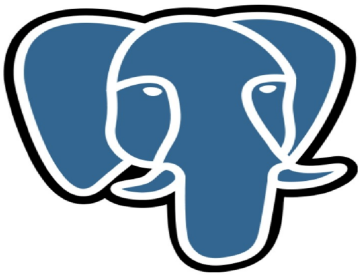


POLIBRÁS NET

PostgreSQL Essentials

Objetivos do Curso

* Introdução ao PostgreSQL
  - O que é o PostgreSQL
  - Principais funcionalidades
  - Plataformas suportadas
  - Limitações do PostgreSQL
  - Instalação Linux/Windows
* Conceito de banco de dados
  - Banco de dados relacionais
  - o Banco de dados objeto-relacional
* Interfaces de acesso ao PostgreSQL
  - Conexão JDBC
  - Conexão Delphi/ Visual Borland C++
  - Conexão .NET
  - Introdução ao psql
  - Operação do psql
  - pgAdmin
* SQL
  - A linguagem SQL
  - Palavras-chave
  - Identificadores
  - Valores nulos
  - Comentários
  - Tipos de dados
* Gerenciamento de tabelas
  - Introdução
  - Visualizando a estrutura de tabelas
  - Colunas de sistema
  - Sintaxe de criação de tabelas
  - Comando ALTER TABLE
  - Alteração de tabelas e colunas
  - Comentários em objetos
  - Eliminação de tabelas

Notação Utilizada no Curso

Itálico **: Utilizamos para nomes de objetos de banco de dados, arquivos e outros elementos que devem ser fornecidos pelo usuário.**

`Courier New` **: Os comandos, palavras-chave e saídas de comandos são grafados no tipo Courier.**

`Courier New` **: Utilizamos Courier New em negrito para enfatizar uma palavra-chave apresentada pela primeira vez e valores padrão.**

[ ] : Utilizamos colchetes sempre que uma palavra-chave for opcional.

{ } : Utilizamos chaves para delimitar uma lista de itens onde um deles deva ser escolhido.

... : Utilizamos três pontos para indicar que aquele item pode se repetir. Quando utilizado nos exemplos, indica uma parte não importante da informação foi removida.

Capítulo 1

Introdução ao PostgreSQL

O que é o PostgreSQL ?

O PostgreSQL é um dos bancos de dados abertos mais utilizados atualmente, possui recursos avançados e compete igualmente com muitos bancos de dados comerciais.

O banco de dados PostgreSQL nasceu na Universidade de Berkeley, em 1986, como um projeto acadêmico e se encontra hoje na versão 9.0, sendo um projeto mantido pela comunidade de Software Livre.

A coordenação de desenvolvimento do PostgreSQL é executada pelo *PostgreSQL Global Development Group* que conta com um grande número de desenvolvedores ao redor do mundo.

Principais Funcionalidades

- **Banco de dados objeto-relacional**
- **Herança entre tabelas**
- **Sobrecarga de funções**
- **Colunas do tipo array**
- **Suporte a transações (padrão ACID - Acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade) - http://pt.wikipedia.org/wiki/ACID**
- **Lock por registro (row level locking)**
- **Integridade referencial**
- **Extensão para GIS (dados georreferenciados)**
- **Acesso via drivers ODBC e JDBC, além do suporte nativo em várias linguagens**
- **Interface gráfica de gerenciamento**
- **Uso otimizado de recursos do sistema operacional**

Principais Funcionalidades

- **Suporte aos padrões ANSI SQL 92 e 99**
- **Triggers, views e functions (PL/pgSQL, Perl, Python e Tcl)**
- **Mecanismo de rules**
- **Suporte ao armazenamento de BLOBs (binary large objects)**
- **Sub-queries e queries definidas na cláusula FROM**
- **Backup online**
- **Sofisticado mecanismo de tuning**
- **Suporte a conexões de banco de dados seguras (criptografia)**
- **Modelo de segurança para o acesso aos objetos do banco de dados por roles**
- **Replicação de banco de dados (Warm Standby, Slony, PgPool)**
- **Joins: Implementa todos os tipos de join definidos pelo padrão SQL99: *inner join, left, right, full outer join, natural join*.**
- **Recurso de tablespace**
- **Backup físico offline (*Point in Time Recovery*)**
- **Transações com *Savepoint* e *Two-Phase Commit***
- **Parâmetros IN/OUT em funções**
- **Uso de roles**
- **Mecanismo próprio de logs**
- **Particionamento de tabelas**
- ***Autovacuum* integrado**
- **Ordenação por nulos**
- **HOT (*Heap Only Tuples*)**

Plataformas Suportadas

- **IBM AIX**
- **FreeBSD, OpenBSD, NetBSD**
- **HP-UX**
- **Irix**
- **Linux**
- **MacOS X**
- **Microsoft Windows (suporte nativo desde a versão 8.0)**
- **SCO Open Server**
- **Sun Solaris**
- **Tru64 Unix**

Limitações do PostgreSQL

Tamanho máximo de um banco	**ilimitado**
Tamanho máximo de uma tabela	**32 TB**
Tamanho máximo de um registro	**1.6 TB**
Tamanho máximo de um campo	**1 GB**
Máximo de linhas numa tabela	**ilimitado**
Máximo de colunas numa tabela	**250 à 1600**
Máximo de índices numa tabela	**ilimitado**

Capítulo 2

Entendendo um Banco de Dados

Bancos de Dados Relacionais

- **O modelo relacional surgiu devido às seguintes necessidades:**

  Aumentar a independência de dados nos sistema gerenciadores de banco de dados;

  Prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados;

  O modelo foi apresentado num artigo publicado em 1970, mas só nos anos 80 foi implementado.

- **A estrutura fundamental do modelo relacional é a relação. Uma relação é constituída por um ou mais atributos (campos), que traduzem o tipo de dados a armazenar. Cada instância do esquema (linha), designa-se por tupla (registro)**

  – **O modelo relacional implementa estruturas de dados organizadas em relações (tabelas).**

Banco de Dados Objeto-Relacional

– **O PostgreSQL é normalmente considerado um sistema gerenciador de banco de dados relacional (SGBD-R, ou RDBMS, em inglês). Entretanto, o PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional (SGBD-OR).**

– **Por ser objeto-relacional, o PostgreSQL suporta recursos inexistentes a um banco de dados puramente relacional, tais como: herança entre tabelas, arrays em colunas e sobrecarga de funções.**

Observe que "objeto-relacional" não é sinônimo de "orientado a objetos", um termo cabível a muitas linguagens de programação modernas.

Capítulo 3

Interfaces de Acesso ao PostgreSQL

Interfaces de Acesso ao Banco de Dados

O PostgreSQL pode ser acessado a partir de várias linguagens, entre elas estão:

C/C++
Java (JDBC)
PHP, JSP, ColdFusion
TCL/Tk
Perl
Python
ODBC (ASP, Delphi ou qualquer linguagem que suporte ODBC)

Em todo caso, sempre é possível utilizar a libpq para escrever o seu próprio driver, caso a linguagem utilizada ainda não tenha suporte ao PostgreSQL.

Conexão JDBC

- **Abaixo um exemplo de conexão utilizando driver JDBC:**

```
public  Connection  connect(){
    driver  =  "org.postgresql.Driver";
    url = "jdbc:postgresql://172.16.129.13:5432/teste?user=postgres"; try{
        Class.forName(driver).newInstance(); con =
        DriverManager.getConnection(url);
    }
    catch (Exception e){ System.out.println("Erro:");
        e.printStackTrace();
    }
    return  con;
}
```

Para o url de conexão temos as opções:

User: Usuário para conexão.

Password: Senha para a conexão.

Database: Nome do banco de dados a ser acessado.

Port: Porta de conexão.

IP: Endereço IP do servidor.

Conexão com Delphi / Borland C++

- Existem várias formas de se conectar ao PostgreSQL, onde existem fornecedores de drivers de acesso direto, preferível ao acesso via ODBC.

- ZeosLib (livre): *http://sourceforge.net/projects/zeoslib/*

- Vita Voom: *http://www.vitavoom.com/*

Conexão com .NET

- Também existem fornecedores de drivers de acesso direto para .NET, preferíveis ao acesso ODBC.

- Npgsql (livre): http://npgsql.projects.postgresql.org/

- *CoreLab:* http://crlab.com/pgsqlnet/

Introdução ao psql

- O psql é o modo interativo do PostgreSQL para acesso e manipulação dos bancos de dados:

```
psql [ -h hostname -p port -U user -W ] [ database ]
```

Onde:

hostname: Nome ou IP do servidor (padrão é localhost) port: porta de conexão (padrão é 5432)

User: Usuário postgres (padrão é o usuário de sistema operacional)

database: Nome do banco de dados.

- A opção – W força a entrada da senha do usuário.

Comandos Básicos do sql

Dentro do psql, a lista de todos os comandos pode ser acessada com \?, alguns dos mais utilizados:

```
\h  <comando>     exibe  a  ajuda  sobre  o  comando
\c  [banco]           conecta  num  banco  de  dados  específico
\d                    lista  de  objetos  de  banco
\d  <tabela>          descrição  da  tabela  específica
\d{t|i|s|v}           lista  de  tabelas,  índices,  sequências  e  views
\d{p|S|l}             lista  de  permissões,  tabelas  de  sistemas  e  BLOBs
\da                   lista  de  funções  agregadas
\dd  <obj>            lista  os  comentários  sobre  um  objeto
\df                   lista  de  funções
\do                   lista  de  operadores
\dT                   lista  de  tipos  de  dados
\e                    edita  a  query  atual  num  editor  de  texto
\g                    executa  a  query  do  buffer
\r              apaga  os  comandos  do  buffer
\i  <arq>       lê  e  executa  comandos  a  partir  de  um  arquivo
\l              lista  de  todos  os  bancos  de  dados
\o  <file>     envia  os  resultados  para  um  arquivo
\q              sai  do  psql
\s  <file>     salva  os  'comandos'  num  arquivo
\t              mostra  somente  as  tuplas
\H              utiliza  tags  HTML  para  exibir  resultados
\w              salva  o  comando  atual  num  arquivo
\x              exibe  os  registros  de  forma  expandida  (padrão:  off)
\timing         mostra  o  tempo  de  execução  dos  comandos
\!  [cmd]       executa  um  comando  (cmd)  no  sistema  operacional
```

- **Para conectar a outro banco de dados:**
  `\c[onnect]  [  DBNAME  [  USER  ]  ]`
- **Para editar o último comando ou um arquivo:**
  `\e  [nome-do-arquivo.sql]`
- **Para salvar todos os comandos do history num arquivo:**
  `\s  nome-do-arquivo.sql`
- **Para executar um arquivo de script:**
  `\i  nome-do-arquivo.sql`
- **Para gravar a saída num arquivo:**
  `\o  nome-do-arquivo.sql`
- **Para exibir o tempo de execução dos comandos:**
  `\timing`

Executando Comandos no sql

- Existem duas maneiras possíveis de executar comandos no psql, a maneira interativa e a execução de comandos existentes em arquivos (`\i`).

- Na maneira interativa, basta digitar os comandos no prompt do psql. O prompt tem o seguinte formato:

  `banco= #` O sinal **#** indica que se trata de um superusuário

  `banco= >` *Prompt* para usuários regulares

- Um comando SQL pode ser divido em múltiplas linhas e deve ser finalizado com um ";" (ou '`\g`"). Uma vez que uma nova linha é iniciada o *prompt* muda para o formato abaixo:

```
banco->
```

- O *prompt* também informa se existe algum parênteses ou aspas em aberto.

```
banco-> CREATE TABLE emp (
banco(>
```

- O comando **`\e`** abre o editor de texto padrão *vi* para editar um comando.

- Entretanto, é possível alterar o editor para qualquer outro existente no sistema operacional. Para tanto, é necessário executar os seguintes passos no Linux (*bash*):

```
export EDITOR=/usr/bin/nano
```

- A alteração do editor será válida somente durante a sessão. Para tornar esta modificação definitiva, deve-se inserir este comando no **`.bashrc`**.

Outras Opções do psql

- É possível alterar o comportamento padrão do psql através de opções de chamada:

```
-l        lista todos os bancos e sai

-f file   executa as queries definidas no arquivo

-o file   manda as saídas para o arquivo definido

-s        uma confirmação é pedida antes de cada query

-S        considera cada linha um comando sem a

          necessidade de ponto-e-vírgula

-c        executa um comando sql
```

- *Exemplo:*

```
$ psql -l
        List of databases
   Name    |  Owner   | Encoding
-----------+----------+----------
 curso     | postgres | LATIN1
 postgres  | postgres | LATIN1
```

- A combinação do comando psql com a opção `-f` (arquivo com script) torna possível automatizar rotinas através de shell scripts.

- O status de saída do comando psql pode ser verificado visualizando o conteúdo da variável shell `$?`.

-> O `psql` retorna os seguintes valores para o sistema operacional:

- 0: Se tudo correu normalmente
- 1: No caso de erro no psql (parâmetros incorretos)
- 2: Se houve falha na comunicação com o servidor
- 3: Se ocorreu algum erro nos comandos SQL (a variável `ON_ERROR_STOP` deve estar `ON`)

Variáveis do psql

- O **psql** possui variáveis de sistema e variáveis que podem ser definidas pelo usuário. A utilização destas variáveis no **psql** é semelhante à da shell no Linux.

- **Usando variáveis:**

```
banco=>  \set
banco=>  \set  var_tabela  cidades
banco=>  \echo  :var_tabela
cidades
banco=>  SELECT  *  FROM  :var_tabela;
banco=>  \unset  var_tabela
```

- **Variáveis mais importantes do psql:**

```
DBNAME           nome do banco corrente
ENCODING         mostra a codificação utilizada
HISTSIZE         número de comandos guardados no
histórico
HOST mostra o servidor atual
LASTOID          último OID afetado por uma query
ON_ERROR_STOP    quando ON, pára a execução de scripts
PORT       porta de conexão ao servidor
USER       usuário conectado ao banco
SINGLELINE considera cada linha um comando
SINGLESTEP  pede confirmação antes de cada comando
ECHO_HIDDEN      mostra o comando gerado pelos
metacomandos (\d, \df, ...)
```

- É possível gravar em um arquivo a configuração das variáveis. Assim, não é necessário repetir a configuração em cada execução do **psql.**

- As configurações devem ser armazenadas no arquivo **$HOME/.psqlrc**
- A configuração deve ser feita através do comando **\set**

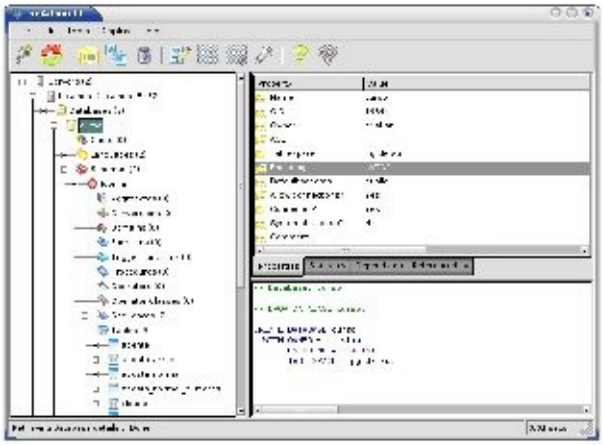
- Exemplo do arquivo .psqlrc**:**

```
\set  ON_ERROR_STOP  on
\set  SINGLESTEP  on
\set  HISTSIZE  1000
```

Interfaces Gráficas

- O PgAdmin é uma interface de modo gráfico para manipulação de bancos de dados. Ele faz o mesmo que o psql, porém, com recursos visuais.

- Instalação via Yum
Yum: yum install pgadmin3



Projetos Relacionados

- PgAdmin
  Site: *http://www.pgadmin.org*
  Gratuito e disponível para Windows

- PgManager
  Site: *http://www.sqlmanager.net/products/postgresql/manager*
  Pago (Windows e Linux), mas com versão Lite gratuita.

- PhpPgAdmin
  Site: *http://phppgadmin.sourceforge.net*
  Ferramenta web gratuita

- Há várias outras ferramentas gratuitas e pagas disponíveis na Internet

Capítulo 4

Linguagem SQL

- SQL (*Structured Query Language*) é uma linguagem declarativa de acesso a banco de dados.

- Por ser uma linguagem padronizada, a migração para o PostgreSQL é facilitada para aqueles que conhecem SQL.

- O PostgreSQL está em conformidade com a maior parte das especificações SQL92 e SQL99.

- A linguagem SQL não considera a caixa dos comandos (*case insensitive*). Entretanto, a caixa faz diferença para os literais entre aspas.

"Podem ocorrer diferenças entre os "dialetos" SQL. Alguns tipos de dados também podem diferir entre um banco de dados e outro."

Palavras-chave e Identificadores

- Palavras-chave são termos SQL que possuem um significado sintático reservado para o servidor PostgreSQL.

  - Todos os comandos SQL são palavras-chave reservadas.

- Aspas duplas normalmente não são necessárias, mas podem ser utilizadas em torno de identificadores. O seu uso indica que o identificador deve ser interpretado literalmente:

```
SELECT * FROM estados;
SELECT * FROM "Estados";
ERROR: Relation 'Estados' does not exist
```

- Este erro ocorre porque o PostgreSQL busca uma tabela literalmente denominada `Estados`.

- Todos os identificadores que não estejam entre aspas duplas são interpretados como se estivessem em minúsculo.

Identificadores

- Nas seguintes situações um identificador deve ser usado com aspas duplas:

  - Quando um objeto de banco de dados tem um nome idêntico a uma palavra-chave.
  - Quando o identificador tem letras maiúsculas, espaços ou acentos.

- Tanto identificadores como palavras-chave devem ter comprimento máximo de 64 caracteres.

  - O analisador sintático trunca os identificadores acima deste limite.

- Identificadores devem começar com letras ou *underscore*, podendo ser seguido de números.

  - É possível utilizar aspas duplas para burlar a restrição de identificadores para que comecem com números:

```
create table "123" (nome text);
```

Comentários

- O PostgreSQL oferece duas maneiras de colocar comentários em blocos SQL.

- Comentários de uma única linha:

```
SELECT 'teste'    -- Este é um comentário
               -- de uma linha
AS exemplo;
```

- Comentários de múltiplas linhas (similar à linguagem C):

```
SELECT 'teste'    /* Este é um comentário
                  com varias linhas
                  /* comentário aninhado */
               */ AS exemplo;
```

"O uso de comentários aninhados facilita comentar grandes porções de código que incluem outros comentários."

Tipos de Dados

- Todo dado representado pelo PostgreSQL possui um tipo de dados associado.

- Tipos numéricos:

  - Smallint (int2) -32.768 até 32.767

  - Integer (int4) -2.147.483.648 até 2.147.483.647

  - Bigint (int8) -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807

  - Real precisão de 6 dígitos

  - Double precision precisão de 15 dígitos

  - Numeric sem limite

  - Decimal sem limite (equivalente ao numeric)

  - Serial 1 até 2.147.483.647

  - Bigserial 1 até 9.223.372.036.854.775.807

- Tipos Caractere
  - Character (n), Char (n)
    - tamanho fixo, preenchido com brancos
  - Character Varying (n), Varchar (n)
    - tamanho variável
  - Text
    - Ilimitado
- Objetos binários
  - Suporte a *blobs* (capítulo adiante destinado a este tópico)
- Tipos lógicos e binários

  - Boolean, Bit, Bit varying (n)

  – Os tipos lógicos podem assumir os valores `TRUE`, `FALSE` ou `UNKNOWN` (este último representado por `NULL`)

Tipos de Dados

- Tipos de data e hora

Date	datas
Time	horas
Timestamp	datas e horas
Interval	intervalos de tempo

- Tipos enumeráveis

- Enum

- Representa uma lista de valores possíveis.

- Tipo de dado Array

- **Tipos geométrico**

- Point 16 bytes (x,y)

- Line 32 bytes ((x1,y1),(x2,y2))

- Lseg 32 bytes ((x1,y1),(x2,y2))

- Box 32 bytes ((x1,y1),(x2,y2))

- Path 16+16n bytes ((x1,y1),...)

- Polygon 40+16n bytes ((x1,y1),...)

  – Circle 24 bytes <(x,y),r>

Referencia a Calculo de uma página de dados:
http://wiki.postgresql.org/wiki/FAQ/pt

Valores Nulos

- O identificador `NULL` é usado em campos que não possuem valor associado.

- O valor `NULL` pode ser utilizado com qualquer tipo de dado.

- Importante:

- O `NULL` é diferente de `''` para tipos texto.

- O `NULL` é diferente de `0` para tipos numéricos.

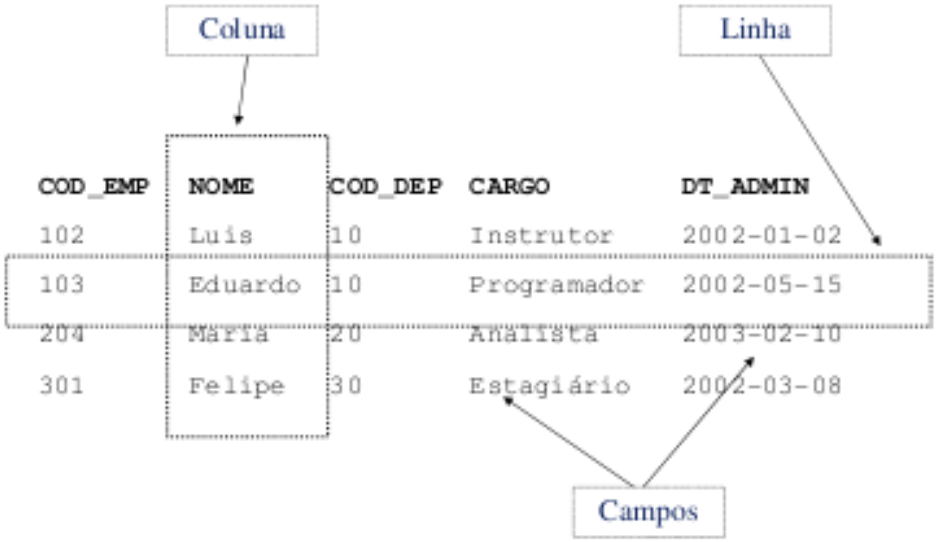
- O valor `NULL` é a ausência de valor para o campo.

- É possível criar restrições para que uma coluna nunca tenha valores nulos.

Capítulo 5

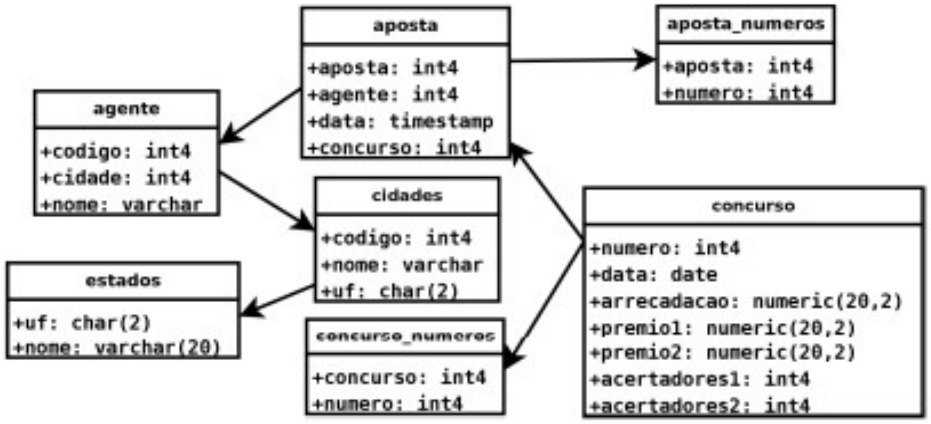
Gerenciando Tabelas

O que é uma Tabela?



Tabelas Utilizadas no Treinamento

- Tabelas utilizadas:



Visualizando a Estrutura de Tabelas

- Uma tabela é composta de linhas e colunas e suas interseções são os campos.

  - De uma maneira geral, uma coluna numa tabela tem um nome e um tipo de dado associado.

  - Linhas numa tabela representam registros compostos por campos.

- Não existe ordem nos registros armazenados numa tabela, a ordenação é definida no momento da seleção.

- Uma tabela não precisa possuir nenhuma coluna e normalmente criada vazia, isto é, sem registros.

- O número máximo de colunas de uma tabela é 1600.

"comando `\d`, sem parâmetros, exibe a lista de objetos do banco de dados."

Colunas de Sistema

- O PostgreSQL define uma série de colunas de sistema em todas as tabelas. Estas colunas, a não ser que sejam explicitamente referenciadas, são invisíveis aos usuários.

- As principais colunas de sistema são o OID e o TABLEOID:

- O OID é o identificador único da linha na tabela. O OID não indica o endereço físico da linha, por isto, a seleção via OID não é eficiente.

- O TABLEOID é um identificador único da tabela no banco de dados.

- A combinação do OID com o TABLEOID identifica unicamente uma linha no sistema.

Ex.

```
SELECT TABLEOID, * FROM t_cliente LIMIT 10 ;
```

IMPORTANTE: tanto o **`OID`** como o **`TABLEOID`** são modificados quando um banco de dados é recriado. A criação de tabelas a partir da versão 8.1, por padrão, não trará os **`OIDs`** das linhas.

Sintaxe de Criação de Tabelas

- De maneira simples, podemos criar tabelas da seguinte forma:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE
   table_name ( [
     {    column_name    data_type    [    DEFAULT    default_expr    ]
      0[column_constraint  [  ...  ]  ]
       0|table_constraint
        |  LIKE  parent_table  [  {  INCLUDING  |  EXCLUDING  }
   {  DEFAULTS  |  CONSTRAINTS  |  INDEXES  }  ]  ...  }
    [,  ...  ]
]  )
  0[INHERITS  (  parent_table  [,  ...  ]  )  ]
  0[WITH  (  storage_parameter  [=  value]  [,  ...  ]  )  |  WITH  OIDS
     0|WITHOUT  OIDS  ]
  0[ON  COMMIT  {  PRESERVE  ROWS  |  DELETE  ROWS  |  DROP  }  ]
  0[TABLESPACE  tablespace  ]
```

Comando ALTER TABLE

- O comando **ALTER TABLE** modifica a estrutura de uma tabela em diferentes formas:

- ADD COLUMN
- DROP COLUMN
- SET/DROP DEFAULT
- RENAME TABLE/COLUMN
- OWNER
- SET/DROP NULL
  - ADD/DROP table_constraint
  - ALTER COLUMN TYPE

- Adicionando uma nova coluna:

ALTER TABLE agente ADD COLUMN media numeric(20,2);

- Renomeando uma coluna existente:

ALTER TABLE agente RENAME COLUMN media TO media_apostas;

- Eliminando uma coluna de uma tabela:

ALTER TABLE agente DROP COLUMN media_apostas;

- Alterando o tipo de dados de uma coluna:

ALTER TABLE aposta_numeros ALTER COLUMN numero TYPE varchar;

- Transformando uma coluna varchar para inteiro:

ALTER TABLE aposta_numeros ALTER COLUMN numero TYPE int USING CAST(numero AS int);

Alterando Tabelas e Colunas

- Adicionando a condição NOT NULL a uma coluna:

ALTER TABLE agente ALTER COLUMN codigo SET NOT NULL;

- Removendo a condição NOT NULL de uma coluna:

ALTER TABLE agente ALTER COLUMN cdigo DROP NOT NULL;

- Adicionando um valor default numérico para uma coluna:

ALTER TABLE concurso ALTER COLUMN premio1 SET DEFAULT 0;

- Adicionando um valor default data para uma coluna:

ALTER TABLE aposta ALTER COLUMN data SET DEFAULT now();

- Renomeando uma tabela existente:

ALTER TABLE estados RENAME TO ufs;

Comentários em Objetos

- É possível criar comentários para objetos que ficarão armazenados no banco de dados.

\h COMMENT

- Comentando tabelas:

curso=# COMMENT ON TABLE agente IS 'Agentes Lotéricos';

- Comentando colunas da tabela:

curso=# COMMENT ON COLUMN agente.nome IS 'Nome do Agente';

Eliminando Tabelas

- O comando SQL para eliminar tabelas é o DROP TABLE. Note que o DROP é utilizado para eliminar vários tipos de objetos (tabelas, sequencias, funções, triggers, usuários, bancos de dados, etc...)

- Sintaxe:

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

“Nenhum comando SQL pede confirmação quando removemos um objeto de banco de dados.“

“Ao eliminar uma tabela perdemos todos as linhas que ela possuía, e não existe uma forma de recuperar este dados. Portanto, utilize com cuidado!”

Boas Práticas:

- Sempre verificar as sintaxes rapidamento com o atalho “\h ou \help”

  \h CREATE TABLE

  \h SELECT

  \h ALTER TABLE

- Ler a documentação: http://www.postgresql.org/docs/current/static/

- O PostgreSQL por padrão é autocommit, todo comando que for executado fora de um “BEGIN” (Abertura de Transação) será commitado automaticamente.

Exercícios:

- Alterar a tabela de endereço de clientes o tipo da coluna s_codcliente para o tipo inteiro.

- Dropar a tabela a_endereco_cliente sem deletar a sequence relacionada a ela

- Dropar a tabela a_telefone_cliente dropando todos os objetos relacionados a ela

- Criar uma tabela que contenha um identificador único de registro, uma descricao, um status e um campo para armazenar um arquivo do tipo imagem e um campo valor que será expressado em R$.

- Criar um arquivo sql com algumas consultas, rodar, editar e copiar esse arquivo a partir do prompt do banco de dados.