3) Criação e Manipulação de Tabelas

3.1) Visualizando a estrutura de tabelas criadas

Para visualizar a estrutura de tabelas criadas podemos usar:

- Metacomando \d nometabela no psql ou então
- Selecionar a tabela no PGAdmin

3.2) Entendendo as colunas de sistema

Vejamos alguns exemplos de colunas das tabelas de sistema.

Acesse o psql em qualquer banco e execute:

\dS

\d pg_database

Vejamos algumas das colunas (campos):

```
dml=# \d pg_database
   Table "pg_catalog.pg_database"
   Column     |   Type    | Modifiers
---------------+-----------+-----------
 datname       | name      | not null
 datdba        | oid       | not null
 encoding      | integer   | not null
 datistemplate | boolean   | not null
 datallowconn  | boolean   | not null
 datconnlimit  | integer   | not null
```

datame – representa o nome do banco

encoding – a codificação do banco

datistemplate – se o banco é ou não um template

3.3) Sintaxe de criação de tabelas

Nada melhor que o psql para nos informar toda a sintaxe completa da criação de tabelas:

```
dml=# \h create table
Command:     CREATE TABLE
Description: define a new table
Syntax:
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name ( [
  { column_name data_type [ DEFAULT default_expr ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE parent_table [ { INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | I
NDEXES } ] ... }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  CHECK ( expression ) |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIM
PLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ...
] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDA
TE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]

index_parameters in UNIQUE and PRIMARY KEY constraints are:
```

```
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace ]
```

Alguns exemplos de criação de tabelas

```
create database dba_projeto;
\c dba_projeto

create table clientes (
    cpf varchar(11) primary key,
    nome varchar(45) not null,
    telefone varchar(11),
    email varchar(45),
    data_nasc date not null
);
create table produtos (
    codigo serial primary key,
    descricao varchar(40) not null,
    quantidade int2 not null,
    data_compra date not null
);


create table pedidos (
    codigo serial primary key,
    cpf_cliente varchar(11),
    codigo_produto int4,
    quantidade int2,
    data date,
    valor decimal(12,2)
);
```

Agora criando a tabela pedidos com algumas foreign keys:

```
create table pedidos (
    codigo serial primary key,
    cpf_cliente varchar(11),
    codigo_produto int4,
    quantidade int2,
    data date,
    valor decimal(12,2),
CONSTRAINT pedidos_cli_fk FOREIGN KEY (cpf_cliente) REFERENCES clientes (cpf),
CONSTRAINT pedidos_prod_fk FOREIGN KEY (codigo_produto) REFERENCES produtos
(codigo)
);
```

3.4) Entendendo o comando Alter Table

```
dml=# \h alter table
Command:     ALTER TABLE
Description: change the definition of a table
Syntax:
ALTER TABLE [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column TO new_column
ALTER TABLE name
    RENAME TO new_name
ALTER TABLE name
    SET SCHEMA new_schema

where action is one of:

    ADD [ COLUMN ] column type [ column_constraint [ ... ] ]
    DROP [ COLUMN ] column [ RESTRICT | CASCADE ]
    ALTER [ COLUMN ] column TYPE type [ USING expression ]
    ALTER [ COLUMN ] column SET DEFAULT expression
    ALTER [ COLUMN ] column DROP DEFAULT
    ALTER [ COLUMN ] column { SET | DROP } NOT NULL
    ALTER [ COLUMN ] column SET STATISTICS integer
    ALTER [ COLUMN ] column SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
    ADD table_constraint
    DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
    DISABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE REPLICA TRIGGER trigger_name
    ENABLE ALWAYS TRIGGER trigger_name
    DISABLE RULE rewrite_rule_name
    ENABLE RULE rewrite_rule_name
    ENABLE REPLICA RULE rewrite_rule_name
    ENABLE ALWAYS RULE rewrite_rule_name
    CLUSTER ON index_name
    SET WITHOUT CLUSTER
    SET WITHOUT OIDS
    SET ( storage_parameter = value [, ... ] )
    RESET ( storage_parameter [, ... ] )
    INHERIT parent_table
    NO INHERIT parent_table
    OWNER TO new_owner
    SET TABLESPACE new_tablespace
```

3.5) Alterando tabelas e colunas

Adicionar campo, remover campo, adicionar constraint, remover constraint, alterar valor default, renomear campo, renomear tabela, alterar tipo de dado de campo (>=8.0).

Adicionar Um Campo
ALTER TABLE tabela ADD COLUMN campo tipo;
ALTER TABLE produtos ADD COLUMN descricao text;

Remover Campo
ALTER TABLE tabela DROP COLUMN campo;
ALTER TABLE produtos DROP COLUMN descricao;
ALTER TABLE produtos DROP COLUMN descricao CASCADE; -- Cuidado com CASCADE

Adicionar Constraint
ALTER TABLE tabela ADD CONSTRAINT nome;
ALTER TABLE produtos ADD COLUMN descricao text CHECK (descricao <> '');
ALTER TABLE produtos ADD CHECK (nome <> '');
ALTER TABLE produtos ADD CONSTRAINT unique_cod_prod UNIQUE (cod_prod);
ALTER TABLE produtos ADD FOREIGN KEY (cod_produtos) REFERENCES grupo_produtos;
ALTER TABLE produtos ADD CONSTRAINT vendas_fk FOREIGN KEY (cod_produtos)
REFERENCES produtos (codigo);

Remover Constraint
ALTER TABLE tabela DROP CONSTRAINT nome;
ALTER TABLE produtos DROP CONSTRAINT produtos_pk;
ALTERAR VALOR DEFAULT DE CAMPO:

Mudar Tipo de Dados de Campo (Só >=8.0):
ALTER TABLE tabela ALTER COLUMN campo TYPE tipo;
ALTER TABLE produtos ALTER COLUMN preco TYPE numeric(10,2);
ALTER TABLE produtos ALTER COLUMN data TYPE DATE USING CAST (data AS DATE);

Mudar Nome De Campo
ALTER TABLE tabela RENAME COLUMN campo_atual TO campo_novo;
ALTER TABLE produtos RENAME COLUMN cod_prod TO cod_produto;

Setar/Remover Valor Default de Campo
ALTER TABLE tabela ALTER COLUMN campo SET DEFAULT valor;
ALTER TABLE produtos ALTER COLUMN cod_prod SET DEFAULT 0;
ALTER TABLE produtos ALTER COLUMN preco SET DEFAULT 7.77;
ALTER TABLE tabela ALTER COLUMN campo DROP DEFAULT;
ALTER TABLE produtos ALTER COLUMN preco DROP DEFAULT;

Adicionar/Remover NOT NULL
ALTER TABLE produtos ALTER COLUMN cod_prod SET NOT NULL;
ALTER TABLE produtos ALTER COLUMN cod_prod DROP NOT NULL;

Renomear Tabela
ALTER TABLE tabela RENAME TO nomenovo;
ALTER TABLE produtos RENAME TO equipamentos;
Adicionar Constraint (Restrição)
ALTER TABLE produtos ADD CONSTRAINT produtos_pk PRIMARY KEY (codigo);
ALTER TABLE vendas ADD CONSTRAINT vendas_fk FOREIGN KEY (codigo) REFERENCES produtos(codigo_produto);
ALTER TABLE vendas ADD CONSTRAINT vendas_fk FOREIGN KEY (codigo) REFERENCES produtos; -- Neste caso usa a chave primária da tabela produtos

Remover Constraint (Restrição)
ALTER TABLE produtos DROP CONSTRAINT produtos_pk;
ALTER TABLE vendas DROP CONSTRAINT vendas_fk;

3.6) Comentários em objetos

Para comentar objetos no PostgreSQL usamos o comando COMMENT, que não é compatível com o SQL, mas pertente ao PostgreSQL.

```
dml=# \h comment
Command:     COMMENT
Description: define or change the comment of an object
Syntax:
COMMENT ON
{
  TABLE object_name |
  COLUMN table_name.column_name |
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (sourcetype AS targettype) |
  CONSTRAINT constraint_name ON table_name |
```

```
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  FUNCTION func_name ( [ [ argmode ] [ argname ] argtype [, ...] ] ) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR op (leftoperand_type, rightoperand_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TRIGGER trigger_name ON table_name |
  TYPE object_name |
  VIEW object_name
} IS 'text'
```

Resumindo:
COMMENT ON TIPO nomeobjeto IS 'Comentario';

Exemplo:

COMMENT ON TABLE clientes IS 'Comentario da Tabela Clientes';
No PGAdmin, ao selecionar a tabela vemos o comentário sobre a mesma.

3.7) Eliminando tabelas

Para eliminar tabelas usamos o comando SQL '**drop**'. Vejamos sua sintaxe:

```
dml=# \h drop table
Command:     DROP TABLE
Description: remove a table
Syntax:
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Bem simples e sem muitas opções.

IF EXISTS – Não dispara erro, caso a tabela não exista. Útil para scripts.

CASCADE – força a exclusão, excluíndo automaticamente os objetos que dependem desta tabela, como views.

RESTRICT – Opção padrão, que se recusa a remover a tabela caso exista algum objeto dependente da tabela.

Exemplos:

```
DROP TABLE clientes;

DROP TABLE IF EXISTS clientes;
DROP TABLE clientes CASCADE;
```