**# Formas Normais -> Diretrizes na Modelagem de Dados. NÃO SÂO REGRAS !**

1. **Forma normal:** 
   1. **Um campo não pode ser divisível (Atomicidade).**
   2. **Um campo não pode ser vetorizado, ou seja, com armazenem de múltiplos valores.**

**Ter um campo que identifique exclusivamente cada registro (chave primária - PK).**

**Forma Normal:**

* 1. **Todo Campo Não Chave Depende da Totalidade das Chaves**

1. **Forma Normal:**
   1. **Todo Campo não Chave, que dependente de outros campos não chave, devem ser movidos para outra tabela.**

**# COMANDOS IMPORTANTES**

STATUS 🡪 PARA VISUALIZAR O BANCO EM USO E TAMBÉM O COMANDO DE DEMITER, ENTRE OUTROS (COMANDO DE ESTRUTURA)

**COMANDOS DE ADM DO BANCO (DBA)**

* SHOW DATABASES; 🡪 PARA VISUALIZAR OS BANCOS
* USE banco de dados 🡪 PARA SELECIONAR UM BANCO PARA USO
* SHOW TABLES; 🡪 PARA VISUALIZAR AS TABELAS DO BANCO

SHOW CREATE TABLE *tabela* 🡪 VISUALIZAR DESCRIÇÃO DETABALHADA DA TABELA

* DESC *tabela* 🡪 BOA PRÁTICA USAR UM DESC ANTES DE DÁ UM INSERT INTO EM TABELAS
* DROP DATABASE *nomeDoBanco* 🡪 APAGAR BANCO
* CREATE VIEW *V\_nomeDaView* AS query... 🡪 Views são amplamente utilizadas em um ambiente corporativo e uma boa prática e colocar um sufixo “V\_” no nome da View, pois as mesas ficam junto das tabelas (show tables). Além disso, nas views também devem ser considerado as regras de negócio, como colocar uma função ifnull(), caso um campo não seja obrigatório e possa retornar vazio
  + Particularidadecs de View: views com Join só é permitida update, view sem Join é permitido todas as operações DML (insert, delete, update, ect)

**DDL**

* ALTER TABLE *tabela* CHANGE -- Normalmente usado para mudar o nome da tabela
* ALTER TABLE *tabela* MODIFY -- Normalmente usado para mudar o tipo da tabela
* ALTER TABLE *tabela* ADD -- Adicionando uma coluna à tabela
* ALTER TABLE *tabela* DROP COLUMN -- Deletando uma coluna da tabela

**DML**

* DELETE/UPDATE 🡪 BOA PRÁTICA USAR UM COUNT(\*), E TALVEZ UM SELECT, ANTES DE DÁ DELETE/UPDATE EM TABELAS E SEMPRE FAZER PELA PK
* ORDER BY -> Pode ser ordenado pelo nome da coluna ou posição dela na tabela. **Complemento:** ASC, ou DESC
* HAVING 🡪 É UMA BOA COMBINAÇÃO COM COUNT(\*)
* ORDER GROUP 🡪 É UMA BOA COMBINAÇÃO COM COUNT(\*)
* IFNULL(*nome da coluna*, *substituto*)
* COUNT(\*) 🡪 CONTAGEM DE REGISTROS, OU SEJA LINHAS.

**Exemplo:** SELECT *SEXO*, COUNT(\*) FROM *CLIENTE* GROUP BY *SEXO* HAVING COUNT(\*) *> 2*;

**# TIPAGENS IMPORANTES**

* PRIMARY KEY
* FOREIGN KEY
* AUTO\_INCREMENT
* NOT NULL
* UNIQUE
* CONSTRAINT -- SÃO AS REGRAS QUE GARANTEM A INTEGRIDADE REFERENCIAL DAS CHAVES (PK OU FK).
  + No mercado, o mais correto é definir as CONSTRTAINT isoladamente da criação das tabelas. Além disso, deve-se colocar os nomes de forma bem descritiva, assim possibilitamos facilmente saber do que a CONSTRTAINT se trata e possibilitando até desabilitar e habilitar dependendo da situação, como no caso de um inserção em massa.

**Exemplo**:

* Nomenclatura Padrão: *PK ou FK* \_ *Tabela Pertencente* \_ *Tabela de Onde Vem*

ALTER TABLE *CLIENTE* ADD CONSTRAINT *PK\_CLIENTE*

PRIMARY KEY(*IDCLIENTE*);

ALTER TABLE TELEFONE ADD CONSTRAINT **FK\_ TELEFONE \_CLIENTE**

FOREIGN KEY(ID\_CLIENTE) REFERENCES CLIENTE(IDCLIENTE);

**# DICA DE CONSTRUÇÃO DE QUERY**

-- SEMPRE COMECE SUA QUERY COM A PROJEÇÃO DO QUE VOCÊ QUER VER NA TELA (SELECT). LEMBRE, SELECT É PROJEÇÃO NÃO SELEÇÃO

-- IMPORTANTE SEPARAR E IDENTAR A QUERY

-- SELECAO, PROJECAO E JUNCAO

* PROJECAO -> É TUDO QUE VOCE QUER VER NA TELA
* SELECAO -> É UM SUBCONJUNTO DO CONJUNTO TOTAL DE REGISTROS DE UMA TABELA. A CLAUSULA DE SELECAO É O WHERE

-- FILTRANDO VALORES NULOS

**Exemplo:** SELECT *NOME*, *SEXO, ENDERECO* FROM *CLIENTE* WHERE *EMAIL* **IS NULL**;

**Exemplo:** SELECT *NOME*, *SEXO, ENDERECO* FROM *CLIENTE* WHERE *EMAIL* **IS NOT NULL**;

* JUNÇÃO

-- 2 TABELAS, **Exemplo:**

SELECT NOME, SEXO, BAIRRO, CIDADE ***/\* PROJECAO \*/***

FROM CLIENTE ***/\* ORIGEM \*/***

INNER JOIN ENDERECO ***/\*JUNCAO \*/***

ON IDCLIENTE = ID\_CLIENTE

WHERE SEXO = 'F'; ***/\* SELECAO \*/***

-- 3 TABELAS, **Exemplo:**

SELECT CLIENTE.NOME, CLIENTE.SEXO, ENDERECO.BAIRRO, ENDERECO.CIDADE, TELEFONE.TIPO, TELEFONE.NUMERO

FROM CLIENTE

INNER JOIN ENDERECO

ON CLIENTE.IDCLIENTE = ENDERECO.ID\_CLIENTE

INNER JOIN TELEFONE

ON CLIENTE.IDCLIENTE = TELEFONE.ID\_CLIENTE;

-- OU USANDO PONTEIRAMENTO, O QUE AJUDA A PERFORMACE DO BANCO DE DADOS, **Exemplo:**

SELECT C.NOME, C.SEXO, E.BAIRRO, E.CIDADE, T.TIPO, T.NUMERO

FROM CLIENTE C

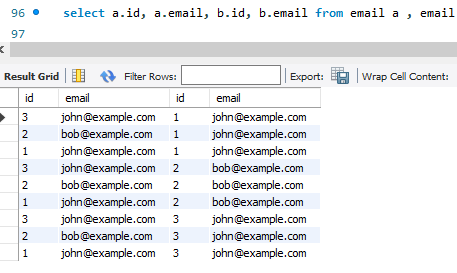
INNER JOIN ENDERECO E

ON C.IDCLIENTE = E.ID\_CLIENTE

INNER JOIN TELEFONE T

ON C.IDCLIENTE = T.ID\_CLIENTE;

**# junção de uma tabela consigo mesma (autojunção ou self-join)**

****

Qoruando realizamos uma junção de uma tabela consigo mesma (autojunção ou self-join) sem uma cláusula "WHERE" específica, obtemos todas as combinações possíveis entre os registros da tabela.

Em uma autojunção, a tabela é tratada como duas instâncias separadas, geralmente com a utilização de aliases para diferenciá-las. Isso significa que cada registro da tabela é combinado com todos os outros registros da mesma tabela, incluindo ele mesmo.

O resultado será uma tabela resultante com todas as combinações possíveis de registros entre as duas instâncias, exibindo os valores dos campos de cada instância.

É importante observar que, em algumas situações, realizar uma autojunção sem uma cláusula "WHERE" específica pode resultar em um grande número de linhas no resultado, especialmente se a tabela tiver muitos registros. Portanto, é comum utilizar uma cláusula "WHERE" para filtrar os resultados e obter apenas as combinações desejadas.

Espero que isso esclareça a questão. Se você tiver mais dúvidas, estou à disposição para ajudar. **# DELIMITER**

O **delimiter** é um comando SQL que é usado para definir um novo caractere de delimitação para separar as instruções SQL. O caractere de delimitação padrão é o ponto e vírgula (;), mas às vezes é necessário usar um caractere de delimitação diferente, especialmente quando se trabalha com procedimentos armazenados e gatilhos. O **delimiter** é usado para definir um novo caractere de delimitação, permitindo que várias instruções SQL sejam executadas como uma única instrução. O **delimiter** não é classificado como DDL ou DML, mas é um comando de administração de banco de dados que é usado para gerenciar o ambiente do banco de dados e obter informações sobre o banco de dados e suas tabelas.

* STATUS 🡪 CONSULTAR O DEMILITER ATIVO

**# STORED PROCEDURES**

**Stored** Procedures (Procedimentos Armazenados) são rotinas definidas no banco de dados que armazenam tarefas repetitivas e aceitam parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual. Eles podem reduzir o tráfego na rede, melhorar a performance de um banco de dados, criar tarefas agendadas, diminuir riscos, criar rotinas de processamento, entre outras funcionalidades. Stored Procedures são de extrema importância para os DBAs e desenvolvedores, pois permitem executar rotinas complexas de manipulação de dados a partir da linguagem/ferramenta utilizada. Stored Procedures são definidos no banco de dados, identificados por um nome pelo qual podem ser invocados, e podem executar uma série de instruções, receber parâmetros e retornar valores. Eles são usados em conjunto com outros comandos SQL para obter informações e gerenciar o ambiente do banco de dados.

EXEMPLO:

* **CRIAR PROCEDURE:**

CREATE PROCEDURE CAD\_CURSO(P\_NOME VARCHAR(30), P\_HORAS INT(30), P\_PRECO FLOAT(10,2))

BEGIN

INSERT INTO CURSOS VALUES(NULL,P\_NOME,P\_HORAS,P\_PRECO);

END

* **VOLTAR COM O DEMILITER PADRÃO**

DEMILITER ;

* **CHAMAR PROCEDURE COM PARAMETROS:**

CALL CAD\_CURSO('BI SQL SERVER',35,3000.00);

* **CONSULTAR PROCEDURE EXISTENTES NO DATABASE**

SHOW PROCEDURE STATUS WHERE Db = ‘nomeDoDataBase”; *(as aspas são obrigatórias)* 🡪 Esta consulta retorna uma lista de todas as procedures no banco de dados especificado

SHOW CREATE PROCEDURE *nome\_da\_procedure; 🡪 Esta consulta retorna o código SQL usado para criar a procedure especificada. Isso inclui o nome da procedure, os parâmetros de entrada e saída, e o código SQL que a procedure executa.*

SHOW CREATE PROCEDURE *mydatabase.myprocedure; 🡪 Esta consulta retornará o código SQL usado para criar a procedure*

**# ESTRUTURA DE UMA TRIGGER**

**Trigger** é um tipo especial de **Stored Procedures** que é executado automaticamente quando um evento ocorre no servidor de banco de dados, como uma instrução INSERT, DELETE ou UPDATE em uma tabela, e pode ser usado para registro de eventos, auditoria, replicação síncrona de tabelas, imposição de autorizações de segurança e prevenção de transações inválidas

CREATE TRIGGER *NOME*

BEFORE/AFTER INSERT/DELETE/UPDATE ON *TABELA*

FOR EACH ROW ~~(PARA CADA LINHA)~~

BEGIN ~~-> INICIO~~

QUALQUER COMANDO SQL

END ~~-> FIM~~

**OBS: NÃO PODE TER 2 TRIGGER NO MESMO MOMENTO(BEFORE/AFTER) REFERENTE AO MESMO COMANDO(INSERT/DELETE/UPDATE).**

**OBS: VALORES 🡪 NEW / OLD**

**EXEMPLO:**

**CREATE TABLE BKP\_USUARIO(**

**IDBACKUP INT PRIMARY KEY AUTO\_INCREMENT,**

**IDUSUARIO INT,**

**NOME VARCHAR(30),**

**LOGIN VARCHAR(30)**

**);**

**DELIMITER $**

**CREATE TRIGGER BACKUP\_USER**

**BEFORE DELETE ON USUARIO**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO BKP\_USUARIO VALUES(**

**NULL, OLD.IDUSUARIO, OLD.NOME, OLD.LOGIN**

**);**

**END**

**$**

**# Cursores**

**# SUBQUERIES**

**/\***

**VENDEDOR QUE VENDEU MENOS EM MARCO**

**E O SEU NOME**

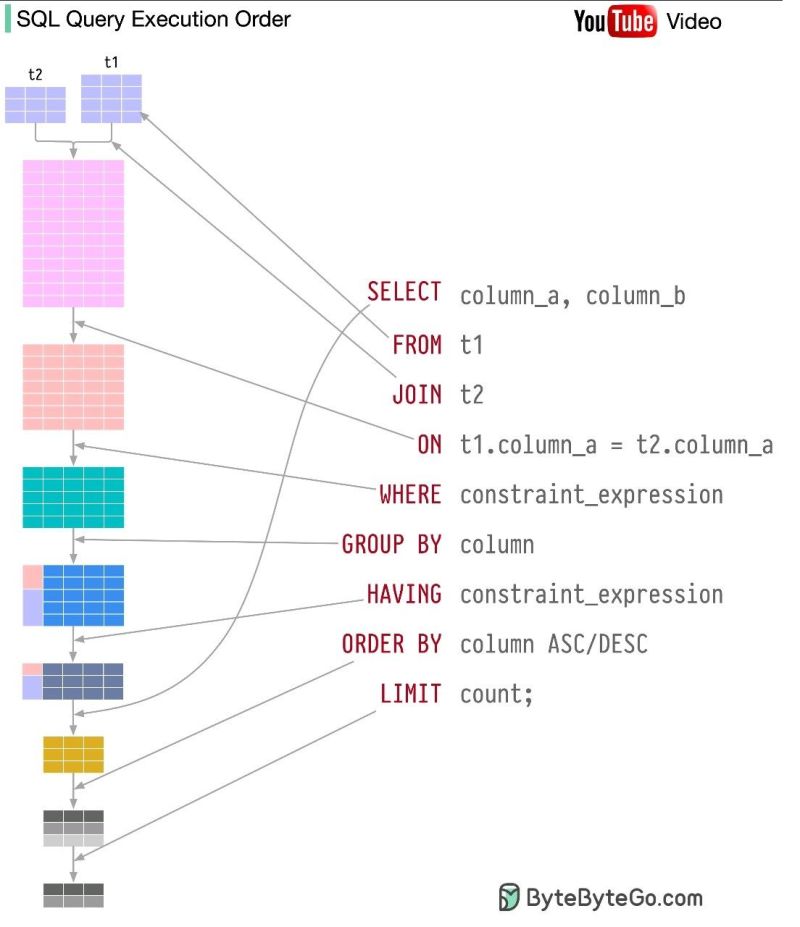
**NOME E O VALOR QUE VENDEU MAIS EM MARCO**

**QUEM VENDEU MAIS QUE O VALOR MEDIO DE FEV**

**\*/**

* SELECT NOME, MARCO FROM VENDEDORES -- > **ALTER QUERY (query de fora)**  
  WHERE MARCO = (SELECT MIN(MARCO) FROM VENDEDORES); 🡪 Isso é uma INNER query (query de dentro)
* SELECT NOME, MARCO FROM VENDEDORES   
  WHERE MARCO = (SELECT MAX(FEVEREIRO) FROM VENDEDORES);
* SELECT NOME, MARCO FROM VENDEDORES   
  WHERE MARCO > (SELECT AVG(MARCO) FROM VENDEDORES);

**OBS: O resultado da INNER QUERY sempre tem que bater com a quantidade de colunas da condição. Exemplo: Neste exercício, temos 1 resultado (média, min, ou max) batendo com 1 coluna ‘marco’.**



**/\* FUNCOES DE AGREGACAO NUMERICAS \*/ 🡪 OPERACÕES DE COLUNA !!**

**Importante Mencionar:**

* Quando você usa o GROUP BY, os registros são agrupados em conjuntos com base nos valores dessas colunas, e as funções agregadas, como AVG(), SUM(), COUNT(), etc., são aplicadas a cada grupo.

CREATE TABLE VENDEDORES(

IDVENDEDOR INT PRIMARY KEY AUTO\_INCREMENT,

NOME VARCHAR(30),

SEXO CHAR(1),

JANEIRO FLOAT(10,2),

FEVEREIRO FLOAT(10,2),

MARCO FLOAT(10,2)

);

INSERT INTO VENDEDORES VALUES(NULL,'CARLOS','M',76234.78,88346.87,5756.90);

INSERT INTO VENDEDORES VALUES(NULL,'MARIA','F',5865.78,6768.87,4467.90);

INSERT INTO VENDEDORES VALUES(NULL,'ANTONIO','M',78769.78,6685.87,6664.90);

INSERT INTO VENDEDORES VALUES(NULL,'CLARA','F',5779.78,446886.87,8965.90);

INSERT INTO VENDEDORES VALUES(NULL,'ANDERSON','M',676545.78,77544.87,578665.90);

INSERT INTO VENDEDORES VALUES(NULL,'IVONE','F',57789.78,44774.87,68665.90);

INSERT INTO VENDEDORES VALUES(NULL,'JOAO','M',4785.78,66478.87,6887.90);

INSERT INTO VENDEDORES VALUES(NULL,'CELIA','F',89667.78,57654.87,5755.90);

/\* MAX - TRAZ O VALOR MAXIMO DE UMA COLUNA \*/

SELECT MAX(FEVEREIRO) AS MAIOR\_FEV

FROM VENDEDORES;

/\* MIN - TRAZ O VALOR MINIMO DE UMA COLUNA \*/

SELECT MIN(FEVEREIRO) AS MENOR\_FEV

FROM VENDEDORES;

/\* AVG - TRAZ O VALOR MEDIO DE UMA COLUNA \*/

SELECT AVG(FEVEREIRO) AS MEDIA\_FEV

FROM VENDEDORES;

/\* VARIAS FUNCOES \*/

SELECT MAX(JANEIRO) AS MAX\_JAN,

MIN(JANEIRO) AS MIN\_JAN,

AVG(JANEIRO) AS MEDIA\_JAN

FROM VENDEDORES;

/\*TRUNCATE \*/

SELECT MAX(JANEIRO) AS MAX\_JAN,

MIN(JANEIRO) AS MIN\_JAN,

TRUNCATE(AVG(JANEIRO),2) AS MEDIA\_JAN

FROM VENDEDORES;

/\* A30 - AGREGANDO COM SUM() \*/

SELECT SUM(JANEIRO) AS TOTAL\_JAN

FROM VENDEDORES;

SELECT SUM(JANEIRO) AS TOTAL\_JAN,

SUM(FEVEREIRO) AS TOTAL\_FEV,

SUM(MARCO) AS TOTAL\_MAR

FROM VENDEDORES;

/\* VENDAS POR SEXO \*/

SELECT SEXO, SUM(MARCO) AS TOTAL\_MARCO

FROM VENDEDORES

GROUP BY SEXO;

**/\* OPERACOES EM LINHA \*/**

/\* SOMAR LINHAS \*/

SELECT \* FROM VENDEDORES;

SELECT NOME,

JANEIRO,

FEVEREIRO,

MARCO,

(JANEIRO+FEVEREIRO+MARCO) AS "TOTAL",

TRUNCATE((JANEIRO+FEVEREIRO+MARCO)/3,2) AS "MEDIA"

FROM VENDEDORES;

SELECT NOME,

(JANEIRO+FEVEREIRO+MARCO) AS "TOTAL",

TRUNCATE((JANEIRO+FEVEREIRO+MARCO)/3,2) AS "MEDIA"

FROM VENDEDORES;

SELECT NOME,

TRUNCATE((JANEIRO+FEVEREIRO+MARCO)/3,2) AS "MEDIA"

FROM VENDEDORES;

/\* APLICANDO UM % \*/

SELECT NOME,

JANEIRO,

FEVEREIRO,

MARCO,

(JANEIRO+FEVEREIRO+MARCO) AS "TOTAL",

(JANEIRO+FEVEREIRO+MARCO) \* .25 AS "DESCONTO",

TRUNCATE((JANEIRO+FEVEREIRO+MARCO)/3,2) AS "MEDIA"

**FONTE IMPORTANTES:**

* [O curso completo de Banco de Dados e SQL, sem mistérios! | Udemy](https://www.udemy.com/course/bancos-de-dados-relacionais-basico-avancado/learn/lecture/6070382#overview)
  + [42. 36 - Cardinalidade e Obrigatoriedade](https://www.udemy.com/course/bancos-de-dados-relacionais-basico-avancado/learn/lecture/19043256#content)
  + [Revisão ilustrada sobre Foreign Keys](https://www.udemy.com/course/bancos-de-dados-relacionais-basico-avancado/learn/lecture/6070382#content)
  + [1°, 2° e 3° Formas Normais](https://www.udemy.com/course/bancos-de-dados-relacionais-basico-avancado/learn/lecture/6288728#content)
  + Dicionário de Dados