 FULL OUTER JOIN

FULL JOIN ou FULL OUTER JOIN é a junção de todos o os registros/linhas/instâncias de todas as tabelas presentes nesta SELECT.

7.7 CROSS JOIN

O **CROSS JOIN** é um tipo de JOIN que **retorna o produto cartesiano** entre duas tabelas. Isso significa que **cada linha da primeira tabela será combinada com todas as linhas da segunda tabela**. Basicamente é o que vimos a respeito do produto cartesiano. MUITO POUCO UTILIZADO ESSE TIPO DE JOIN.