



IgorAvilaPereira /
iobd2023_2sem



<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security



Dicas

[Jump to bottom](#)

Igor Avila Pereira edited this page on Jul 17 · 2 revisions

SQL

Operadores

- Lógicos and or not = <> != < <= > >=
- Números, inteiros e reais: + - * / % ^ between

Funções Numéricas

- abs(x), div(x, y), mod(x, y), ceil(x), floor(x), round(x), trunc(x), exp(x), ln(x), log(x), power(x, y), sqrt(x), random(), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), degrees(x), radians(x) e etc.

Funções para Strings

- length(x), lower(x), upper(x), trim(x), strpos(x, y), substr(x, y), substr(x, y, z), replace(x, y, z), repeat(x, y), translate(x, y, z) md5(x), to_char(x, y) e etc.

Principais Tipos de Dados - PostgreSQL

Nome	Descrição
boolean	booleano lógico (verdade/falso)
character varying [(n)]	cadeia de caracteres de comprimento variável
character [(n)]	cadeia de caracteres de comprimento fixo
date	data de calendário (ano, mês,dia)
double precision	número de ponto flutuante de precisão dupla
integer	inteiro de quatro bytes com sinal
money	quantia monetária
numeric [(p, s)]	numérico exato com precisão selecionável
real	número de ponto flutuante de precisão simples
serial	inteiro de quatro bytes com auto-incremento
text	cadeia de caracteres de comprimento variável
timestamp	data e hora
time	hora

Conversão de Tipos (cast)

```
select cast('true' as boolean);
select cast('12' as integer);
select cast('12.34' as real);
select cast('2000-12-31' as date);
select cast('12:00:00' as time);
select cast('2000-12-31 12:00:00' as timestamp);
select cast('3 months 10 days' as interval);
select cast(false as integer);
select cast(12.34 as integer);
```



CREATE DATABASE

```
CREATE DATABASE banco;
```



CREATE TABLE

```
create table tabela2 (
    campo1 serial primary key,
    campo2 real default 0,
    campo3 varchar(100),
```



```
    campo4 date default current_date
);
```

CHECK

```
CREATE TABLE products (
    product_no integer,
    name text DEFAULT 'Igor',
    -- valores da coluna "price" não podem ser nulos (NOT NULL) e devem ser maior que zero
    price numeric NOT NULL CHECK (price > 0)
);
```



PRIMARY KEY

```
create table tabela2 (
    -- pk do tipo serial (inteiro auto-incrementado)
    campo1 serial primary key,
    campo2 real default 0,
    campo3 varchar(100),
    campo4 date default current_date
);
```



FOREIGN KEY

```
CREATE TABLE products (
    -- criando a pk (tipo serial - inteiro auto-incrementado)
    product_no serial PRIMARY KEY,
    name text,
    price numeric
);
CREATE TABLE orders (
    -- criando a pk (tipo integer)
    order_id integer PRIMARY KEY,
    -- demais colunas
    quantity integer,
    -- fk referente a products (product_no)
    product_no integer REFERENCES products (product_no)
);
```



FOREIGN KEY ON DELETE CASCADE

```
CREATE TABLE products (
    product_no serial PRIMARY KEY,
    name text,
    price numeric
);
```



```
);
CREATE TABLE orders (
  order_id serial PRIMARY KEY,
  shipping_address text
);
CREATE TABLE order_items (
  -- fk que restringe/impede o DELETE
  product_no integer REFERENCES products ON DELETE RESTRICT,
  -- fk DELETE CASCADE
  order_id integer REFERENCES orders ON DELETE CASCADE,
  -- demais colunas
  quantity integer,
  -- pk composta
  PRIMARY KEY (product_no, order_id)
);
```

[Vídeo complementar](#)

UNIQUE

Assegura que os dados contidos em uma coluna (ou um grupo de colunas) é única no que diz respeito a todas as linhas da tabela.

```
CREATE TABLE example (
  a integer,
  b integer,
  c integer,
  UNIQUE (a, c)
);
```



ALTER TABLE

```
-- adicionando uma nova coluna "description" na tabela "products"
ALTER TABLE products ADD COLUMN description text;
-- removendo uma nova coluna "description" da tabela "products"
ALTER TABLE products DROP COLUMN description;
-- adicionando um check
ALTER TABLE products ADD CHECK (name <> '');
-- a coluna "product_no" não pode ser null
ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;
-- adicionando uma nova chave estrangeira
ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;
-- valor padrão da coluna "price" será 7.77
ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77;
-- removendo o valor padrão da coluna "price"
ALTER TABLE products ALTER COLUMN price DROP DEFAULT;
-- renomeando uma coluna
ALTER TABLE products RENAME COLUMN product_no TO product_number;
```



```
-- a tabela "products" terá um novo nome "items"  
ALTER TABLE products RENAME TO items;
```

DROP TABLE/DROP DATABASE

```
-- excluindo a tabela "tabela"  
DROP TABLE table;  
  
-- testa antes de tentar excluir a tabela "tabela"  
DROP TABLE IF EXISTS table;  
  
-- excluindo o banco "banco"  
DROP DATABASE banco;  
  
-- testa antes de tentar excluir o banco "banco"  
DROP DATABASE IF EXISTS banco;
```



INSERT

```
insert into tabela2 (campo2, campo3, campo4) values (123.45, 'abc', '2000-12-31');  
-- campo1 = autoincrementado
```



INSERT Múltiplo

```
-- Insere 5 registros em uma mesma instrução INSERT  
insert into montadora (codigo, nome) values  
(1, 'Ford'),  
(2, 'Chevrolet'),  
(3, 'Volkswagen'),  
(4, 'Fiat'),  
(5, 'Gurgel');
```



UPDATE

```
update tabela1 set campo2 = valor2+campo2 where campo1 = valor1;
```



SELECT

```
SELECT * from canal;
```



Apelidar colunas retornadas de um SELECT

```
-- A coluna "a" terá o nome de "value" e a soma "b+c" será chamada de "sum"
SELECT a AS value, b + c AS sum FROM ...
```



ORDER BY

```
-- ascendente/alfabética
SELECT * from canal order by nome ASC;
-- descendente
SELECT * from canal order by nome DESC;
```



Concatenação (Operador ||)

```
select 'abc' || 'def' as resultado; -- concatenação
```



LIKE/ILIKE

```
select 'abc' like 'a%' as resultado; -- começa com a?
select 'abc' like '_b_' as resultado; -- tem um caracter qualquer, b e um caracter qual
select 'abc' like '%c' as resultado; -- acaba com c?
select 'abcdefghi' like '%def%' as resultado; -- contém def?
select 'abcdefghi' not like '%def%' as resultado; -- não contém def?
```



IN/NOT IN

```
select 'a' in ( 'a', 'e', 'i', 'o', 'u' ) as resultado;
select 'x' not in ( 'a', 'e', 'i', 'o', 'u' ) as resultado;
```



EXISTS

- Se retornar pelo menos uma linha, o resultado de EXISTS é "verdade"
- Se a subconsulta não retorna nenhuma linha, o resultado de EXISTS é "falso"

```
SELECT col1 FROM tab1 WHERE EXISTS (SELECT 1 FROM tab2 WHERE col2 = tab1.col2);
```



DELETE

```
delete from tabela1 where campo1 = valor1;
```



Transações (BEGIN/COMMIT/ROLLBACK)

```
-- confirmando uma transação
begin;
select * from produto where codigo = 10;
update produto set quantidade = quantidade-1 where codigo = 10;
commit;

-- cancelando uma transação
begin;
select * from produto where codigo = 10;
update produto set quantidade = quantidade-1 where codigo = 10;
rollback;
```



Data/Hora

- **Constantes:** *current_date, current_time, current_timestamp*
- **Operadores:** + - *between*
- **Funções:** *now(), extract(day dow doy month year hour minute second), age(x)* e *to_date(x, y)*
- **Intervalos:** *day(s) month(s) year(s) hour(s) minute(s) second(s)*

```
select current_date-cast('1 day' as interval) as ontem;
select current_date+cast('1 day' as interval) as amanha;
SELECT CURRENT_DATE + interval '1 DAY' as amanha;
select
    extract(day from current_date) as dia,
    extract(dow from current_date) as dia_semana,
    extract(doy from current_date) as dia_ano;
select cast('2005-12-31' as date) between cast('2000-12-31' as date) and cast('2010-12-
select cast('2005-12-31' as date) not between cast('2000-12-31' as date) and cast('2010
select to_char(current_date, 'DD/MM/YYYY') as dma;
select to_date('31/12/2000', 'DD/MM/YYYY') as amd;
select NOW() + '28 DAYS'; -- 28 dias depois de agora
select current_date - interval '30 DAY' -- 30 dias antes de agora
```



CASE/WHEN/ELSE/END

```
-- ex1: Trocar 1 por one, 2 por two e os demais números por "other"
SELECT * FROM test;
a
---
```



```
1
2
3
```

```
SELECT a,
       CASE
         WHEN a=1 THEN 'one'
         WHEN a=2 THEN 'two'
         ELSE 'other' END
FROM test;
```

```
a | case
---+-----
1 | one
2 | two
3 | other
```

-- ex2: data por extenso

```
select
  case extract(dow from current_date)
    when 0 then 'domingo'
    when 1 then 'segunda'
    when 2 then 'terca'
    when 3 then 'quarta'
    when 4 then 'quinta'
    when 5 then 'sexta'
    when 6 then 'sabado'
  end || ', ' || to_char(current_date, 'DD') || ' de ' ||
  case extract(month from current_date)
    when 1 then 'janeiro'
    when 2 then 'fevereiro'
    when 3 then 'marco'
    when 4 then 'abril'
    when 5 then 'maio'
    when 6 then 'junho'
    when 7 then 'julho'
    when 8 then 'agosto'
    when 9 then 'setembro'
    when 10 then 'outubro'
    when 11 then 'novembro'
    when 12 then 'dezembro'
  end || ' de ' || to_char(current_date, 'YYYY') as hoje
```

LIMIT/OFFSET

-- Pulei os primeiros 10 programas do canal CAR e limitei em 10 registros
select * from programa where canal = 'CAR' limit 10 offset 10;



DISTINCT

```
-- nomes dos programas do canal CAR, sem repetições
select distinct nome from programa where canal = 'CAR' order by nome asc;
```



COUNT

```
-- quantas HDs de 500GB?
select count(*) as quantidade from produto where lower(descricao) like '%hd % 500%gb%';
```



GROUP BY

```
-- qual a quantidade de produtos por departamento?
select departamento, count(*) as quantidade from produto group by departamento;
```



HAVING

```
-- quais departamentos possuem menos de 10 produtos?
select departamento, count(*) as quantidade from produto group by departamento having c
-- where: filtro antes do group by
-- having: filtro depois do group by
```



INNER JOIN/CROSS JOIN/FULL JOIN/RIGTH JOIN/LEFT JOIN

```
create table montadora (
    codigo integer not null,
    nome varchar(100) not null,
    primary key (codigo)
);

create table modelo (
    codigo integer not null,
    nome varchar(100) not null,
    montadora integer,
    foreign key (montadora) references montadora(codigo),
    primary key (codigo)
);

insert into montadora (codigo, nome) values
(1, 'Ford'),
(2, 'Chevrolet'),
(3, 'Volkswagen'),
(4, 'Fiat'),
(5, 'Gurgel');
```



```
insert into modelo (codigo, nome, montadora) values
(11, 'Escort', 1),
(12, 'Corsa', 2),
(13, 'Gol', 3),
(14, 'Uno', 4),
(15, 'Countach', null);
```

-- produto cartesiano/CROSS JOIN

ex1: `select * from montadora, modelo;`

ex2: `SELECT * FROM montadora CROSS JOIN modelo;`

montadora			modelo		
codigo	nome		codigo	nome	montadora
1	Ford		11	Escort	1
1	Ford		12	Corsa	2
1	Ford		13	Gol	3
1	Ford		14	Uno	4
1	Ford		15	Countach	
2	Chevrolet		11	Escort	1
2	Chevrolet		12	Corsa	2
2	Chevrolet		13	Gol	3
2	Chevrolet		14	Uno	4
2	Chevrolet		15	Countach	
3	Volkswagen		11	Escort	1
3	Volkswagen		12	Corsa	2
3	Volkswagen		13	Gol	3
3	Volkswagen		14	Uno	4
3	Volkswagen		15	Countach	
4	Fiat		11	Escort	1
4	Fiat		12	Corsa	2
4	Fiat		13	Gol	3
4	Fiat		14	Uno	4
4	Fiat		15	Countach	
5	Gurgel		11	Escort	1
5	Gurgel		12	Corsa	2
5	Gurgel		13	Gol	3
5	Gurgel		14	Uno	4
5	Gurgel		15	Countach	

(25 registros)

-- JOIN/INNER JOIN

ex1: `select * from montadora join modelo on modelo.montadora = montadora.codigo;`

ex2: `select * from montadora inner join modelo on modelo.montadora = montadora.codigo;`

montadora			modelo		
codigo	nome		codigo	nome	montadora
1	Ford		11	Escort	1
2	Chevrolet		12	Corsa	2
3	Volkswagen		13	Gol	3
4	Fiat		14	Uno	4

(4 registros)

-- LEFT JOIN

```
select * from montadora left join modelo on modelo.montadora = montadora.codigo;
```

montadora			modelo		
codigo	nome		codigo	nome	montadora
1	Ford		11	Escort	1
2	Chevrolet		12	Corsa	2
3	Volkswagen		13	Gol	3
4	Fiat		14	Uno	4
5	Gurgel				

(5 registros)

-- RIGHT JOIN

```
select * from montadora right join modelo on modelo.montadora = montadora.codigo;
```

montadora			modelo		
codigo	nome		codigo	nome	montadora
1	Ford		11	Escort	1
2	Chevrolet		12	Corsa	2
3	Volkswagen		13	Gol	3
4	Fiat		14	Uno	4
			15	Countach	

(5 registros)

-- FULL JOIN (LEFT e RIGHT JOIN junto)

```
select * from montadora full join modelo on modelo.montadora = montadora.codigo;
```

montadora			modelo		
codigo	nome		codigo	nome	montadora
1	Ford		11	Escort	1
2	Chevrolet		12	Corsa	2
3	Volkswagen		13	Gol	3
4	Fiat		14	Uno	4
			15	Countach	
5	Gurgel				

(6 registros)

UNION

-- com union

```
(select mesa.nome as mesa
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-09' as date))
union
(select mesa.nome as mesa
```



```
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-16' as date))
order by 1 asc;
```

INTERSECT

```
-- com intersect
(select mesa.nome as mesa
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-09' as date))
intersect
(select mesa.nome as mesa
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-16' as date))
order by 1 asc;
```



EXCEPT

```
-- com except
(select mesa.nome as mesa
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-16' as date))
except
(select mesa.nome as mesa
from comanda
    join mesa on comanda.mesa = mesa.codigo
where comanda.data = cast('2020-08-09' as date))
order by 1 asc;
```



Blob's - Bytea e OID (Arquivos)

bytea

```
CREATE TABLE publicidade.banner (
    id SERIAL PRIMARY KEY,
    arquivo bytea,
    legenda text,
    altura integer,
    largura integer,
    link text, -- http://www.g1.com
    tipo text CHECK (tipo = 'SUPERIOR' OR tipo = 'INFERIOR') DEFAULT 'SUPERIOR',
    qtde_cliques INTEGER DEFAULT 0
);
```



```

1) INSERT INTO publicidade.banner (arquivo, legenda, link, tipo) VALUES
(pg_read_binary_file('/tmp/globo.png'), 'Clique Aqui', 'http://www.g1.com', 'SUPERIOR');

2) INSERT INTO publicidade.banner (arquivo, legenda, link, tipo) VALUES
(pg_read_file('/tmp/globo.png')::bytea, 'Clique Aqui', 'http://www.g1.com', 'SUPERIOR');

```

No JAVA:

```

-- classe de modelo
public class Banner {
    private int id;
    private byte[] arquivo;
    private String legenda;
    private int largura;
    private int altura;
    private String link;
    private String tipo;
    private int qtdeCliques;

    // getters and setters...
}

-- classe de persistência
public class BannerDAO {
    private ConexaoPostgreSQL conexaoPostgreSQL;

    public Banner obter(int id) throws SQLException{
        Banner b = new Banner();
        this.conexaoPostgreSQL = new ConexaoPostgreSQL();
        Connection conn = this.conexaoPostgreSQL.getConexao();
        String sql = "SELECT * FROM publicidade.banner WHERE id = ?";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        preparedStatement.setInt(1, id);
        ResultSet rs = preparedStatement.executeQuery();
        if (rs.next()){
            b.setId(rs.getInt("id"));
            b.setAltura(rs.getInt("altura"));
            b.setLargura(rs.getInt("largura"));
            b.setLegenda(rs.getString("legenda"));
            b.setLink(rs.getString("link"));
            b.setQtdeCliques(rs.getInt("qtde_cliques"));
            b.setTipo(rs.getString("tipo"));
            b.setArquivo(rs.getBytes("arquivo"));
        }
        conn.close();
        return b;
    }

    public void adicionar(Banner banner, String dir) throws SQLException, FileNotFoundException{
        this.conexaoPostgreSQL = new ConexaoPostgreSQL();
        Connection conn = this.conexaoPostgreSQL.getConexao();
        String sql = "INSERT INTO publicidade.banner "
            + " (arquivo, legenda, link, tipo) VALUES " +

```

```

        "(?,?,?,?);";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);

        // dir => diretorio + nome_arquivo + extensao
        File file = new File(dir);
        FileInputStream fis = new FileInputStream(file);
        preparedStatement.setBinaryStream(1, fis, file.length());

        preparedStatement.setString(2, banner.getLegenda());
        preparedStatement.setString(3, banner.getLink());
        preparedStatement.setString(4, banner.getTipo());
        preparedStatement.executeUpdate();
        conn.close();
    }

    public ArrayList<Banner> listar() throws SQLException {
        ArrayList<Banner> vetBanner = new ArrayList<Banner>();
        this.conexaoPostgreSQL = new ConexaoPostgreSQL();
        Connection conn = this.conexaoPostgreSQL.getConexao();
        String sql = "SELECT * FROM publicidade.banner";
        PreparedStatement preparedStatement = conn.prepareStatement(sql);
        ResultSet rs = preparedStatement.executeQuery();
        while (rs.next()){
            Banner b = new Banner();
            b.setId(rs.getInt("id"));
            b.setAltura(rs.getInt("altura"));
            b.setLargura(rs.getInt("largura"));
            b.setLegenda(rs.getString("legenda"));
            b.setLink(rs.getString("link"));
            b.setQtdeCliques(rs.getInt("qtde_cliques"));
            b.setTipo(rs.getString("tipo"));
            b.setArquivo(rs.getBytes("arquivo"));
            vetBanner.add(b);
        }
        conn.close();
        return vetBanner;
    }
    // ...
}

// Main
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws SQLException, FileNotFoundException {

        // escrita
        Banner bannerVetorial = new Banner();
        bannerVetorial.setLink("http://vetorial.net");
        bannerVetorial.setLegenda("clique aqui e contrate sua banda larga");
        bannerVetorial.setTipo("SUPERIOR");
        new BannerDAO().adicionar(bannerVetorial, "/home/iapereira/vetorial.png");

        // leitura
    }
}

```

```

BannerDAO bannerDAO = new BannerDAO();
Banner bannerGlobo = bannerDAO.obter(id);
ImageIcon imageIcon = new ImageIcon(bannerGlobo.getArquivo());
JFrame jFrame = new JFrame();
jFrame.setLayout(new FlowLayout());
jFrame.setSize(500, 500);
JLabel jLabel = new JLabel();
jLabel.setIcon(imageIcon);
jFrame.add(jLabel);
jFrame.setVisible(true);
jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

```

oid (Arquivos Grandes)

```

CREATE TABLE largeObjects_Devmedia
(
    cod_imagem INTEGER,
    nome_imagem VARCHAR(30),
    local_imagem oid,
    CONSTRAINT pk_cod_imagem PRIMARY KEY(cod_imagem)
);

```



```

INSERT INTO public.largeobjects_devmedia(cod_imagem, nome_imagem, local_imagem)
VALUES (1, 'naruto_shippuden', lo_import('D:/imagens/naruto_shippuden.jpg'));

```

Percebaa que no momento de inserção dos dados na tabela utilizamos a função específica *lo_import()*, que é utilizada para carregar imagens para a tabela de sistema pg_largeobjects.

Obs: Caso os dados não sejam inseridos na tabela, é necessário atribuir as devidas permissões no banco de dados, usando a seguinte instrução:

```

GRANT SELECT, INSERT, UPDATE ON pg_largeobject TO PUBLIC;

```



De forma similar a importação da imagem para a base de dados, podemos também exportá-la para a nossa máquina utilizando a função *lo_export()* com as informações de OID e o local no qual será armazenada a imagem como parâmetros , de acordo com a seguinte instrução:

```

SELECT lo_export(32784, 'D:/imagens/naruto_shippuden.jpg');

```



Temos também a função *lo_unlink()*, que é utilizada para realizar a remoção do objeto, como podemos observar na instrução a seguir:

```

SELECT lo_unlink(32784);

```



Links Complementares:

- 🔥 <https://www.postgresql.org/docs/7.4/jdbc-binary-data.html>
- <https://www.cybertec-postgresql.com/en/binary-data-performance-in-postgresql/>
- <http://postgresqlbr.blogspot.com/2013/04/trate-com-blobs-e-clobs-diretamente-no.html>
- <https://www.devmedia.com.br/trabalhando-com-large-objects-no-postgresql/34167>

SCHEMA

Um banco de dados pode conter um ou mais esquemas, que por sua vez contêm tabelas. Os esquemas também contêm outros tipos de objetos, incluindo tipos de dados, funções e operadores. EX: tanto *schema1* quanto *myschema* podem conter tabelas denominadas *mytable*. Ao contrário dos bancos de dados, os esquemas não são rigidamente separados: um usuário pode acessar objetos em qualquer um dos esquemas do banco de dados ao qual está conectado, se tiver privilégios para isso.

Ex:

```
DROP DATABASE IF EXISTS exemplo_bd_com_esquema;
CREATE DATABASE exemplo_bd_com_esquema;

\c exemplo_bd_com_esquema;

DROP SCHEMA IF EXISTS esquema;
-- criando um schema
CREATE SCHEMA esquema;

-- adicionando o esquema no conjunto de esquemas
SET search_path TO public, esquema;

DROP TABLE IF EXISTS esquema.tabela;
-- criar tabela no schema esquema (sem ser no public)
CREATE TABLE esquema.tabela (
    id serial primary key,
    nome text
);

INSERT INTO esquema.tabela (nome) VALUES ('Igor');

-- removendo schema em cascata
-- DROP SCHEMA esquema CASCADE;

-- criando uma tabela no esquema public
CREATE table teste (
    id serial primary key,
    nome text
);
```




```
-- acessar tabela de um schema
SELECT * FROM esquema.tabela;
```

VIEWS

Views são consideradas pseudo-tables, ou seja, elas são usadas junto com a instrução SELECT para apresentar subconjuntos de dados presentes em tabelas reais. Assim, podemos apresentar as colunas e linhas que foram selecionadas da tabela original ou associada. E como as Views possuem permissões separadas, podemos utilizá-las para restringir mais o acesso aos dados pelos usuários, para que veja apenas o que é necessário.

Quando uma visão é definida, o sistema de banco de dados armazena sua definição ao invés do resultado da expressão SQL que a definiu. Sempre que a relação visão é usada, ela é sobreposta pela expressão da consulta armazenada, de maneira que, sempre que a consulta for solicitada, a relação visão será recomputada.

As visões em SQL são geradas a partir do comando create view. A cláusula padrão é:

```
CREATE VIEW <nome da visão> AS <expressão de consulta>;
```



Caso não necessitemos mais de uma dada visão, podemos eliminá-la por meio do comando:

```
DROP VIEW <nome da visão>;
```



Ex:

```
-- listando todas as colunas da tabela COMPANY
SELECT * FROM COMPANY
```



id	name	age	address	salary
1	Paul	32	California	20000
2	Allen	25	Texas	15000
3	Teddy	23	Norway	20000
4	Mark	25	Rich-Mond	65000
5	David	27	Texas	85000
6	Kim	22	South-Hall	45000
7	James	24	Houston	10000

```
-- criando a view que seleciona de COMPANY somente as colunas: id, name e age
CREATE VIEW COMPANY_VIEW AS SELECT ID, NAME, AGE FROM COMPANY;
```

```
-- consultando a view
SELECT * FROM COMPANY_VIEW;
```

id	name	age
1	Paul	32
2	Allen	25
3	Teddy	23

```
4 | Mark | 25
5 | David | 27
6 | Kim | 22
7 | James | 24
(7 rows)
```

```
-- removendo a view
DROP VIEW COMPANY_VIEW;
```

Links:

- [DevMedia](#)
- [Guru99](#)
- [Tutorials Point](#)
- [Documentação Oficial - PostgreSQL](#)

PSQL

```
psql -U postgres
```



	<code>\?</code>	listar todos os comandos do psql
	<code>\l</code>	listar todos os bancos de dados
	<code>\c BANCO</code>	conectar no banco de dados BANCO
	<code>\i ARQUIVO</code>	executar comandos SQL do arquivo ARQUIVO no banco de dados
atual	<code>\d</code>	listar todas as tabelas do banco de dados atual
	<code>\d TABELA</code>	mostrar a estrutura da tabela TABELA do banco de dados atual
	<code>\du</code>	lista usuários
	<code>\du+</code>	lista usuários e seus privilégios
	<code>\h</code>	listar todos os comandos SQL
	<code>\h COMANDO</code>	mostrar a sintaxe do comando SQL COMANDO
	<code>\df</code>	listar todas os stored procedure de um B.D
	<code>\q</code>	sair do psql

Modelo Relacional (Modelagem Lógica)

- Entidades Forte, Fraca e Associativa tornam-se, com grande frequência, tabelas.
- Atributos identificadores tornam-se chaves primárias.
- Relacionamentos $1:n$ exigem a criação de uma coluna adicional na tabela referente ao n do relacionamento, denominada de chave estrangeira.

- Relacionamentos com atributos, geralmente, fazem com que estes relacionamentos sejam mapeados como tabelas.
- Relacionamentos $n:m$ (muitos para muitos) devem ser quebrados em 2 relacionamentos $1:n$ e exigem a criação de uma tabela intermediária
- Atributos multivalores tornam-se tabelas.
- Atributos compostos podem se transformar em 1) colunas (o que a literatura diz) ou em uma 2) nova tabela + um relacionamento $1:n$ com a tabela resultante da entidade que, anteriormente, tinha o atributo composto (solução prática que permite mais uma instância do atributo composto).
- Herança/Especialização/Generalização podem gerar (1) uma única tabela, (2) uma tabela para cada entidade filha ou (3) uma tabela para cada entidade.

ER (Modelagem Conceitual)

1. Evite loops
2. Cuidado com o sentido da cardinalidade (para não inverter). Favor dar uma olhada nos vídeos que fiz;
3. Para "fugir" de entidades fracas, crie sempre um atributo identificador forte para as entidades.
4. Para "fugir" de entidades associativas, crie sempre uma outra entidade forte.
5. Evite ao máximo relacionamentos ternários: são permitidos, mas de forma geral, podem ser quebrados em 2 relacionamentos binários;
6. Um atributo pode ser multivalorado ou composto (mas NÃO as duas coisas ao mesmo tempo)
 - Ex: Telefone (quero cadastrar mais de um, mas ao mesmo tempo quero quebrar cada telefone em dois sub-atributos - 1) ddd e o 2) número)
 - Se quisesse que telefone tivesse estas duas características ao mesmo tempo, teria que criar uma nova entidade (Telefone) e criar um relacionamento $1:n$ com a entidade desejada (Ex: Uma instância da entidade Pessoa pode ter 1 ou n Telefones. Sendo que a Entidade Telefone tem código (atributo identificador), ddd e número como atributos)
 - É possível criar a entidade Telefone como entidade fraca em relação a entidade Pessoa mas entraria em conflito com a dica de evitar o uso de entidades fracas (Dica 3) heheheheh kkkkk
7. Se um relacionamento tem atributo, recomendo criar uma entidade forte
8. Nomes de Entidades são substantivos (de preferência no singular).
9. Nomes de Relacionamentos são verbos
10. Evite utilizar Herança

PostgreSQL

Entrando pelo Terminal

```
# 1
psql -U postgres

# 2
export PGPASSWORD='postgres'; psql -h 'localhost' -U 'postgres'

# 3
psql -h 'localhost' -U 'postgres'
```



Dump pelo Terminal (Linux)

```
PGPASSWORD=<SENHA> pg_dump --host <HOST> --port <PORT> --username <USERNAME> --for
```



Herança entre Tabelas

```
DROP DATABASE IF EXISTS heranca;

CREATE DATABASE heranca;

\c heranca;

create table funcionario (
    matricula int,
    nome varchar,
    data_nascimento date,
    primary key(matricula)
);

create table gerente (
    percentParticipacaLucro int,
    telCel varchar
) inherits (funcionario);

-- Os dados são inseridos somente na tabela funcionários e não na tabela gerente.
insert into funcionario values (2000, 'Maria', '02/02/1980');

-- Ao inserir um gerente, automaticamente os atributos herdados (matricula, nome, dataN
insert into gerente values (1000, 'Hesley', '01/01/1975', 10, '99999999');

-- select * from gerente;
-- 1000;"Hesley";"1975-01-01";10;"99999999"

-- select * from funcionario;
-- 2000;"Maria";"1980-02-02"
-- 1000;"Hesley";"1975-01-01"

-- select * from only funcionario;
```



```
-- Retorna somente os funcionários que não são gerentes.  
-- 2000;"Maria";"1980-02-02"  
  
-- Como no SELECT, os comando UPDATE e DELETE também suportam o uso do "ONLY".
```

<https://www.devmedia.com.br/heranca-no-postgresql/9182>

Stored Procedure

Procedimento armazenado ou Stored Procedure é uma coleção de comandos em SQL, que podem ser executadas em um Banco de dados de uma só vez, como em uma função

```
CREATE FUNCTION soma(text, text) RETURNS char AS  
$$  
DECLARE  
    resultado text;  
BEGIN  
    resultado := $1 || $2;  
    return resultado;  
END;  
$$ LANGUAGE 'plpgsql';
```



Trigger

```
-- criando uma função/stored procedure que retorna uma trigger  
CREATE OR REPLACE FUNCTION processa_empregados_audit() RETURNS TRIGGER AS  
$$  
BEGIN  
    IF (TG_OP = 'DELETE') THEN  
        INSERT INTO empregados_audit (operacao, usuario, data, nome)  
        VALUES ('E', USER, NOW(), OLD.nome);  
        RETURN OLD;  
    ELSIF (TG_OP = 'UPDATE') THEN  
        INSERT INTO empregados_audit (operacao, usuario, data, nome)  
        VALUES ('A', USER, NOW(), NEW.nome);  
        RETURN NEW;  
    ELSIF (TG_OP = 'INSERT') THEN  
        INSERT INTO empregados_audit (operacao, usuario, data, nome)  
        VALUES ('I', USER, NOW(), NEW.nome);  
        RETURN NEW;  
    END IF;  
    RETURN NULL;  
END;  
$$ LANGUAGE 'plpgsql';
```



-- definindo comportamento de gatilho

CREATE TRIGGER empregados_audit AFTER INSERT OR UPDATE OR DELETE ON empregados FOR EACH

DCL

Criar um Usuário

1) CREATE ROLE name;

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]
    SUPERUSER | NOSUPERUSER
  | CREATEDB | NOCREATEDB
  | CREATEROLE | NOCREATEROLE
  | INHERIT | NOINHERIT
  | LOGIN | NOLOGIN
  | REPLICATION | NOREPLICATION
  | BYPASSRLS | NOBYPASSRLS
  | CONNECTION LIMIT connlimit
  | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
  | VALID UNTIL 'timestamp'
  | IN ROLE role_name [, ...]
  | IN GROUP role_name [, ...]
  | ROLE role_name [, ...]
  | ADMIN role_name [, ...]
  | USER role_name [, ...]
  | SYSID uid
```



2) CREATE USER name [[WITH] option [...]]

```
    SUPERUSER | NOSUPERUSER
  | CREATEDB | NOCREATEDB
  | CREATEROLE | NOCREATEROLE
  | INHERIT | NOINHERIT
  | LOGIN | NOLOGIN
  | REPLICATION | NOREPLICATION
  | BYPASSRLS | NOBYPASSRLS
  | CONNECTION LIMIT connlimit
  | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
  | VALID UNTIL 'timestamp'
  | IN ROLE role_name [, ...]
  | IN GROUP role_name [, ...]
  | ROLE role_name [, ...]
  | ADMIN role_name [, ...]
  | USER role_name [, ...]
  | SYSID uid
```

Remover um Usuário

1) DROP ROLE name;



2) DROP USER name;

Renomear um Usuário

```
ALTER USER <nome_antigo_usuario>  
  RENAME TO <nome_novo_usuario>;
```



Listar Usuários

```
SELECT rolname FROM pg_roles;
```



Usuário com Login

```
1) CREATE ROLE name LOGIN;  
2) CREATE USER name;
```



Usuário Superuser

```
1) CREATE ROLE name SUPERUSER  
2) CREATE USER name SUPERUSER
```



Usuário com Capacidade de Criar BD's

```
CREATE ROLE name CREATEDB
```



Usuário que Pode Criar/Alterar outros Usuários/papéis

```
CREATE ROLE name CREATEROLE
```



Usuário com Senha

```
CREATE ROLE name PASSWORD 'string'.
```



Logando com um Usuário

```
psql -h localhost -U <usuario> <banco>;
```



Alterar Privilégios

```
ALTER ROLE  
-- ex:  
ALTER ROLE <usuario> SET <privilegio> TO OFF | ON;
```



Remover Algum Privilégio Específico

```
use ALTER ROLE rolename RESET varname.
```



Atribuindo um Privilégio Para um Usuário

```
GRANT <privilegio> ON <object> TO <username>;  
-- ex. o usuário techonthenet pode fazer select, insert, update e delete na tabela prod  
GRANT SELECT, INSERT, UPDATE, DELETE ON products TO techonthenet;  
  
-- ex: dando todos os privilégios para usuário teste2 na tabela pessoa  
GRANT ALL ON pessoa_id_seq TO teste2;  
GRANT ALL ON pessoa TO teste2;  
  
-- ex. o usuário teste3 pode fazer apenas consultas (SELECT) na tabela pessoa  
GRANT SELECT ON pessoa TO teste3;
```



Revogando um privilégio de um determinado usuário

```
REVOKE <privileges> ON <object> FROM <user>;  
-- ex o usuário techonthenet não pode fazer nada na tabela products:  
REVOKE ALL ON products FROM techonthenet;  
-- ex: o usuário techonthenet não pode fazer delete e update na tabela products  
REVOKE DELETE, UPDATE ON products FROM techonthenet;
```



Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
TRUNCATE	Ability to perform TRUNCATE statements on the table.
REFERENCES	Ability to create foreign keys (requires privileges on both parent and child tables).
TRIGGER	Ability to create triggers on the table.
CREATE	Ability to perform CREATE TABLE statements.
ALL	Grants all permissions.

Permitindo que um Usuário faça Tudo em um Esquema

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA <schema_name> TO <username>;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA <schema_name> TO <username>;
```



Revogando que um Usuário Faça Tudo em um Esquema

```
REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA <schema_name> TO <username>;
REVOKE ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA <schema_name> TO <username>;
```



▼ Pages 8

Find a page...

▶ Home

▶ Bibliografia

▼ Dicas

- SQL
 - Operadores
 - Funções Numéricas
 - Funções para Strings
 - Principais Tipos de Dados - PostgreSQL
 - Conversão de Tipos (cast)
 - CREATE DATABASE
 - CREATE TABLE
 - CHECK
 - PRIMARY KEY

FOREIGN KEY

FOREIGN KEY ON DELETE CASCADE

UNIQUE

ALTER TABLE

DROP TABLE/DROP DATABASE

INSERT

INSERT Múltiplo

UPDATE

SELECT

Apelidar colunas retornadas de um SELECT

ORDER BY

Concatenação (Operador ||)

LIKE/ILIKE

IN/NOT IN

EXISTS

DELETE

Transações (BEGIN/COMMIT/ROLLBACK)

Data/Hora

CASE/WHEN/ELSE/END

LIMIT/OFFSET

DISTINCT

COUNT

GROUP BY

HAVING

INNER JOIN/CROSS JOIN/FULL JOIN/RIGTH JOIN/LEFT JOIN

UNION

INTERSECT

EXCEPT

Blob's - Bytea e OID (Arquivos)

- bytea

- oid (Arquivos Grandes)

SCHEMA

VIEWS

PSQL

Modelo Relacional (Modelagem Lógica)

ER (Modelagem Conceitual)

PostgreSQL

- Entrando pelo Terminal

- Dump pelo Terminal (Linux)

- Herança entre Tabelas

Stored Procedure

Trigger

DCL

Criar um Usuário
Remover um Usuário
Renomear um Usuário
Listar Usuários
Usuário com Login
Usuário Superuser
Usuário com Capacidade de Criar BD's
Usuário que Pode Criar/Alterar outros Usuários/papéis
Usuário com Senha
Logando com um Usuário
Alterar Privilégios
Remover Algum Privilégio Específico
Atribuindo um Privilégio Para um Usuário
Revogando um privilégio de um determinado usuário
Permitindo que um Usuário faça Tudo em um Esquema
Revogando que um Usuário Faça Tudo em um Esquema

▸ **Ementa**

▸ **Listas**

▸ **Programa**

▸ **Setup**

▸ **Trabalhos**

Clone this wiki locally

https://github.com/IgorAvilaPereira/iobd2023_2sem.wiki.git

