CRIADO PELA FERNANDA MAKI HIROSE

dados: são valores armazenados que não sofrem nenhum tipo de alteração
modelo relacional: classifica e organiza as informações em tabelas com linhas e colunas
tabelas: podem ser carro, produto, animal, perfil de usuário, status de compra, produto de pedido, histórico de dados
- primary key: ele é o índice para criar relacionamentos com outras tabelas, ele nunca se repete, o seu valor precisa existir
- foreign key: faz referência a uma primary key para criar relacionamentos
- sistemas de gerenciamento de banco de dados: softwares para gerenciar o banco de dados
- o postgres: é um modelo cliente-servidor (para usar ele usa-se a parte do servidor e a parte do cliente, exemplo: terminal), é possível restaurar seus dados, ele tem suporte procedural com perl, python, etc, é possível fazer consultas complexas e common table expressions, suporte a dados geográficos (postGIS), controle de concorrência multi-versão
site oficial: https://www.postgresql.org/
documentação: https://www.postgresql.org/docs/

instalação: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
- o local de destino do arquivo a ser instalado: deixe padrão
- select components: instale todos
- data directory: pode clicar em next
- senha: crie uma senha
- porta: deixe a porta padrão
- database cluster: default locale
- no windows procure por 'services' - procure por 'postgres.sql'
- para ver o programa procure por 'pgadmin'

- postgresql.conf: apresenta todas as configurações do postgresql
view pg_settings: é possível visualizar todas as configurações atuais:
SELECT name, settings
FROM pg_settings;
listen addresses: endereços tcp/ip que o postgresql vai executar/liberar conexões
port: é a porta que o postgresql vai ouvir, a padrão é 5432
max_connections: números máximos de conexões simultâneas (uma conexão consome muitos dados, então cuidado)
superuser_reserved_connections: números de conexões reservadas para super usuários (se acontecer algum problema o superusuário pode ir lá e resolver o problema)
authentication_timeout: o tempo máximo para o cliente abrir uma conexão (se passar do tempo vai dar um erro)
password_encryption: é a criptografia das senhas dos novos usuários
ssl: é uma conexão por ssl
shared_buffers: tamanho da memória compartilhada no postgresql para cache/buffer de tabelas, índices, etc (tudo que roda em disco é muito mais lento que memória, tudo que é preciso rodar mais rápido é colocado em cache)
work_mem: tamanho da memória para operações de agrupamento e ordenação (ORDER BY, DISTINCT, MERGE JOINS)
maintenance_work_mem: tamanho da memória para operações como VACUUM, INDEX, ALTER TABLE

- pg_hba.conf: é responsável pelo controle de autenticação dos usuários
trust: conexão sem requisição de senha
reject: rejeitar conexões
md5: criptografia md5
password: senha sem criptografia
gss: generic security service application program interface
sspi: security support provider interface - somente para windows
krb5: kerberos v5

ident: utiliza o usuário do sistema operacional do cliente via ident server
peer: utiliza o usuário do sistema operacional do cliente
ldap: ldap server
radius: radius server
cert: autenticação via certificado ssl do cliente
pam: pluggable authentication modules
dica extra: sempre use o seu ip no banco de dados, para que ninguém além da sua máquina consiga entrar e não coloque asterisco no servidor de produção (é uma falta de segurança)

- pg_ident.conf: mapeia os usuários do sistema operacional com os usuários do banco de dados

- comandos administrativos: no seu computador vá em 'services' , procure por 'postgresql', se você clicar com o botão direito você vai ver todos os comandos administrativos

- binários do postgresql: createdb (cria um banco de dados), createuser (cria um usuário), dropdb (apaga um banco de dados), dropuser (apaga um usuário), initdb (cria arquivos e diretórios), pg_clt (controla o banco de dados), pg_basebackup (vai criar um backup de toda a estrutura), pg_dump / pg_dumpall (é um pseudo backup), pg_restore (restaura os arquivos), pspl (junta um banco de dados via terminal), reindexdb (faz um reindex), vacuumdb (reorganiza todas as tabelas)

- cluster: é uma coleção de banco de dados que compartilham a mesma configuração, pode ter um cluster rodando em cada porta

- schemas: é um conjunto de objetos e relações (tabelas, funções, views), o mysql trata ele como um banco de dados

se você não estiver conseguindo se conectar com o banco de dados você precisa:
- ir no curso da dio 'conceito e melhores práticas com banco de dados postgresql' e ir no vídeo 'conheça a ferramenta pgadmin'

mão a massa
no seu banco de dados vá em 'file' - 'preferences' - 'auto completion' - 'keywords in uppercase' -> ative essa opção

criar um server group: clique em cima de 'servers' - 'create' - 'server group' - dê um nome para ele
criar um servers: clique em cima de 'servers' - 'create' - 'server' - em 'name' dê um nome, pode ser o mesmo nome do server group, deixe um comentário
criando conexões: nas conexões: no 'host' coloque o endereço da sua máquina (127.0.0.1), a porta: (5432), maintenance database (coloque o database que você está configurando), username (é o seu nome de usuário), senha: (coloque a senha)
criando o banco de dados: clique no seu database - 'tools' - 'query tools', é aqui que você pode escrever os seus comandos, o ícone que executa a ação

exemplo de ação criada:
CREATE DATABASE auladb;

depois disso clique com o botão direito no seu database - 'refresh', vá em 'tools' - 'query tools', aqui você pode escrever seus comandos

users/roles/groups
- roles: papéis ou funções
- users: usuários
- groups: perfis com permissões em comum ou específicas

administrando users/roles/groups
no banco de dados digite:
CREATE ROLE name [[WITH] option []]

dentro do option pode aparecer:
SUPERUSER (vai criar um super usuário, evite de usar ou se usar use com cautela)
NOSUPERUSER (não vai criar um super usuário)
CREATEDB (vai criar um banco de dados)
NOCREATEDB (não vai criar um banco de dados)
CREATE ROLE nomedarole (vai nova uma nova role)
NOCREATEROLE (não vai criar uma nova role)
INHERIT (sempre que a role pertencer a uma outra role ela vai herdar as permissões da outra role)
INHERIT (não vai herdar as permissões da outra role)
LOGIN (pode se conectar no banco de dados)
NOLOGIN (não pode se conectar no banco de dados)
REPLICATION (vai fazer backup ou replicação)
NOREPLICATION (não vai fazer backup ou replicação)
BYPASSRLS (trabalha com a segurança)
NOBYPASSRLS (trabalha com a segurança)
CONNECTION LIMIT numerolimite (quantas conexões simultâneas a role pode ter dentro do banco de dados)
[ENCRYPTED] PASSWORD 'password' (a senha pode ser criptografada, por padrão toda senha do banco de dados é criptografada, mas tem como mudar para outra criptografia)
PASSWORD NULL (a senha não é obrigatória)
VALID UNTIL 'timestamp' (tempo limite que você quer que a role tenha acesso ao banco de dados)
IN ROLE role_name [, ...] (a role vai tem que ser a role definida pelo in role)
ROLE role_name [, ...] (a role vai pertencer ao novo grupo da role sendo criada)
ADMIN role_name [, ...] (todas as roles passarão a fazer parte da nova role e terão acessos administrativos)
USER role_name [, ...] (usamos o user como a role)
SYSID uid (usa-se apenas para incompatibilidade)

ASSOCIAÇÃO ENTRE ROLES

---

quando uma role assume as permissões de outra role:
necessário a opção INHERIT

no momento da 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]

exemplo

```
CREATE ROLE professores
        NOCREATEDB
        NOCREATEROLE
        INHERIT
        NOLOGIN
        NOBYPASSRLS
        CONNECTION LIMIT -1;

CREATE ROLE daniel LOGIN CONNECTION LIMIT 1 PASSWORD '123' IN ROLE professores;
```

(a role daniel passa a assumir as permissões da role professores)

CREATE ROLE daniel LOGIN CONNECTION LIMIT 1 PASSWORD '123' ROLE professores;
(a role professores passa a assumir as permissões da role daniel)

CREATE ROLE daniel LOGIN CONNECTION LIMIT 1 PASSWORD '123';
GRANT professores TO daniel;
(a administração e a associação entre as duas roles)

desassociar membros entre roles
REVOKE [role que será revogada] FROM [role que terá suas permissões revogadas]

alterando uma role
ALTER ROLE nomedarole [WITH] option […]

excluindo uma role
DROP ROLE nomedarole;

comentando uma linha
-- DROP ROLE nomedarole;

configurações para logar com o usuário postgres
https://jquerydicas.blogspot.com/2013/10/psql-nao-e-reconhecido-como-um-comando.html
entre nas pastas após encontrar o local no comando dir psql.exe /s /b
depois digite no terminal: psql -U postgres

checar se a role foi criada
no terminal (no path de dir psql.exe /s /b) digite:
psql
(digite a senha)
\du (vai listar todas as roles criadas)
SELECT * FROM pg_roles ; (mostra todas as roles disponíveis no banco de dados)
\q (para sair do banco de dados)
psql -U nomedarole nomedobancodedados (vai checar apenas uma role)

grants (privilégios)
- tabela
- coluna
- sequence
- database
- domain
- foreign data wrapper
- foreign server
- function
- language
- large object
- schema
- tablespace
- type

database
GRANT {{ CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
TO nomedarole [, ...] [WITH GRANT OPTION]

schema

```
GRANT {{ CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
        ON SCHEMA schema_name [, ...]
        TO nomedarole [, ...] [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 nomedarole [, ...] [ WITH GRANT OPTION ]
```

table - exemplo

```
CREATE TABLE IF NOT EXISTS banco (
        numero INTEGER NOT NULL,
        nome VARCHAR(50) NOT NULL,
        ativo BOOLEAN NOT NULL DEFAULT TRUE,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY (numero)
);
```

descrevendo: default true = vê se o banco está ativo ou inativo
default current_timestamp = o valor default é o valor da hora atual

```
CREATE TABLE IF NOT EXISTS agencia (
        banco_numero INTEGER NOT NULL,
        numero INTEGER NOT NULL,
        nome VARCHAR(80) NOT NULL,
        ativo BOOLEAN NOT NULL DEFAULT TRUE,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY (banco_numero, numero),
        FOREIGN KEY (banco_numero) REFERENCES banco (numero)
);
```

descrevendo: a primary key está passando os campos que nunca podem se repetir, na foreign key mostra o campo que é referência de outro campo

```
CREATE TABLE IF NOT EXISTS cliente (
        numero BIGSERIAL PRIMARY KEY,
        nome VARCHAR(120) NOT NULL,
        email VARCHAR(250) NOT NULL,
        ativo BOOLEAN NOT NULL DEFAULT TRUE,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS conta_corrente (
        banco_numero INTEGER NOT NULL,
        agencia_numero INTEGER NOT NULL,
        numero BIGINT NOT NULL,
        digito SMALLINT NOT NULL,
        cliente_numero BIGINT NOT NULL,
        ativo BOOLEAN NOT NULL DEFAULT TRUE,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY (banco_numero, agencia_numero, numero, digito, cliente_numero),
        FOREIGN KEY (banco_numero, agencia_numero) REFERENCES agencia (banco_numero,
numero),
        FOREIGN KEY (cliente_numero) REFERENCES cliente (numero)
```

```
);

CREATE TABLE IF NOT EXISTS tipo_transacao (
        id SMALLSERIAL PRIMARY KEY,
        nome VARCHAR(50) NOT NULL,
        ativo BOOLEAN NOT NULL DEFAULT TRUE,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS cliente_transacoes (
        id BIGSERIAL PRIMARY KEY,
        banco_numero INTEGER NOT NULL,
        agencia_numero INTEGER NOT NULL,
        conta_corrente_numero BIGINT NOT NULL,
        conta_corrente_digito SMALLINT NOT NULL,
        cliente_numero BIGINT NOT NULL,
        tipo_transacao_id SMALLINT NOT NULL,
        valor NUMERIC(15, 2) NOT NULL,
        data_criacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (banco_numero, agencia_numero, conta_corrente_numero,
conta_corrente_digito, cliente_numero) REFERENCES conta_corrente (banco_numero,
agencia_numero, numero, digito, cliente_numero)
);
```

revoke - database

```
REVOKE [ GRANT OPTION FOR ]
        {{ CREATE | CONNECT | TEMPORARY | TEMP} [, ...] | ALL [PRIVILEGES] }
        ON DATABASE database_name [, ...]
        FROM { [ GROUP ] nomedarole | PUBLIC} [, ...]
        [ CASCADE | RESTRICT ]
```

revore - schema

```
REVOKE [ GRANT OPTION FOR ]
        {{ CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
        ON SCHEMA schema_name [, ...]
        FROM { [ GROUP ] nomedarole | PUBLIC} [, ...]
        [ CASCADE | RESTRICT ]
```

revoke - table

```
REVOKE [ GRANT OPTION FOR ]
        {{ SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER } [, ...] |
ALL [PRIVILEGES] }
        ON { [TABLE] table_name [, ...]
                | ALL TABLES IN SCHEMA schema_name [, ...] }
        FROM { [ GROUP ] role_name | PUBLIC } [, ...]
        [ CASCADE | RESTRICT ]
```

revogando todas as permissões

```
REVOKE ALL ON ALL TABLES IN SCHEMA [schema] FROM [role];
REVOKE ALL ON SCHEMA [schema] FROM [role];
REVOKE ALL ON DATABASE [database] FROM [role];
```

database

é um banco de dados, dentro dele tem schemas e 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 do cluster postgresql

schemas
é um grupo de objetos como tabelas, types, views, funções, entre outros. é possível relacionar objetos entre diversos schemas, é possível relacionar objetos entre diversos schemas

objetos
são as tabelas, views, funções, types, sequences, entre outros, pertencentes ao schema

database - comandos

```
CREATE DATABASE name
        [ [with] [owner [=] user_name ]
                [ TEMPLATE [=] template ]
                [ ENCODING [=] encoding ]
                [ LC_COLLATE [=] lc_collate ]
                [ LC_CTYPE[=] lc_ctype ]
                [ TABLESPACE [=] tablespace_name ]
                [ ALLOW_CONNECTIONS [=] allowconn]
                [ CONNECTION LIMIT [=] connlimit ]
                [ IS_TEMPLATE [=] istemplate ]

ALTER DATABASE name RENAME TO new_name
ALTER DATABASE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
ALTER DATABASE name SET TABLESPACE new_tablespace

DROP DATABASE [nome]
```

schema - comandos

```
CREATE SCHEMA schema_name [ AUTHORIZATION nomedarole ]
ALTER SCHEMA name RENAME TO new_name
ALTER SCHEMA name OWNER TO { new_owner | CURRENT_USER | SESSION_USER }

DROP SCHEMA [nome]
```

melhores práticas:

```
CREATE SCHEMA IF NOT EXISTS chema_name [ AUTHORIZATION nomedarole ]
DROP SCHEMA IF EXISTS [nome];
```

colunas: são os atributos da tabela
linhas: são as tuplas da tabela, é onde são contidos os valores, os dados
tipos de dados:
- numeric types
- monetary types
- character types
- binary data types
- date/time types
- boolean type
- enumerated types
- geometric types
- network address types
- geometric types
- network address types
- bit string types
- text search types
- UUID type
- XML type
- JSON types

- arrays
- composite types
- range types
- domain types
- object identifier types
- pg_lsn type
- pseudo-types

tipo numérico
- smallint: -32,769 até +32,767
- integer: -2,147,483,648 até 2,147,483,647
- bigint: -9,223,372,036,854,775,808 até 9,223,372,036,854,775,807
- decimal/numeric: até 131072 dígitos antes do ponto decimal e até 16383 depois do ponto decimal
- real: 6 dígitos
- double precision: 15 dígitos
- smallserial: 1 até 32767
- serial: 1 até 2147483647
- bigserial: 1 até 9,223,372,036,854,775,807

caracteres
character varying(n), varchar(n): tamanho de variável com limite
character(n), char(n): valor fixo
text: tamanho ilimitado

horas
timestamp [(p)] [without time zone]
(maior valor: 294276 AD, menor valor: 4713 BC, traz o ano, mês, dia, hora, minuto e segundo)

timestamp [(p)] with time zone
(maior valor: 294276 AD, menor valor: 4713 BC, traz o ano, mês, dia, hora, minuto e segundo)

date
(maior valor: 5874897 AD, menor valor: 4713 BC, traz o ano, mês e dia)

time [(p)] [without time zone]
(maior valor: 24:00:00, menor valor: 00:00:00, traz a hora)

timestamp [(p)] with time zone
(maior valor: 24:00:00-1459, menor valor: 00:00:00+1459, traz a hora)

interval [fields] [(p)]
(maior valor: 178000000, menor valor: -178000000, traz a diferença entre duas datas)

booleanos
boolean
(traz true ou false)

DML
Data Manipulation Language (Linguagem de Manipulação de dados)
INSERT, UPDATE, DELETE, SELECT
(alguns consideram o select como DML, outros como DQL, que significa data query language ou linguagem de consulta de dados)

DLL
Data Definition Language (Linguagem de definição de dados)

CREATE, ALTER, DROP

create
```
CREATE [objeto] [nome do objeto] [opções]
```

alter
```
ALTER [objeto] [nome do objeto] [opções]
```

drop
```
DROP [objeto] [nome do objeto] [opções]
```

database
```
CREATE DATABASE dadosbancarios;
ALTER DATABASE dadosbancarios OWNER TO diretoria;
DROP DATABASE dadosbancarios;
```

schema
```
CREATE SCHEMA IF NOT EXISTS bancos;
ALTER SCHEMA bancos OWNER TO diretoria;
DROP SCHEMA IF EXISTS bancos;
```

tabelas
```
CREATE TABLE [IF NOT EXISTS] [nome da tabela] (
        [nome do campo] [tipo] [regras] [opções]
)

ALTER TABLE [nome da tabela] [opções];
DROP TABLE [nome da tabela];
```

exemplo 1
```
CREATE TABLE IF NOT EXISTS banco (
        codigo INTEGER PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        data_criacao TIMESTAMP NOT NULL DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS banco (
        codigo INTEGER,
        name VARCHAR(50) NOT NULL,
        data_criacao TIMESTAMP NOT NULL DEFAULT NOW(),
        PRIMARY KEY (codigo)
);

ALTER TABLE banco ADD COLUMN tem_poupanca BOOLEAN;
DROP TABLE IF EXISTS banco;
```

insert
```
INSERT INTO [nome da tabela] ([campos da tabela,])
VALUES ([valores de acordo com a ordem dos campos acima,]);

INSERT INTO [nome da tabela] ([campos da tabela,])
SELECT ([valores de acordo com a ordem dos campos acima,]);
```

exemplo 1
```
INSERT INTO banco (codigo, nome, data_criacao)
VALUES (100, 'Banco do Brasil', now());
```

```
INSERT INTO banco (codigo, nome, data_criacao)
SELECT 100, 'Banco do Brasil', now();
```

idempotência
não usamos idempotência nesse caso, usamos 'on conflict'

```
INSERT INTO agencia (banco_numero, numero, nome) VALUES (341, 1, 'Centro da cidade') ON CONFLICT (banco_numero, numero) DO NOTHING;
```

UPDATE

```
UPDATE [nome da tabela] SET
[campo1] = [novo valor do campo1],
[campo2] = [novo valor do campo2],
...
[WHERE + condições]
```

atenção: muito cuidado com os updates. sempre utilize-os com condição.

exemplo 1

```
UPDATE banco SET
codigo = 500
WHERE codigo = 100;

UPDATE banco SET
data_criacao = now()
WHERE data_criacao IS NULL;
```

DELETE

```
DELETE FROM [nome da tabela]
[WHERE + condições]
```

atenção: muito cuidado com os deletes. sempre utilize-os com condição.

exemplo 1

```
DELETE FROM banco
WHERE codigo = 512;

DELETE FROM banco
WHERE nome = 'Conta Digital';
```

SELECT

```
SELECT [campos da tabela]
FROM [nome da tabela]
[WHERE + condições]
```

atenção: evite usar SELECT *, e automações em bancos de dados

exemplo 1

```
SELECT codigo, nome
FROM banco;
```

```
SELECT codigo, nome
FROM banco
WHERE data_criacao > '2019-10-15 15:00:00';
```

exemplo 2
```
SELECT nome FROM cliente WHERE email ILIKE '%gmail.com';
```
(o % vai fazer um select de todos os campos que terminal com gmail.com)

exemplo 3
```
SELECT numero, nome FROM banco WHERE ativo IS TRUE;
```
(vai fazer um select onde o ativo for verdadeiro)

alerta: não é recomendado fazer um select e abrir parênteses e com isso usar o ILIKE, porque isso come muitos recursos do seu banco de dados e não é a forma mais inteligente, tem mais chances de dar erro

select *
evitar usar, pois vai retornar todas as tabelas, isso come muitos recursos

IDEMPOTÊNCIA

é uma propriedade que possibilita uma ação ser executada mais de uma vez, usamos: IF EXISTS
- se você tem dúvida se um campo pode ficar nulo, por exemplo: número de telefone, não é todo mundo que tem, então não é necessário ficar junto com a tabela 'cliente' por exemplo, pode criar outra tabela

- cuidado com as constraints, quanto mais regras você tiver mais vai dificultar na hora de fazer um update ou um insert
- cuidado com o excesso de fk, porque elas também são constraints
- cuidado com o tamanho indevido de colunas, exemplo: coluna CEP VARCHAR(255)

CONDIÇÃO (WHERE/AND/OR)

=
>
>=
<
<=
<>
!=
LIKE
ILIKE
IN
(a primeira condição sempre é where, as demais é 'and' ou 'or')

idempotência
```
SELECT (campos) FROM tabela1
WHERE EXISTS (
        SELECT (campo,)
        FROM tabela2
        WHERE campo1 = valor1
        [AND/OR campoN = valorN]
);
```

(não é uma boa prática, melhor prática é usar 'left join')

TRUNCATE

o truncate esvazia a tabela.

TRUNCATE [TABLE] [ONLY] nomedatabela [*] [, ...]
[RESTART IDENTITY | CONTINUE IDENTITY] [CASCADE | RESTRICT]

usa-se restart identity ou continue identity, o padrão é usar continue identity
restart identity: não é recomendado, nele você passa um id e a tabela recomeça a partir dele, não é nada recomendado
continue identity: vai limpar a tabela e a tabela começa do id que parou
restrict: é o padrão, se você tiver foreign keys ele não vai apagar a tabela por ter foreign keys
cascade: vai apagar tudo da tabela e se tiver referências em outras tabelas, ele vai apagar as referências

COMO VER AS COLUNAS SEM PRECISAR FAZER SELECT *

SELECT column_name FROM information_schema.columns WHERE table_name = 'banco';

(vai pegar o nome das colunas da tabela 'banco')

SELECT column_name, data_type FROM information_schema.columns WHERE table_name = 'banco';
(vai pegar a coluna 'data type' da tabela 'banco')

FUNÇÕES AGREGADAS

https://www.postgresql.org/docs/11/functions-aggregate.html essas são as funções agregadas, vamos ver algumas

avg
SELECT AVG(valor) FROM cliente_transacoes;
(vai calcular a média de todos os valores da coluna 'valor' na tabela 'cliente_transacoes')

count
SELECT COUNT (numero) FROM cliente;
(vai contar quantas vezes aparece os números na coluna 'numero' da tabela cliente)

SELECT COUNT (numero), email FROM cliente
WHERE email ILIKE '%gmail.com'
GROUP BY 'email';
(vai mostrar a quantidade de g-mail)

SELECT COUNT (id), tipo_transacao_id
FROM cliente_transacoes
GROUP BY tipo_transacao_id
HAVING COUNT (id) > 150;
(o having é muito bom para evitar de usar registros duplicados em uma tabela)

max
SELECT MAX (valor) FROM cliente;

(vai pegar o maior valor)

```
SELECT MAX (valor), tipo_transacao_id
FROM cliente_transacoes
GROUP BY tipo_transacao_id;
```

(quando tem mais de um valor passa o group by)

min

```
SELECT MIN (numero) FROM cliente;
```

(vai pegar o menor número)

```
SELECT MIN (valor), tipo_transacao_id
FROM cliente_transacoes
GROUP BY tipo_transacao_id;
```

(quando tem mais de um valor passa o group by)

sum

```
SELECT SUM (valor)
FROM cliente_transacoes;
```

(soma os valores)

```
SELECT SUM (valor), tipo_transacao_id
FROM cliente_transacoes
GROUP BY tipo_transacao_id ASC;
```

(quando tem mais de um valor passa o group by)

```
SELECT SUM (valor), tipo_transacao_id
FROM cliente_transacoes
GROUP BY tipo_transacao_id DESC;
```

JOIN - RELACIONAMENTO DE TABELAS

---

join (inner)
pegam os registros que têm em comum

```
SELECT tabela1.campos, tabela2.campos
FROM tabela1
JOIN tabela2 ON tabela2.campos  = tabela1.campos ;
```

left join (outer)
vai trazer todos os campos da tabela 1 e os campos em comum da tabela 2

```
SELECT tabela1.campos, tabela2.campos
FROM tabela1
LEFT JOIN tabela2 ON tabela2.campos  = tabela1.campos;
```

right join (outer)
vai trazer todos os campos da tabela 2 e os campos em comum da tabela 1

```
SELECT tabela1.campos, tabela2.campos
FROM tabela1
RIGHT JOIN tabela2 ON tabela2.campos  = tabela1.campos;
```

full join
ele traz os dois registros relacionados da tabela 1 e 2, os registros da tabela 1 que não tem relação com a tabela 2 e depois os registros da tabela 2 que não tem relação com a tabela 1

alerta: não é recomendado usar com frequência, pois vai trazer mais dados que o necessário

```
SELECT tabela1.campos, tabela2.campos
FROM tabela1
FULL JOIN tabela2 ON tabela2.campos  = tabela1.campos;
```

cross join
alerta: use o cross join para situações extremas, porque gera muitos dados, todos os campos da tabela 1 se relacionarão com os campos da tabela 2

```
SELECT tabela1.campos, tabela2.campos
FROM tabela1
CROSS JOIN tabela2
```

exemplos

```
SELECT count(distinct banco.numero)
FROM banco
JOIN agencia ON agencia.banco_numero = banco.numero;
JOIN conta_corrente ON conta_corrente.banco_numero = banco.numero;
AND conta_corrente.agencia_numero = agencia.numero
JOIN cliente ON cliente.numero = conta_corrente.cliente_numero;
```

(o 'count' fez a conta dos números e o 'distinct' mostrou o total
é possível adicionar mais 'join' dentro de um 'select' e é possível usar o 'and'
)

COMMON TABLE EXPRESSIONS

---

ajuda a organizar blocos de código (statements) para consultas muito grandes, gera tabelas temporárias e cria relacionamento entre elas, dentro dos blocos de código podem ter select, insert, update ou delete. depois do 'with' você coloca o nome da tabela temporária.

```
WITH [nome1] AS (
        SELECT (campos,)
        FROM tabela_A
        [WHERE]
),      [nome2] AS (
        SELECT (campos,)
        FROM tabela_B
        [WHERE]
)
SELECT [nome1].(campos,), [nome2].(campos,)
FROM [nome1]
JOIN [nome2]
```

exemplo

```
WITH tbl_tmp_banco AS (
        SELECT numero, nome
        FROM banco
)
SELECT numero, nome
FROM tbl_tmp_banco;

WITH params AS (
        SELECT 213 AS banco_numero
), tbl_tmp_banco AS (
```

```
        SELECT numero, nome
        FROM banco
        JOIN params ON params.banco_numero = banco_numero
)
SELECT numero, nome
FROM tbl_tmp_banco;

SELECT banco.numero, banco.nome
FROM banco
JOIN (
        SELECT 213 AS banco_numero
)       params ON params.banco_numero = banco.numero;

WITH clientes_e_transacoes AS (
        SELECT cliente.nome AS cliente_nome,
                        tipo_transacao.nome AS tipo_transacao_nome,
                        cliente_transacoes.valor AS tipo_transacao_valor
        FROM cliente_transacoes
        JOIN cliente ON cliente.numero = cliente_transacoes.cliente_numero
        JOIN tipo_transacao ON tipo_transacao.id = cliente_transacoes.tipo_transacao_id
        JOIN banco ON banco.numero = cliente_transacoes.banco_numero and banco.nome
ILIKE '%Itaú%'
)
SELECT cliente_nome, tipo_transacao_nome, tipo_transacao_valor
FROM clientes_e_transacoes;
```

VIEWS

---

as views são camadas para as tabelas, em vez de fazer um select você pode fazer uma view, elas são mais seguras, porque as pessoas vão acessar elas e não diretamente o banco de dados, é possível fazer select, **insert**, **update** e **delete**, as views que fazem referência a apenas 1 tabela só aceitam os tipos de comando em negrito

estrutura geral

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
        [ WITH (view_option_name [= view_option_value] [, ...] ) ]
        AS query
        [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

create or replace: cria ou substitui a view
temp/temporary: a view só vai existir na sessão
recursive: é um select dentro da view que vai chamar a própria view até esgotar uma determinada opção
with cascaded local check option: vão validações para os comandos insert, update e delete da sua view

idempotência

```
CREATE OR REPLACE VIEW vw_bancos AS (
        SELECT numero, nome, ativo
        FROM banco
);

SELECT numero, nome, ativo
FROM vw_bancos;
```

```
CREATE OR REPLACE VIEW vw_bancos (banco_numero, banco_nome, banco_ativo) AS (
        SELECT numero, nome, ativo
        FROM banco
);

SELECT banco_numero, banco_nome, banco_ativo
FROM vw_banco;
```

insert - update - delete
- funcionam apenas para views com apenas 1 tabela

```
INSERT INTO vw_bancos (numero, nome, ativo) VALUES (100, 'Banco CEM', TRUE);
UPDATE vw_bancos SET nome = 'Bancos 100' WHERE numero = 100;
DELETE FROM vw_bancos WHERE numero = 100;
```

temporary

```
CREATE OR REPLACE TEMPORARY vw_bancos AS (
        SELECT numero, nome, ativo
        FROM banco
);

SELECT numero, nomes, ativo
FROM vw_bancos;
```

descrevendo: o temporary é uma view qualquer, mas que funciona na janela ativa do seu navegador ou do seu terminal

recursive
- obrigatório a existência dos campos da view e do union all
- union (unifica), union all (não unifica)

```
CREATE OR REPLACE RECURSIVE VIEW vw_funcionarios (id, gerente, funcionario) AS (
        SELECT id, CAST(" AS VARCHAR) AS gerente, nome
        FROM funcionarios
        WHERE gerente IS NULL
        UNION ALL
        SELECT funcionarios.id, gerentes.nome, funcionarios.gerente, funcionarios.nome
        FROM funcionarios
        JOIN vw_funcionarios ON vw_funcionarios.id = funcionarios.gerente
        JOIN funcionarios gerentes ON gerentes.id = vw_funcionarios.id
);

SELECT id, gerente, funcionario
FROM vw_funcionarios
```

with option

```
CREATE OR REPLACE VIEW bancos_ativos AS (
        SELECT numero, nome, ativo
        FROM banco
        WHERE ativo IS TRUE
) WITH LOCAL CHECK OPTION;

INSERT INTO bancos_ativos (numero, nome ativo) VALUES (51, 'Banco Boa Ideia', False)
```

descrevendo: vai dar falso isso, porque no parâmetro que é para ser true está como false
alerta: se no lugar de LOCAL estiver o CASCADED, vai validar todas as views do 'bancos_ativos'

TRANSAÇÕES

muitos códigos reunidos em apenas 1 transação, onde o resultado precisa ser tudo ou nada. começa com o 'begin' e termina com o 'commit', se acontecer algum erro, desfaz tudo que está dentro do 'begin' e do 'commit', o 'savepoint' faz o código retornar para o código dentro dele caso aconteça um erro, o 'rollback' ignora o que o código do 'savepoint'

exemplo

```
BEGIN;

        UPDATE conta SET valor = valor - 100.00
        WHERE nome = 'Alice';

SAVEPOINT my_savepoint;

        UPDATE conta SET valor = valor + 100.00
        WHERE nome =        'Bob';

ROLLBACK TO my_savepoint;

        UPDATE conta SET valor = valor + 100.00
        WHERE nome =        'Wally';

COMMIT;
```

FUNÇÕES

somente algumas funções tem o recurso de ter transações, existem 4 tipos de funções:
- query language function: funções escritas em SQL
- procedural language functions (funções escritas em, por exemplo, PL/pgSQL ou PL/py)
- internal functions
- C-language functions

porém vamos falar sobre USER DEFINED FUNCTIONS
funções que podem ser criadas pelo usuário, o postgres deixa você usar linguagens procedurais, ou seja, funções que podem ser escritas em outras linguagens, como: PL/PY, PL/PHP, PL/RUBY, PL/JAVA, PL/LUA, **PL/PGSQL** (essa é a que vamos falar)

idempotência
CREATE OR REPLACE FUNCTION [nome da função]
- mesmo nome
- mesmo tipo de retorno
- mesmo número de parâmetros/argumentos

returns
- tipo de retorno (data type):
- integer
- char/varchar
- boolean
- row
- table
- json

segurança
SECURITY INVOKER: a função vai ser executada com as permissões de quem executa
SECURITY DEFINER: a função é executada com as permissões de quem criou a função

comportamento
IMMUTABLE: não podem alterar o banco de dados, funções que garantem o mesmo resultado para os mesmos argumentos/parâmetros da função. evitar a utilização de selects, pois tabelas podem sofrer alterações

STABLE: não podem alterar o banco de dados, funções que garantem o mesmo resultado para os mesmos argumentos/parâmetros da função. trabalha melhor com tipos de current_timestamp e outros tipos de variáveis. podem conter selects.

VOLATILLE: comportamento padrão. aceita todos os cenários.

segurança e boas práticas
CALLED ON NULL INPUT: padrão, se qualquer um dos parâmetros/argumentos for NULL, a função será executada.

RETURNS NULL ON NULL INPUT: se qualquer um dos parâmetros/argumentos for NULL, a função retornará NULL.

recursos
COST: custo/row em unidades de CPU
ROWS: número estimado de linhas que será analisada pelo planner

funções escritas em SQL
não é possível utilizar transações, dentro o dólar fica o corpo da função, no SQL não retornamos a função com 'return' e sim com 'select'

nesse exemplo, a função precisa retornar 101.

```
CREATE OR REPLACE FUNCTION func_somar(INTEGER, INTEGER)
RETURNS INTEGER
SECURITY DEFINER
RETURNS NULL ON NULL INPUT
LANGUAGE SQL
AS $$
	SELECT $1 + $2
$$;

SELECT func_somar(1, 100)
```

outro exemplo (o 'coalesce' retorna o primeiro número nulo)

```
CREATE OR REPLACE FUNCTION func_somar(INTEGER, INTEGER)
RETURNS INTEGER
SECURITY DEFINER
CALLED ON NULL INPUT
LANGUAGE SQL
AS $$
	SELECT $1 + $2
$$;

SELECT COALESCE(null, 'daniel')
```

função PLPGSQL

apresenta um ‘begin’ uma transação, se qualquer erro acontecer no meio da função, um rollback vai ser executado, de preferência faça funções que aceitem transações

```
CREATE OR REPLACE FUNCTION banco_add(p_numero INTEGER, p_nome VARCHAR, p_ativo
BOOLEAN)
RETURNS BOOLEAN
LANGUAGE PLPGSQL
AS $$
DECLARE variavel_id INTEGER;
BEGIN
	SELECT INTO variavel_id numero FROM banco WHERE nome = p_nome;
	IF variavel_id IS NULL THEN
		INSERT INTO banco (numero, nome, ativo) VALUES (p_numero, p_nome,
p_ativo);
	ELSE
		RETURN FALSE;
	END IF;
	SELECT INTO variavel_id numero FROM banco WHERE nome = p_nome;
	IF variavel_id IS NULL THEN
		RETURN FALSE;
	ELSE
		RETURN TRUE;
	END IF;
END; $$;

SELECT banco_add(13, ‘Banco Azarado’, true)
```