Curso de SQL

Relacionamentos:

Ligação entre tabela

Chave Primária (Pk):

Coluna com valores únicos

Chave Composta:

Composição de duas ou mais colunas para gerar uma combinação única

Chave Estrangeira (Fk):

Coluna que armazena a chave primária de outra tabela

1:1 Um para um

1:\* Um para muitos

\*:\* Muitos para muitos

Obs:

Tabelas que “fazem tudo” geram anomalia

Anomalia de inserção: - Impedem a inserção de registros devido a falta de dados

Anomalia de exclusão: - Impede a exclusão de registros devido ao relacionamento com outra tabela

Anomalia de alteração: - Impede a alteração de registros devido ao relacionamento com outra tabela

A ação de separar as diferentes informações em tabelas distintas é o que se chama de normalização de banco de dados.

Benefícios:

Evitar anomalias

Facilitar a manutenção

Maximizar a performance

Manter a integridade dos dados

\*Normalização

- 5 formas normais

Normalizar resulta em um numero maior de tabelas no banco.

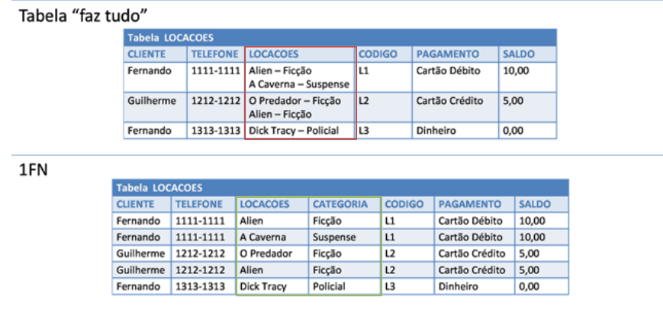
Formas normais:

Primeira forma normal:

Cada linha de tabela deve representar um registro

Cada célula de tabela deve conter um único valor

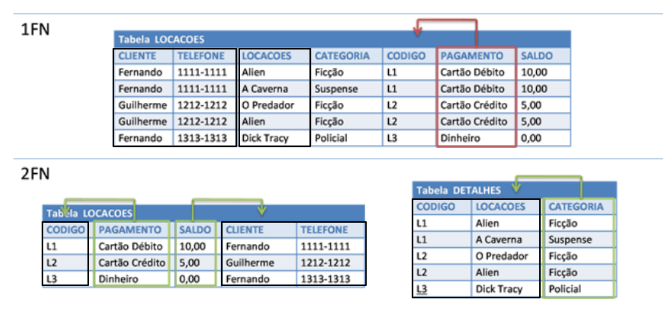
Obs: Deve-se desmembrar os campos, ex: cidade e estado em uma única coluna



Segunda forma normal

Obrigatoriamente deve estar na 1fn

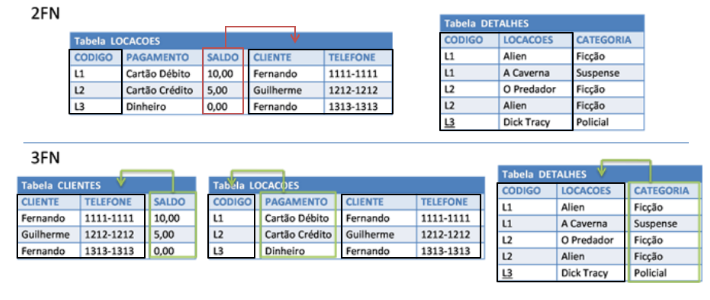
Atributos não chave da tabela devem depender de alguma das chaves da tabela



Terceira forma normal

Obrigatoriamente deve estar na 2fn

Atributos não chave da tabela depender exclusivamente da chave primária da tabela



Quarta e quinta forma normal

Separam em novas tabelas valores que ainda estejam redundantes em uma mesma coluna



Tipos de dados

Boolean: verdadeiro ou falso (0 ou 1)

String:

char

Varchar

Data e hora:

Date

Time

Datetime

Temestamp

# [Utilizando o Postgres - dicas de comandos SQL e essenciais para a manipulação de dados](https://www.codigofonte.net/dicas/bancodedados/561_utilizando-o-postgres-dicas-de-comandos-sql-e-essenciais-para-a-manipula-o-de-dados" \o "Utilizando o Postgres - dicas de comandos SQL e essenciais para a manipulação de dados)

Vou demostrar a seguir uma lista dos principais comandos SQL e outros comandos básicos e essenciais para a manipulação de dados no PostgreSQL.

Vamos lá!

**SELECT**

A declaração SELECT é utilizada para selecionar os dados de um banco de dados.

SELECT nome FROM clientes

**WHERE**

A cláusula WHERE é usada para extrair apenas os registros que cumprir um determinado critério.

SELECT \* FROM clientes

  WHERE Clie\_nome = 'Adelar Poggere'

**AND & OR**

E o operador apresenta um recorde se tanto a condição primeira e à segunda condição é verdadeira.

SELECT \* FROM clientes WHERE clie\_nome ='Adelar Poggere'

  AND clie\_nome = 'Ademar Bonetti'

* OR

SELECT \* FROM clientes WHERE clie\_nome ='Adelar Poggere'

  OR clie\_nome = 'Ademar Bonetti'

* Combinação AND & OR

SELECT \* FROM clientes WHERE clie\_nome ='Adelar Poggere'

  AND (clie\_nome ='Ademar Bonetti' OR clie\_nome='Adelvo Basquera')

**ORDER BY**

É utilizado para classificar o resultado-estabelecidos por uma determinada coluna.

SELECT \* FROM clientes

  ORDER BY clie\_nome ASC -- DESC

**INSERT**

INSERT INTO A afirmação é usada para inserir uma nova linha em uma tabela.

INSERT INTO clientes

  VALUES (4,'Denilson', 'Andrade', 'Rio Grande do Sul', 'Francisco Beltrão')

**UPDATE**

A declaração UPDATE é utilizado para atualizar os registros existentes em uma tabela.

UPDATE clientes

  SET endereco='Francisco', cidade='Francisco Beltrão'

   WHERE nome='Denilson' AND Sobrenome='Andrade'

**DELETE**

O DELETE é usado para excluir linhas em uma tabela.

DELETE FROM clientes

  WHERE Nome = 'Denilson' AND Sobrenome = 'Andrade'

**DISTINCT**

Em uma tabela, algumas das colunas podem conter valores duplicados.

Este não é um problema, no entanto, às vezes você irá querer listar apenas  
os diferentes (distintos) valores em uma tabela.

SELECT DISTINCT nome FROM clientes

**Como Criar uma View**

CREATE VIEW cli AS

 SELECT \*

  FROM clientes

   WHERE clie\_codigo = '20498';

**LIKE**

Localiza todos os cliente cujo nome termina com "s" da tabela  "Clientes".

SELECT \* FROM clientes

 WHERE clie\_nome LIKE '%s';

Localiza todos os cliente cujo nome Inicia com "S" da tabela  "Clientes".

SELECT \* FROM clientes

  WHERE clie\_nome LIKE 'S%';

Localiza todos os cliente cujo nome contenha "Den" da tabela  "Clientes".

SELECT \* FROM clientes

  WHERE clie\_nome LIKE '%Den%';

Localiza todos os cliente cujo nome não contenha "Den" da tabela  "Clientes".

SELECT \* FROM clientes

  WHERE clie\_nome NOT LIKE '%Den%'

Localiza todas os Clientes com a segunda letra "e" idenpendente da primeira Chamado coringa

SELECT \* FROM clientes

  WHERE UPPER(clie\_nome) LIKE '\_E%'

**IN**

Localiza apenas os clientes cujo codigo seja igual a '20037','20500','20501','20917'

SELECT \* FROM clientes

  WHERE clie\_codigo IN ('20037','20500','20501','20917')

Localiza Todos os clientes cujo codigo seja diferente de 20037','20500','20501','20917'

SELECT \* FROM clientes

  WHERE clie\_codigo NOT IN ('20037','20500','20501','20917')

**BETWEEN**

O operador seleciona uma série de dados entre dois valores.  
Os valores podem ser números, texto ou datas.  
Localiza Todos os clientes cujo codigo seja de "1" a "100"

SELECT \* FROM clientes

  WHERE clie\_codigo

   BETWEEN '01' AND '100'

**Alias**

Você pode dar uma tabela ou uma coluna outro nome usando um alias.  
Isto pode ser uma boa coisa a se fazer se o nome da tabela ou coluna.  
forem muto complexo

* Alias de Colunas

SELECT clie\_codigo AS cod,\* FROM clientes

* Alias de Tabelas

SELECT c.clie\_codigo, c.Clie\_nome FROM clientes AS c

  WHERE c.clie\_codigo='1'

**JOIN**

A palavra-chave JOIN é usada em uma instrução SQL para consultar os dados de duas ou mais tabelas, com base em uma relação entre determinadas colunas nestas tabelas.

Tabelas em um banco de dados são, muitas vezes, relacionadas umas às outras com as teclas.

Uma chave primária é uma coluna (ou uma combinação de colunas), com um valor único para cada linha. Cada chave primária valor deve ser exclusivo dentro da tabela.

O objetivo é vincular os dados em conjunto, em tabelas, sem repetição de todos os dados em cada tabela.

* **INNER JOIN**: Regresso filas quando há, pelo menos, um jogo em ambas as tabelas

SELECT \* FROM clientes

  INNER JOIN movfiscal

   ON clie\_codigo = mfis\_codentidade

    WHERE clie\_codigo = '20301'

* **LEFT JOIN**: Retornar todas as linhas da tabela à esquerda, mesmo quando não há jogos no quadro do direito

SELECT \* FROM clientes

  LEFT JOIN movfiscal

   ON clie\_codigo = mfis\_codentidade

    WHERE clie\_codigo = '20301'

* **RIGHT JOIN**: Retornar todas as linhas da tabela à direita, mesmo se não houver jogos no quadro da esquerda

SELECT \* FROM clientes

  RIGHT JOIN movfiscal

   ON clie\_codigo = mfis\_codentidade

    WHERE clie\_codigo = '20301'

* **FULL JOIN**: Regresso filas quando há um jogo em um dos quadros

SELECT \* FROM clientes

  FULL JOIN movfiscal

   ON clie\_codigo = mfis\_codentidade

    WHERE clie\_codigo = '20301'

**UNION**

O operador UNION é usado para combinar o resultado-conjunto de dois ou mais SELECT.  
Observe que cada SELECT declaração no âmbito da União devem ter o mesmo número de colunas.  
As colunas devem ter também os tipos de dados semelhantes. Além disso, as colunas em cada SELECT declaração deve ser na mesma ordem.

SELECT clie\_codigo, clie\_nome FROM clientes

  UNION

   SELECT mfis\_codentidade, mfis\_historico FROM movfiscal

* **UNION ALL**

SELECT clie\_codigo, clie\_nome FROM clientes

  UNION ALL

   SELECT mfis\_codentidade, mfis\_historico FROM movfiscal

**SELECT INTO**

A declaração SELECT INTO seleciona dados de uma tabela e insere-lo em uma tabela diferente.  
A declaração SELECT INTO é mais frequentemente usado para criar cópias de segurança das tabelas.

SELECT \* INTO new\_Clientes

  FROM Clientes

* **SELECT INTO** - Juntando Tabelas

SELECT clie\_codigo,mfis\_historico

  INTO new\_junta

   FROM Clientes

  INNER JOIN movfiscal

   ON clie\_codigo = mfis\_codentidade

    WHERE clie\_codigo = '20301'

**CREATE DATABASE**

A declaração CREATE DATABASE é utilizado para criar um banco de dado

CREATE DATABASE teste

**CREATE TABLE**

O CREATE TABLE é usado para criar uma tabela em um banco de dados.

CREATE TABLE Clientes

(

cod int,

Nome varchar(255),

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)

)

**SQL Constraints**

Restrições são utilizados para limitar o tipo de dados que pode ir em uma tabela.  
Constrangimentos pode ser especificado quando uma tabela é criada (com CREATE TABLE), ou após a tabela é criada (com o ALTER TABLE).

Iremos focar as seguintes restrições:

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT
* **NOT NULL**

CREATE TABLE Clientes

(

cod int NOT NULL,

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)

)

* **UNIQUE**

O único condicionalismo identifica exclusivamente cada registro em um banco de dados tabela.

CREATE TABLE Clientes

(

cod int NOT NULL UNIQUE,

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)

)

* **PRIMARY KEY**

A PRIMARY KEY constraint identifica exclusivamente cada registro em um banco de dados tabela.

CREATE TABLE Clientes

(

cod integer PRIMARY KEY,

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)

)

* **FOREIGN KEY**

Uma chave estrangeira em uma tabela aponta para uma PRIMARY KEY em uma outra tabela.

CREATE TABLE Fiscal

(

id integer PRIMARY KEY,

cod integer REFERENCES Clientes (cod),

transacao integer

)

ou

CREATE TABLE Fiscal

(

Id int NOT NULL,

transacao int NOT NULL,

PRIMARY KEY (Id),

FOREIGN KEY (cod) REFERENCES Clientes(cod)

)

* **CHECK**

É usada para limitar o valor intervalo que pode ser colocado em uma coluna.

CREATE TABLE Clientes

(

cod int NOT NULL CHECK (Cod > 0),

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)

)

* **DEFAULT**

É utilizado para inserir um valor padrão para uma coluna

CREATE TABLE Clientes

(

cod int NOT NULL ,

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255)DEFAULT 'Dois Vizinhos'

)

**CREATE INDEX**

Um índice pode ser criado em uma tabela para encontrar os dados de forma mais rápida e eficiente.  
Os usuários não podem ver os índices, eles são usados apenas para acelerar pesquisas / consultas.

CREATE INDEX index\_name

  ON Clientes (Nome)

ou

CREATE INDEX index\_name

  ON new\_junta (clie\_codigo)

**DROP INDEX**

É utilizado para excluir um índice em uma tabela.

DROP INDEX index\_name;

**ALTER TABLE**

É usado para adicionar, apagar ou alterar colunas em uma tabela existente.

* ADD

ALTER TABLE Clientes

  ADD Nome01 varchar(150);

* DROP

ALTER TABLE Clientes

  DROP COLUMN Nome01;

* ALTER

ALTER TABLE Clientes ALTER COLUMN Nome01 TYPE VARCHAR(30);

* RENAME - Mudar o Nome da coluna

ALTER TABLE Clientes RENAME COLUMN Nome01 TO Nome02;

* RENAME para Mudar o Nome da Tabela

ALTER TABLE Clientes RENAME TO Clientes01;

**AUTO INCREMENT**

Muitas vezes, gostaríamos que o valor da chave primária campo a ser criado automaticamente cada vez que um novo registro seja inserido.

CREATE TABLE Clientes

(

Cod int NOT NULL AUTO\_INCREMENT,

Nome varchar(255) NOT NULL,

SobreNome varchar(255),

endereco varchar(255),

Cidade varchar(255),

PRIMARY KEY (Cod)

)

**Função NOW pega data e hora**

SELECT NOW()

**NULL "IS"**

SELECT \* FROM Clientes WHERE clie\_codigo IS NULL

SELECT \* FROM Clientes WHERE clie\_codigo IS NOT NULL

**Funções SQL agregadas**

* AVG()   -  Retorna o valor médio
* COUNT() -  Retorna o número de linhas
* MAX()   -  Retorna o maior valor
* MIN()   -  Retorna o menor valor
* SUM()   -  Devolve a soma

SELECT AVG(clie\_codigo) FROM Clientes

SELECT COUNT(clie\_codigo) FROM Clientes

SELECT MAX(clie\_codigo) FROM Clientes

SELECT MIN(clie\_codigo) FROM Clientes

SELECT SUM(clie\_codigo) FROM Clientes

**GROUP BY**

A declaração GROUP BY é utilizada em conjugação com as funções agregadas ao grupo o resultado-definido por um ou mais coluna

SELECT SUM(clie\_codigo) FROM Clientes

  GROUP BY clie\_codigo

**HAVING**

A cláusula HAVING foi adicionado ao SQL porque a palavra-chave WHERE não pode ser utilizado com funções agregadas.

SELECT SUM(clie\_codigo) FROM Clientes

  GROUP BY clie\_codigo

   HAVING SUM(clie\_codigo)<2000

**UPPER**

UPPER () função converte o valor de um campo com letras maiúsculas.

SELECT UPPER(clie\_nome) FROM Clientes

**LOWER**

LOWER () função converte o valor de um campo com letras Minúsculas.

SELECT LOWER(clie\_nome) FROM Clientes

**ROUND**

Afunção Round () é utilizada para arredondar um campo numérico para o número de casas decimais especificado.

SELECT ROUND(clie\_codigo,12) FROM Cliente