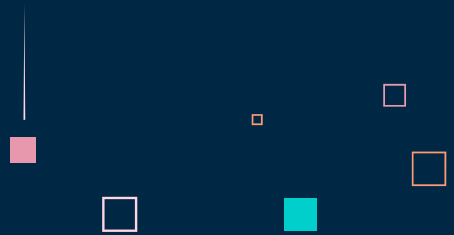
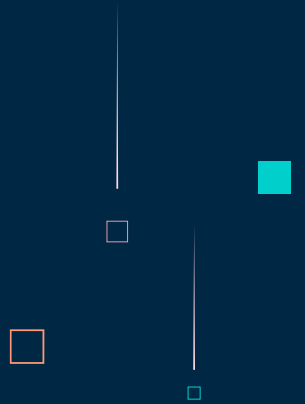
SQL

Conceitos complementares

AGENDA

- Domínios
- Transações
- Conversão de tipos
- Views
- Funções
- Triggers





DOMÍNIO

- É um tipo de dado definido pelo usuário, baseado em outro tipo de dado (primitivo ou outro domínio)
- Pode serem definidas restrições (CHECK)
- Pode ser usado na conversão de um valor também
- Se criado usando um schema, o domínio pertencerá somente à esse schema.
- Ideal para abstrair restrições comuns aos campos, facilitando a manutenção.
- Sintaxe padrão:

CRIAR DOMÍNIO

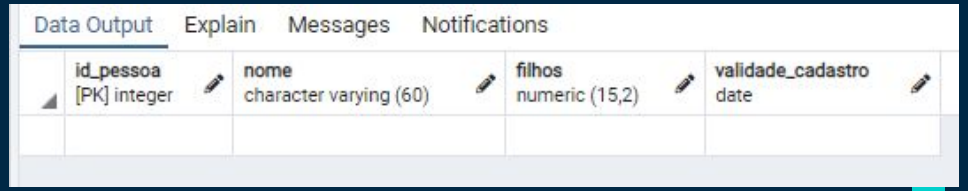
```
CREATE DOMAIN name AS data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]
```

Onde ***constraint*** is:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

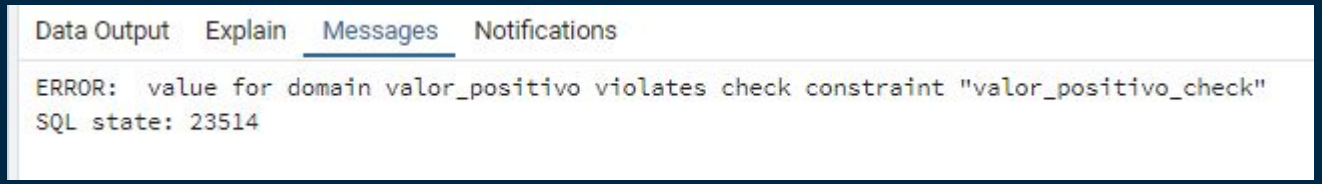
EXEMPLO DOMÍNIO

```
CREATE DOMAIN nomes AS VARCHAR(60) NOT NULL;

CREATE DOMAIN data_vencimento AS date CHECK (VALUE >=
current_date);

CREATE DOMAIN valor_positivo AS Numeric(15,2) CHECK (VALUE >= 0);

CREATE TABLE pessoas(
    id_pessoa serial NOT NULL,
    nome nomes,
    filhos valor_positivo,
    validade_cadastro data_vencimento,
    PRIMARY KEY(id_pessoa)
)
```



EXEMPLO DOMÍNIO

```
INSERT INTO pessoas (nome, filhos, validade_cadastro) VALUES
('Fernando',-1,'31/12/2021')
```




```
INSERT INTO pessoas (nome, filhos, validade_cadastro) VALUES
('Fernando',1,'31/12/2020')
```

EXEMPLO DOMÍNIO

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK(
   VALUE ~ '^\d{5}$'
OR VALUE ~ '^\d{5}-\d{4}$'
);

CREATE TABLE us_snail_addy (
  address_id SERIAL PRIMARY KEY,
  street1 TEXT NOT NULL,
  street2 TEXT,
  street3 TEXT,
  city TEXT NOT NULL,
  postal us_postal_code NOT NULL
);
```

TRANSAÇÕES

- É um mecanismo que permite manter a integridade dos dados quando estão sendo acessados/manipulados ao mesmo tempo.
- Pode ser tipo BLOCK ou MVCC - Controle de Simultaneidade multiversão.
- BLOCK é a forma tradicional, que bloqueia acesso à tabela para evitar

```
START TRANSACTION [ transaction_mode [, ...] ]

where transaction_mode is one of:
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED |
READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

TRANSAÇÕES - MVCC

- Desta forma, cada instrução SQL “vê” uma versão do banco de dados, como se fosse um “retrato” do BD, evitando que sejam exibidos dados inconsistentes durante a execução dos comandos.

- Esse processo se dá o nome de “isolamento de transação”

- Esse modelo de concorrência é uma evolução do tradicional bloqueio, permitindo que operações possam ser realizadas simultaneamente.

TRANSAÇÕES - MVCC

- Com o MVCC, a leitura nunca bloqueia a gravação e a gravação nunca bloqueia a leitura.
- Bloqueios também estão disponíveis no postgres (em Tabela e Linha)
- No isolamento das transações, existem 4 níveis, sendo o mais restrito o *Serializable*.
- Serializable: Garantia de que um conjunto de transações desse tipo seja executada como se fossem executadas uma por vez respeitando alguma ordem, ou seja, executasse uma após a outra.
- Os outros três são formas tratar concomitantemente as execuções ao mesmo tempo.

TRANSAÇÕES - FENÔMENOS

- **Leitura Suja:** Uma transação lê dados gravados por uma transação simultânea não confirmada.
- **Leitura não repetível:** Uma transação relê os dados lidos anteriormente e descobre que foram modificados por outra transação confirmada após a leitura inicial.
- **Leitura Fantasma:** Uma transação executa novamente uma consulta e descobre que o conjunto de linhas que satisfazem a condição mudou devido a outra transação confirmada recentemente.
- **Anomalia de serialização:** Resultado da confirmação bem sucedida de um grupo de transações, é inconsistente com a execução de todas as transações uma após a outra.

TIPOS DE TRANSAÇÕES x FENÔMENOS

Nível de Isolamento	Leitura suja	Leitura não repetível	Leitura fantasma	Anomalia de serialização
Leia sem compromisso (Read uncommitted)	Permitido, mas não no PG	Possível	Possível	Possível
Leia comprometido (Read committed)	Não é possível	Possível	Possível	Possível
Leitura repetível (Repeatable read)	Não é possível	Não é possível	Permitido, mas não no PG	Possível
Serializável (Serializable)	Não é possível	Não é possível	Não é possível	Não é possível

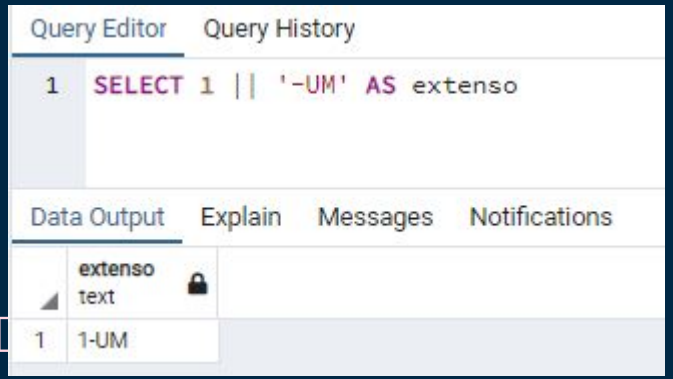
TRANSAÇÕES - PRINCIPAIS COMANDOS

- **SET TRANSACTION** — set the characteristics of the current transaction
- **START TRANSACTION** — start a transaction block
- **ROLLBACK** - aborta a transação atual
- **COMMIT** — commit the current transaction
- **END** - confirma a transação atual
- **ABORT** — abort the current transaction
- **BEGIN** — start a transaction block

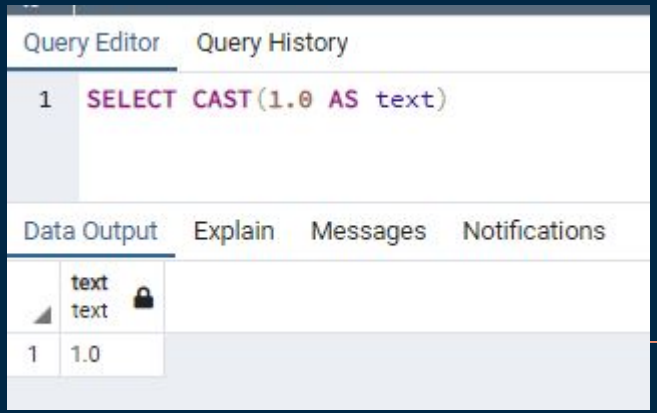
CONVERSÕES

- É possível converter o tipo dos dados.
- Em alguns casos, o PostgreSQL faz isso automaticamente.
- Em outros, pode ser necessário o usuário fazer uma conversão explícita com CAST(dado AS tipo) ou dado::tipo.

Ex conversão automática:



Ex conversão explícita



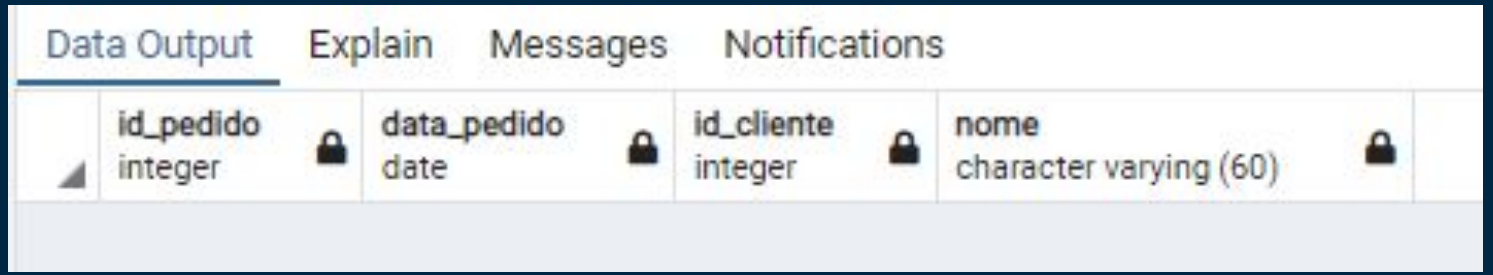
VIEWS

- É uma consulta salva onde a tabela não é materializada.
- Serve para armazenar consultas complexas ou de alto uso, facilitando o acesso a consulta.
- Sintaxe:

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

VIEWS - EXEMPLOS

```
CREATE OR REPLACE VIEW v_pedidos AS
SELECT p.id_pedido, p.data_pedido, p.id_cliente, c.nome FROM
clientes c INNER JOIN pedidos p USING (id_cliente);

SELECT * FROM v_pedidos;
```



FUNÇÕES

- Postgres possui a linguagem procedural Pl/pgSQL que permite principalmente:
  - Criação de Funções
  - Criação de Gatilhos (triggers)
  - Criação Procedimentos
  - Adicionar estruturas de controle ao SQL
  - Realizar cálculos complexos
- As funções em PL/pgSQL podem ser usadas em qualquer lugar que as funções de sistema podem ser usadas.

FUNÇÕES

- As funções são executadas no servidor.
- Sintaxe padrão é:

```
CREATE FUNCTION somefunc (integer, text) RETURNS integer
AS $$
' corpo da função'
$$ LANGUAGE plpgsql;
```

Sendo o corpo da função um bloco de código:

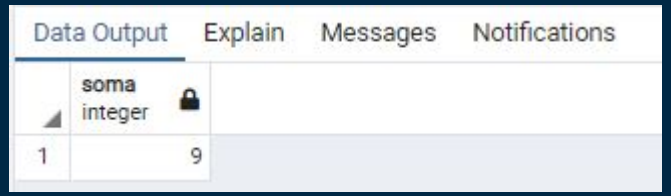
```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

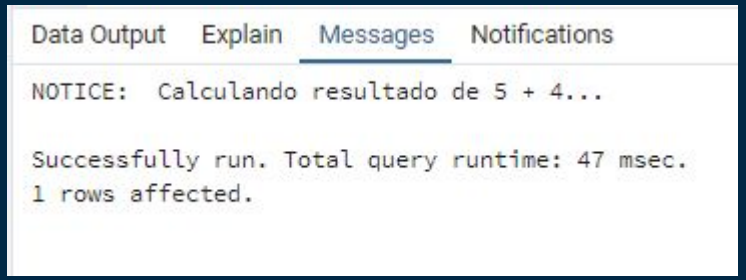
Variáveis.

Comandos da função.

FUNÇÕES

```
CREATE OR REPLACE FUNCTION soma(a integer, b integer) RETURNS integer
AS $$
DECLARE
    resultado integer;
BEGIN
    RAISE NOTICE 'Calculando resultado de % + %...', a, b;  --Exibir
mensagem.
    resultado = a + b;
    return resultado;
END;
$$ LANGUAGE plpgsql;

SELECT soma(5,4); --chamando a função.
```





FUNÇÕES

```
CREATE OR REPLACE FUNCTION preco_produto(_id_produto integer, _data date)
RETURNS Numeric AS $$
DECLARE
    preco produtos_precos.valor% TYPE;
BEGIN

    SELECT valor INTO preco
    FROM produtos_precos_y
    WHERE id_produtos_precos = _id_produto AND
            (_data BETWEEN data_inicio AND data_fim OR (_data >
data_inicio AND data_fim is null));

    return COALESCE(preco,0);
END;
$$ LANGUAGE PLPGSQL;

SELECT preco_produto(1,'01/05/2019');
```

Pega o tipo da coluna da tabela.

Armazena o resultado da consulta na variável.

	preco_produto numeric
1	2.95

TRIGGERS ou GATILHOS

- Pl/pgSQL permite a criação de gatilhos para alteração de dados ou eventos de um banco de dados.
- Primeiro se cria uma função de gatilho, com retorno **`trigger`** ou **`event_trigger`**.
- Depois se cria o gatilho com o **`CREATE TRIGGER`** referenciando a função e o evento que a disparará.
- Quando uma função é invocada por um gatílho, várias variáveis são criadas automaticamente e podem ser usadas na função.

TRIGGERS ou GATILHOS

- **NEW**: Novos valores em INSERT/UPDATE, antes de confirmar.
- **OLD**: valores antigos em DELETE/UPDATE, antes de executar.
- **TG_NAME**: nome da trigger atual.
- **TG_WHEN**: Texto. Indica a definição da trigger: BEFORE, AFTER, or INSTEAD OF.
- **TG_LEVEL**: Texto. Indice a triger esta definida para ROW STATEMENT.
- **TG_OP**: Texto. Indica se a trigger foi ativada por um INSERT, UPDATE, DELETE, or TRUNCATE.
- **TG_RELID**: Oid. retorna o id do objeto da tabela que invocou a trigger.
- **TG_TABLE_NAME**: Nome. Retorna o nome da tabela que invocou a trigger.
- **TG_TABLE_SCHEMA**: Name. Retorna o schema da tabela que invocou a trigger.

TRIGGERS ou GATILHOS - EXEMPLO

1

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);
```

2

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
  BEGIN
    -- Check that empname and salary are given
    IF NEW.empname IS NULL THEN
      RAISE EXCEPTION 'empname cannot be null';
    END IF;
    IF NEW.salary IS NULL THEN
      RAISE EXCEPTION '% cannot have null salary', NEW.empname;
    END IF;

    -- Who works for us when they must pay for it?
    IF NEW.salary < 0 THEN
      RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
    END IF;

    -- Remember who changed the payroll when
    NEW.last_date := current_timestamp;
    NEW.last_user := current_user;
    RETURN NEW;
  END;
$emp_stamp$ LANGUAGE plpgsql;
```

3

```
CREATE TRIGGER emp_stamp BEFORE INSERT
OR UPDATE ON emp
    FOR EACH ROW EXECUTE FUNCTION
emp_stamp();
```

TRIGGERS ou GATILHOS - EXEMPLO

1

```
CREATE TABLE log(
      id_log serial NOT NULL,
      tabela VARCHAR(30),
      operacao VARCHAR(30),
      old text,
      new text,
      datahora TIMESTAMP,
      PRIMARY KEY(id_log)
);

select * from log;
```

2

```
CREATE OR REPLACE FUNCTION registra_log() RETURNS trigger AS $$
DECLARE

BEGIN
        INSERT INTO log (tabela,operacao,old,new,datahora)
                VALUES (TG_TABLE_NAME,TG_OP,
                                OLD.id_cliente || '-' || OLD.nome || '-'||
OLD.id_grupo || '-' || OLD.cidade,
                                NEW.id_cliente || '-' || NEW.nome || '-'||
NEW.id_grupo || '-' || NEW.cidade,
                                current_timestamp);
        RAISE NOTICE 'Log Inserido com sucesso';
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

3

```
CREATE TRIGGER registra_log_clientes
AFTER INSERT OR UPDATE OR DELETE ON clientes
FOR EACH ROW EXECUTE FUNCTION registra_log();
```

TRIGGERS ou GATILHOS - EXEMPLO

```
INSERT INTO clientes VALUES
(99992,'Fernando',1,'Toledo','PR')
```

```
UPDATE clientes
SET nome = 'Fernando Botelho'
WHERE id_cliente = 99992
```

```
DELETE FROM clientes WHERE id_cliente = 99992
```

Data Output | Explain | Messages | Notifications

	id_log [PK] integer	tabela character varying (30)	operacao character varying (30)	old text	new text	datahora timestamp without time zone
1	2	clientes	INSERT	[null]	99992-Fernando-1-Toledo	2021-08-28 02:19:10.606423
2	3	clientes	UPDATE	99992-Fernando-1-Toledo	99992-Fernando Botelho-1-Toledo	2021-08-28 02:22:02.102108
3	4	clientes	DELETE	99992-Fernando Botelho-1...	[null]	2021-08-28 02:22:48.888413

REFERÊNCIAS

https://www.postgresql.org/docs/13/domains.html

https://www.postgresql.org/docs/13/sql-createdomain.html

https://www.postgresql.org/docs/13/sql-start-transaction.html

https://www.postgresql.org/docs/13/typeconv.html

https://www.postgresql.org/docs/13/sql-createview.html

https://www.postgresql.org/docs/13/plpgsql-overview.html

https://www.postgresql.org/docs/13/plpgsql-structure.html

https://www.postgresql.org/docs/13/plpgsql-trigger.html