Conceitos e Melhores Práticas com BD PostgreSQL

1. Fundamentos de BD
2. Modelo Relacional
3. Introdução ao PostgreSQL

1 – Fundamentos de BD

Dados são valores brutos, observações documentadas, registros soltos, etc., tudo isso sem tratamento. Já informações são dados estruturados, organizados, conjuntos de dados relacionados entre si, tudo isso gerando valor, que criam sentido aos dados e dessa informação extraímos conhecimento.

2 – Modelo Relacional

Modelar significa criar um modelo e esse modelo ira explicar as características de funcionamento, vai explicar o comportamento de um software. Pensamos aqui então em modelagem de dados, sendo assim, um modelo de dados irá nos mostrar como os dados estão organizados, de que tipo são e como eles vão se relacionar entre si, de maneira a ser gerada uma informação.

Então o Modelo Relacional é o modelo de dados representativo que se baseia no principio de que todos os dados que serão armazenados, serão armazenados em tabelas e se dividirão em linhas (tuplas) e colunas. Sendo as tuplas os valores das tabelas organizados e as colunas os atributos destes dados.

Tabelas são um conjunto de dados organizados por colunas e linhas(tuplas) com um objetivo em comum, por exemplo uma tabela de contatos com nomes e números de telefone. As colunas são os atributos da tabela, por exemplo nome, telefone, endereço, e-mail. Já as linhas/tuplas são os dados/valores em si.

O que pode ser definido como tabelas:

* Coisas tangíveis

Elementos físicos (carro, produto, animal)

* Funções

Perfis de usuários, status de compras.

* Eventos ou ocorrências

Produtos de um pedido, histórico de dados

Colunas importantes

* Chave Primária / Primary Key / PK

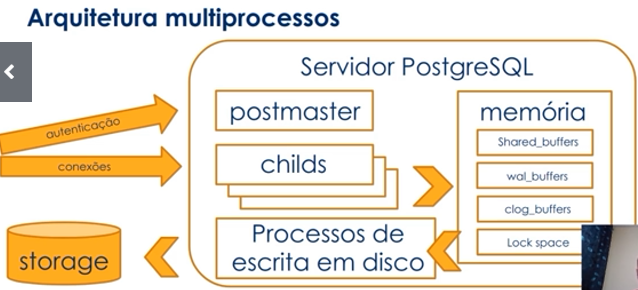
Conjunto de um ou mais campos que nunca se repetem. Identidade da tabela. Utilizados como índice de referência na criação de relacionamentos entre tabelas.

* Chave Estrangeira / Foreign Key / FK

Valor de referência a uma PK de outra tabela ou da mesma tabela para criar relacionamento.

Surge no contexto do Modelo Relacional os SGBDs (Sistemas de Gerenciamento/Gestão de BD) que são os responsáveis para gerir todo o relacionamento entre as tabelas. Conjunto de softwares responsáveis pelo gerenciamento de todo o BD que facilitam na sua administração. Exemplos de SGBDs: Oracle, PostgreSQL, MySQL, Microsoft SQL Server, MongoDB, entre outros.

3 – Introdução ao PostgreSQL

 É um SGBD que trabalha de uma forma relacional. Foi desenvolvido em 1986 pelo departamento de Ciência da Computação da Universidade da Califórnia em Berkeley. Ele é um SGBD opensource, portanto seu código pode ser utilizado, modificado e distribuído por qualquer pessoa para qualquer finalidade.

Por ser um SGBD cujo modelo é Cliente/Servidor há processos que irão acontecer somente no lado do cliente e processos que irão acontecer somente no servidor. Os processos do servidor estão evidenciados acima. Já os processos executados do lado do cliente podem ser a Interface Gráfica, o terminal, a aplicação, entre outros.

Suas principais características são:

* Ser OpenSource;
* Permite o Point in time recovery
  + Por exemplo caso haja um problema no BD as 13h é possível realizar a recuperação até as 12:59h, ou seja, no ponto anterior praticamente imediato do problema.
* Linguagem procedural com suporte a várias linguagens de programação (perl, python, etc)
* Suporte a views, functions, procedures, triggers
* Consultas complexas e Commom table expressions (CTE)
* Suporte a dados geográficos (PostGIS)
* Controle de concorrência multi-versão

Postgresql.conf

Arquivo onde fica armazenada todas as configurações do servidor PostgreSQL. Alguns parâmetros só podem ser alterados com a reinicialização do BD. A view pg\_settings, cujo acesso é feito dentro do BD, guarda todas as configurações atuais, tudo o que está em execução no momento.

Por padrão sua localização fica dentro da pasta PGDATA.

CONFIGURAÇÕES DE CONEXÃO

* Listen\_addresses – endereços TCP/IP que o SGBD vai escutar/liberar conexões
* Port – porta TCP do servidor. Padrão 5432
* Max\_connections – n° máximo de conexões simultâneas
* Superuser\_reserved\_connections – números de conexões (slots) reservados para conexões ao BD de super usuários.

CONFIGURAÇÕES DE AUTENTICAÇÃO

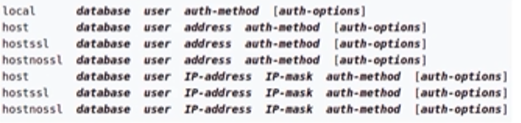
* Authentication\_timeout – tempo máximo de segundos para o cliente conseguir conexão com o servidor
* Password\_encryption – algoritmo de criptografia das senhas
* SSL – habilita conexões de criptografia SSL (somente se o PostgreSQL foi compilado com suporte SSL)

CONFIGURAÇÕES DE MEMÓRIA

* Shared\_buffers – tamanho da memória para cache/buffer de tabelas, índices e demais relações
* Work\_mem – tamanho da memória para operações de agrupamento e ordenação
* Maintenance\_work\_mem – tamanho da memória para operações como VACUUM, INDEX, ALTER TABLE

PG\_HBA.conf

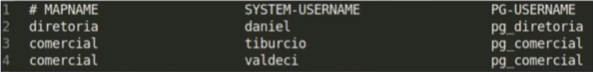
Arquivo responsável pelo controle de autenticação dos usuários no servidor PostgreSQL. Formatos que o arquivo pode ser:



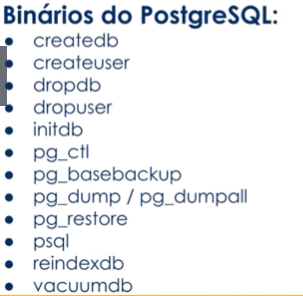
Por esse arquivos são passados os métodos de autenticação para acesso ao BD, existindo diversos métodos, como por exemplo: TRUST, REJECT, MD5, etc.

PG\_IDENT.conf

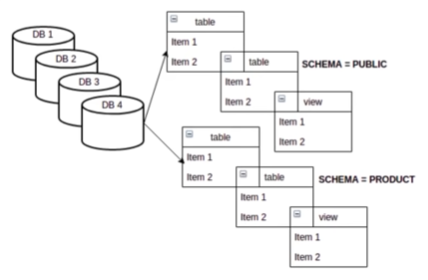
Ele realiza o mapeamento dos usuários do sistema operacional com os usuários do BD. Fica localizado no diretório de dados PGDATA de sua instalação. A opção ident deve ser utilizada no arquivo pg\_hba.conf.



Comandos Administrativos



Arquitetura/Hierarquia

Cluster

Coleção de BDs que compartilham a mesma configuração (arquivos de configuração) do PostgreSQL e do sistema operacional (porta, listen\_addresses, etc).



Banco de Dados (database)

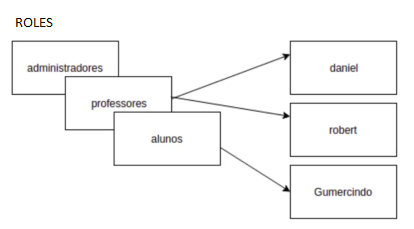
Conjunto de schemas com seus objetos/relações (tabelas, funções, views, etc).

Users/Roles/Groups

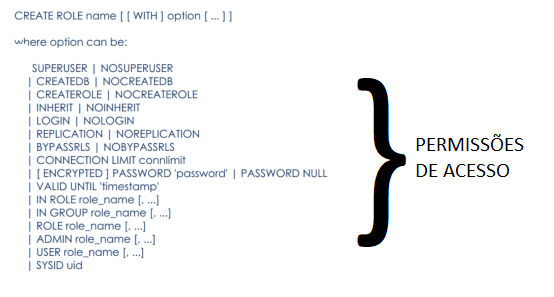
Roles (papéis ou funções), users (usuários) e grupo de usuários são “contas”, perfis de atuação em um BD, que possuem permissões em comum ou específicas.

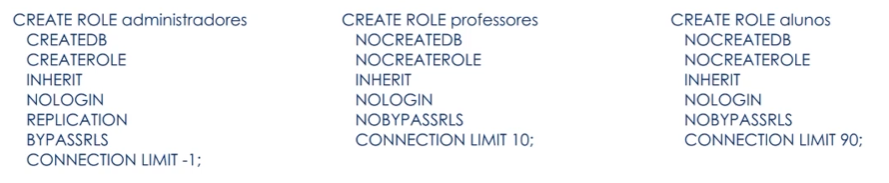
O conceito de usuários e grupos administrativos eram distintos em na versão 8.1 do PostgreSQL. Hoje eles se tornaram uma única entidade chamado “role”. Por significar cargo (função) ou “papel” (como atua), ROLE identifica o próprio papel e quais funções é responsável por exercer no contexto de segurança do BD.

De forma abstrata, uma role pode se comportar como um usuário, como um grupo ou ter ambos os comportamentos ao mesmo tempo. Ela pode conter e ser contida por outra role. Seu conceito está atrelado a definição de permissões, privilégios e garantias de acesso aos objetos do BD.

Uma ROLE ADMINISTRADOR tem acessos ilimitados no BD.

Já uma ROLE PROFESSORES tem permissão de escrita em apenas duas tabelas e a de ler qualquer tabela. Por estarem inseridas dentro da ROLE PROFESSORES a ROLE DANIEL E ROBERT possuem as mesmas características e da ROLE PROFESSORES.

Já uma ROLE ALUNOS tem apenas a permissão de leitura de uma tabela. A ROLE GUMERCINDO passa a ter as mesmas permissões da ROLE ALUNOS.



Associação entre Roles

Quando uma role assume as permissões de outra rola. Necessário a opção inherit.

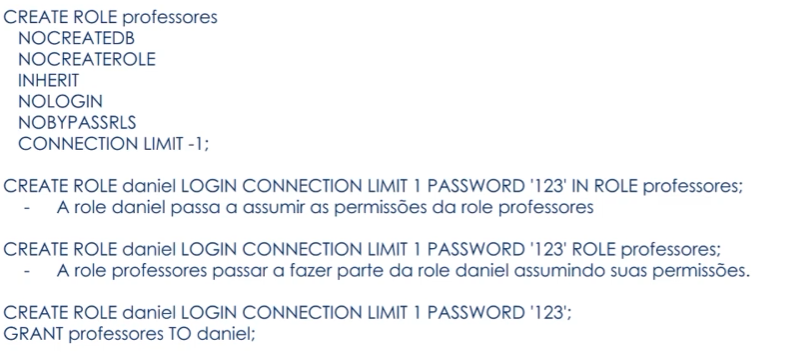
No momento de criação da role:

- IN ROLE (passa a pertencer a role informada)

- ROLE (a role informada passa a pertencer a nova role)

Ou após a criação da role:

- GRANT [role a ser concedida] TO [role a assumir as permissões]



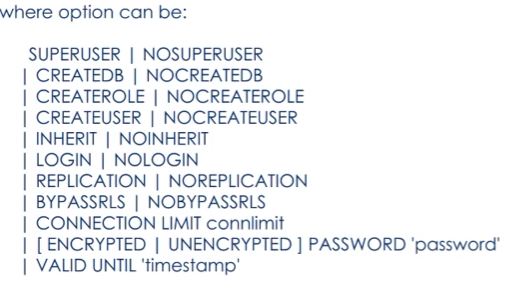
**Desassociar membros entre roles**

REVOKE [role que será revogada] FROM [role que terá suas permissões revogadas]

REVOKE professores FROM Daniel;

**Alterando uma role**

ALTER ROLE role\_specification [with] option [ ... ]



**Excluindo uma role**

DROP ROLE role\_specification;

Administrando Acessos (GRANT)

São os privilégios de acesso aos objetos do BD.



DATABASE

GRANT {{CREATE | CONNECT | TEMPORARY | TEMP } [...] | ALL [PRIVILEGES] }

ON DATABASE database\_name [, ...]

TO role\_specification [, ...] [WITH GRANT OPTION]

SCHEMA

GRANT {{CREATE | USAGE} [...] | ALL [PRIVILEGES] }

ON SCHEMA schema\_name [, ...]

TO role\_specification [, ...] [WITH GRANT OPTION]

TABLE

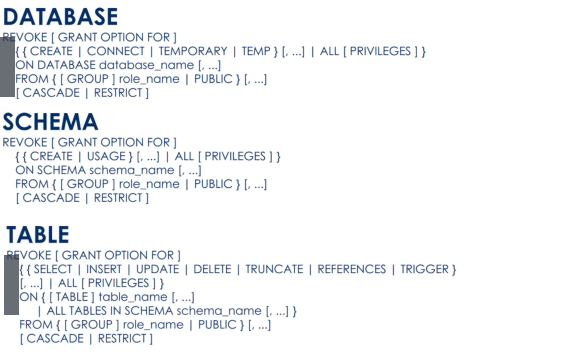
GRANT {{ SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER } [...] | ALL [PRIVILEGES] }

ON { [ TABLE ] table\_name [, ...]

| ALL TABLES IN SCHEMA schema\_name [, ...] }

TO role\_specification [, ...] [WITH GRANT OPTION]

REVOKE – Retira as permissões da role





Database, Schemas e Objetos

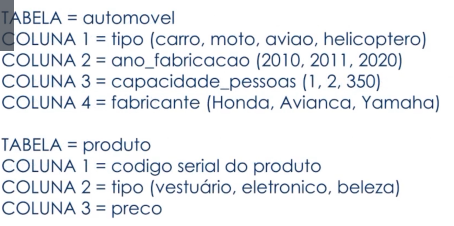
Database – é o BD em si. Conjunto de schemas e seus objetos, como tabelas, types, views, funções, entre outros. Seus schemas e objetos não podem ser compartilhados entre si. Cada database é separado um do outro compartilhando apenas usuários/roles e configurações de cluster PostrgreSQL.

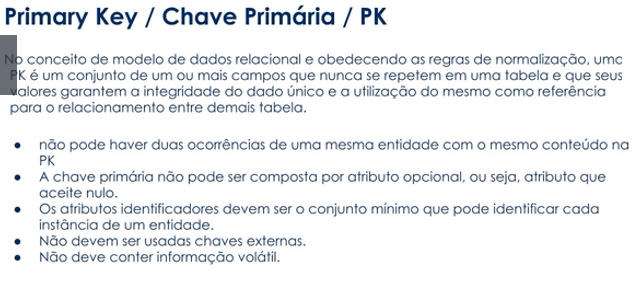
Schemas – grupo de objetos. É possível relacionar objetos entre diversos schemas. Por exemplo: schema public e schema curso podem ter tabelas com o mesmo nome que se relacionam entre si

Objetos – Tabelas, views, funções, types, sequences, entre outros, pertencentes aos schemas.

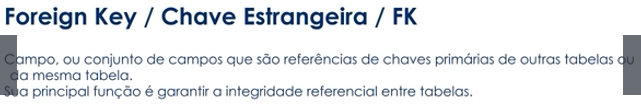


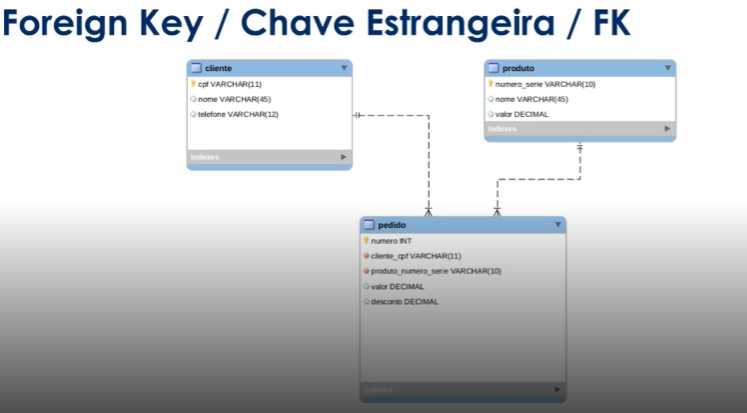






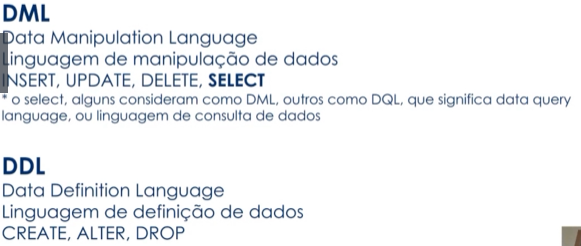


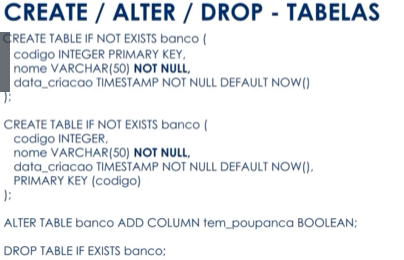


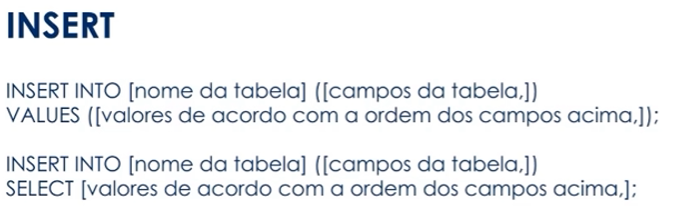


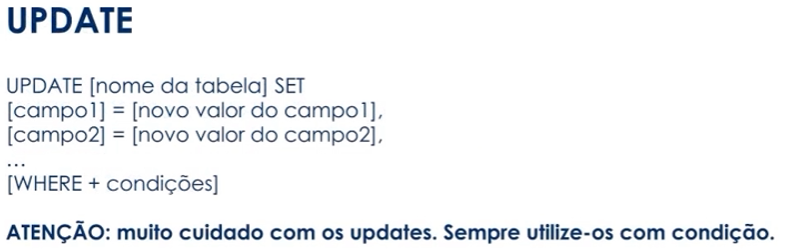




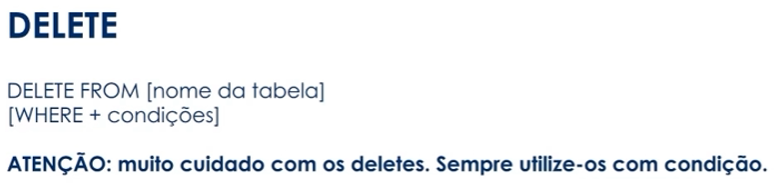




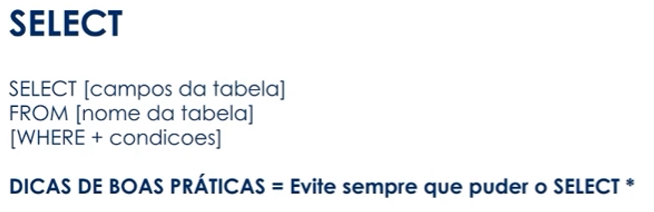


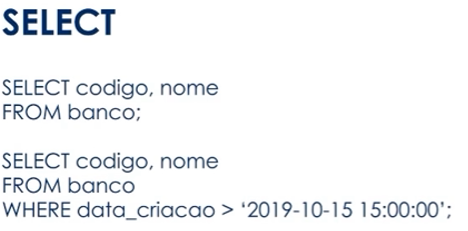












DDL – Data Definition Language

DML – Data Manipulation Language. Conhecido como CRUD também.

Idempotência

Propriedades que algumas ações/operações possuem possibilitando-as de serem executadas diversas vezes sem alterar o resultado após a aplicação inicial.

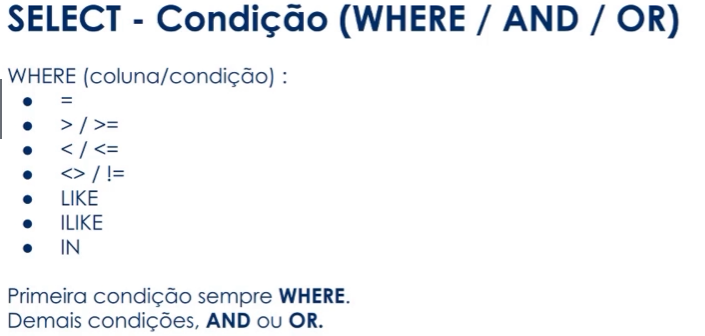
Exemplo: IF EXISTS

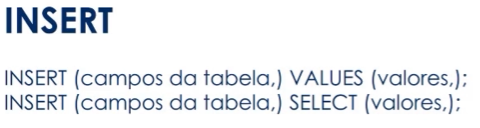
Comandos pertinentes ao DDL e DML

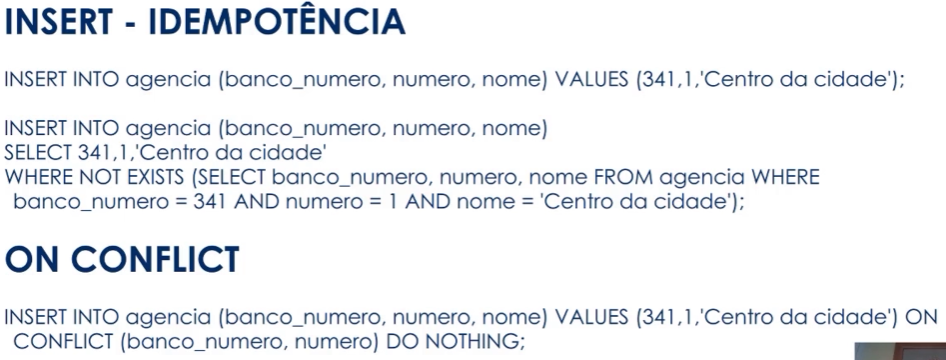
Melhores Práticas em DDL

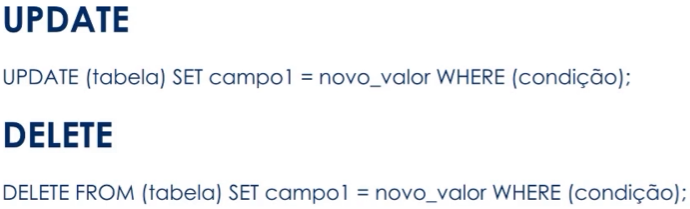
* É importante as tabelas possuírem campos que realmente serão utilizados e que sirvam de atributo direto a um objetivo em comum.
* Criar/Acrescentar colunas que são “atributos básicos” do objeto;
* Cuidado com regras (constraints), quanto mais constraints mais dificil serão algumas ações, devido as validações que são necessárias;
* Cuidado como excesso de FKs
* Cuidado com o tamanho indevido de colunas

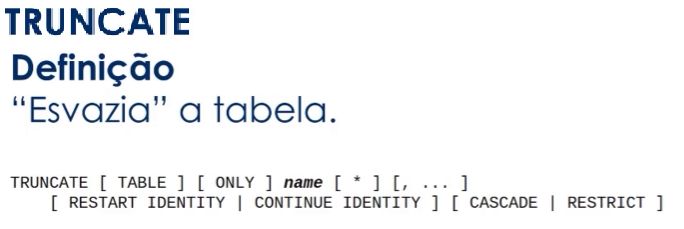
Ex.: coluna CEP VARCHAR(255)











Funções Agregadas

AVG – retorna a média dos valores

Ex.: SELECT AVG(valor) FROM cliente\_transacoes;

COUNT – retorna a quantidade

Ex.: SELECT COUNT(numero), email

FROM cliente

WHERE email ILIKE '%gmail.com'

GROUP BY email;

SELECT COUNT(id), tipo\_transacao\_id

FROM cliente\_transacoes

GROUP BY tipo\_transacao\_id

HAVING COUNT(id) > 150

MAX – retorna o maior valor da coluna selecionada

Ex.: SELECT MAX(valor)

FROM cliente\_transacoes;

SELECT MAX(valor), tipo\_transacao\_id

FROM cliente\_transacoes

GROUP BY tipo\_transacao\_id;

MIN – retorna o menor valor da coluna selecionada

Ex.: SELECT MIN(valor)

FROM cliente\_transacoes;

SELECT MIN(valor), tipo\_transacao\_id

FROM cliente\_transacoes

GROUP BY tipo\_transacao\_id;

SUM – retorna a soma

Ex.: SELECT SUM(VALOR)

FROM cliente\_transacoes;

SELECT SUM(valor), tipo\_transacao\_id

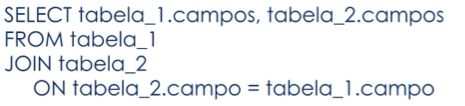
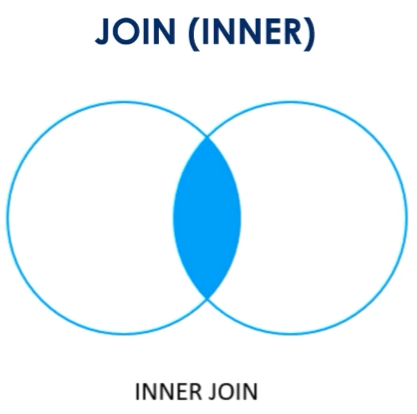
FROM cliente\_transacoes

GROUP BY tipo\_transacao\_id

ORDER BY tipo\_transacao\_id DESC;

Relações entre tabelas

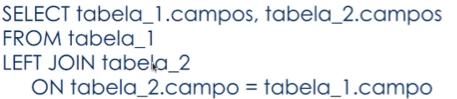




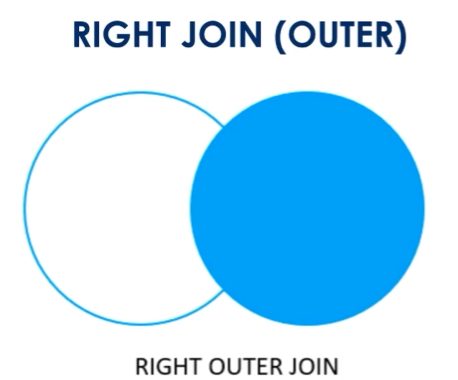
Retorna apenas o que as duas tabelas tem em comum.

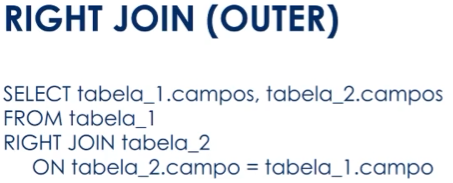
Tabelas que estiverem no relacionamento a esquerda retornarão as tabelas por completo.

Por exemplo temos 10 registros em uma tabela A e 20 registros em uma tabela B. Porém na tabela B há apenas 2 registros que se relacionam com a tabela A. Ao fazer o LEFT JOINT o resultado será os 10 registros da tabela A trazendo os resultados da tabela B que estão no relacionamento com a tabela A. Os outros 8 registros que não tem relacionamento com a tabela B são retornados com o valor NULO.

 No código ao lado pedimos para serem retornados todos os registros da tabela 1 e se houver relação com a tabela 2 retornar o valor, senão retornar o valor NULO.



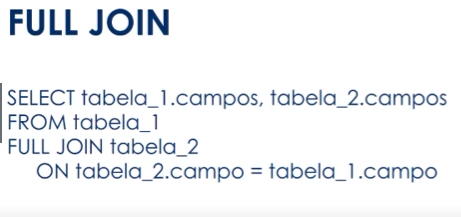
É como o left join porém prioriza as relações a sua direita.

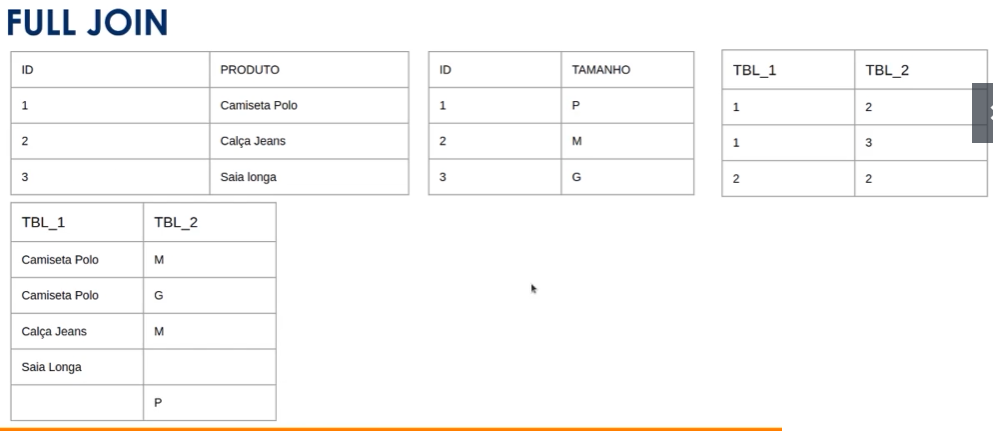


Retorna todos os valores da tabela 2 e na coluna da relação com a tabela 1 caso não houver relação retorna NULO.



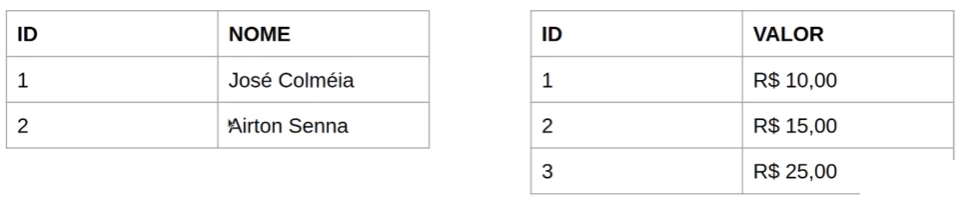
Retorna todas as relações possíveis.





Cross Join

Todos os dados de uma tabela serão cruzados com todos os dados da tabela referenciada no CROSS JOIN criando uma matriz.



COMMOM TABLE EXPRESSIONS

Formas de auxiliar na organização de “statements”, ou seja, blocos de códigos, para consultas muito grandes, gerando tabelas temporárias e criando relacionamentos entre elas.

Dentro dos statements podem ter SELECTs, INSERTs, UPDATEs ou DELETEs.



VIEWs

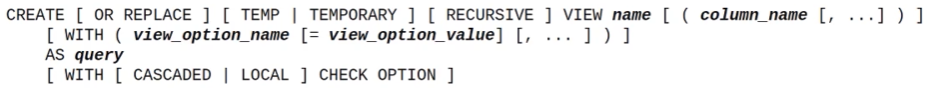
São visões, são camadas para as tabelas que ficam à frente das tabelas, à frente das consultas SQL sendo robustas (tendo diversos JOINS, etc) ou não, são “alias” para uma ou mais queries.

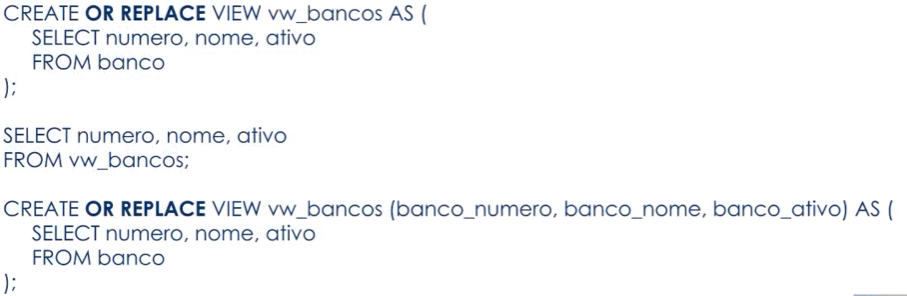
Elas são consideradas boas práticas para segurança, ao criarmos uma view para cada tabela existente fazemos que as pessoas que consumam o BD consumam as views e não tem acesso as tabelas em si.

Aceitam comandos de SELECT, INSERT, UPDATE e DELETE;

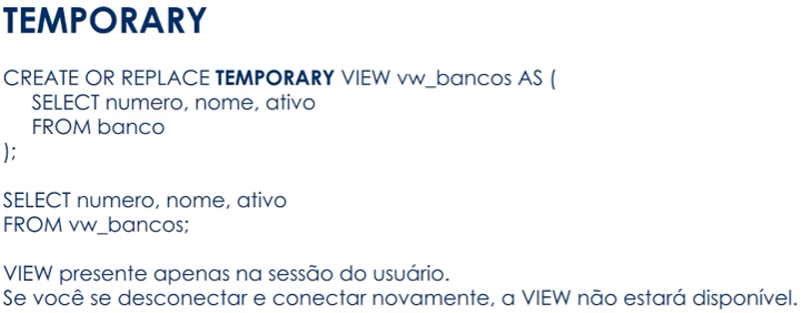
Qual a condição para uma VIEW aceitar comandos de INSERT, UPDATE ou DELETE?

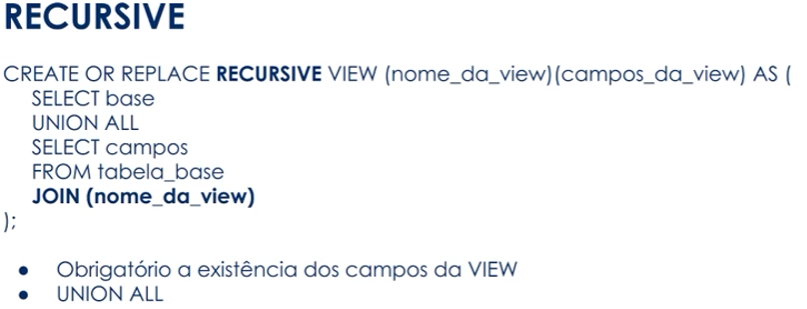
Possuir apenas uma única tabela em seu conteúdo respeitando as regras de obrigatoriedade de cada coluna da tabela.









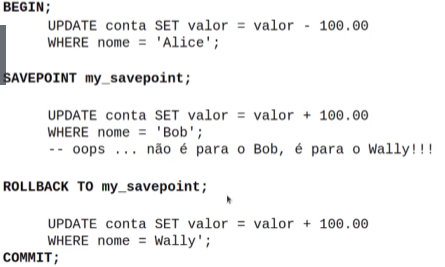




TRANSAÇÕES

Conceito fundamental de todos os sistemas de BD. Conceito de múltiplas etapas/códigos reunidos em apenar 1 transação, onde o resultado precisa ser tudo ou nada.

EXEMPLO



BEGIN – inicia transação;

SAVEPOINT – salva a transação até o momento de sua chamada

ROLLBACK – impede a transação de ser executada caso tenha algum erro;

COMMIT – executa a transação

Funções

Conjunto de códigos que são executados dentro de uma transação com a finalidade de facilitar a programação e obter o reaproveitamento/reutilização de códigos.

Existem 4 tipos de funções:

* Query language functions (funções escritas em SQL)
* Procedural language functions (funções escritas em, por exemplo, PL/pgSQL ou PL/py
* Internal functions
* C-language functions

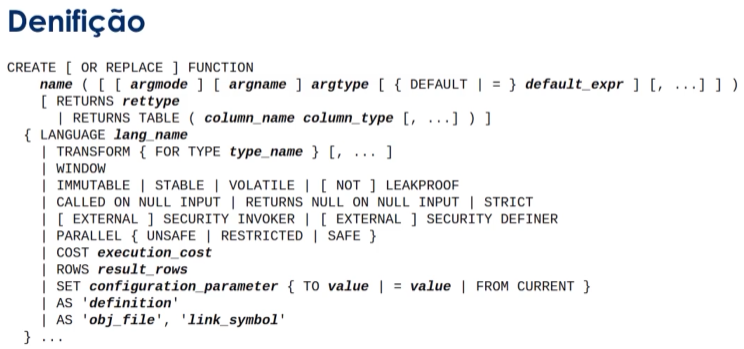
Porém, o foco aqui é falar sobre USER DEFINED FUNCTIONS.

Funções que podem ser criadas pelo usuário.

Linguagens que podemos utilizar na USER DEFINED FUNCTIONS:

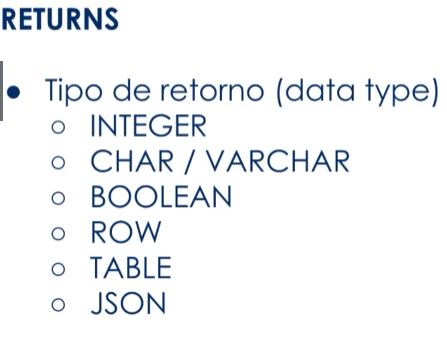
* SQL
* PL/PGSQL
* PL/PY
* PL/PHP
* PL/RUBY
* PL/JAVA
* PL/LUA
* ...........

<http://www.postgresql.org/docs/11/external-pl.html>





Previne erros na aplicação, dá segurando no BD, traz uma melhor prática de programação.

Languages que iremos utilizar:

* SQL
* PLPGSQL

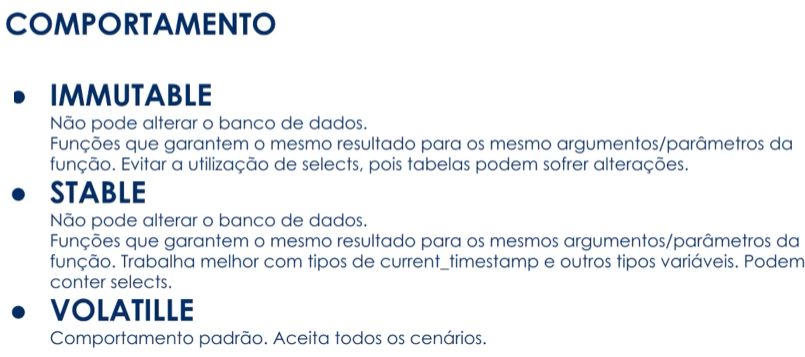
SEGURANÇA

* SECURITY
  + INVOKER (padrão) – permite que a função seja executada com as permissões do usuário que está executando a função;

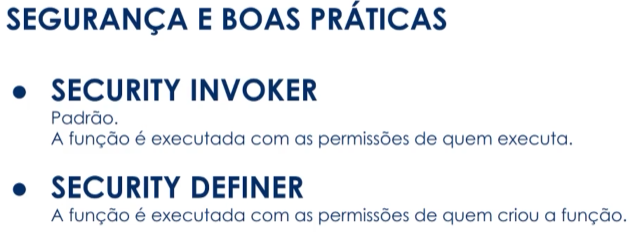
EX.: usuário não pode fazer INSERT e a função possui um INSERT a ser executado, desta forma a função não irá executar por conta da permissão do usuário

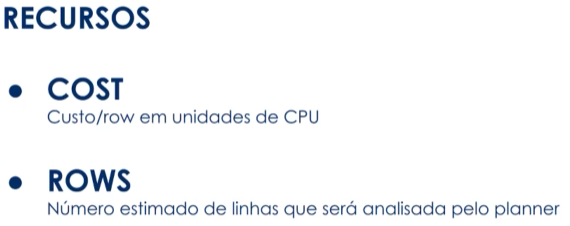
* + DEFINER – permite que o usuário execute a função conforme as permissões do usuário que criou a função;

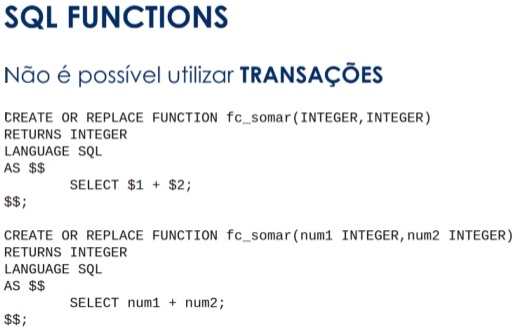
EX.: função foi criada por um superusuário então mesmo que o usuário que esteja executando a função não possa fazer um INSERT por exemplo, a função executará da mesma forma devido em sua criação ter sido criada por um super usuário.

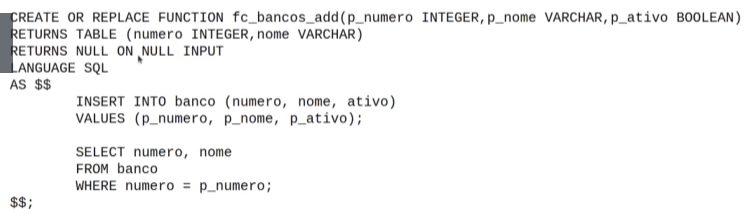


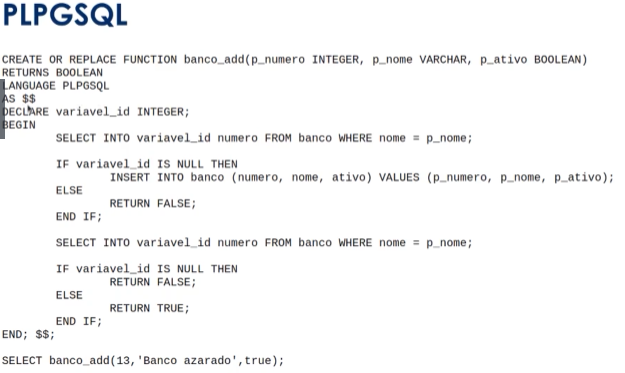












BIBLIOGRÁFIA

<https://www.vivaolinux.com.br/artigo/PostgreSQL-94-O-conceito-de-Role>