cc BY NC ND

PostgreSQL Prático

(versão 8.1.4)

Ribamar FS – ribafs@users.sourceforge.net – http://ribafs.tk

17 de setembro de 2006

ÍNDICE

Capítulo **Página**

1 – Introdução 4

2 - Instalação 8
- 2.1 - No Linux
- 2.2 - No Windows

3 - DDL (Data Definition Language) 13
- 3.1 - Criação e exclusão de bancos, esquemas, tabelas, views, Constraints, etc
- 3.2 - Alterações nos objetos dos bancos
- 3.3 - Índices, Tipos de Dados e Integridade Referencial

4 - DML (Data Manipulation Language) 34
- 4.1 - Consultas (select,insert, update e delete)
- 4.2 - Consultas JOINS
- 4.3 - Sub Consultas

5 - Funções Internas 45
- 5.1 - Strings
- 5.2 - Matemáticas
- 5.3 - Agrupamento (Agregação)
- 5.4 - Data/Hora
- 5.5 - Formatação de Tipos de Dados
- 5.6 - Conversão de Tipos (CAST)

6 - Funções Definidas pelo Usuário e Triggers 55
- 6.1 - SQL
- 6.2 - PlpgSQL
- 6.3 – Triggers

7 - DCL (Data Control Language) - Administração 68
- 7.1 - Usuários, grupos e privilégios

8 - Transações 72

9 – Administração 75
- 9.1 - Backup e Restore
- 9.2 - Importar e Exportar
- 9.3 - Converter
- 9.4 - Otimização e Desempenho

10 - Replicação 84

11 - Configurações 86
- 10.1 - Copiar o script de inicialização dos contribs
- 10.2 - Adicionar ao Path
- 10.3 - Configurar acessos (pg_hba.conf)
- 10.4- Configurações diversas (postgresql.conf)

12 – Metadados (Catálogo) 92

13 - Conectividade 105
- 13.1 - Com Java (JDBC)
- 13.2 - Com aplicativos Windows (ODBC)
- 13.3 - Com PHP
- 13.4 - Exemplos de conexão com PHP, Java e VB

14 - Ferramentas 108
14.1 - psql
14.2 - phpPgAdmin
14.3 - PgAdmin
14.4 - EMS PostgreSQL
14.5 - Azzurry Clay (modelagem com o Eclipse)
14.6 - dbVisualizer
14.7 - OpenOffice Base
15 – Apêndices 124
15.1 – Planejamento e Projeto de Bancos de Dados
15.2 – Implementação de Banco de Dados com o PostgreSQL
15.3 - Integridade Referencial - PostgreSQL
15.4 – Dicas Práticas de uso do SQL
15.5 – Dicas sobre Desempenho e Otimizações do PostgreSQL
16 – Exercícios 149

17 - Referências 154

1 - Introdução

História dos SGDBs
Anos 60 - utilizados sistemas gerenciadores de arquivos (ISAM e VSAM), usados até hoje.
Anos 70 - Gerenciadores de Bancos de dados de rede. Extinguiram-se nos anos 90.
Anos 80 - SGBDRs (Oracle, DB2, SQLServer)
Anos 90 - SGBDOR (Oracle, DB2, PostgreSQL e Informix)
Anos 90 - SGBDOO (Caché)

SGBD = Composto por programas de gerenciamento, armazenamento e acesso aos dados, com a finalidade de tornar ágil e eficiente a manipulação dos dados.

Dicionário de dados - metadados, dados sobre os dados, ou seja, informações sobre a estrutura dos bancos de dados (nomes de tabelas, de campos, tipos de dados, etc).

DBA - Database Administrator, com as funções de:
- Definir e modificar esquemas, estruturas de armazenamento e métodos de acesso
- Liberar privilégios de acesso
- Especificação de restrição de integridade

Simplificando temos (no PostgreSQL), em termos de estrutura:
- Um SGBD é formado por bancos de dados, tablespaces, usuários e alguns programas auxiliares;
- Um banco de dados é formado pelos esquemas e linguagens;
- Um esquema é formado por funções de agrupamento, funções, triggers, procedures, sequências, tabelas e views;
- Tabelas são formadas por campos, constraints, índices e triggers.
- Em termos de dados uma tabela é formada por registros e campos.

Segundo a Wikipedia (http://pt.wikipedia.org):
...
A apresentação dos dados pode ser semelhante à de uma planilha eletrônica, porém os sistemas de gestão de banco de dados possuem características especiais para o armazenamento, classificação e recuperação dos dados.

Os bancos de dados são utilizados em muitas aplicações, abrangendo praticamente todo o campo dos programas de computador. Os bancos de dados são o método de armazenamento preferencial para aplicações multiusuário, nas quais é necessário haver coordenação entre vários usuários. Entretanto, são convenientes também para indivíduos, e muitos programas de correio eletrônico e organizadores pessoais baseiam-se em tecnologias padronizadas de bancos de dados.

Em Março, 2004, AMR Research (como citado em um artigo da CNET News.com listado na secção de "Referências") previu que aplicações de banco de dados de código aberto seriam amplamente aceitas em 2006.

Esquemas – são subdivisões de bancos de dados, cuja função é permitir um melhor nível de organização.

Projetos de mesma categoria, que precisem acessar uns aos outros devem ficar em um mesmo banco, podendo ficar em esquemas separados.

Tabelas – são subdivisões de um esquema, nelas realmente ficam armazenados os dados de um banco. Uma tabela parece realmente com uma tabela em papel, tipo planilha, com linhas e colunas. Cada linha representa um registro de banco de dados e cada cruzamento de coluna com linha representa um campo de tabela.

Tipo de Dados de um campo restringe o conjunto de valores (domínio) que pode ser atribuído ao campo e atribui semântica aos dados armazenados. Um campo do tipo numérico não aceita dados do tipo texto ou similar.

Citação da Introdução do documento sobre otimização do PostgreSQL

POSTGRESQL é um SGBD objeto-relational (SGBDOR) desenvolvido via Internet por um grupo de desenvolvedores espalhados pelo globo. É uma alternativa de código fonte-aberta para SGBDs comerciais como Oracle e Informix.
O POSTGRESQL foi desenvolvido originalmente na Universidade de Califórnia em Berkeley. Em 1996, um grupo começou o desenvolvimento do SGBD na Internet. Eles usam e-mail para compartilhar idéias e servidores de arquivos para compartilhar código. POSTGRESQL é agora comparável à SGBDs comerciais em termos de características, desempenho e confiança. Hoje tem transações, views, procedimentos armazenados, e constranints de integridade referencial. Apóia um número grande de interfaces de programação, como ODBC, Java (JDBC), TCL/TK, PHP, Perl e Python, entre outros. POSTGRESQL continua avançando a um tremendo passo, graças a um grupo talentoso de desenvolvedores via Internet. (Bruce Momjian - 16th January 2003)

Projeto POSTGRES (1986-1994): Partiu do projeto do SGBD Ingres de Berkeley. Projetista: Michael Stonebraker.
Em 1995 dois estudantes de Berkeley (Jolly Chen e Andrew Yu) adicionam suporte a SQL. Seu novo nome: Postgres95. Foi totalmente reescrito em C e também adotou a SQL. Foi originalmente patrocinado pelo DARPA, ARO, NSF e ESL Inc.
Em 1996: Disponibilizado na Internet sob o nome de PostgreSQL.

O PostgreSQL aniversariou no dia 08/07/2006, quando completou 10 anos (08/07/1996). Seu décimo aniversário foi comemorado nos dias 08 e 09 de julho próximo, em Toronto, Canadá, com algumas conferências sobre o mesmo. Atualmente está na versão 8.1.4 (14/09/2006).

Para saber mais sobre a história do PostgreSQL visite o site oficial em:
http://www.postgresql.org/docs/current/interactive/history.html

Ou em português em:
http://pgdocptbr.sourceforge.net/pg80/history.html

Características:
- O PostgreSQL suporta grande parte do SQL ANSI, inclusive do SQL 2003, além de oferecer outros recursos importantes, como:
- Comandos complexos
- Chaves estrangeiras (Foreign Key)
- Gatilhos (Triggers)
- Visões (views)
- Integridade de Transações
- Controle de Simultaneidade Multiversão (MVCC)
- Suporta múltiplas transações online concorrentes entre usuários.
- Suporte a Rules (sistema de regras que reescreve diretivas SQL)
- Criação de tabelas temporárias (CREATE TEMP TABLE nome(listadecampos tipos);)

Traz também opções de extensão pelo usuário:
• Tipos de dados
• Funções
• Operadores
• Funções de Agregação (Agrupamento)
• Métodos de Índice
• Linguagens Procedurais (Stored Procedures)

Licença
Sua licença é BSD, portanto pode ser utilizado, modificado e distribuído por qualquer pessoa ou empresa para qualquer finalidade, sem qualquer encargo, em quaisquer dos sistemas operacionais suportados.

Empresas que Utilizam PostgreSQL
BASF (PDF format)
Fujitsu
Apple
RedHat
Sun
Pervasive
Mohawk Software
Proximity
Radio Paradise
Shannon Medical Center
Spiros Louis Stadium
The Dravis Group OSS Report
Vanten Inc.
SRA
Rambler
Netezza
VA Software
Travel Post
National Weather Service

Aplicações Corporativas de Alto Volume: Uma Solução com o PostgreSQL
A utilização da dupla PostgreSQL+Linux nas empresas cresce rapidamente e é um exemplo de como produtos Open Source podem ajudar empresas a racionalizar os custos de TI. Uma das características do PostgreSQL é a sua capacidade de lidar com um grande volume de dados. E-xistem aplicações em produção com tabelas possuindo mais de 100 milhões de linhas. No Brasil, existem casos de sucesso de empresas lidando com bases com dezenas de milhões de registros gerenciadas pelo PostgreSQL.

Uma das maiores implantações de PostgreSQL no Brasil é na Atrium Telecom, empresa de tele-fonia corporativa de São Paulo. O PostgreSQL é utilizado como banco de dados do sistema de billing e tem uma base de dados de mais de 100GB e efetua 1 milhão de transações diárias. As maiores tabelas do sistema contam com mais de 70 milhões de linhas.

A utilização do banco de dados PostgreSQL é cada vez mais ampla nas empresas que buscam um servidor de banco de dados altamente sofisticado, com alta performance, estável e capacitado para lidar com grandes volumes de dados. O fato de ser um produto Open Source, sem custos de licença para nenhum uso, torna o PostgreSQL uma alternativa extremamente atraente para empresas que buscam um custo total de propriedade (TCO) menor para os ativos de TI.

Citação de: http://www.dib.com.br/dib%20cd/LC2003/P%C3%A1ginas/LC2003_Conf.html

Metrô de São Paulo e DATAPREV também utilizam o PostgreSQL.

Sobre o Autor

Ribamar FS

Desenvolvedor de aplicativos web para a Intranet do DNOCS (Departamento Nacional de Obras Contra as Secas). Desenvolve atualmente em PHP com PostgreSQL.
Trabalhou no DNOCS por algum tempo como administrador de redes Linux e FreeBSD.

É graduado em Engenharia Civil pela Universidade de Fortaleza (UNIFOR)
Com especialização em Irrigação e Drenagem pela UFC/IRYDA
Cursando Especialização em Java na UNIFOR
Concluiu o Curso de PostgreSQL pela dbExpert (São Paulo) e pelo Evolução (Fortaleza)
Concluiu o curso de Administração Linux pelo Evolução (Fortaleza)
Foi escritor colaborador da Revista Forum Access (na área de Access)
É escritor colaborador da Revista Web Mobile (artigo sobre Joomla 02/2006)
Foi professor de cursos de extensão na UNIFOR (PHP+MySQL e PHP + PostgreSQL) em 2005 e 2006
Apresentou palestra sobre PostgreSQL na UNIFOR no dia 29/03/2006.
Apresentou palestra sobre PostgreSQL na UFC no dia 21/09/2006 (II Semana de Software Livre da UFC).

Compartilha seus conhecimentos através do site:
http://ribafs.tk (http://www.geocities.com/ribafsindex) e http://www.ribafs.net

2 – Instalação

Instalação no Windows XP

Lembrar que: não instala em sistema de arquivos FAT-32, mesmo que seja o XP em FAT-32, ele não instala. Precisa instalar em NTFS e não instala no XP Start Edition.

- Fazer download do site oficial (www.postgresql.org) (hoje postgresql-8.1.4-1.zip)
- Executar o arquivo postgresql-8.1.msi
- Selecionar idioma e Start. Depois em Próximo.
- Na tela Informações de Instalações existem muitas informações importantes:
    - Sugere a leitura da FAQ
    - Fala das licenças dos diversos softwares a serem instalados
    - As versões 95, 98 e Me do Windows não são suportadas pelo PostgreSQL
    - Usar obrigatoriamente em sistema de arquivos NTFS
    - Instalar como serviço (mesmo que deixe como manual)
    - O PostgreSQL não executa com usuário que tenha privilégios de administrador
    - Os drivers jdbc estão no subdiretório \jdbc, que deve ser adicionada ao CLASSPATH
- Na Tela "Opções de Instalação" marque:
    - Suporte para idioma nativo (importante para ter as mensagens em pt_BR)
    - E outros que considere importantes e clique em Próximo
- Na tela "Configuração do Serviço":
    - Poderá optar entre instalar como serviço ou não. Como serviço é mais prático. Clique em Próximo (ele criará uma senha)
    - Obs.: Caso já tenha instalado o PostgreSQL antes nesta máquina deverá remover o usuário "postgres" antes de continuar:
    - Painel de controle - Ferramentas administrativas - Gerenciamento do computador - Usuários e grupos locais - Usuários. Remova o "postgres"
    - Agora clique em Próximo e Sim
- Na tela "Inicializar o agrupamento de bancos de dados:
    - Caso precise acessar sua máquina de outra remota marque Endereços
    - Em Locale selecione Português Brasil
    - Em Codificação selecione LATIN1
    - Entre com uma senha e repita. Altere o usuário se for o caso e Próximo.
- Na tela "Habilitar Linguagens Procedurais" deixe marcada PL/pgsql e Próximo
- Na tela "Habilitar Módulos Contrib" marque os desejados e Próximo
- Na tela "Habilitar PostGIS em templae1" marque se precisar que todos os bancos tragam o PostGIS e Próximo e Próximo.
- Após instalar, na tela "Instalação concluída" recomenda-se que se cadastrar na lista pgsql-announce, que envia informações semanais sobre novas versões e correções de error. Basta clicar no botão, fazer o cadastro e Concluir.

Editar postgresql.conf e adicionar a linha (datestyle = 'sql european'), após a existente.

Pré-requisitos para instalação do PostgreSQL num UNIX:

- make do GNU (gmake ou make)
- compilador C, preferido GCC mais recente
- gzip
- biblioteca readline (para psql)
- gettext (para NLS)
- kerberos, openssl e pam (opcional, para autenticação)

Instalação no Linux

Várias distribuições já contam com binários para instalação do PostgreSQL (Ubuntu, Debian, Slackware, RedHat, Fedora, etc).

Em uma instalação padrão do Ubuntu veja o que precisa para instalar os fontes:

Antes de instalar:
  sudo apt-get install build-essential
  sudo apt-get install libreadline-dev
  sudo apt-get install zlib1g-dev
  sudo apt-get install gettext
E use make ao invés de gmake.

Mas caso queira ter um controle maior instalando os fontes, apenas faça o download e descompacte (gosto de descompactar em /usr/local/src e instalar no diretório default, que é /usr/local/pgsql).
Instalar pelos binários da distribuição tem as vantagens de já instalar e configurar praticamente tudo automaticamente, mas instalar dos fontes dá um maior controle sobre as configurações (você sabe que tudo ficará no /usr/local/pgsql), possibilidade de instalar sempre a última versão.
Aqui a instalação é no modo texto, mas mesmo assim não dá trabalho. Após descompactar visualize ou edite o arquivo INSTALL e siga as recomendações resumidas existentes no início do arquivo, reproduzidas abaixo:

make distclean (adicionei, para o caso de ter que repetir os procedimentos)
./configure
make (build – construir)
su (mudar para superusuário, ou no Ubuntu usar sudo para as linhas abaixo)
make install (instalar)
groupadd postgres (criar o grupo postgres)
useradd -g postgres -d /usr/local/pgsql postgres (criar o usuário postgres)
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data (tornar o postgres dono da pasta data)
passwd postgres
su - postgres   (se logar como postgres)
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 & (startar)
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test

Opcionalmente:
./configure –enable-nls=pt_BR –with-openssl (para mensagens em português e autenticação SSL).

Copiar o script de inicialização "linux" para o /etc/init.d (Debians):

De /usr/local/src/postgresql-8.1.4/contrib/start-script/linux para /etc/init.d/postgresql

Dar permissão de execução: chmod u+x /etc/init.d/postgresql

Se no Ubuntu ou outro Debian:
su - postgres
gedit .bash_profile (e adicione a linha):
PATH=/usr/local/pgsql/bin:$PATH

Pós Instalação (sh,bash,ksh e zsh):
LD_LIBRARY_PATH=/usr/local/pgsql/lib
export LD_LIBRARY_PATH
Ou no ~/.bash_profile do usuário postgres

initdb – inicializa o cluster, cria os scripts de configuração default.
postmaster – inicia o processo do servidor responsável por escutar por pedidos de conexão.

Para suporte aos locales do Brasil usar:
/usr/local/pgsql/bin/initdb -locale=pt_BR -D /usr/local/pgsql/data

A instalação via fontes (sources) em algumas distribuições muito enxutas, voltadas para para desktop, pode não funcionar da primeira vez, pois faltarão algumas bibliotecas, compiladores, etc.

Após a instalação está criado o agrupamento principal (cluster main) de bancos de dados do PostgreSQL.

Caso não se tenha confiança nos usuários locais é recomendável utilizar a opção -W, --pwprompt ou –pwfile do initdb, que atribuirá uma senha ao superusuário.
No arquivo pg_hba.conf utilizar autenticação tipo md5, password ou cript, antes de iniciar o servidor pela primeira vez.

Quando o programa que inicia o servidor (postmaster) está em execução, é criado um PID e armazenado dentro do arquivo postmaster.pid, dentro do subdiretório data. Ele impede que mais de um processo postmaster seja executado usando o mesmo cluster e diretório de dados.

Baixar PostgreSQL via Anonymous CVS:
Baixar CVS de - http://www.nongnu.org/cvs/

Instalar e Logar com qualquer senha:
cvs -d :pserver:anoncvs@anoncvs.postgresql.org:/projects/cvsroot login
Baixar fontes:
cvs -z3 -d :pserver:anoncvs@anoncvs.postgresql.org:/projects/cvsroot co -P pgsql
Isto irá instalar o PostgreSQL num subdiretório pgsql do diretório atual.

Atualizar a última instalação via CVS:
Acesse o diretório pgsql e execute - cvs -z3 update -d -P
Isto irá baixar somente as alterações ocorridas após a última instalação.

Também podemos criar um arquivo .cvsrc no home do usuário com as duas linhas:
cvs -z3
update -d -P

Atualização do PostgreSQL entre Versões

Caso você tenha uma versão que não seja 8.1.x e esteja querendo instalar a 8.1.4, então precisa fazer um backup dos seus dados e restaurar logo após a instalação como sugerido em seguida. Será assumido que sua instalação foi em:
/usr/local/pgsql e seus dados no sub data. Caso contrário atenha-se ao seu path para ajustes.

1 – Atenção para que seus bancos não estejam recebendo atualização durante o backup. Se preciso proíba acesso no pg_hba.conf.
2 – Efetuando backup:
pg_dumpall > bancos.sql .Para preservar os OIDs use a opção -o no pg_dumpall.
3 – Pare o servidor
pg_ctl stop ou outro comando

Caso queira instalar a nova versão no mesmo diretório da anterior
mv /usr/local/pgsql /usr/local/pgsql.old
Então instale a nova versão, crie o diretório de dados e start o novo servidor.
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data

Finalmente, restore seus dados com
/usr/local/pgsql/bin/psql -d postgres -f bancos.sql

Para mais detalhes sobre os procedimentos de instalação, veja itens 14.5 e 14.6 do manual.

Plataformas Suportadas
Atualmente o PostgreSQL suporta muitas plataformas, entre elas o Windows, Linux, FreeBSD, NetBSD, OpenBSD, Mac OS e diversos outros. Plataformas suportadas e as não suportadas na seção 14.7 do manual oficial.

No PostgreSQL o processo postmaster escuta por conexões dos clientes.
Existem mais dois processos também iniciados, ambos com nome postgres. Eles cuidam da gravação dos logs ou tabelas e da manutenção das estatísticas.
Para cada conexão com uma aplicação cliente é criado um novo processo com o mesmo nome do usuário da conexão. Por isso é importante que cada aplicativo tenha seu usuário e se tenha um maior controle.

Os arquivos de configuração (postgresql.conf, pg_hba.conf e pg_ident.conf) a partir da versão 8 podem ficar em diretório diferente do PGDATA.

Sugestão de Padrão

- Nomes de bancos no plural
- Nomes de tabelas no singular
- Exemplo:
  - banco – clientes
  - tabela - cliente

Criar Novo Cluster

Caso sinta necessidade pode criar outros clusters, especialmente indicado para grupos de tabelas com muito acesso.
O comando para criar um novo cluster na versão atual (8.1.3) do PostgreSQL é:

```
banco=# \h create tablespace
Comando:     CREATE TABLESPACE
Descrição:   define uma nova tablespace
Sintaxe:
CREATE TABLESPACE nome_tablespace [ OWNER usuário ] LOCATION 'diretório'
```

Exemplo:

```
CREATE TABLESPACE ncluster OWNER usuário LOCATION '/usr/local/pgsql/nc';
CREATE TABLESPACE ncluster [OWNER postgres] LOCATION 'c:\\ncluster';
```

O diretório deve estar vazio e pertencer ao usuário.

Criando um banco no novo cluster:

```
CREATE DATABASE bdcluster TABLESPACE = ncluster;
```

Obs: Podem existir numa mesma máquina vários agrupamentos de bancos de dados (cluster) gerenciados por um mesmo ou por diferentes postmasters.

Se usando tablespace o gerenciamento será de um mesmo postmaster, se inicializados por outro initdb será por outro.

Setar o Tablespace default:

```
SET default_tablespace = tablespace1;
```

Listar os Tablespaces existentes:

```
\db
SELECT spcname FROM pg_tablespace;
```

Detalhes extras no item 14.5 do manual oficial.

3 - DDL (Data Definition Language)

3.1 - Criação e exclusão de bancos, esquemas, tabelas, views, etc

Obs.: Nomes de objetos e campos não podem usar hífen (-). Alternativamente usar sublinhado (_).

campo-1 Inválido
campo_1 Válido

Nomes de Identificadores

Utiliza-se por convenção as palavras chaves do SQL em maiúsculas e os identificadores dos objetos que criamos em minúsculas.

Identificadores digitados em maiúsculas serão gravados em minúsculas, a não ser que venham entre aspas "".

Revisões da Linguagem SQL

SQL – 1989
SQL – 1992
SQL – 1999
SQL – 2003

Divisões da SQL

DML – Linguagem de Manipulação de Dados
DDL – Linguagem de Definição de Dados
DCL – Linguagem de Controle de Dados (autorização de dados e licença de usuários para controlar quem tem acesso aos dados).
DQL – Linguagem de Consulta de Dados (Tem apenas um comando: SELECT).

Exemplo Gráfico de Consultas (Tabela, com campos C1, C2)
(Adaptação de exemplo da Wikipedia (http://pt.wikipedia.org))

Tabela T		Consulta	Resultado	
C1	C2	SELECT * FROM T	C1	C2
1	a		1	a
2	b		2	b
C1	C2	SELECT C1 FROM T	C1	
1	a		1	
2	b		2	
C1	C2	SELECT * FROM T WHERE C1=1	C1	C2
1	a		1	a
2	b			
C1	C2	SELECT C1 FROM T WHERE C2=b	C1	
1	A		2	
2	B			

Criar Banco
banco=# \h create database
Comando: CREATE DATABASE
Descrição: cria um novo banco de dados
Sintaxe:

```
CREATE DATABASE nome
    [ [ WITH ] [ OWNER [=] dono_bd ]
           [ TEMPLATE [=] modelo ]
           [ ENCODING [=] codificação ]
           [ TABLESPACE [=] tablespace ] ]
           [ CONNECTION LIMIT [=] limite_con ] ]
```

CREATE DATABASE nomebanco;

Excluindo Um Banco
DROP DATABASE nomebanco;

Listar os bancos existentes:
\l - - No psql
psql -l (no prompt)
SELECT datname FROM pg_database;

Quando se cria um novo banco de dados sem indicar o modelo, o que de fato estamos fazendo é clonar o banco de dados template1.

Criar um banco para outro usuário:

CREATE DATABASE nomebanco OWNER nomeuser;
createdb -O nomeusuario nomebanco

Obs.: requer ser superusuário para poder criar banco para outro usuário.

Criar Tabela
postgres=# \h create table
Comando: CREATE TABLE
Descrição: define uma nova tabela
Sintaxe:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE nome_tabela ( [
  { nome_coluna tipo_dado [ DEFAULT expressão_padrão ] [ restrição_coluna [ ... ] ]
  | restrição_tabela
  | LIKE tabela_pai [ { INCLUDING | EXCLUDING } DEFAULTS ] }
  [, ... ]
] )
[ INHERITS ( tabela_pai [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

onde restrição_coluna é:

```
[ CONSTRAINT nome_restrição ]
{ NOT NULL |
 NULL |
 UNIQUE [ USING INDEX TABLESPACE tablespace ] |
 PRIMARY KEY [ USING INDEX TABLESPACE tablespace ] |
 CHECK (expressão) |
 REFERENCES tabela_ref [ ( coluna_ref ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
  [ ON DELETE ação ] [ ON UPDATE ação ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

e restrição_tabela é:

```
[ CONSTRAINT nome_restrição ]
{ UNIQUE ( nome_coluna [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
 PRIMARY KEY ( nome_coluna [, ... ] ) [ USING INDEX TABLESPACE tablespace ] |
 CHECK ( expressão ) |
 FOREIGN KEY ( nome_coluna [, ... ] ) REFERENCES tabela_ref [ ( coluna_ref [, ... ] ) ]
  [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE ação ] [ ON UPDATE
ação ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

Obs.: Atenção: nesta versão (8.1.3) WITH OID é opcional. As tabelas são criadas sem OID.

\d – visualizar tabelas e outros objetos
\d nometabela – visualizar estrutura da tabela

```
CREATE TABLE primeira_tabela (
  primeiro_campo text,
  segundo_campo integer
);
```

Excluindo Tabela

```
DROP TABLE primeira_tabela;
```

Valor Default (padrão) Para Campos

Ao definir um valor default para um campo, ao ser cadastrado o registro e este campo não for informado, o valor default é assumido. Caso não seja declarado explicitamente um valor default, o valor nulo (NULL) será o valor default.

```
CREATE TABLE produtos (
  produto_no integer,
  descricao text,
  preco numeric DEFAULT 9.99
);
```

Constraints (Restrições)

CHECK

Ao criar uma tabela podemos prever que o banco exija que o valor de um campo satisfaça uma expressão

```
CREATE TABLE produtos (
  produto_no integer,
  descricao text,
  preco numeric CHECK (preco > 0)
);
```

***Dando nome à restrição check*. Isso ajuda a tornar mais amigável as mensagens de erro.**

```
CREATE TABLE produtos (
  produto_no integer,
  descricao text,
  preco numeric CONSTRAINT preco_positivo CHECK (preco > 0)
);
```

```
CREATE TABLE produtos (
  produto_no integer,
  descricao text,
  desconto numeric CHECK (desconto > 0 AND desconto < 0.10),
  preco numeric CONSTRAINT preco_positivo CHECK (preco > 0),
  check (preco > desconto)
);
```

Constraint NOT NULL

Obrigar o preenchimento de um campo. Ideal para campos importantes que não devem ficar sem preenchimento. Mas devemos ter em mente que até um espaço em branco atende a esta restrição.

```
CREATE TABLE produtos (
  cod_prod   integer NOT NULL CHECK (cod_prod > 0),
```

```
  nome      text    NOT NULL,
  preco     numeric
);
```

Obs importante: nulos não são checados. UNIQUE não aceita valores repetidos, mas aceita vários nulos (já que estes não são checados). Cuidado com NULLs.

Unique Constraint

Obrigar valores exclusivos para cada campo em todos os registros

```
CREATE TABLE produtos (
  cod_prod   integer UNIQUE,
  nome       text,
  preco      numeric
);

CREATE TABLE produtos (
  cod_prod   integer,
  nome       text,
  preco      numeric,
  UNIQUE (cod_prod)
);

CREATE TABLE exemplo (
  a integer,
  b integer,
  c integer,
  UNIQUE (a, c)
);

CREATE TABLE produtos (
  cod_prod   integer CONSTRAINT unq_cod_prod UNIQUE,
  nome       text,
  preco      numeric
);
```

Evitando duplicação com nulos:

```
create table teste(
        id serial not null,
        parent integer null,
        component integer not null
);

postgres=# create unique index naoduplic on teste using btree (component) where (parent is null);
```

Chaves Primárias (Primary Key)
A chave primária de uma tabela é formada internamente pela combinação das constraints UNIQUE e NOT NULL. Uma tabela pode ter no máximo uma chave primária. A teoria de bancos de dados relacional dita que toda tabela deve ter uma chave primária. O PostgreSQL não obriga que uma tabela tenha chave primária, mas é recomendável seguir, a não ser que esteja criando uma tabela para importar de outra que contenha registros duplicados para tratamento futuro ou algo parecido.

```
CREATE TABLE produtos (
  cod_prod   integer UNIQUE NOT NULL,
  nome       text,
  preco      numeric
);

CREATE TABLE produtos (
  cod_prod   integer PRIMARY KEY,
  nome       text,
  preco      numeric
);

CREATE TABLE exemplo (
  a integer,
  b integer,
  c integer,
  PRIMARY KEY (a, c)
);
```

Chave Estrangeira (Foreign Key)
Criadas com o objetivo de relacionar duas tabelas, mantendo a integridade referencial entre ambas. Especifica que o valor da coluna (ou grupo de colunas) deve corresponder a algum valor existente em um registro da outra tabela. Normalmente queremos que na tabela estrangeira existam somente registros que tenham um registro relacionado na tabela principal. Como também garantir que não se remova um registro na tabela principal que tenha registros relacionados na estrangeira.

Tabela primária

```
CREATE TABLE produtos (
  cod_prod   integer PRIMARY KEY,
  nome       text,
  preco      numeric
);

CREATE TABLE pedidos (
  cod_pedido  integer PRIMARY KEY,
  cod_prod    integer,
  quantidade  integer,
  CONSTRAINT pedidos_fk FOREIGN KEY (cod_prod) REFERENCES produtos (cod_prod)
);
```

```
CREATE TABLE t0 (
 a integer PRIMARY KEY,
 b integer,
 c integer,
 FOREIGN KEY (b, c) REFERENCES outra_tabela    - - a coluna de destino será a PK
);

CREATE TABLE t1 (
 a integer PRIMARY KEY,
 b integer,
 c integer,
 FOREIGN KEY (b, c) REFERENCES outra_tabela (c1, c2)
);
```

OBS.: Preferir sempre criar FK, utilizando a palavra reservada FOREIGN KEY e não somente com REFERENCES.

Obviamente, o número de colunas e tipo na restrição devem ser semelhantes ao número e tipo das colunas referenciadas.

SIMULANDO ENUM

Para simular a constraint enum do MySQL, podemos usar a constraint check.
Dica do site "PostgreSQL & PHP Tutorials".

```
CREATE TABLE pessoa(
        codigo int null primary key,
        cor_favorita varchar(255) not null,
        check (cor_favorita IN ('vermelha', 'verde', 'azul'))
);

INSERT INTO pessoa (codigo, cor_favorita) values (1, 'vermelha'); -- OK
INSERT INTO pessoa (codigo, cor_favorita) values (1, 'amarela'); -- Erro, amarelo não consta
```

Herança
Podemos criar uma tabela que herda todos os campos de outra tabela existente.

```
CREATE TABLE cidades (
  nome       text,
  populacao   float,
  altitude    int     -- (em pés)
);

CREATE TABLE capitais (
  estado      char(2)
) INHERITS (cidades);
```

capitais assim passa a ter também todos os campos da tabela cidades.

Segundo uma entrevista (vide DBFree Magazine No. 2) com a equipe de desenvolvimento do PostgreSQL, evite utilizar herança de tabelas.

Esquemas (Schema)

\dn – visualizar esquemas

Um banco de dados pode conter vários esquemas e dentro de cada um desses podemos criar várias tabelas. Ao invés de criar vários bancos de dados, criamos um e criamos esquemas dentro desse. Isso permite uma maior flexibilidade, pois uma única conexão ao banco permite acessar todos os esquemas e suas tabelas. Portanto devemos planejar bem para saber quantos bancos precisaremos, quantos esquemas em cada banco e quantas tabelas em cada esquema.

Cada banco ao ser criado traz um esquema public, que é onde ficam todas as tabelas, caso não seja criado outro esquema. Este esquema public não é padrão ANSI. Caso se pretenda ao portável devemos excluir este esquema public e criar outros. Por default todos os usuários criados tem privilégio CREATE e USAGE para o esquema public.

Criando Um Esquema
CREATE SCHEMA nomeesquema;

Excluindo Um Esquema
DROP SCHEMA nomeesquema;

Aqui, quando o esquema tem tabelas em seu interior, não é possível apagar dessa forma, temos que utilizar:

DROP SCHEMA nomeesquema CASCADE;
Que apaga o esquema e todas as suas tabelas, portanto muito cuidado.
Obs.: O padrão SQL exige que se especifique RESTRICT (default no PostgreSQL) OU CASCADE, mas nenhum SGBD segue esta recomendação.

Obs.: é recomendado ser explícito quanto aos campos a serem retornados, ao invés de usar * para todos, entrar com os nomes de todos os campos. Assim fica mais claro. Além do mais a consulta terá um melhor desempenho.

Acessando Tabelas Em Esquemas
SELECT * FROM nomeesquema.nometabela;

Privilégios Em Esquemas
\dp – visualizar permissões

REVOKE CREATE ON SCHEMA public FROM PUBLIC; - - Remove o privilégio CREATE de todos os usuários.

Obtendo Informações sobre os Esquemas:

\dn
\df current_schema*

SELECT current_schema();
SELECT current_schemas(true);
SELECT current_schemas(false);
Visões (views)
\dp – visualizar views e outros objetos

Que são VIEWS?

São uma maneira simples de executar e exibir dados selecionados de consultas complexas em bancos.
Em que elas são úteis? Elas economizam grande quantidade de digitação e esforço e apresentam somente os dados que desejamos.

Criando Uma View

```
CREATE VIEW recent_shipments
      AS SELECT count(*) AS num_shipped, max(ship_date), title
        FROM shipments
        JOIN editions USING (isbn)
        NATURAL JOIN books AS b (book_id)
        GROUP BY b.title
        ORDER BY num_shipped DESC;
```

Usando Uma View

```
SELECT * FROM recent_shipments;
SELECT * FROM recent_shipments
      ORDER BY max DESC
      LIMIT 3;
```

Destruindo Uma View

```
DROP VIEW nomeview;
```

Criar as Tabelas que servirão de Base

```
CREATE TABLE client (
        clientid SERIAL NOT NULL PRIMARY KEY,
        clientname VARCHAR(255)
);

CREATE TABLE clientcontact (
        contactid SERIAL NOT NULL PRIMARY KEY,
        clientid int CONSTRAINT client_contact_check REFERENCES client(clientid),
        name VARCHAR(255),
        phone VARCHAR(255),
        fax VARCHAR(255),
        emailaddress VARCHAR(255)
);

CREATE VIEW client_contact_list AS
SELECT client.clientid, clientname, name, emailaddress FROM client, clientcontact
WHERE client.clientid = clientcontact.clientid;
```

Estando no psql e digitando \d podemos visualizar também as views.

O nome da visão deve ser distinto do nome de qualquer outra visão, tabela, seqüência ou índice no mesmo esquema.
A visão não é materializada fisicamente. Em vez disso, a consulta é executada toda vez que a visão é referenciada em uma consulta.
Fazer livre uso de visões é um aspecto chave de um bom projeto de banco de dados SQL.

As visões podem ser utilizadas em praticamente todos os lugares onde uma tabela real pode ser utilizada. Construir visões baseadas em visões não é raro.

Atualmente, as visões são somente para leitura: o sistema não permite inserção, atualização

ou exclusão em uma visão. É possível obter o efeito de uma visão atualizável criando regras que reescrevem as inserções, etc. na visão como ações apropriadas em outras tabelas. Para obter informações adicionais consulte o comando CREATE RULE.

CREATE VIEW vista AS SELECT 'Hello World';
é ruim por dois motivos: o nome padrão da coluna é ?column?, e o tipo de dado padrão da coluna é unknown. Se for desejado um literal cadeia de caracteres no resultado da visão deve ser utilizado algo como CREATE VIEW vista AS SELECT text 'Hello World' AS hello;

Veja capítulo 4 do Livro "Pratical PostgreSQL"
Supondo que uma consulta seja de particular interesse para uma aplicação, mas que não se deseja digitar esta consulta toda vez que for necessária, então é possível criar uma view baseada na consulta, atribuindo um nome a esta consulta pelo qual será possível referenciá-la como se fosse uma tabela comum.

```
CREATE VIEW minha_view AS
  SELECT cidade, temp_min, temp_max, prcp, data, localizacao
     FROM clima, cidades
     WHERE cidade = nome;
SELECT * FROM minha_visao;
```

Fazer livre uso de visões é um aspecto chave de um bom projeto de banco de dados SQL. As visões permitem encapsular, atrás de interfaces que não mudam, os detalhes da estrutura das tabelas, que podem mudar na medida em que as aplicações evoluem.

As visões podem ser utilizadas em praticamente todos os lugares onde uma tabela real pode ser utilizada. Construir visões baseadas em visões não é raro.

RULES
O comando CREATE RULE cria uma regra aplicada à tabela ou visão especificada.

Uma regra faz com que comandos adicionais sejam executados quando um determinado comando é executado em uma determinada tabela.

É importante perceber que a regra é, na realidade, um mecanismo de transformação de comando, ou uma macro de comando.

É possível criar a ilusão de uma visão atualizável definindo regras ON INSERT, ON UPDATE e ON DELETE, ou qualquer subconjunto destas que seja suficiente para as finalidades desejadas, para substituir as ações de atualização na visão por atualizações apropriadas em outras tabelas.

Existe algo a ser lembrado quando se tenta utilizar regras condicionais para atualização de visões: é obrigatório haver uma regra incondicional INSTEAD para cada ação que se deseja permitir na visão. Se a regra for condicional, ou não for INSTEAD, então o sistema continuará a rejeitar as tentativas de realizar a ação de atualização, porque acha que poderá acabar tentando realizar a ação sobre a tabela fictícia da visão em alguns casos.

```
banco=# \h create rule
Comando:     CREATE RULE
Descrição:   define uma nova regra de reescrita
Sintaxe:
CREATE [ OR REPLACE ] RULE nome AS ON evento
   TO tabela [ WHERE condição ]
   DO [ ALSO | INSTEAD ] { NOTHING | comando | ( comando ; comando ... ) }
```

O comando CREATE RULE cria uma regra aplicada à tabela ou visão especificada.

evento
Evento é um entre SELECT, INSERT, UPDATE e DELETE.

condição
Qualquer expressão condicional SQL (retornando boolean). A expressão condicional não pode fazer referência a nenhuma tabela, exceto NEW e OLD, e não pode conter funções de agregação.

INSTEAD
INSTEAD indica que os comandos devem ser executados em vez dos (instead of) comandos originais.

ALSO
ALSO indica que os comandos devem ser executados adicionalmente aos comandos originais.
Se não for especificado nem ALSO nem INSTEAD, ALSO é o padrão.

comando
O comando ou comandos que compõem a ação da regra. Os comandos válidos são SELECT, INSERT, UPDATE, DELETE e NOTIFY.

Dentro da condição e do comando, os nomes especiais de tabela NEW e OLD podem ser usados para fazer referência aos valores na tabela referenciada. O NEW é válido nas regras ON INSERT e ON UPDATE, para fazer referência à nova linha sendo inserida ou atualizada. O OLD é válido nas regras ON UPDATE e ON DELETE, para fazer referência à linha existente sendo atualizada ou excluída.

Obs.: É necessário possuir o privilégio RULE na tabela para poder definir uma regra para a mesma.
Exemplos:

```
CREATE RULE me_notifique AS ON UPDATE TO datas DO ALSO NOTIFY datas;

CREATE RULE r1 AS ON INSERT TO TBL1 DO
(INSERT INTO TBL2 VALUES (new.i); NOTIFY TBL2);

CREATE RULE "_RETURN" AS ON SELECT TO minha_visão DO INSTEAD
SELECT * FROM minha_tabela;  -- Ao invés de selecionar da visão seleciona da tabela.
```

Banco de dados modelo intocado
Existe um modelo de banco de dados que sempre se preserva original, que é o template0. O template template1 pode incorporar objetos e acaba algumas vezes ficando inviável seu uso como modelo. Quando isso acontece podemos substitui-lo com uma cópia do template0.

Criando banco de dados baseado em outro modelo
```
CREATE DATABSASE nomebanco TEMPLATE template0;
createdb -T template0 nomebanco
```

Recriando o template1
```
\c testes
postgres=# UPDATE pg_database SET datistemplate=false WHERE datname='template1';
testes=# DROP DATABASE template1;
testes=# CREATE DATABASE template1 TEMPLATE template0 ENCODING 'latin1';
testes=# \c template1
template1=# VACUUM FULL FREEZE;
template1=# VACUUM FULL;
template1=# UPDATE pg_database SET datistemplate=true WHERE datname='template1';
```

Agora temos um template1 original e limpo.

3.2 - Alterações nos objetos dos bancos

Adicionar campo, remover campo, adicionar constraint, remover constraint, alterar valor default, alterar nome de campo, alterar nome de 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;

3.3 - Índices, Tipos de Dados e Integridade Referencial

É importante conhecer bem o máximo de recursos existentes no banco, especialmente aqueles relacionados às nossas necessidades. Assim trabalhamos com mais eficiência e criamos bancos mais leves e com mais potencial. Os tipos de dados são fatores de desempenho.

Exemplo:
Se um campo tipo inteiro irá precisar de valores até 100 e nunca mudará esta faixa. Não devemos usar este campo com o tipo INT8, quando o INT2 atende e sobra.
De forma semelhante escolher todos os demais campos da tabela com bom senso.
Mais Detalhes no Capítulo 8 do Manual:
http://pgdocptbr.sourceforge.net/pg80/datatype.html

Índices

Os índices são recursos do SGBD para melhorar o desempenho de consultas. Mas como o uso de índices também tem um preço é importante planejar bem e conhecer as particularidades antes de adicionar um índice.

Cada vez que um registro é inserido ou atualizado a tabela de índices também é atualizada.

Quando criamos consultas SQL, que pesquisam tabelas com muitos registros e esta consulta usa a cláusula WHERE, então os campos que fazem parte da cláusula WHERE são bastante indicados para índice, para que melhore o desempenho da consulta.

Os índices são uma forma de melhorar o desempenho de bancos de dados. Ao invés de procurar de forma sequencial, o servidor procura pelo índice, como se faz uma busca em índices de livros e vai-se diretamente à página procurada.

O índice é passado para cada registro adicionado ou removido.

É difícil criar regras genéricas para determinar que índices devem ser definidos. Muita experiência por parte do administrador e muita verificação experimental é necessária na maioria dos casos.

Criar um índice:
CREATE INDEX nomeindice ON tabela (campo);

Regra geral para nome de índice: idx_nometabela_nomecampo
Obs.: índices não importantes ou não utilizados devem ser removidos.

Remover índice:
DROP INDEX nomeindice;

Criar um índice Único:
CREATE UNIQUE INDEX nomeindice ON tabela (campo);
Obs.: Somente os índices tipo B-tree podem ser do tipo Unique.

Criar um índice com várias colunas:
CREATE INDEX idx_clientes_ps ON clientes (codigo, nome);

Boa indicação para consultas com WHERE...AND. Ao usar OR o índice não será utilizado pelo PostgreSQL:
SELECT nome FROM clientes WHERE codigo = 12 AND nome = 'João';

Usar índices com várias colunas com moderação. Índices com mais de 3 colunas tem grandes possibilidades de não serem utilizados internamente.

Tipos de Índices
O PostgreSQL suporta atualmente quatro tipos de índices: B-tree (árvore B), R-tree (árvore R), Hash e GiST.

B-tree -> é o tipo padrão (assume quando não indicamos). São índices que podem tratar consultas de igualdade e de faixa, em dados que podem ser classificados.
Indicado para consultas com os operadores: <, <=, =, >=, >. Também pode ser utilizado com LIKE, ILIKE, ~ e ~*.

R-tree -> tipo mais adequado a dados espaciais. Adequado para consultas com os operadores: <<, &<, &>, >>, @, ~=, &&.

Hash ->indicados para consultas com comparações de igualdade simples. É desencorajado seu uso. Em seu lugar recomenda-se o B-tree.

GiST ->

Criando índices de tipos diferentes:
CREATE INDEX nome ON tabela USING tipo (campo);

tipo: BTREE, RTREE, HASH, GIST

Obs.: Somente os tipos B-tree e GiST suportam índices com várias colunas.
Índices com mais de um campo somente será utilizado se as cláusulas com os campos indexados forem ligados por AND.
Um índice com mais de 3 campos dificilmente será utilizado.

Índice Parcial
Criado apenas sobre um subconjunto dos registros de uma tabela, definido numa expressão durante a criação do índice parcial. É um recurso para melhorar o desempenho dos índices, já que atualiza somente parte dos registros.

Obs.: na maioria dos casos a vantagem de um índice parcial sobre um índice integral não é muita.

Exemplos:

Examinando a Utilização dos Índices
A verificação de uso de índices deve ser feita com os comandos EXPLAIN e ANALYZE, sendo que o comando ANALYZE sempre deve ser executado antes. O comando ANALYZE coleta estatísticas sobre a distribuição dos valores na tabela.
Devem ser utilizados dados reais e o conjunto de dados de teste nunca deve ser pequeno.

Atentar para usar índices nos campos das Cláusulas
- FOREIGN KEY
- ORDER BY
- WHERE
- ON
- GROUP BY
- HAVING

Exemplos prático da vantagem do Índice

- Uma tabela contendo os CEPs do Brasil, com 633.401 registros.
Esta tabela sem nenhum índice executa a consulta abaixo:

```
\timing
SELECT * FROM cep_tabela WHERE cep = '60420440';
```

Em 7691 ms

- Pós adicionar um índice:
```
ALTER TABLE cep_tabela ADD CONSTRAINT cep_pk PRIMARY KEY (cep);
```

A mesma consulta anterior agora gasta apenas 10 ms.
Isso num AMD Duron 1300, 128MB de RAM).

Índice Funcional
```
CREATE INDEX nomeindice ON tabela (lower (nomecampo));
```

Ótimo artigo no iMasters
http://www.imasters.com.br/artigo.php?cn=1897&cc=23
http://www.imasters.com.br/artigo.php?cn=1922&cc=23
http://www.imasters.com.br/artigo.php?cn=1959&cc=23

Vide manual oficial, capítulo 11 para detalhes.

Tipos de Dados Mais Comuns

Numéricos			
Tipo	**Tamanho**	**Apelido**	**Faixa**
smallint (INT2)	2 bytes	inteiro pequeno	-32768 a +32767
integer (INT ou INT4)	4 bytes	inteiro	-2147483648 até +2147483647
bigint (INT8)	8 bytes	inteiro longo	-9223372036854775808 a +9223372036854775807
numeric (p,e)			tamanho variável, precisão especificada pelo usuário. Exato e sem limite
decimal (p,e)			e – escala (casas decimais) p – precisão (total de dígitos, inclusive estala)
real (float)	4 bytes	ponto flutuante	precisão variável, não exato e precisão de 6 dígitos
double precision	8 bytes	dupla precisão	precisão variável, não exato e precisão de 15 dígitos
int (INT4)			mais indicado para índices de inteiros
Caracteres			
character varying(n)		varchar(n)	comprimento variável, com limite
character(n)		char(n)	comprimento fixo, completa com brancos
text			comprimento variável e ilimitado
Desempenho semelhante para os tipos caractere.			
Data/Hora			
timestamp[(p)] [witout time zone]	8 bytes	data e hora sem zona	4713 AC a 5874897 DC
timestamp [(p)][with time zone]	8 bytes	data e hora com zona	4713 AC a 5874897 DC
interval	12 bytes	intervalo de tempo	178000000 anos a 178000000 anos
date	4 bytes	somente data	4713 AC até 32767 DC
time [(p)] [without time zone]	8 bytes	somente a hora	00:00:00.00 até 23:59:59.99
time [(p)] [with time zone]	8 bytes	somente a hora	00:00:00.00 até 23:59:59.99
[(p)] - é a precisão, que varia de 0 a 6 e o defualt é 2.			

Tipos de Dados Mais Comuns (Continuação)

Boleanos			
Tipo	**Tamanho**	**Apelido**	**Faixa**
TRUE		Representações:	't', 'true', 'y', 'yes' e '1'
FALSE		Representações:	'f', 'false', 'n', 'no', '0'
Apenas um dos dois estados. O terceiro estado, desconhecido, é representado pelo NULL.			

Exemplo de consulta com boolean:

```
CREATE TABLE teste1 (a boolean, b text);
INSERT INTO teste1 VALUES (TRUE, 'sic est');
INSERT INTO teste1 VALUES (FALSE, 'non est');
SELECT * FROM teste1;
```

Retorno:

```
 a |    b
---+---------
 t | sic est
 f | non est
```

Alerta: a entrada pode ser: 1/0, t/f, true/false, TRUE/FALSE, mas o retorno será semp re t/f.

Obs.: Para campos tipo data que permitam NULL, devemos prever isso na consulta SQL e passar NULL sem delimitadores e valores não NULL com delimitadores.

Obs: Evite o tipo MONEY que está em obsolescência. Em seu lugar use NUMERIC.
Prefira INT (INTEGER) em lugar de INT4, pois os primeiros são padrão SQL. Em geral evitar os nomes INT2, INT4 e INT8, que não são padrão. O INT8 ou bigint não é padrão SQL.
Em índices utilize somente INT, evitando smallint e bigint, que nunca serão utilizados.

Tipos SQL Padrão
bit, bit varying, boolean, char, character varying, character, varchar, date, double precision, integer, interval, numeric, decimal, real, smallint, time (com ou sem zona horária), timezone (com ou sem zona horária).

O tipo NUMERIC pode realizar cálculos exatos. Recomendado para quantias monetárias e outras quantidades onde a exatidão seja importante. Isso paga o preço de queda de desempenho comparado aos inteiros e flutuantes.

Pensando em portabilidade evita usar NUMERIC(12) e usar NUMERIC (12,0).

A comparação de igualdade de dois valores de ponto flutuante pode funcionar conforme o esperado ou não.

O PostgreSQL trabalha com datas do calendário Juliano.
Trabalha com a faixa de meio dia de Janeiro de 4713 AC (ano bisexto, domingo de lua nova) até uma data bem distante no futuro. Leva em conta que o ano tem 365,2425 dias.

SERIAL
No PostgreSQL um campo criado do “tipo” SERIAL é internamente uma seqüência.

Os principais SGBDs utilizam alguma variação deste tipo de dados (auto-incremento).
Serial é o “tipo” auto-incremento do PostgreSQL. Quando criamos um campo do tipo SERIAL ao inserir um novo registro na tabela com o comando INSERT omitimos o campo tipo SERIAL, pois ele será unserido automaticamente pelo PostgreSQL.

CREATE TABLE serial_teste (codigo SERIAL, nome VARCHAR(45));
INSERT INTO serial_teste (nome) VALUES ('Ribamar FS');

Obs.: A regra é nomear uma seqüência “serial_teste_codigo_seq”,ou seja, tabela_campo_seq.

select * from serial_teste_codigo_seq;

Esta consulta acima retorna muitas informações importantes sobre a seqüência criada: nome, valor inicial, incremento, valor final, maior e menor valor além de outras informações.

Veja que foi omitido o campo código mas o PostgreSQL irá atribuir para o mesmo o valor do próximo registro de código. Por default o primeiro valor de um serial é 1, mas se precisarmos começar com um valor diferente veja a solução abaixo:

Setando o Valor Inicial do Serial
ALTER SEQUENCE tabela_campo_seq RESTART WITH 1000;

CHAR corresponde a CHAR(1).
VARCHAR corresponde a uma cadeia de tamanho sem limites.

Diferença de Desempenho
Internamente o PostgreSQL armazena em tabelas separados os valores longos, para não interferirem no acesso dos valores curtos da coluna. O maior valor permitido para uma cadeia de caracteres é de 1GB. Para valores maiores usar TEXT ou VARCHAR sem especificar comprimento.

Tipos de Dados Geométricos

Geometric Types

Name	Storage Size	Representation	Description
point	16 bytes	Point on the plane	(x,y)
line	32 bytes	Infinite line (not fully implemented)	((x1,y1),(x2,y2))
lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
box	32 bytes	Rectangular box	((x1,y1),(x2,y2))
path	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)
path	16+16n bytes	Open path	[(x1,y1),...]
polygon	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)
circle	24 bytes	Circle	<(x,y),r> (center and radius)

Tipos de Dados de Redes

Network Address Types

Name	Storage Size	Description
`cidr`	12 or 24 bytes	IPv4 and IPv6 networks
`inet`	12 or 24 bytes	IPv4 and IPv6 hosts and networks
`macaddr`	6 bytes	MAC addresses

Tipos de Dados Array

Podemos ter campos com tipos de dados que não são simples, mas arrays.

```
CREATE TABLE salario (
    nome            text,
    apgamento  integer[],
    agendamento   text[][]
);

CREATE TABLE tictactoe (
    quadrado   integer[3][3]
);
```

Entrando os valores:

'{{1,2,3},{4,5,6},{7,8,9}}'

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {"meeting"}}');
ERROR:  multidimensional arrays must have array expressions with matching
dimensions
```

Precisa ter a mesma quantidade de elementos.

```
INSERT INTO sal_emp
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {"training", "presentation"}}');

INSERT INTO sal_emp
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"breakfast", "consulting"}, {"meeting", "lunch"}}');



SELECT * FROM sal_emp;
 name  |      pay_by_quarter       |                 schedule
-------+---------------------------+-------------------------------------------
 Bill  | {10000,10000,10000,10000} | {{meeting,lunch},{training,presentation}}
 Carol | {20000,25000,25000,25000} | {{breakfast,consulting},{meeting,lunch}}
(2 rows)
```

O construtor `ARRAY` também pode ser usado:

```
INSERT INTO sal_emp
    VALUES ('Bill',
    ARRAY[10000, 10000, 10000, 10000],
    ARRAY[['meeting', 'lunch'], ['training', 'presentation']]);

INSERT INTO sal_emp
    VALUES ('Carol',
    ARRAY[20000, 25000, 25000, 25000],
    ARRAY[['breakfast', 'consulting'], ['meeting', 'lunch']]);
```

Acessando:

SELECT name FROM sal_emp WHERE pay_by_quarter[1] <> pay_by_quarter[2];

SELECT pay_by_quarter[3] FROM sal_emp;

Faixa de valores- inferior:superior:

SELECT schedule[1:2][1:1] FROM sal_emp WHERE name = 'Bill';

SELECT array_dims(ARRAY[1,2] || 3);

SELECT array_prepend(1, ARRAY[2,3]);

SELECT array_cat(ARRAY[1,2], ARRAY[3,4]);

SELECT 1 || ARRAY[2,3] AS array;

SELECT ARRAY[1,2] || ARRAY[[3,4]] AS array;

```
SELECT f1[1][-2][3] AS e1, f1[1][-1][5] AS e2
 FROM (SELECT '[1:1][-2:-1][3:5]={{{1,2,3},{4,5,6}}}'::int[] AS f1) AS ss;
```

4 - DML (Data Manipulation Language)

SQL (Structure Query Language) - É uma linguagem declarativa, onde você diz ao computador o que deseja fazer e deixa a máquina decidir a forma correta de chegar ao resultado.

Para o primeiro contato com o PostgreSQL e para ter certeza de que o mesmo está corretamente instalado e configurado, podemos digitar na linha de comando do sistema operacional (como usuário do postgresql):

psql --version
psql -l

O psql é o programa de gerenciamento e uso do PostgreSQL pelo usuário local.
Com ele podemos fazer praticamente tudo que se pode fazer com o PG.

Alguns programas estão disponíveis na linha de comando do sistema operacional, permitindo criar e excluir bancos, criar e excluir usuários, entre outros. Os programas aí disponíveis dependem da versão instalada, do sistema operacional e da forma que foi instalado.

Quem instala através dos fontes (sources) tem um sub-diretório chamado contrib, onde estão os demais programas desenvolvidos pela comunidade de programadores do PG. Neste caso para instalar um destes programas execute "make; make install" estando no respectivo diretório. Um exemplo é o pgbench.

Os comandos via linha de comandos do SO, normalmente terminam com "db" e são formados com apenas uma palavra, createdb, por exemplo. Já de dentro do psql, eles normalmente são formados por duas palavras, como por exemplo,
CREATE DATABASE.

Os comandos a seguir serão executados na linha de comando do SO. Supondo que o super-usuário seja "postgres".

Forma mais geral de uso:
nome_comando opção -U nomeuser

Criar um banco de dados:
createdb controle_estoque -U postgres

Visualizar o banco criado:
psql -l -U postgres
Excluir o banco criado:
dropdb controle_estoque -U postgres

Ajuda sobre os comandos:
nome_comando --help

Acessar o banco criado através do terminal interativo de gerenciamento do PostgreSQL (psql):
psql controle_estoque -U postgres

```
D:\Arquivos de programas\PostgreSQL\8.1\bin>psql controle_estoque -U postgres
Bem vindo ao psql 8.1.3, o terminal iterativo do PostgreSQL.
Digite:  \copyright para mostrar termos de distribuição
        \h para ajuda com comandos SQL
        \? para ajuda com comandos do psql
        \g ou terminar com ponto-e-vírgula para executar a consulta
        \q para sair

controle_estoque=#
```

Este é o prompt do psql. Veja que já nos recebe com boas vindas e com dicas de como podemos a qualquer momento receber ajuda. Especialmente atente para os comandos:

\h - para receber ajuda sobre comandos SQL. \h comando - ajuda sobre um comando
\? - ajuda sobre os comandos de operação do terminal psql
; - é o comando para indicar ao PG que execute nossa seqüência de comandos
\q - para sair do psql

Obs.: Aceita quebras de linha para uma seqüência de comandos.

Mesmo que possamos utilizar ferramentas gráficas ou Web para gerenciar o PG, é altamente recomendado que nos familiarizemos com a sintaxe dos comandos para entender como os comandos são executados internamente e ter maior domínio sobre o PG. Depois dessa fase, os que resistem aos encantos do psql :) podem usar uma das citadas ferramentas.

Vamos executar alguns comandos do psql e algumas pequenas consultas para ficarmos mais à vontade.

\l -- lista bancos, donos e codificação
\d -- descreve tabela, índice, seqüência ou view (visão)
\du -- lista usuários e permissões
\dg -- lista grupos
\dp -- lista privilégios de acesso à tabelas, views (visões) e sequências

psql controle_estoque -U postgres

```
controle_estoque=# SELECT version();
                                 version
-----------------------------------------------------------------
 PostgreSQL 8.1.3 on i686-pc-mingw32, compiled by GCC gcc.exe (GCC) 3.4.2 (mingw-special)
```

Para distinguir convencionou-se que as palavras chave do SQL sejam escritas em maiúsculas, mas podem ser escritas em minúsculas sem problema para o interpretador de comandos.
SELECT 25*4;
SELECT current_date;

4.1 - Consultas Básicas em SQL

SELECT – selecionar registros de tabelas
banco=# \h select -- da ajuda via psql
Comando: SELECT
Descrição: recupera (retorna) registros de uma tabela ou visão (view)
Sintaxe:

```
SELECT [ ALL | DISTINCT [ ON ( expressão [, ...] ) ] ]
   * | expressão [ AS nome_saída ] [, ...]
   [ FROM item_de [, ...] ]
   [ WHERE condição ]
   [ GROUP BY expressão [, ...] ]
   [ HAVING condição [, ...] ]
   [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
   [ ORDER BY expressão [ ASC | DESC | USING operador ] [, ...] ]
   [ LIMIT { contador | ALL } ]
   [ OFFSET início ]
   [ FOR { UPDATE | SHARE } [ OF nome_tabela [, ...] ] [ NOWAIT ] ]
```

ASC é o default

Item_de pode ser um dos:

```
   [ ONLY ] nome_tabela [ * ] [ [ AS ] alias [ ( alias_coluna [, ...] ) ] ]
   ( select ) [ AS ] alias [ ( alias_coluna [, ...] ) ]
   nome_função ( [ argumento [, ...] ] ) [ AS ] alias [ ( alias_coluna [, ...] | definição_coluna [, ...] ) ]
   nome_função ( [ argumento [, ...] ] ) AS ( definição_coluna [, ...] )
   item_de [ NATURAL ] tipo_junção item_de [ ON condição_junção | USING ( coluna_junção [, ...] ) ]
```

Sintaxe resumida:
SELECT lista_de_campos FROM expressão_de_tabela

A lista_de_campos é o retorno da consulta.

Exemplos:

```
1) SELECT siape AS "Matricula do Servidor" FROM pessoal;
2) SELECT pessoal.siape, pessoal.senha, locacoes.lotacao
   FROM pessoal, lotacoes WHERE pessoal.siape = lotacoes.siape
   ORDER BY lotacoes.lotacao;
```

DISTINCT – Escrita logo após SELECT desconsidera os registros duplicados, retornando apenas registros exclusivos.

SELECT DISTINCT email FROM clientes;
ALL é o contrário de DISTINCT e é o padrão, retornando todos os registros, duplicados ou não.

Ao fazer uma consulta, um registro será considerado igual a outro se pelo menos um campo for diferente. E os todos os valores NULL serão considerados iguais.

CLÁUSULA WHERE - Filtra o retorno de consultas.

Operadores aceitos:
=, >, <, <>, !=, >=, <=

```
SELECT nome FROM clientes WHERE email = 'ribafs@ribafs.org';
SELECT nome FROM clientes WHERE idade > 18;
SELECT nome FROM clientes WHERE idade < 21;
SELECT nome FROM clientes WHERE idade >= 18;
SELECT nome FROM clientes WHERE idade <= 21;
SELECT nome FROM clientes WHERE UPPER(estado) != 'CE';
SELECT nome FROM clientes WHERE email = 'ribafs@ribafs.org';
```

BETWEEN, LIKE, OR, AND, NOT, EXISTS, IS NULL, IS NOT NULL, IN

```
SELECT nome FROM clientes WHERE idade  BETWEEN 18 and 45;
SELECT nome FROM clientes WHERE email LIKE '%@gmail.com';
SELECT nome FROM clientes WHERE idade >18 21 OR idade < 21; -- entre 18 e 21
SELECT nome FROM clientes WHERE idade >= 18 AND UPPER(estado) = 'CE';
SELECT nome FROM clientes WHERE idade NOT BETWEEN 18 AND 21;
SELECT * FROM datas WHERE EXISTS(SELECT * FROM datas2 WHERE datas.data =
datas2.data);
SELECT nome FROM clientes WHERE estado IS NULL;
SELECT nome FROM clientes WHERE estado IS NOT NULL;
SELECT nome FROM clientes WHERE estado IN ('CE', 'RN');
```

GROUP BY - Geralmente utilizada com funções de agrupamento (de agregação), como também com HAVING. Agrupa o resultado dos dados por um ou mais campos de uma tabela. Utilizado para agrupar registros (linhas) da tabela que compartilham os mesmos valores em todas as colunas (campos) da lista. Normalmente feito para calcular agrupamentos (agregações) aplicáveis aos grupos.

Exemplos:

SELECT SUM(horas) FROM empregados; -- Traz a soma das horas de todos os empregados
SELECT empregado, SUM(horas) FROM empregados GROUPBY empregado; -- Traz a soma das horas de cada empregado. Veja que “empregado” deve aparecer em GROUP BY, já que os campos de retorno diferentes do usado na função de agrupamento devem vir no GROUP BY.

Dica: Quando se utiliza uma função de agrupamento num campo da lista do SELECT, os demais campos da lista deverão ser agrupados. Exemplo:

SELECT codigo, nome, count(valor) FROM vendas GROUP BY codigo, nome.
Exemplo:

```
SELECT c.nome, COUNT(p.quant) AS quantos
FROM clientes c, pedidos p
WHERE c.codigo = p.cod_cliente
GROUP BY (p.cod_cliente);
```

HAVING - Filtra o retorno de GROUP BY. Não altera o resultado,apenas filtra.

Exemplo:
SELECT cliente, SUM(uant) AS total
FROM pedidos GROUP BY cliente
HAVING total > 50;   -- ou HAVING SUM(quant) > 50;

ORDER BY - Ordena o resultado da consulta por um ou mais campos em ordem ascendente (default) ou descendente.

Exemplos:
ORDER BY cliente; -- pelo cliente e ascendente
ORDER BY cliente DESC; -- descendente
ORDER BY cliente, quantidade; -- pelo cliente e sub ordenado pela quantidade
ORDER BY cliente DESC, quant ASC;

No exemplo ordenando por dois campos:

SELECT * FROM pedidos ORDER BY cliente, quantidade; A saída ficaria algo como:

Antônio – 1
Antônio – 2
João   - 1
Pedro   - 1
Pedro   - 2

INSERT – Inserir registros em tabelas.
banco=# \h insert
Comando: INSERT
Descrição: insere novos registros em uma tabela

Sintaxe:
INSERT INTO tabela [(lista_de_campos)]
   { DEFAULT VALUES | VALUES ({ expressão | DEFAULT } [, ...]) | consulta }

DEFAULT - Se ao criar a tabela definirmos campos com valor default, ao inserir registros e omitir o valor para estes campos, o servidor os cadastrará com o valor default.

Exemplo (forma completa):
Na tabela o campo idade tem valor default 18.

INSERT INTO clientes (codigo, nome, idade) VALUES (1, “Ribamar FS”);
Neste exemplo será cadastrado para a idade o valor 18.

Forma Abreviada:
INSERT INTO clientes VALUES (1, “Ribamar FS”);
Não é recomendada, por não ser clara nem adequada para trabalho em grupo. Caso utilizemos esta forma somos obrigados a inserir os campos na exata ordem em que estão na tabela.

Inserindo com SubConsulta:
INSERT INTO clientes (codigo, nome, idade) VALUES
(SELECT fnome, fidade FROM funcionarios WHERE cli = 'S');
SELECT firstname, lastname, city, state INTO newfriend FROM friend;

UPDATE - Atualizar registros de tabelas
banco=# \h update
Comando: UPDATE
Descrição: atualiza registros de uma tabela

Sintaxe:
UPDATE [ONLY] tabela SET coluna = { expressão | DEFAULT } [, ...]
[FROM lista_de]
[WHERE condição]

Exemplos:
UPDATE clientes SET idade = idade + 1; -- Todos os registros de clientes serão atualizados

UPDATE pedidos SET quant = quant + 3
WHERE cliente IN (SELECT codigo FROM clientes WHERE idade > 18);

DELETE - Remover registros de tabelas
banco=# \h delete
Comando: DELETE
Descrição: apaga registros de uma tabela

Sintaxe:
DELETE FROM [ONLY] tabela
[USING lista_util]
[WHERE condição]

Exemplos:
DELETE FROM pedidos; -- Cuidado, excluirá todos os registros da tabela pedidos
DELETE FROM pedidos WHERE (codigo IS NULL); - - Remove sem confirmação nem com opção de desfazer.

4.2 - Junções de Tabelas ou Consultas

As junções SQL são utilizadas quando precisamos selecionar dados de duas ou mais tabelas.

Existem as junções com estilo non-ANSI ou theta (junção com WHERE)

E as junções ANSI join (com JOIN). As junções ANSI podem ser de dois tipos, as INNER JOINS e as OUTER JOINS. A padrão é a INNER JOIN. INNER JOIN pode ser escrito com apenas JOIN.

Exemplo ANSI:

SELECT p.siape, p.senha, l.lotacao FROM pessoal p CROSS JOIN lotacoes l;
Tipos de Junções

INNER JOIN - Onde todos os registros que satisfazem à condição serão retornados.

Exemplo:
```
SELECT p.siape, p.nome, l.lotacao
FROM pessoal p INNER JOIN lotacoes l
ON p.siape = l.siape ORDER BY p.siape;
```

Exemplo no estilo theta:
```
SELECT p.siape, p.nome, l.lotacao
FROM pessoal p, lotacoes l
WHERE p.siape = l.siape ORDER BY p.siape;
```

OUTER JOIN que se divide em **LEFT OUTER JOIN** e **RIGHT OUTER JOIN**

LEFT OUTER JOIN ou simplesmente LEFT JOIN - Somente os registros da tabela da esquerda (left) serão retornados, tendo ou não registros relacionados na tabela da direita.

Primeiro, é realizada uma junção interna. Depois, para cada linha de T1 que não satisfaz a condição de junção com nenhuma linha de T2, é adicionada uma linha juntada com valores nulos nas colunas de T2. Portanto, a tabela juntada possui, incondicionalmente, no mínimo uma linha para cada linha de T1.

A tabela à esquerda do operador de junção exibirá cada um dos seus registros, enquanto que a da direita exibirá somente seus registros que tenham correspondentes aos da tabela da esquerda.
Para os registros da direita que não tenham correspondentes na esquerda serão colocados valores NULL.

Exemplo (voltar todos somente de pessoal):
```
SELECT p.siape, p.nome, l.lotacao
FROM pessoal p LEFT JOIN lotacoes l
ON p.siape = l.siape ORDER BY p.siape;
```

Veja que pessoal fica à esquerda em “FROM pessoal p LEFT JOIN lotacoes l”.

RIGHT OUTER JOIN
Inverso do LEFT, este retorna todos os registros somente da tabela da direita (right).
Primeiro, é realizada uma junção interna. Depois, para cada linha de T2 que não satisfaz a condição de junção com nenhuma linha de T1, é adicionada uma linha juntada com valores nulos nas colunas de T1. É o oposto da junção esquerda: a tabela resultante possui, incondicionalmente, uma linha para cada linha de T2.

Exemplo (retornar somente os registros de lotacoes):

```
SELECT p.siape, p.nome, l.lotacao
FROM pessoal p RIGHT JOIN lotacoes l
ON p.siape = l.siape ORDER BY p.nome;
```

FULL OUTER JOIN
Primeiro, é realizada uma junção interna. Depois, para cada linha de T1 que não satisfaz a condição de junção com nenhuma linha de T2, é adicionada uma linha juntada com valores nulos nas colunas de T2. Também, para cada linha de T2 que não satisfaz a condição de junção com nenhuma linha de T1, é adicionada uma linha juntada com valores nulos nas colunas de T1.

E também as:
CROSS JOIN e SELF JOIN (para si mesmo).

Vide item 7.2.1.1 do manualoficial para mais detalhes e exemplos.

LIMIT
LIMIT (limite) juntamente com OFFSET (deslocamento) permite dizer quantas linhas desejamos retornar da consulta. Podemos retornar desde apenas uma até todas.

Sintaxe:
SELECT *lista_de_campos*
FROM *expressão*
[LIMIT { *número* | ALL }] [OFFSET *inicio*]

LIMIT ALL – mesmo que imitir LIMIT.
OFFSET inicio – orienta para que a consulta retorne somente a partir de inicio.
OFFSET 0 – mesmo que omitir OFFSET.
LIMIT 50 OFFSET 11 – Deverá trazer 50 registros do 11 até o 60, caso existam.

Obs.: Quando se utiliza LIMIT é importante utilizar a cláusula ORDER BY para estabelecer uma ordem única para as linhas do resultado. Caso contrário, será retornado um subconjunto imprevisível de linhas da consulta; pode-se desejar obter da décima a vigésima linha, mas da décima a vigésima de qual ordem? A ordem é desconhecida a não ser que seja especificado ORDER BY. Isto é uma conseqüência inerente ao fato do SQL não prometer retornar os resultados de uma consulta em qualquer ordem específica, a não ser que ORDER BY seja utilizado para impor esta ordem.

Exemplos:
SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET 1;
Irá retornar os registros do 1 até o 20.

SELECT * FROM news_m LIMIT $inicio, $n_resultados

O comando "SELECT * FROM news_m LIMIT $n_resultados OFFSET $inicio"
irá pesquisar as noticias da tabela "news_m" começando do resultado "$inicio" e irá listar "$n_resultados".

Exemplo: "SELECT * FROM news_m LIMIT 3 OFFSET 2" irá exibir 3 notícias a partir da 2a. notícia da tabela, ou seja, irá exibir as notícias 2, 3 e 4 da nossa tabela "news_m".

4.3 – Sub consultas
São consultas dentro de consultas.

Subconsulta escalar é um comando SELECT comum, entre parênteses, que retorna exatamente um registro, com um campo.

```
select nome, (select max(preco) from produtos where codigo=1) as "maior preço" from
produtos;

SELECT * FROM tabela1 WHERE tabela1.col1 =
(SELECT col2 FROM tabela2 WHERE col2 = valor);
SELECT name FROM customer WHERE customer_id NOT IN ( SELECT customer_id FROM
salesorder );
SELECT 'test' AS test, id FROM (SELECT * FROM books) AS example_sub_query;
SELECT firstname, state,
CASE
 WHEN state = 'PA' THEN 'close'
 WHEN state = 'NJ' OR state = 'MD' THEN 'far'
 ELSE 'very far'
END AS distance
FROM friend;
```

Expressões de Sub Consultas

EXISTS

```
SELECT campo1 FROM tabela1 WHERE EXISTS
  (SELECT 1 FROM tabela2 WHERE campo2 = tabela1.campo2);
```

Combinando CASE e EXISTS

```
CREATE TEMPORARY TABLE frutas (id SERIAL PRIMARY KEY, nome TEXT);
INSERT INTO frutas VALUES (DEFAULT, 'banana');
INSERT INTO frutas VALUES (DEFAULT, 'maçã');

CREATE TEMPORARY TABLE alimentos (id SERIAL PRIMARY KEY, nome TEXT);
INSERT INTO alimentos VALUES (DEFAULT, 'maçã');
INSERT INTO alimentos VALUES (DEFAULT, 'espinafre');

SELECT nome, CASE WHEN EXISTS (SELECT nome FROM frutas WHERE nome=a.nome)
             THEN 'sim'
             ELSE 'não'
         END AS fruta
FROM alimentos a;
```

IN

```
SELECT nome, CASE WHEN nome IN (SELECT nome FROM frutas)
             THEN 'sim'
             ELSE 'não'
         END AS fruta
FROM alimentos;
```

NOT IN
ANY/SOME

```
SELECT nome, CASE WHEN nome = ANY (SELECT nome FROM frutas)
             THEN 'sim'
```

```
            ELSE 'não'
         END AS fruta
FROM alimentos;
```

CASE WHEN

```
CREATE TABLE notas (
   nota decimal(4,2) CONSTRAINT chknota
             CHECK (nota BETWEEN 0.00 AND 10.00)
);

INSERT INTO notas VALUES(10);
INSERT INTO notas VALUES(9.2);
INSERT INTO notas VALUES(9.0);
INSERT INTO notas VALUES(8.3);
INSERT INTO notas VALUES(7.7);
INSERT INTO notas VALUES(7.4);
INSERT INTO notas VALUES(6.4);
INSERT INTO notas VALUES(5.8);
INSERT INTO notas VALUES(5.1);
INSERT INTO notas VALUES(5.0);
INSERT INTO notas VALUES(0);
SELECT CASE
        WHEN nota < 3 THEN 'E'  -- 0 a 3
        WHEN nota < 5 THEN 'D'          -- 3 a 5
        WHEN nota < 7 THEN 'C'          -- 5 a 7
        WHEN nota < 9 THEN 'B'          -- 7 a 9
        ELSE 'A'                        -- 9 a 10
      END AS conceito,
      COUNT(*) AS quantidade,
      MIN(nota) AS menor,
      MAX(nota) AS maior,
      ROUND(AVG(nota)) AS media
FROM notas
GROUP BY CASE
         WHEN nota < 3 THEN 'E' -- Aqui podemos utilizar expressões
         WHEN nota < 5 THEN 'D'
         WHEN nota < 7 THEN 'C'
         WHEN nota < 9 THEN 'B'
         ELSE 'A'
       END
ORDER BY conceito;
```

Mostrando os Dados em uma única linha:

```
SELECT COUNT(CASE WHEN nota BETWEEN 9.00 AND 10.00 THEN 1 ELSE NULL END)
AS A,
      COUNT(CASE WHEN nota BETWEEN 7.00 AND  8.99 THEN 1 ELSE NULL END) AS B,
      COUNT(CASE WHEN nota BETWEEN 5.00 AND  6.99 THEN 1 ELSE NULL END) AS C,
      COUNT(CASE WHEN nota BETWEEN 3.00 AND  4.99 THEN 1 ELSE NULL END) AS D,
      COUNT(CASE WHEN nota BETWEEN 0.00 AND  2.99 THEN 1 ELSE NULL END) AS E
FROM notas;
```

Mostrar cada nota junto com a menor nota, a maior nota, e a média de todas as notas.

```
SELECT nota,
      (SELECT MIN(nota) FROM notas) AS menor,
      (SELECT MAX(nota) FROM notas) AS maior,
      (ROUND(SELECT AVG(nota) FROM notas)) AS media
FROM notas;
```

5 - Funções Internas

5.1 – Funções de Strings

Concatenação de Strings **- dois || (pipes)**
SELECT 'ae' || 'io' || 'u' AS vogais;   --vogais -------- aeiou
SELECT CHR(67)||CHR(65)||CHR(84) AS "Dog"; -- Dog CAT

Quantidade de Caracteres de String
char_length - retorna o número de caracteres
SELECT CHAR_LENGTH('UNIFOR'); - -Retorna 6

Ou SELECT LENGTH('Database'); - - Retorna 8

Converter para minúsculas
SELECT LOWER('UNIFOR');

Converter para maiúsculas
SELECT UPPER('universidade');

Posição de caractere
SELECT POSITION ('@' IN 'ribafs@gmail.com'); -- Retorna 7
Ou SELECT STRPOS('Ribamar' ,'mar'); - - Retorna 5

Substring
SUBSTRING(string [FROM inteiro] [FOR inteiro])
SELECT SUBSTRING ('Ribamar FS' FROM 9 FOR 10); - - Retorna FS
SUBSTRING(string FROM padrão);
SELECT SUBSTRING ('PostgreSQL' FROM '.......'); - - Retorna Postgre
SELECT SUBSTRING ('PostgreSQL' FROM '...$'); - -Retorna SQL

Primeiros .... e últimos ...$

Ou
SUBSTR ('string', inicio, quantidade);
SELECT SUBSTR ('Ribamar', 4, 3); - - Retorna mar

Substituir todos os caracteres semelhantes
SELECT TRANSLATE(string, velho, novo);
SELECT TRANSLATE('Brasil', 'il', 'ão'); - - Retorna Brasão
SELECT TRANSLATE('Brasileiro...leiro', 'eiro', 'eira');

Remover Espaços de Strings
SELECT TRIM('   SQL - PADRÃO   ');

Calcular MD5 de String
SELECT MD5('ribafs'); - - Retorna 53cd5b2af18063bea8ddc804b21341d1

Repetir uma string n vezes
SELECT REPEAT('SQL-', 3); - - Retorna SQL-SQL-SQL-

Sobrescrever substring em string
SELECT REPLACE ('Postgresql', 'sql', 'SQL'); - - Retorna PostgreSQL

Dividir Cadeia de Caracteres com Delimitador
SELECT SPLIT_PART('PostgreSQL', 'gre', 2); - -Retorna SQL
SELECT SPLIT_PART('PostgreSQL', 'gre', 1); - -Retorna Post

Iniciais Maiúsculas

INITCAP(text) - INITCAP ('olá mundo') - - Olá Mundo

Remover Espaços em Branco
TRIM ([leading | trailing | both] [characters] from string)- remove caracteres da direita e da esquerda. trim (both 'b' from 'babacatebbbb'); - - abacate
RTRIM (string text, chars text) - Remove os caracteres chars da direita (default é espaço)
rtrim('removarrrr', 'r') - - remova
LTRIM - (string text, chars text) - Remove os caracteres chars da esquerda
ltrim('absssremova', 'abs') - - remova

Detalhes no item 9.4 do Manual:
http://pgdocptbr.sourceforge.net/pg80/functions-string.html

Like e %
SELECT * FROM FRIENDS WHERE LASTNAME LIKE 'M%';

O ILIKE é case INsensitive e o LIKE case sensitive.

~~ equivale ao LIKE
~~* equivale equivale ao ILIKE
!~~ equivale ao NOT LIKE
!~~* equivale equivale ao NOT ILIKE

```
... LIKE '[4-6]_6%' -- Pegar o primeiro sendo de 4 a 6,
                    -- o segundo qualquer dígito,
                    -- o terceiro sendo 6 e os demais quaisquer
```

% similar a *
_ similar a ? (de arquivos no DOS)

Correspondência com um Padrão
O PostgreSQL disponibiliza três abordagens distintas para correspondência com padrão: o operador LIKE tradicional do SQL; o operador mais recente SIMILAR TO (adicionado ao SQL:1999); e as expressões regulares no estilo POSIX. Além disso, também está disponível a função de correspondência com padrão substring, que utiliza expressões regulares tanto no estilo SIMILAR TO quanto no estilo POSIX.

SELECT substring('XY1234Z', 'Y*([0-9]{1,3})'); - - Resultado: 123
SELECT substring('XY1234Z', 'Y*?([0-9]{1,3})'); - - Resultado: 1

SIMILAR TO

O operador SIMILAR TO retorna verdade ou falso conforme o padrão corresponda ou não à cadeia de caracteres fornecida. Este operador é muito semelhante ao LIKE, exceto por interpretar o padrão utilizando a definição de expressão regular do padrão SQL.

'abc' SIMILAR TO 'abc'   verdade
'abc' SIMILAR TO 'a'   falso
'abc' SIMILAR TO '%(b|d)%' verdade
'abc' SIMILAR TO '(b|c)%'   falso

SELECT 'abc' SIMILAR TO '%(b|d)%';   -- Procura b ou d em 'abc' e no caso retorna TRUE
REGEXP

SELECT 'abc' ~ '.*ab.*';

~ distingue a de A
~* não distingue a de A
!~ distingue expressões distingue a de A
!~* distingue expressões não distingue a de A
'abc' ~ 'abc' -- TRUE
'abc' ~ '^a' -- TRUE
'abc' ~ '(b|j)' -- TRUE
'abc' ~ '^(b|c)' -- FALSE

5.2 – Funções Matemáticas

Operadores Lógicos:
AND, OR e NOT. TRUE, FALSE e NULL

Operadores de Comparação:
<, >, <=, >=, =, <> ou !=
a BETWEEN x AND y
a NOT BETWEEN x AND y
expressão IS NULL
expressão IS NOT NULL
expressão IS TRUE
expressão IS NOT TRUE
expressão IS FALSE
expressão IS NOT FALSE
expressão IS UNKNOWN
expressão IS NOT UNKNOWN

OPERADOR NULL

Em SQL NULL é para valores inexistentes. Regra geral: NULL se propaga, o que significa que com quem NULL se combina o resultado será um NULL.
NULL não zero, não é string vazia nem string de comprimento zero.
Um exemplo: num cadastro de alunos, para o aluno que ainda não se conhece a nota, não é correto usar zero para sua nota, mas sim NULL.
Não se pode efetuar cálculos de expressões onde um dos elementos é NULL.

COMPARANDO NULLs
NOT NULL com NULL -- Unknown
NULL com NULL -- Unknown

CONVERSÃO DE/PARA NULL
NULLIF() e COALESCE()

NULLIF(valor1, valor2)

NULLIF – Retorna NULL se, e somente se, valor1 e valor2 forem iguais, caso contrário retorna valor1.

Algo como:
if (valor1 == valor2){
then NULL
else valor1;

Retorna valor1 somente quando valor1 == valor2.

COALESCE – retorna o primeiro de seus argumentos que não for NULL. Só retorna NULL quando todos os seus argumentos forem NULL.

Uso: mudar o valor padrão cujo valor seja NULL.

create table nulos(nulo int, nulo2 int, nulo3 int);

insert into nulos values (1,null,null);

select coalesce(nulo, nulo2, nulo3) from nulos; - - Retorna 1, valor do campo nulo;

select coalesce(nulo2, nulo3) from nulos; - - Retorna NULL, pois ambos são NULL.

GREATEST - Retorna o maior valor de uma lista - SELECT GREATEST(1,4,6,8,2); - - 8

LEAST - Retorna o menor valor de uma lista.

Todos os valores da lista devem ser do mesmo tipo e nulos são ignorados.
Obs.: Ambas as funções acima não pertencem ao SQL standard, mas são uma extensão do PostgreSQL.

CONCATENANDO NULLs
A regra é: NULL se propaga. Qualquer que concatene com NULL gerará NULL.

STRING || NULL -- NULL

Usos:
- Como valor default para campos que futuramente receberão valor.
- Valor default para campos que poderão ser sempre inexistentes.

Operadores Matemáticos
+, -, *, /, % (módulo, resto de divisão de inteiros), ^(potência), !(fatorial), @(valor absoluto)
| / - rais quadrada (| / 25.0 = 5)
| | / - raiz cúbica (| | / 27.0 = 3)

Algumas funções Matemáticas
ABS(x) - valor absoluto de x
CEIL(numeric) - arredonda para o próximo inteiro superior
DEGREES(valor) - converte valor de radianos para graus
FLOOR(numeric) - arredonda para o próximo inteiro inferior
MOD(x,y) - resto da divisão de x por y
PI() - constante PI (3,1415...)
POWER(x,y) - x elevado a y
RADIANS(valor) - converte valor de graus para radianos
RANDOM() - valor aleatório entre 0 e 1
ROUND(numeric) - arredonda para o inteiro mais próximo
ROUND(v, d) - arredonda v com d casas decimais
SIGN(numeric) - retorna o sinal da entrada, como -1 ou +1
SQRT(X) - Raiz quadrada de X
TRUNC (numeric) - trunca para o nenhuma casa decimal
TRUNC (v numeric, s int) - trunca para s casas decimais

5.3 – Funções de Agrupamento (Agregação)
As funções de agrupamento são usadas para contar o número de registros de uma tabela.

avg(expressão)
count(*)
count(expressão)
max(expressão)
min(expressão)
stddev(expressão)
sum(expressão)
variance(expressão)

Onde expressão, pode ser "ALL expressão" ou “DISTINCT expressão”.

As funções de Agrupamento (agregação) não podem ser utilizadas na cláusula WHERE. Devem ser utilizadas entre o SELECT e o FROM. Num SELECT que usa uma função agregada, as demais colunas devem fazer parte da cláusula GROUP BY. Somente podem aparecer após o SELECT ou na cláusula HAVING. De uso proibido nas demais cláusulas.

Obs.: Ao contar os registros de uma tabela com a função COUNT(campo) e esse campo for nulo em alguns registros, estes registros não serão computados, por isso cuidado com os nulos nas funções de agregação.

A cláusula HAVING normalmente vem precedida de uma cláusula GROUP BY e obrigatoriamente contém funções de agregação.

ALERTA: Retornam somente os registros onde o campo pesquisado seja diferente de NULL.
NaN - Not a Number (Não é um número)

UPDATE tabela SET campo1 = 'NaN';

SELECT MIN(campo) AS "Valor Mínimo" FROM tabela;

Caso tenha problema com esta consulta use:
SELECT campo FROM tabela ORDER BY campo ASC LIMIT 1;

SELECT MAN(campo) AS "Valor Máximo" FROM tabela;

Caso tenha problema com esta consulta use:
SELECT campo FROM tabela ORDER BY campo DESC LIMIT 1;

5.4 – Funções de Data/Hora

Operações com datas:
timestamp '2001-09-28 01:00' + interval '23 hours' -> timestamp '2001-09-29 00:00'
date '2001-09-28' + interval '1 hour' -> timestamp '2001-09-28 01:00'
date '01/01/2006' – date '31/01/2006'
time '01:00' + interval '3 hours'time -> '04:00'
interval '2 hours' - time '05:00' -> time '03:00:00'

Função age (retorna Interval) - Diferença entre datas
age(timestamp)interval (Subtrai de hoje)
age(timestamp '1957-06-13') -> 43 years 8 mons 3 days
age(timestamp, timestamp)interval Subtrai os argumentos
age('2001-04-10', timestamp '1957-06-13') -> 43 years 9 mons 27 days

Função extract (retorna double)
Extrai parte da data: ano, mês, dia, hora, minuto, segundo.
select extract(year from age('2001-04-10', timestamp '1957-06-13'))
select extract(month from age('2001-04-10', timestamp '1957-06-13'))
select extract(day from age('2001-04-10', timestamp '1957-06-13'))

Data e Hora atuais (retornam data ou hora)
SELECT CURRENT_DATE;
SELECT CURRENT_TIME;
SELECT CURRENT_TIME(0);
SELECT CURRENT_TIMESTAMP;
SELECT CURRENT_TIMESTAMP(0);

Obtendo o dia do mês:
SELECT DATE_PART('DAY', CURRENT_TIMESTAMP) AS Dia;

Somar dias e horas a uma data:
SELECT CAST('06/04/2006' AS DATE) + INTERVAL '27 DAYS' AS Data;

Função now (retorna timestamp with zone)
now() - Data e hora corrente (timestamp with zone);
Não usar em campos somente timestamp.

Função date_part (retorna double)
SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
Resultado: 16 (day é uma string, diferente de extract)

Função date_trunc (retorna timestamp)
SELECT date_trunc('year', TIMESTAMP '2001-02-16 20:38:40');
Retorna 2001-02-16 00:00:00

Convertendo (CAST)
select to_date('1983-07-18', 'YYYY-MM-DD')
select to_date('19830718', 'YYYYMMDD')

Função timeofday (retorna texto)
select timeofday() -> Fri Feb 24 10:07:32.000126 2006 BRT

Interval
interval [(*p*)]
to_char(interval '15h 2m 12s', 'HH24:MI:SS')
date '2001-09-28' + interval '1 hour'
interval '1 day' + interval '1 hour'
interval '1 day' - interval '1 hour'
900 * interval '1 second'
Interval trabalha com as unidades: second, minute, hour, day, week, month, year, decade, century, millenium ou abreviaturas ou plurais destas unidades.
Se informado sem unidades '13 10:38:14' será devidamente interpretado '13 days 10 hours 38 minutes 14 seconds'.

CURRENTE_DATE - INTERVAL '1' day;
TO_TIMESTAMP('2006-01-05 17:56:03', 'YYYY-MM-DD HH24:MI:SS')

Tipos Geométricos:
CREATE TABLE geometricos(ponto POINT, segmento LSEG, retangulo BOX, poligono POLYGON, circulo CIRCLE);

ponto (0,0),
segmento de (0,0) até (0,1),
retângulo (base inferior (0,0) até (1,0) e base superior (0,1) até (1,1)) e
círculo com centro em (1,1) e raio 1.
INSERT INTO geometricos VALUES ('(0,0)','((0,0),(0,1))', '((0,0),(0,1))', '((0,0),(0,1),(1,1),(1,0))','((1,1),1)');

Tipos de Dados para Rede:
Para tratar especificamente de redes o PostgreSQL tem os tipos de dados cidr, inet e macaddr.

cidr – para redes IPV4 e IPV6
inet – para redes e hosts IPV4 e IPV6
macaddr – endereços MAC de placas de rede
Assim como tipos data, tipos de rede devem ser preferidos ao invés de usar tipos texto para guardar IPs, Máscaras ou endereços MAC.
Veja um exemplo em Índices Parciais e a documentação oficial para mais detalhes.

5.5 - Formatação de Tipos de Dados

TO_CHAR - Esta função deve ser evitada, pois será descontinuada.

TO_DATE
date TO_DATE(text, text); Recebe dois parâmetros text e retorna date.
Um dos parâmetros é a data e o outro o formato.

SELECT TO_DATE('29032006','DDMMYYYY'); - Retorna 2006-03-29

TO_TIMESTAMP
tmt TO_TIMESTAMP(text,text) - Recebe dois text e retorna timestamp with zone

SELECT TO_TIMESTAMP('29032006 14:23:05','DDMMYYYY HH:MI:SS'); - Retorna 2006-03-29 14:23:05+00

TO_NUMBER
numeric TO_NUMBER(text,text)
SELECT TO_NUMBER('12,454.8-', '99G999D9S'); Retorna -12454.8
SELECT TO_NUMBER('12,454.8-', '99G999D9'); Retorna 12454.8
SELECT TO_NUMBER('12,454.8-', '99999D9'); Retorna 12454

Detalhes no item 9.8 do manual.

5.6 - Conversão Explícita de Tipos (CAST)
CAST (expressão AS tipo) AS apelido; -- Sintaxe SQL ANSI

Outra forma:

Tipo (expressão);

Exemplo:

SELECT DATE '10/05/2002' - DATE '10/05/2001'; -- Retorna a quantidade de dias
- -entre as duas datas

Para este tipo de conversão devemos:
Usar float8 ao invés de double precision;
Usar entre aspas alguns tipos como interval, time e timestamp

Obs.: aplicações portáveis devem evitar esta forma de conversão e em seu lugar usar o CAST explicitamente.

A função CAST() é utilizada para converter explicitamente tipos de dados em outros.
SELECT CAST(2 AS double precision) ^ CAST(3 AS double precision) AS "exp";
SELECT ~ CAST('20' AS int8) AS "negativo"; - Retorna -21

```
SELECT round(CAST (4 AS numeric), 4); - Retorna 4.0000
SELECT substr(CAST (1234 AS text), 3);
SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
```

Funções Diversas

```
SELECT CURRENT_DATABASE();
SELECT CURRENT_SCHEMA();
SELECT CURRENT_SCHEMA(boolean);
SELECT CURRENT_USER;
SELECT SESSION_USER;
SELECT VERSION();

SELECT CURRENT_SETTING('DATESTYLE');
SELECT HAS_TABLE_PRIVILEGE('usuario','tabela','privilegio');
SELECT HAS_TABLE_PRIVILEGE('postgres','nulos','insert'); - - Retorna: t
SELECT HAS_DATABASE_PRIVILEGE('postgres','testes','create'); - - Retorna: t
SELECT HAS_SCHEMA_PRIVILEGE('postgres','public','create'); - - Retorna: t

SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

Arrays

```
SELECT ARRAY[1.1,2.2,3.3]::INT[] = ARRAY[1,2,3];
SELECT ARRAY[1,2,3] = ARRAY[1,2,8];
SELECT ARRAY[1,3,5] || ARRAY[2,4,6];
SELECT 0 || ARRAY[2,4,6];
```

Array de char com 48 posições e cada uma com 2:
campo char(2) [48]

Funções Geométricos

```
area(objeto) - - area(box '((0,0), (1,1))');
center(objeto) - - center(box '((0,0), (1,2))');
diameter(circulo double) - - diameter(circle '((0,0), 2.0)');
height(box) - - height(box '((0,0), (1,1))');
length(objeto) - - length(path '((-1,0), (1,0))');
radius(circle) - - radius(circle '((0,0), 2.0)');
width(box) - - width(box '((0,0), (1,1))');
```

Funções para Redes

Funções cidr e inet

host(inet) - - host('192.168.1.5/24') - - 192.168.1.5
masklen(inet) - - masklen('192.168.1.5/24') - - 24
netmask(inet) - - netmask('192.168.1.5/24') - - 255.255.255.0
network(inet) - - network('192.168.1.5/24') - - 192.168.1.0/24

Função macaddr

trunt(macaddr) - - trunc(maraddr '12:34:34:56:78:90:ab') - - 12:34:56:00:00:00

Funções de Informação do Sistema

current_database()
current_schema()
current_schemas(boolean)
current_user()
inet_client_addr()
inet_client_port()
inet_server_addr()
inet_server_port()
pg_postmaster_start_time()
version()
has_table_privilege(user, table, privilege) - dá privilégio ao user na tabela
has_table_privilege(table, privilege) - dá privilégio ao usuário atual na tabela
has_database_privilege(user, database, privilege) - dá privilégio ao user no banco
has_function_privilege(user, function, privilege) - dá privilégio ao user na função
has_language_privilege(user, language, privilege) - dá privilégio ao user na linguagem
has_schema_privilege(user, schema, privilege) - dá privilégio ao user no esquema
has_tablespace_privilege(user, tablespace, privilege) - dá privilégio ao user no tablespace

current_setting(nome) - valor atual da configuração
set_config(nome, novovalor, is_local) - seta parâmetro de retorna novo valor

pg_start_backup(label text)
pg_stop_backup()

pg_column_size(qualquer)
pg_tablespace_size(nome)
pg_database_size(nome)
pg_relation_size(nome)
pg_total_relation_size(nome)
pg_size_pretty(bigint)

pg_ls_dir(diretorio)
pg_read_file(arquivo text, offset bigint, tamanho bigint)
pg_stat_file(arquivo text)

6 - Funções Definidas pelo Usuário e Triggers

O PostgreSQL oferece quatro tipos de funções:
- Funções escritas em SQL
- Funções em linguagens de procedimento (PL/pgSQL, PL/Tcl, PL/php, PL/Java, etc)
- Funções internas (rount(), now(), max(), count(), etc).
- Funções na linguagem C

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [, ...] ] )
    [ RETURNS rettype ]
  { LANGUAGE langname
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Para reforçar a segurança é interessante usar o parâmetro SECURITY DEFINER, que especifica que a função será executada com os privilégios do usuário que a criou.

SECURITY INVOKER indica que a função deve ser executada com os privilégios do usuário que a chamou (padrão).

SECURITY DEFINER especifica que a função deve ser executada com os privilégios do usuário que a criou.

Uma grande força do PostgreSQL é que ele permite a criação de funções pelo usuário em diversas linguagens: SQL, PlpgSQL, TCL, Perl, Phyton, Ruby.

Para ter exemplos a disposição vamos instalar os do diretório "tutorial" dos fontes do PostgreSQL:

Acessar /usr/local/src/postgresql-8.1.3/src/tutorial e executar:

make install

Feito isso teremos 5 arquivos .sql.

O syscat.sql traz consultas sobre o catálogo de sistema, o que se chama de metadados (metadata).
O basic.sql e o advanced.sql são consultas SQL.
O complex.sql trata da criação de um tipo de dados pelo usuário e seu uso.
O func.sql traz algumas funções em SQL e outras em C.

6.1 – Funções em SQL

O que outros SGBDs chamam de stored procedures o PostgreSQL chama de funções, que podem ser em diversas linguagens.

```
CREATE OR REPLACE FUNCTION olamundo() RETURNS int4
AS 'SELECT 1' LANGUAGE 'sql';

SELECT olamundo() ;

CREATE OR REPLACE FUNCTION add_numeros(nr1 int4, nr2 int4) RETURNS int4
AS 'SELECT $1 + $2' LANGUAGE 'sql';
SELECT add_numeros(300, 700) AS resposta ;
```

Podemos passar como parâmetro o nome de uma tabela:

```
CREATE TEMP TABLE empregados (
        nome text,
        salario numeric,
        idade integer,
        baia point
);

INSERT INTO empregados VALUES('João',2200,21,point('(1,1)'));
INSERT INTO empregados VALUES('José',4200,30,point('(2,1)'));

CREATE FUNCTION dobrar_salario(empregados) RETURNS numeric AS $$
        SELECT $1.salario * 2 AS salario;
$$ LANGUAGE SQL;

SELECT nome, dobrar_salario(emp.*) AS sonho
        FROM empregados
        WHERE empregados.baia ~= point '(2,1)';
```

Algumas vezes é prático gerar o valor do argumento composto em tempo de execução. Isto pode ser feito através da construção ROW.

```
SELECT nome, dobrar_salario(ROW(nome, salario*1.1, idade, baia)) AS sonho
        FROM empregados;
```

Função que retorna um tipo composto. Função que retorna uma única linha da tabela empregados:

```
CREATE FUNCTION novo_empregado() RETURNS empregados AS $$
        SELECT text 'Nenhum' AS nome,
        1000.0 AS salario,
        25 AS idade,
        point '(2,2)' AS baia;
$$ LANGUAGE SQL;
```

Ou

```
CREATE OR REPLACE FUNCTION novo_empregado() RETURNS empregados AS $$
	SELECT ROW('Nenhum', 1000.0, 25, '(2,2)')::empregados;
$$ LANGUAGE SQL;
```

Chamar assim:

```
SELECT novo_empregado();
```

ou

```
SELECT * FROM novo_empregado();
```

Funções SQL como fontes de tabelas

```
CREATE TEMP TABLE teste (testeid int, testesubid int, testename text);
INSERT INTO teste VALUES (1, 1, 'João');
INSERT INTO teste VALUES (1, 2, 'José');
INSERT INTO teste VALUES (2, 1, 'Maria');

CREATE FUNCTION getteste(int) RETURNS teste AS $$
  SELECT * FROM teste WHERE testeid = $1;
$$ LANGUAGE SQL;

SELECT *, upper(testename) FROM getteste(1) AS t1;
```

Tabelas Temporárias - criar tabelas temporárias (TEMP), faz com que o servidor se encarregue de removê-la (o que faz logo que a conexão seja encerrada).

```
CREATE TEMP TABLE nometabela (campo tipo);
```

Funções SQL retornando conjunto

```
CREATE FUNCTION getteste(int) RETURNS SETOF teste AS $$
	SELECT * FROM teste WHERE testeid = $1;
$$ LANGUAGE SQL;

SELECT * FROM getteste(1) AS t1;
```

Funções SQL polimórficas

As funções SQL podem ser declaradas como recebendo e retornando os tipos polimórficos anyelement e anyarray.

```
CREATE FUNCTION constroi_matriz(anyelement, anyelement) RETURNS anyarray AS $$
	SELECT ARRAY[$1, $2];
$$ LANGUAGE SQL;

SELECT constroi_matriz(1, 2) AS intarray, constroi_matriz('a'::text, 'b') AS textarray;

CREATE FUNCTION eh_maior(anyelement, anyelement) RETURNS boolean AS $$
	SELECT $1 > $2;
$$ LANGUAGE SQL;
SELECT eh_maior(1, 2);
```

Mais detalhes no capítulo 31 do manual.

6.2 - Funções em PlpgSQL

As funções em linguagens procedurais no PostgreSQL, como a PlpgSQL são correspondentes ao que se chama comumente de Stored Procedures.

Por default o PostgreSQL só traz suporte às funções na linguagem SQL. Para dar suporte à funções em outras linguagens temos que efetuar procedimentos como a seguir.
Para que o banco postgres tenha suporte à linguagem de procedimento PlPgSQL executamos na linha de comando como super usuário do PostgreSQL:

createlang plpgsql –U nomeuser nomebanco
A PlpgSQL é a linguagem de procedimentos armazenados mais utilizada no PostgreSQL, devido ser a mais madura e com mais recursos.

```
CREATE FUNCTION func_escopo() RETURNS integer AS $$
DECLARE
  quantidade integer := 30;
BEGIN
  RAISE NOTICE 'Aqui a quantidade é %', quantidade;  -- A quantidade aqui é 30
  quantidade := 50;
  --
  -- Criar um sub-bloco
  --
  DECLARE
     quantidade integer := 80;
  BEGIN
     RAISE NOTICE 'Aqui a quantidade é %', quantidade;  -- A quantidade aqui é 80
  END;
  RAISE NOTICE 'Aqui a quantidade é %', quantidade;  -- A quantidade aqui é 50
  RETURN quantidade;
END;
$$ LANGUAGE plpgsql;

=> SELECT func_escopo();

CREATE FUNCTION instr(varchar, integer) RETURNS integer AS $$
DECLARE
  v_string ALIAS FOR $1;
  index    ALIAS FOR $2;
BEGIN
  -- algum processamento neste ponto
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION concatenar_campos_selecionados(in_t nome_da_tabela) RETURNS
text AS $$
BEGIN
  RETURN in_t.f1 || in_t.f3 || in_t.f5 || in_t.f7;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION somar_tres_valores(v1 anyelement, v2 anyelement, v3 anyelement)
RETURNS anyelement AS $$
DECLARE
  resultado ALIAS FOR $0;
BEGIN
  resultado := v1 + v2 + v3;
  RETURN resultado;
END;
$$ LANGUAGE plpgsql;

SELECT somar_tres_valores(10,20,30);
```

Utilização de tipo composto:

```
CREATE FUNCTION mesclar_campos(t_linha nome_da_tabela) RETURNS text AS $$
DECLARE
  t2_linha nome_tabela2%ROWTYPE;
BEGIN
  SELECT * INTO t2_linha FROM nome_tabela2 WHERE ... ;
  RETURN t_linha.f1 || t2_linha.f3 || t_linha.f5 || t2_linha.f7;
END;
$$ LANGUAGE plpgsql;

SELECT mesclar_campos(t.*) FROM nome_da_tabela t WHERE ... ;
```

Temos uma tabela (datas) com dois campos (data e hora) e queremos usar uma função para manipular os dados desta tabela:

```
CREATE or REPLACE FUNCTION data_ctl(opcao char, fdata date, fhora time) RETURNS
char(10) AS '
DECLARE
   opcao ALIAS FOR $1;
   vdata ALIAS FOR $2;
   vhora ALIAS FOR $3;
   retorno char(10);
BEGIN
   IF opcao = ''I'' THEN
        insert into datas (data, hora) values (vdata, vhora);
      retorno := ''INSERT'';
   END IF;
   IF opcao = ''U'' THEN
        update datas set data = vdata, hora = vhora where data=''1995-11-01'';
      retorno := ''UPDATE'';
   END IF;
   IF opcao = ''D'' THEN
        delete from datas where data = vdata;
      retorno := ''DELETE'';
   ELSE
      retorno := ''NENHUMA'';
   END IF;
   RETURN retorno;
END;
' LANGUAGE plpgsql;
//select data_ctl('I','1996-11-01', '08:15');
```

```
select data_ctl('U','1997-11-01','06:36');
select data_ctl('U','1997-11-01','06:36');
```

Mais Detalhes no capítulo 35 do manual oficial.

Funções que Retornam Conjuntos de Registros (SETS)

```
CREATE OR REPLACE FUNCTION codigo_empregado (codigo INTEGER)
  RETURNS SETOF INTEGER AS '
  DECLARE
        registro RECORD;
        retval INTEGER;
  BEGIN
        FOR registro IN SELECT * FROM empregados WHERE salario >= $1 LOOP
              RETURN NEXT registro.departamento_cod;
        END LOOP;
        RETURN;
  END;
' language 'plpgsql';

select * from codigo_empregado (0);
select count (*), g from codigo_empregado (5000) g group by g;
```

Funções que retornam Registro

Para criar funções em plpgsql que retornem um registro, antes precisamos criar uma variável composta do tipo ROWTYPE, descrevendo o registro (tupla) de saída da função.

```
CREATE TABLE empregados(
  nome_emp text,
  salario int4,
  codigo int4 NOT NULL,
  departamento_cod int4,
  CONSTRAINT empregados_pkey PRIMARY KEY (codigo),
  CONSTRAINT empregados_departamento_cod_fkey FOREIGN KEY (departamento_cod)
     REFERENCES departamentos (codigo) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION
)

CREATE TABLE departamentos (codigo INT primary key,  nome varchar);

CREATE TYPE dept_media AS (minsal INT, maxsal INT, medsal INT);

create or replace function media_dept() returns dept_media as
'
declare
        r dept_media%rowtype;
        dept record;
        bucket int8;
        counter int;
  begin
        bucket := 0;
        counter := 0;
        r.maxsal :=0;
        r.minsal :=0;
```

```
        for dept in select sum(salario) as salario, d.codigo as departamento
                from empregados e, departamentos d where e.departamento_cod = d.codigo
                group by departamento loop
          counter := counter + 1;
          bucket := bucket + dept.salario;
          if r.maxsal <= dept.salario or r.maxsal = 0 then
                r.maxsal := dept.salario;
          end if;
          if r.minsal <= dept.salario or r.minsal = 0 then
                r.minsal := dept.salario;
          end if;
        end loop;

        r.medsal := bucket/counter;

        return r;
  end
  ' language 'plpgsql';
```

Funções que Retornam Conjunto de Registros (SETOF, Result Set)
Também requerem a criação de uma variável (tipo definidopelo user)

```
CREATE TYPE media_sal AS
        (deptcod int, minsal int, maxsal int, medsal int);

CREATE OR REPLACE FUNCTION medsal() RETURNS SETOF media_sal AS
'
DECLARE
        s media_sal%ROWTYPE;
        salrec RECORD;
        bucket int;
        counter int;
BEGIN
        bucket :=0;
        counter :=0;
        s.maxsal :=0;
        s.minsal :=0;
        s.deptcod :=0;
        FOR salrec IN SELECT salario AS salario, d.codigo AS departamento
                FROM empregados e, departamentos d WHERE e.departamento_cod =
d.codigo ORDER BY d.codigo LOOP
                IF s.deptcod = 0 THEN
                        s.deptcod := salrec.departamento;
                        s.minsal := salrec.salario;
                        s.maxsal := salrec.salario;
                        counter := counter + 1;
                        bucket := bucket + salrec.salario;
                ELSE
                        IF s.deptcod = salrec.departamento THEN
                                IF s.maxsal <= salrec.salario THEN
                                        s.maxsal := salrec.salario;
                                END IF;
                                IF s.minsal >= salrec.salario THEN
```

```
                    s.minsal := salrec.salario;
                END IF;
                counter := counter +1;
            ELSE
                s.medsal := bucket/counter;
                RETURN NEXT s;
                s.deptcod := salrec.departamento;
                s.minsal := salrec.salario;
                s.maxsal := salrec.salario;
                counter := 1;
                bucket := salrec.salario;
            END IF;
        END IF;
    END LOOP;
    s.medsal := bucket/counter;
    RETURN NEXT s;
    RETURN;
END '
LANGUAGE 'plpgsql';

select * from medsal()
```

Relacionando:

```
select d.nome, a.minsal, a.maxsal, a.medsal
from medsal() a, departamentos d
where d.codigo = a.deptcod
```

6.3 - Triggers (Gatilhos)

Capítulo 32 do manual oficial. e:
http://pgdocptbr.sourceforge.net/pg80/sql-createtrigger.html
Até a versão atual não existe como criar funções de gatilho na linguagem SQL.

Uma função de gatilho pode ser criada para executar antes (BEFORE) ou após (AFTER) as consultas INSERT, UPDATE OU DELETE, uma vez para cada registro (linha) modificado ou por instrução SQL. Logo que ocorre um desses eventos do gatilho a função do gatilho é disparada automaticamente para tratar o evento.

A função de gatilho deve ser declarada como uma função que não recebe argumentos e que retorna o tipo TRIGGER.
Após criar a função de gatilho, estabelecemos o gatilho pelo comando CREATE TRIGGER.
Uma função de gatilho pode ser utilizada por vários gatilhos.

As funções de gatilho chamadas por gatilhos-por-instrução devem sempre retornar NULL.

As funções de gatilho chamadas por gatilhos-por-linha podem retornar uma linha da tabela (um valor do tipo HeapTuple) para o executor da chamada, se assim o decidirem.

Sintaxe:

```
CREATE TRIGGER nome { BEFORE | AFTER } { evento [ OR ... ] }
   ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]
   EXECUTE PROCEDURE nome_da_função ( argumentos )
```

O gatilho fica associado à tabela especificada e executa a função especificada nome_da_função quando determinados eventos ocorrerem.

O gatilho pode ser especificado para disparar antes de tentar realizar a operação na linha (antes das restrições serem verificadas e o comando INSERT, UPDATE ou DELETE ser tentado), ou após a operação estar completa (após as restrições serem verificadas e o INSERT, UPDATE ou DELETE ter completado).

evento
Um entre INSERT, UPDATE ou DELETE; especifica o evento que dispara o gatilho. Vários eventos podem ser especificados utilizando OR.

Exemplos:

```
CREATE TABLE empregados(
  codigo int4 NOT NULL,
  nome varchar,
  salario int4,
  departamento_cod int4,
  ultima_data timestamp,
  ultimo_usuario varchar(50),
  CONSTRAINT empregados_pkey PRIMARY KEY (codigo) )

CREATE FUNCTION empregados_gatilho() RETURNS trigger AS $empregados_gatilho$
    BEGIN
        -- Verificar se foi fornecido o nome e o salário do empregado
        IF NEW.nome IS NULL THEN
            RAISE EXCEPTION 'O nome do empregado não pode ser nulo';
        END IF;
        IF NEW.salario IS NULL THEN
            RAISE EXCEPTION '% não pode ter um salário nulo', NEW.nome;
        END IF;

        -- Quem paga para trabalhar?
        IF NEW.salario < 0 THEN
            RAISE EXCEPTION '% não pode ter um salário negativo', NEW.nome;
        END IF;

        -- Registrar quem alterou a folha de pagamento e quando
        NEW.ultima_data := 'now';
        NEW.ultimo_usuario := current_user;
        RETURN NEW;
    END;
$empregados_gatilho$ LANGUAGE plpgsql;

CREATE TRIGGER empregados_gatilho BEFORE INSERT OR UPDATE ON empregados
    FOR EACH ROW EXECUTE PROCEDURE empregados_gatilho();

INSERT INTO empregados (codigo,nome, salario) VALUES (5,'João',1000);
INSERT INTO empregados (codigo,nome, salario) VALUES (6,'José',1500);
INSERT INTO empregados (codigo,nome, salario) VALUES (7,'Maria',2500);

SELECT * FROM empregados;
```

```
INSERT INTO empregados (codigo,nome, salario) VALUES (5,NULL,1000);

NEW – Para INSERT e UPDATE
OLD – Para DELETE

CREATE TABLE empregados (
   nome  varchar NOT NULL,
   salario     integer
);

CREATE TABLE empregados_audit(
   operacao    char(1)   NOT NULL,
   usuario     varchar      NOT NULL,
   data        timestamp NOT NULL,
   nome    varchar      NOT NULL,
   salario     integer
);

CREATE OR REPLACE FUNCTION processa_emp_audit() RETURNS TRIGGER AS
$emp_audit$
   BEGIN
      --
      -- Cria uma linha na tabela emp_audit para refletir a operação
      -- realizada na tabela emp. Utiliza a variável especial TG_OP
      -- para descobrir a operação sendo realizada.
      --
      IF (TG_OP = 'DELETE') THEN
         INSERT INTO emp_audit SELECT 'E', user, now(), OLD.*;
         RETURN OLD;
      ELSIF (TG_OP = 'UPDATE') THEN
         INSERT INTO emp_audit SELECT 'A', user, now(), NEW.*;
         RETURN NEW;
      ELSIF (TG_OP = 'INSERT') THEN
         INSERT INTO emp_audit SELECT 'I', user, now(), NEW.*;
         RETURN NEW;
      END IF;
      RETURN NULL; -- o resultado é ignorado uma vez que este é um gatilho AFTER
   END;
$emp_audit$ language plpgsql;

CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON empregados
   FOR EACH ROW EXECUTE PROCEDURE processa_emp_audit();

INSERT INTO empregados (nome, salario) VALUES ('João',1000);
INSERT INTO empregados (nome, salario) VALUES ('José',1500);
INSERT INTO empregados (nome, salario) VALUES ('Maria',250);
UPDATE empregados SET salario = 2500 WHERE nome = 'Maria';
DELETE FROM empregados WHERE nome = 'João';

SELECT * FROM empregados;

SELECT * FROM empregados_audit;
```

Outro exemplo:

```
CREATE TABLE empregados (
   codigo        serial  PRIMARY KEY,
   nome    varchar    NOT NULL,
   salario    integer
);

CREATE TABLE empregados_audit(
   usuario        varchar      NOT NULL,
   data           timestamp NOT NULL,
   id             integer   NOT NULL,
   coluna         text      NOT NULL,
   valor_antigo   text      NOT NULL,
   valor_novo     text      NOT NULL
);

CREATE OR REPLACE FUNCTION processa_emp_audit() RETURNS TRIGGER AS
$emp_audit$
   BEGIN
      --
      -- Não permitir atualizar a chave primária
      --
      IF (NEW.codigo <> OLD.codigo) THEN
         RAISE EXCEPTION 'Não é permitido atualizar o campo codigo';
      END IF;
      --
      -- Inserir linhas na tabela emp_audit para refletir as alterações
      -- realizada na tabela emp.
      --
      IF (NEW.nome <> OLD.nome) THEN
        INSERT INTO emp_audit SELECT current_user, current_timestamp,
                NEW.id, 'nome', OLD.nome, NEW.nome;
      END IF;
      IF (NEW.salario <> OLD.salario) THEN
        INSERT INTO emp_audit SELECT current_user, current_timestamp,
                NEW.codigo, 'salario', OLD.salario, NEW.salario;
      END IF;
      RETURN NULL; -- o resultado é ignorado uma vez que este é um gatilho AFTER
   END;
$emp_audit$ language plpgsql;

CREATE TRIGGER emp_audit
AFTER UPDATE ON empregados
   FOR EACH ROW EXECUTE PROCEDURE processa_emp_audit();

INSERT INTO empregados (nome, salario) VALUES ('João',1000);
INSERT INTO empregados (nome, salario) VALUES ('José',1500);
INSERT INTO empregados (nome, salario) VALUES ('Maria',2500);
UPDATE empregados SET salario = 2500 WHERE id = 2;
UPDATE empregados SET nome = 'Maria Cecília' WHERE id = 3;
UPDATE empregados SET codigo=100 WHERE codigo=1;
ERRO:  Não é permitido atualizar o campo codigo
```

```
SELECT * FROM empregados;

SELECT * FROM empregados_audit;
```

Crie a mesma função que insira o nome da empresa e o nome do cliente retornando o id de ambos

```
create or replace function empresa_cliente_id(varchar,varchar) returns _int4 as
'
declare
        nempresa alias for $1;
        ncliente alias for $2;
        empresaid integer;
        clienteid integer;
begin
        insert into empresas(nome) values(nempresa);
        insert into clientes(fkempresa,nome)   values (currval (''empresas_id_seq''), ncliente);
        empresaid := currval(''empresas_id_seq'');
        clienteid := currval(''clientes_id_seq'');

        return ''{''|| empresaid ||'',''|| clienteid ||''}'';
end;
'
language 'plpgsql';
```

Crie uma função onde passamos como parâmetro o id do cliente e seja retornado o seu nome

```
create or replace function id_nome_cliente(integer) returns text as
'
declare
        r record;
begin
    select into r * from clientes where id = $1;
    if not found then
      raise exception ''Cliente não existente !'';
    end if;
    return r.nome;
end;
'
language 'plpgsql';
```

Crie uma função que retorne os nome de toda a tabela clientes concatenados em um só campo

```
create or replace function clientes_nomes() returns text as
'
declare
     x text;
     r record;
begin
    x:=''Inicio'';
    for r in select * from clientes order by id loop
       x:= x||'' : ''||r.nome;
```

```
    end loop;
    return x||'' : fim'';
end;
'
language 'plpgsql';
```

7 - DCL (Data Control Language)

7.1 - Usuários, grupos e privilégios

De fora do banco utiliza-se comandos sem espaço: createdb, dropdb, etc.
De dentro do banco (psql) os comandos são formados por duas palavras:
CREATE DATABASE, DROP DATABASE, etc.

De dentro do banco:
CREATE USER é agora um alias para CREATE ROLE, que tem mais recursos.

```
banco=# \h create role
Comando:    CREATE ROLE
Descrição:  define um novo papel (role) do banco de dados
Sintaxe:
CREATE ROLE nome [ [ WITH ] opção [ ... ] ]

onde opção pode ser:

     SUPERUSER | NOSUPERUSER
   | CREATEDB | NOCREATEDB
   | CREATEROLE | NOCREATEROLE
   | CREATEUSER | NOCREATEUSER
   | INHERIT | NOINHERIT
   | LOGIN | NOLOGIN
   | CONNECTION LIMIT limite_con
   | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'senha'
   | VALID UNTIL 'tempo_absoluto'
   | IN ROLE nome_role [, ...]
   | IN GROUP nome_role [, ...]
   | ROLE nome_role [, ...]
   | ADMIN nome_role [, ...]
   | USER nome_role [, ...]
   | SYSID uid
```

Caso não seja fornecido ENCRYPTED ou UNENCRYPTED então será usado o valor do parâmetro *password_encryption* (postgresql.conf).

Criar Usuário
CREATE ROLE nomeusuario;

Nas versões anteriores usava-se o parâmetro "CREATEUSER" para indicar a criação de um superusuário, agora usa-se o parâmetro mais adequado SUPERUSER.

Para poder criar um novo usuário local, com senha, devemos setar antes o pg_hba.conf:

```
local    all    all    127.0.0.1/32    password
```

Comentar as outras entradas para conexão local.
Isso para usuário local (conexão via socket UNIX).

Criamos assim:
CREATE ROLE nomeuser WITH ENCRYPTED PASSWORD '********';

Ao se logar: psql -U nomeuser nomebanco.
CREATE ROLE nomeusuario VALID UNTIL 'data'

Excluindo Usuário
DROP USER nomeusuario;

Como usuário, fora do banco:
Criar Usuário
CREATEROLE nomeusuario;

Excluindo Usuário
DROPUSER nomeusuario;
Detalhe: sem espaços.

Criando Superusuário
CREATE ROLE nomeuser WITH SUPERUSER ENCRYPTED PASSWORD '******';

Alterar Conta de Usuário
ALTER ROLE nomeuser ENCRYPTED PASSWORD '******' CREATEUSER

ALTER ROLE nomeuser VALID UNTIL '12/05/2006';
ALTER ROLE fred VALID UNTIL ’infinity’;
ALTER ROLE miriam CREATEROLE CREATEDB;

Obs.: Lembrando que ALTER ROLE é uma extensão do PostgreSQL.

Listando todos os usuários:
SELECT usename FROM pg_user;
A tabela pg_user é uma tabela de sistema (_pg) que guarda todos os usuários do PostgreSQL.

Também podemos utilizar:
\du no psql

Criando Um Grupo
CREATE GROUP nomedogrupo;

Adicionar/Remover Usuários Em Grupos
ALTER GROUP nomegrupo ADD USER user1, user2,user3 ;
ALTER GROUP nomegrupo DROP USER user1, user2 ;

Excluindo Grupo
DROP GROUP nomegrupo;

Obs.: isso remove o grupo mas não remove os usuários do mesmo.

Listando todos os grupos:
SELECT groname FROM pg_group;

Privilégios

Dando Privilégios A Um Usuário

GRANT UPDATE ON nometabela TO nomeusuario;

Dando Privilégios A Um Grupo Inteiro

GRANT SELECT ON nometabela TO nomegrupo;

Removendo Todos os Privilégios de Todos os Users

REVOKE ALL ON nometabela FROM PUBLIC

Privilégios

O superusuário tem direito a fazer o que bem entender em qualquer banco de dados do SGBD.
O usuário que cria um objeto (banco, tabela, view, etc) é o dono do objeto.
Para que outro usuário tenha acesso ao mesmo deve receber privilégios.
Existem vários privilégios diferentes: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE e USAGE.
Os privilégios aplicáveis a um determinado tipo de objeto variam de acordo com o tipo do objeto (tabela, função, etc.).

O comando para conceder privilégios é o GRANT. O de remover é o REVOKE.

GRANT UPDATE ON contas TO joel;
Dá a joel o privilégio de executar consultas update no objeto contas.

GRANT SELECT ON contas TO GROUP contabilidade;

REVOKE ALL ON contas FROM PUBLIC;

Os privilégios especiais do dono da tabela (ou seja, os direitos de DROP, GRANT, REVOKE, etc.) são sempre inerentes à condição de ser o dono, não podendo ser concedidos ou revogados. Porém, o dono do objeto pode decidir revogar seus próprios privilégios comuns como, por exemplo, tornar a tabela somente para leitura para o próprio, assim como para os outros.

Normalmente, só o dono do objeto (ou um superusuário) pode conceder ou revogar privilégios para um objeto.

-- Criação dos grupos

CREATE GROUP adm;

CREATE USER paulo ENCRYPTED PASSWORD 'paulo' CREATEDB CREATEUSER;

-- Criação dos Usuários do Grupo adm

CREATE USER andre ENCRYPTED PASSWORD 'andre' CREATEDB IN GROUP adm;
CREATE USER michela ENCRYPTED PASSWORD 'michela' CREATEDB IN GROUP adm;

O usuário de sistema (super usuário) deve ser um usuário criado exclusivamente para o PostgreSQL. Nunca devemos torná-lo dono de nenhum executável.

Os nomes de usuários são globais para todo o agrupamento de bancos de dados, ou seja, podemos utilizar um usuário com qualquer dos bancos.

Os privilégios DROP, GRANT, REVOKE, etc pertencem ao dono do objeto não podendo ser concedidos ou revogados. O máximo que um dono pode fazer é abdicar de seus privilégios e com isso ninguém mais teria os mesmos e o objeto seria somente leitura para todos.

Dando Privilégios

GRANT SELECT,UPDATE,INSERT ON nometabela TO nomeusuario;

Retirando Privilégios
REVOKE ALL ON nometabela FROM nomeusuario;

Para garantir, sempre remova todos os privilégios antes de delegar algum.

Mais detalhes:
http://pgdocptbr.sourceforge.net/pg80/user-manag.html
http://pgdocptbr.sourceforge.net/pg80/sql-revoke.html
http://pgdocptbr.sourceforge.net/pg80/sql-grant.html

8 – Transações

Uma transação acontece por completo (todas as operações) ou nada acontece.
Também a transação deve garantir um nível de isolamento das demais transações, de maneira que as demais transações somente enxerguem as operações após a transação concluída.

Caso haja um erro qualquer na transação ou falha no sistema o SGBR irá executar um comando ROLLBACK.

Transações são uma forma de dar suporte às operações concorrentes, garantindo a segurança e integridade das informações.
Garantir que duas solicitações diferentes não efetuarão uma mesma operação ao mesmo tempo.

Ao consultar o banco de dados, uma transação enxerga um snapshot (instantâneo) dos dados, como estes eram no exato momento em que a consulta foi solicitada, desprezando as mudanças ocorridas depois disso.

O PostgreSQL trata a execução de qualquer comando SQL como sendo executado dentro de uma transação.

Na versão 8 apareceram os SAVEPOINTS (pontos de salvamento) , que guardam as informações até eles. Isso salva as operações existentes antes do SAVEPOINT e basta um ROLLBACK TO para continuar com as demais operações.

O PostgreSQL mantém a consistência dos dados utilizando o modelo multiversão MVCC (Multiversion Concurrency Control), que permite que leitura não bloqueie escrita nem escrita bloqueie leitura.

O PostgreSQL também conta com um nível de isolamento chamado *serializable* (serializável), que é mais rigoroso e emula execução serial das transações.

```
BEGIN;
UPDATE contas SET saldo = saldo – 100.00 WHERE codigo = 5;
SAVEPOINT meu_ponto_de_salvamento;
UPDATE contas SET saldo = saldo + 100.00 WHERE codigo = 5;
-- ops ... o certo é na conta 6
ROLLBACK TO meu_ponto_de_salvamento;
UPDATE contas SET saldo = saldo + 100.00 WHERE conta = 6;
COMMIT;
```

Exemplos:

```
CREATE TABLE contas(codigo INT2 PRIMARY KEY, nome VARCHAR(40), saldo
NUMERIC());
INSERT INTO contas values (5, 'Ribamar', 500.45);
```

Uma transação é dita um processo atômico, o que significa que ou acontecem todas as suas operações ou então nenhuma será salva.
Vamos iniciar a seguinte transação na tabela acima:

```
BEGIN; -- Iniciar uma transação
UPDATE contas SET saldo = 800.35 WHERE codigo= 5;
```

SELECT nome,saldo FROM contas WHERE codigo = 5;
COMMIT; -- Executar todos os comandos da transação

Agora para testar se de fato todas as operações foram salvas execute:
SELECT nome,saldo FROM contas WHERE codigo = 5;

Vamos a outro teste da atomicidade das transações. Intencionalmente vamos cometer um erro no SELECT (FRON):

BEGIN; -- Iniciar uma transação
UPDATE contas SET saldo = 50.85 WHERE codigo= 5;
SELECT nome,saldo FRON contas WHERE codigo = 5;
COMMIT; -- Executar todos os comandos da transação

Isso causará um erro e o comando ROLLBACK será automaticamente executado, o que garante que nenhuma das operações será realizada.

Então execute a consulta para testar se houve a atualização:

SELECT nome,saldo FRON contas WHERE codigo = 5;

Remover Campo (versões anteriores a 7.3 não contam com esse recurso):

BEGIN;
LOCK TABLE nometabela;
INTO TABLE nomenovo FROM nometabela;
DROP TABLE nometabela;
ALTER TABLE nomenovo RENAME TO nometabela;
COMMIT;

Alterar Tipos de Dados (versões antigas):

BEGIN;
ALTER TABLE tabela ADD COLUMN novocampo novotipodados;
UPDATE tabela SET novocampo = CAST (antigocampo novotipodados);
ALTER TABLE tabela DROP COLUMN antigocampo;
COMMIT;

Transações que não se Concretizam
BEGIN; -- Iniciar uma transação
UPDATE contas SET saldo = 50.85 WHERE codigo= 5;
SELECT nome,saldo FRON contas WHERE codigo = 5;
ROLLBACK; -- Cancelando todos os comandos da transação

BEGIN;
CREATE TABLE teste (id integer, nome text);
INSERT INTO teste VALUES (1, 'Teste1');
INSERT INTO teste VALUES (2, 'Teste2');
DELETE FROM teste;
COMMIT;

BEGIN;

CREATE TABLE teste (id integer, nome text);
INSERT INTO teste VALUES (3, 'Teste3');
INSERT INTO teste VALUES (4, 'Teste4');
DELETE FROM teste;
ROLLBACK;

Detalhes sobre conflitos de bloqueios:
http://www.postgresql.org/docs/current/static/explicit-locking.html

Isolamento de Transações
O nível de isolamento padrão do PostgreSQL é o Read Committed (leitura efetivada). Uma consulta SELECT realizada com este nível perceberá os registros existente no início da consulta. Este é o nível mais flexível.

Existe também o nível serializable, mais rigoroso. Os níveis Read uncommitted e Repeatable read são suportados, mas assumem a forma de um dos dois anteriores.

Setando o Nível de Isolamento de uma transação:
banco=# \h set transaction
Comando: SET TRANSACTION
Descrição: define as características da transação atual
Sintaxe:
SET TRANSACTION modo_transação [, ...]
SET SESSION CHARACTERISTICS AS TRANSACTION modo_transação [, ...]

onde modo_transação é um dos:

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
READ WRITE | READ ONLY

Exemplo:
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Aqui as consultas da transação;
...
COMMIT;

Controle de Simultaneidade no Capítulo 12 do manual oficial.

9 - Administração

9.1 - Backup e Restore

Especialmente quem já teve problemas em HDs e não pode recuperar os dados, sabe da importância dos backups.

Para efetuar backup e restore utilizamos o comando pg_dump em conjunto com o psql.

Obs.: O pg_dump não faz backup de objetos grandes (lo) por default. Caso desejemos também estes objetos no backup devemos utilizar uma saída no formato tar e utilizar a opção -b.

pg_dump -Ftb banco > banco.tar

Backup local de um único banco:
pg_dump -U usuario -d banco > banco.sql
pg_dump -Ft banco > banco.tar

O script normalmente leva a extensão .sql, por convenção, mas pode ser qualquer extensão e o script terá conteúdo texto puro.

Restore de um banco local:
psql -U usuario -d banco < banco.sql
pg_restore -d banco banco.sql
pg_restore -d banco banco.tar

Obs.: Cuidado ao restaurar um banco, especialmente se existirem tabelas sem integridade. Corre-se o risco de duplicar os registros.

Descompactar e fazer o restore em um só comando:
gunzip -c backup.tar.gz | pg_restore -d banco

ou

cat backup.tar.gz | gunzip | pg_restore -d banco
(o cat envia um stream do arquivo para o gunzip que passa para o pg_restore)

Backup local de apenas uma tabela de um banco:
pg_dump -U nomeusuario -d nomebanco -t nometabela > nomescript

Restaurar apenas uma tabela
Para conseguir restaurar apenas uma tabela uma forma é gerar o dump do tipo com tar:
pg_dump -Ft banco -f arquivo.sql.tar
pg_restore -d banco -t tabela banco.sql.tar

Backup local de todos os bancos:
pg_dumpall -U nomeusuario -d nomebanco > nomescript

Backup remoto de um banco:
pg_dump -h hostremoto -d nomebanco | psql -h hostlocal -d banco

Backup em multivolumes (volumes de 200MB):
pg_dump nomebanco | split -m 200 nomearquivo

m para 1Mega, k para 1K, b para 512bytes

Importando backup de versão anterior do PostgreSQL

- Instala-se a nova versão com porta diferente (ex.: 5433) e conectar ambos
- pg_dumpall -p 5432 | psql -d template1 -p 5433

Visualizar comando atual e PID de todos os processos do servidor:

```
SELECT pg_stat_get_backend_pid(s.backendid) AS procpid,
pg_stat_get_backend_activity(s.backendid) AS current_query
FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

Determinação da utilização em disco pelas Tabelas

Tendo um banco com cadastro de CEPs e apenas uma tabela "cep_tabela", mostrar o uso do disco por esta tabela. Precisamos filtrar as tabelas de sistema, veja:

VACUUM ANALYZE;

O utilitário VACUUM recupera espaço em disco ocupado pelos registros excluídos e atualizados, atualiza os dados para as estatísticas usadas pelo planejador de consultas e também protege contra perca de dados quando atingir um bilhão de transações.

```
SELECT relname, relfilenode, relpages FROM pg_class WHERE relname LIKE 'cep_%'
ORDER BY relname;
  relname   | relfilenode | relpages
------------+-------------+----------
 cep_pk     |       25140 |     2441
 cep_tabela |       16949 |    27540
```

O daemon do auto-vacuum

Iniciando na versão 8.1 é um processo opcional do servidor, chamado de autovacuum daemon, cujo uso é para automatizar a execução dos comandos VACUUM e ANALYZE.
Roda periodicamente e checa o uso em baixo nível do coletor de estatísticas.
Não pode ser usado enquanto **stats_start_collector** e **stats_row_level** forem alterados para true. Portanto o postgresql.conf deve ficar assim:

```
stats_start_collector = on
stats_row_level = on
autovacuum = on
```

Por default será executado a casa 60 segundos. Para alterar descomente e mude a linha:
#autovacuum_naptime = 60

Determinar o uso do disco por tabela

SELECT relfinenode, relpages FROM pg_class WHERE relname = 'nometabela'

Cada página usa 8kb.

Tamanho de Índices

```
SELECT c2.relname, c2.relpages
   FROM pg_class c, pg_class c2, pg_index i
   WHERE c.relname = 'customer'
      AND c.oid = i.indrelid
      AND c2.oid = i.indexrelid
   ORDER BY c2.relname;
```

Encontrar as maiores tabelas e índices
SELECT relname, relpages FROM pg_class ORDER BY relpages DESC;

Veja que no resultado também aparece a tabela de índices, e com uso significativo.

Ferramentas Contrib
pgbench – testa desempenho do SGBD.
dbsize – mostra o tamanho de tabelas e bancos
oid2name – retorna OIDs, fileinode e nomes de tabelas

```
D:\ARQUIV~1\POSTGR~1\8.1\bin>oid2name -U postgres -P ********
All databases:
   Oid    Database Name  Tablespace
------------------------------------
  33375   bdcluster           ncluster
  16948   cep_brasil          pg_default
  25146   cep_full            pg_default
  33360   controle_estoque  pg_default
  16879   municipios          pg_default
  33340   pgbench   pg_default
  10793   postgres   pg_default
  10792   template0           pg_default
  33377   template1           pg_default
  16898   testes              pg_default
```

No README desta contrib existe uma boa sugestão para encontrar o tamanho aproximados dos dados de cada objeto interno do PostgreSQL com:

SELECT relpages, relfilenode, relname FROM pg_class ORDER BY relpages DESC;

Cada página tem tipicamente 8KB e o relpages é atualizado pelo comando VACUUM.

Backup Automático de Bancos no Windows com o Agendador de Tarefas

Criação do script backuppg.bat:

```
rem Adaptação de Ribamar FS do original de Ivlison Souza para a lista PostgreSQL Brasil

@echo off
rem (Nome do Usuário do banco para realizar o backup)
REM Dados que precisa alterar:
REM PGUSER
REM PGPASSWORD
REM nome pasta de backup
```

REM nome pasta de instalação do PostgreSQL se diferente de C:\Arquivos de programas\PostgreSQL\8.1\
REM
REM (Nome do usuário do PostgreSQL que executará o script)
SET PGUSER=postgres

rem (Senha do usuário acima)
SET PGPASSWORD=******

rem (Indo para a raiz do disco)
C:

rem (Selecionando a pasta onde será realizada o backup)
chdir C:\backup

rem (banco.sql é o nome que defini para o meu backup
rem (Deletando o backup existente)
del banco*.sql

echo "Aguarde, realizando o backup do Banco de Dados"
rem C:\Arquiv~1\Postgr~1\8.1\bin\pg_dump -i -U postgres -b -o -f "C:\backup\banco.sql" condominio

rem Observação: Caso queira colocar o nome do backup seguindo de uma data é só usar:
for /f "tokens=1,2,3,4 delims=/ " %%a in ('DATE /T') do set Date=%%b-%%c-%%d
rem O comando acima serve para armazenar a data no formato dia-mes-ano na variável Date;

C:\Arquiv~1\Postgr~1\8.1\bin\pg_dump -i -U postgres -b -o -f "C:\backup\banco%Date%.sql" condominio
rem (sair da tela depois do backup)
exit

Configuração do Agendador de Tarefas para executar o script diariamente:

- Iniciar - Programas - Acessórios - Ferramentas de Sistema - Tarefas agendadas
- Adicionar tarefa agendada
- Avançar
- Clique em procurar e indique o backuppg.bat
- Em executar esta tarefa escolha como achar mais adequado (diariamente) e clique em Avançar
- Clique em Avançar e OK. Na próxima tela marque "Executar somente se conectado".
- Então clique em Concluir

- No próximo boot o backup será efetuado a cada dia.

Um bom artigo sobre backup e restauração no PostgreSQL encontra-se no site oficial do PostgreSQL do Brasil: https://wiki.postgresql.org.br/wiki/BackupAndRestore
Veja também a documentação em inglês:

http://www.postgresql.org/docs/8.1/static/app-pgrestore.html
http://www.postgresql.org/docs/8.1/static/app-pgdump.html
http://www.postgresql.org/docs/8.1/static/app-pg-dumpall.html

9.2 - Importar e Exportar

Para importar scripts gerados via pg_dump de dentro do banco devemos utilizar o comando

\i /path/script.sql

\i ./script.sql -- No windows com o arquivo no diretório atual

Para importar arquivos texto com delimitadores, tipo TXT, CSV ou binários utilizamos os comandos do banco (psql), como usuário do banco:

Importando:

\COPY tabela FROM ‘script.csv’

\COPY paises FROM 'clientes.csv';

Exportando:

CREATE TEMP TABLE paises AS SELECT * FROM teste WHERE nome LIKE '%a%';
\COPY paises TO '/usr/teste.copy';

Com Delimitadores

\COPY tabela FROM '/arquivo.csv' DELIMITERS '|';
\COPY tabela TO '/arquivo.txt' DELIMITERS '|';

Obs.: O arquivo teste.copy deve ter permissão de escrita para o user do banco.

Importar uma planilha do Excel ou do Calc do OpenOffice para uma tabela:

Gerando um arquivo CSV no OpenOffice Calc

- Abrir calc e selecionar e copiar a área a importar
- Abrir uma nova planilha
- Clicar com o botão direito sobre a primeira célula e Colar Especial
- Desmarque Colar tudo, marque Números, desmarque Fórmulas e OK
- Tecle Ctrl+S para salvar
- Em Tipo de arquivo escolha Texto CSV, digite o nome e Salvar. Confirme
- Como Delimitador de Campo escolha Tabulação
- Em Delimitador de texto delete as aspas e OK
- Ignore a mensagem de erro, caso apareça.

Importar o arquivo texto CSV para uma tabela com estrutura semelhante à do arquivo csv:

su - postgres
psql nomebanco
\copy nometabela from /home/nomearquivo.csv
No Windows
\copy nometabela from ./arquivo.csv   -- o arquivo estando no path do usuário

Exportar um Banco Access para uso no PostgreSQL ou outros bancos

Selecionar a tabela e Exportar

- Escolher o tipo de arquivos Texto (txt, csv, ...)
- Em avançado: Delimitador de campos – Tabulação
- Qualificador de texto – remover (deixar em branco)

9.3 - Converter
Uma boa forma de converter bancos MySQL para bancos PostgreSQL no Windows é instalando o driver ODBC para o MySQL e para o PostgreSQL.

Então cria-se a comunicação com os dois bancos e exporta-se para o PostgreSQL.

Existem ferramentas comerciais com muitos recursos, como é o caso do EMS Data Export e Import for PostgreSQL: http://www.sqlmanager.net/en/products/postgresql/dataexport

Veja: export to MS Excel, MS Word / RTF, MS Access, HTML, TXT, CSV, PDF, XML and SQL.

Outra opção é exportar para CSV do MySQL e importar pelo PostgreSQL.

9.4 - Otimização e Desempenho
Para isso ajusta-se bem o postgresql.conf, utiliza-se o vacuum, analyze e explain.

Lembrando que na versão 8.1 o vacuum não mais é um programa separado e vem embutido no executável. Mesmo embutido ele é configurável e podemos utilizar ou não e se usar, podemos também configurar sua periodicidade.
Uma ótima fonte de consulta:
http://www.metatrontech.com/wpapers/mysql2postgresql.pdf
Capítulo 21 do manual:
http://pgdocptbr.sourceforge.net/pg80/maintenance.html
Vacuum:
http://pgdocptbr.sourceforge.net/pg80/sql-vacuum.html
Analyze:
http://pgdocptbr.sourceforge.net/pg80/sql-analyze.html

VACUUM
O comando Vacuum tanto recupera espaço em disco, quanto otimiza o desempenho do banco e previne contra perda de dados muito antigos devido ao recomeço do ID das transações, portanto deve ser utilizado constantemente, como também atualiza as estatísticas dos dados utilizados pelo planejador de comandos. Lembrando que na versão 8.1 já vem embutido no executável, podendo apenas ser configurado para que seja executado automaticamente.

Na linha de comando:
vacuumdb -faze ou vacuumdb -fazq.

ANALYZE
O comando ANALYZE coleta estatísticas sobre o conteúdo das tabelas do banco de dados e armazena os resultados na tabela do sistema pg_statistic. Posteriormente, o planejador de comandos utiliza estas estatísticas para ajudar a determinar o plano de execução mais eficiente para os comandos. Caso não atualizemos estas estatísticas com freqüência podemos comprometer o desempenho do banco de dados por uma escolha errada do plano de comandos.
Normalmente operações DELETE ou UPDATE não removem os registros automaticamente. Somente após a execução do VACUUM isso acontece.

Recomendação
Para a maioria das instalações executar o comando VACUUM ANALYZE para todo o banco de dadosuma vez ao dia em horário de pouca utilização. Também podemos utilizar o comando:

vacuumdb -fazq.
Quando foi excluída a maioria dos registros de uma tabela sugere-se a execução do comando VACUUM FULL. Este comando gera um forte bloqueio nas tabelas em que é executado.

Em tabelas cujo conteúdo é excluído periodicamente, como tabelas temporárias, é indicado o uso do comando TRUNCATE ao invés de DELETE.

Exemplo de uso do vacuum. Acesse o banco e execute:
VACUUM VERBOSE ANALYZE nometabela;
De fora do psql usar o comando “vacuumdb -faze” ou “vacuumdb -fazq” (silencioso).

VACUUM VERBOSE ANALYZE autor;

INFO: vacuuming "public.autor"
INFO: "autor": found 0 removable, 0 nonremovable row versions in 0 pages
DETAIL: 0 dead row versions cannot be removed yet.
There were 0 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO: analyzing "public.autor"
INFO: "autor": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows

Em um Banco Completo
Só VACUUM
Ou
VACUUM FULL ANALYZE;

Dicas de Desempenho:
- Adicionar índice à tabela (toda chave primária já contém um índice)
- Adicionar índices aos campos de cláusulas WHERE;
- Evitar campos com tamanho variável. Preferir o CHAR ao VARCHAR.
- Evitar muitos índices
- Evitar índice em tabela muito pequena (poucos registros, não compensa)
- Evitar, sempre que possível, chaves compostas
- Separar bancos em um HD e logs em outro HD
- Aumentar shared buffers (postgresql.conf) de acordo com RAM disponível.

Recomendações: 25% da RAM para shared buffers cache e 2-4% para sort buffer.

bancos em /usr/local/pgsql/data (hda)

logs em /usr/local/pgsql/data/pg_xlog (hdb)

Utilizar links simbólicos para mover tabelas, índices, ... para outro HD.

Ativar o chip DMA

Testar: hdparm -Tr /dev/hda

Ativar o chip: hdparm -d 1 /dev/hda

Desativar: hdparm -d 0 /dev/hda

No postgresql.conf existem configurações para shared_buffers, que quanto maior melhor, respeitando-se a RAM.
O default da versão 8.1.3 é:
shared_buffers = 1000 # min 16 ou max_connections*2 (8KB cada)

Plano de Consulta
O PostgreSQL concebe um plano de comando para cada comando recebido. A escolha do plano correto, correspondendo à estrutura do comando e às propriedades dos dados, é absolutamente crítico para o bom desempenho. Pode ser utilizado o comando EXPLAIN para ver o plano criado pelo sistema para qualquer comando (conjunto executável de instruções). A leitura do plano é uma arte que merece um tutorial extenso, o que este não é; porém, aqui são fornecidas algumas informações básicas.

Os números apresentados atualmente pelo EXPLAIN são:
* **O custo de partida estimado** (O tempo gasto antes de poder começar a varrer a saída como, por exemplo, o tempo para fazer a classificação em um nó de classificação).
* **O custo total estimado** (Se todas as linhas fossem buscadas, o que pode não acontecer: uma consulta contendo a cláusula LIMIT pára antes de gastar o custo total, por exemplo).
* **Número de linhas de saída estimado para este nó do plano** (Novamente, somente se for executado até o fim).
* **Largura média estimada (em bytes)** das linhas de saída deste nó do plano.
EXPLAIN SELECT * FROM NOMETABELA;

Mostra plano de execução interna da consulta, acusando tempo gasto
EXPLAIN SELECT sum(i) FROM tabela1 WHERE i = 4;

Agora a consulta será modificada para incluir uma condição WHERE:
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 1000;
Modificando-se a consulta para restringir mais ainda a condição
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 50;

Adição de outra condição à cláusula WHERE:
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 50 AND stringu1 = 'xxx';

A seguir é feita a junção de duas tabelas, utilizando as colunas sendo discutidas:
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2 WHERE t1.unique1 < 50 AND t1.unique2 = t2.unique2;

Uma forma de ver outros planos é forçar o planejador a não considerar a estratégia que sairia vencedora, habilitando e desabilitando sinalizadores de cada tipo de plano (Esta é uma ferramenta deselegante, mas útil.
SET enable_nestloop = off;
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2 WHERE t1.unique1 < 50 AND t1.unique2 = t2.unique2;
É possível verificar a precisão dos custos estimados pelo planejador utilizando o comando EXPLAIN ANALYZE. Na verdade este comando executa a consulta, e depois mostra o tempo real acumulado dentro de cada nó do plano junto com os custos estimados que o comando EXPLAIN simples mostraria. Por exemplo, poderia ser obtido um resultado como este:

EXPLAIN ANALYZE SELECT * FROM tenk1 t1, tenk2 t2 WHERE t1.unique1 < 50 AND t1.unique2 = t2.unique2;

Reinício do ID de Transações
Para prevenir com segurança o recomeço do ID das Transações devemos utilizar o comando VACUUM em todas as tabelas do banco de dados pelo menos uma vez a cada meio bilhão de transações. Caso o VACUUM não seja executado pelo menos uma vez a cada 2 bilhões de transações ocorrerá a perca de todos os dados do banco. De fato eles não se perdem, voltando dentro de mais 2 bilhões de transações, mas isso não serve de consolo.

Como saber quantas transações ainda falta para a perca dos dados:

SELECT datname AS banco, AGE(datfrozenxid) AS idade FROM pg_database;
Sempre que se executa o comando VACUUM em um banco, a coluna com age começa de 1 bilhão. Ao se aproximar de 2 bilhões devemos executar novamente o comando VACUUM.

Alerta
Caso um banco já esteja com mais de 1,5 bilhões de transações, ao executar o comando VACUUM para o banco inteiro receberá um alerta sobre a necessidade de execução do VACUUM.

10 - Replicação

É o processo de compartilhar e distribuir informações entre diferentes bancos de dados. Estes dados serão mantidos sincronizados e íntegros em relação às regras de integridade referencial e de negócios.

No PostgreSQL algumas formas de realizar replicação são através do contrib dblink e das ferramenta slony e pgcluster.

Para importar o dblink no banco onde queremos replicar:

```
\i /usr/local/pgsql/contrib/dblink.sql
```

Exemplo dbLink - Select

```
select *
from dblink
	(
		'dbname=pgteste
		 hostaddr=200.174.40.63
		 user=paulo
		 password=paulo
		 port=5432',

		'select nome
		 from clientes
		'

	) as t1(nome varchar(30));
```

Exemplo dbLink - Insert

```
select
	dblink_exec(
			'dbname=pgteste
			 hostaddr=200.174.40.63
			 user=paulo
			 password=paulo
			 port=5432',

			 'insert into clientes(nome)
			  values(''roger'')
			'
	);
```

Exemplo dbLink - Update

```
select
	dblink_exec(
			'dbname=pgteste
			 hostaddr=200.174.40.63
			 user=paulo
			 password=paulo
			 port=5432',

			 'update clientes
			  set nome=''Paulo Rogerio''
			  where id = 18
```

```
                    '
        );
Exemplo dbLink - Delete
select
        dblink_exec(
                    'dbname=pgteste
                     hostaddr=200.174.40.63
                     user=paulo
                     password=paulo
                     port=5432',

                     'delete from clientes
                      where id = 18
                    '
        );
```

Temos o contrib dblink e o projerto slony para replicação de bancos do PostgreSQL.
O dblink não vem ativo por default.

Ativando o dblink:

De fora do banco:
psql -U nomeuser nomebanco < /usr/local/pgsql/contrib/dblink.sql

Ou de dentro do banco:
\i /usr/local/pgsql/contrib/dblink.sql

Funções do dblink:
dblink - para SELECT
dblinkexec - para INSERT, UPDATE e DELETE (remotos)

Tutorial sobre replicação no site da dbExperts - www.dbexperts.com.br
Usado para fazer consultas remotas em bancos do PG

dblink -> select
dblinkexec -> insert, update e delete (remotos)
Dica: Remover postmarter.pid em caso de queda anormal do SGBD

Bons documentos sobre replicação:
- Replicação do PostgreSQL com Slony do Marlon Petry
- Backup Quente no PostgreSQL com Replicação do Sílvio César
- Replicando banco de dados PostgreSQL do Rafael Donato

11 – Configurações

Ao instalar o PostgreSQL 8.1.4 via fontes ele cria (e alerta) o arquivo pg_hba.conf com autenticação do tipo trust (conexão local sem senha).

Para autenticar exigindo um dos tipos com senha, devemos antes, ainda no trust, alterar os usuários adicionando senha:

ALTER ROLE nomeuser WITH PASSWORD 'senhadopg';

Somente então devemos alterar o pg_hba.conf para pedir senha e restartar o PostgreSQL.

Numa instalação via fontes da versão 8.1.4 a autenticação padrão é do tipo trust (pg_hba.conf), o que permite acesso local sem senha.

Caso queiramos alterar para que os usuários sejam autenticados com o tipo password, md5 ou crypt, temos que dar a devida senha ao usuário, ainda usando trust e somente após ter criado as senhas dentro do psql ou outra interface, só então mudar o tipo de autenticação no pg_hba.conf.

As configurações principais são feitas nos arquivos pg_hba.conf e postgresql.conf. Se instalado através dos fontes ficam no subdiretório data de instalação do PostgreSQL, normalmente em /usr/local/pgsql. Se instalado via binários da distribuição vai variar com a distribuição. No Slackware estão no diretório /usr/share/postgresql.

O pg_hba.conf controla que máquinas terão acesso ao PostgreSQL e a autenticação dessas máquinas clientes (sem autenticação ou através de outras formas, trust, md5, crypt, etc).

O pg_hba.conf é muito rico e podemos controlar o acesso pelo IP, pela máscara, pelo banco, pelo usuário, pelo método (trust, md5, password, etc).

Trechos do pg_hba.conf:
...
Métodos: "trust", "reject", "md5", "crypt", "password", "krb5", "ident" ou "pam".
O método "password" envia senhas em texto claro; "md5" deve ser preferido já que envia senhas criptografadas. Configurando aqui como md5 as conexões em um cliente como o PHP deverão acontecer com a senha do usuário trazendo o hash md5 respectivo a sua senha e não em texto claro.

Dica: para conexão local, o TYPE local não pode estar comentado, ou seja, abaixo deveria aparecer uma linha com local ao invés de host.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
# IPv4 local connections:
host    all         all         127.0.0.1/32          trust
host    all         all         10.0.0.16/32          password
host    all         all         10.0.2.113/32         md5
host    all         all         0.0.0.0/0.0.0.0       reject
```

No exemplo acima diz que:

- as conexões que vierem de 127.0.0.1 via TCP, são confiáveis e tem acesso garantido.
- As que vierem de 10.0.0.16 deverão vir com senha em texto claro

- As que vierem de 10.0.2.113 deverão vir os hashs md5 das senhas e não texto claro.
- Todas as demais máquinas tem acesso negado (reject).

Exemplos do manual oficial (traduzido para o português do Brasil pelo Halley Pacheco):

Exemplo 19-1. Exemplo de registros do arquivo pg_hba.conf
Permitir qualquer usuário do sistema local se conectar a qualquer banco
de dados sob qualquer nome de usuário utilizando os soquetes do domínio
Unix (o padrão para conexões locais).
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
local all all trust

A mesma coisa utilizando conexões locais TCP/IP retornantes (loopback).
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
host all all 127.0.0.1/32 trust

O mesmo que o exemplo anterior mas utilizando uma coluna em separado para
máscara de rede.
#
TYPE DATABASE USER IP-ADDRESS IP-MASK METHOD
host all all 127.0.0.1 255.255.255.255 trust

Permitir qualquer usuário de qualquer hospedeiro com endereço de IP 192.168.93.x
se conectar ao banco de dados "template1" com o mesmo nome de usuário que "ident"
informa para a conexão (normalmente o nome de usuário do Unix).
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
host template1 all 192.168.93.0/24 ident sameuser

Permitir o usuário do hospedeiro 192.168.12.10 se conectar ao banco de dados
"template1" se a senha do usuário for fornecida corretamente.
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
host template1 all 192.168.12.10/32 md5

Na ausência das linhas "host" precedentes, estas duas linhas rejeitam todas
as conexões oriundas de 192.168.54.1 (uma vez que esta entrada será
correspondida primeiro), mas permite conexões Kerberos V de qualquer ponto
da Internet. A máscara zero significa que não é considerado nenhum bit do
endereço de IP do hospedeiro e, portanto, corresponde a qualquer hospedeiro.
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
host all all 192.168.54.1/32 reject
host all all 0.0.0.0/0 krb5

Permite os usuários dos hospedeiros 192.168.x.x se conectarem a qualquer
banco de dados se passarem na verificação de "ident". Se, por exemplo, "ident"
informar que o usuário é "oliveira" e este requerer se conectar como o usuário
do PostgreSQL "guest1", a conexão será permitida se houver uma entrada
em pg_ident.conf para o mapa "omicron" informando que "oliveira" pode se

conectar como "guest1".
#
TYPE DATABASE USER CIDR-ADDRESS METHOD
host all all 192.168.0.0/16 ident omicron

Se as linhas abaixo forem as únicas três linhas para conexão local, vão
permitir os usuários locais se conectarem somente aos seus próprios bancos de
dados (bancos de dados com o mesmo nome que seus nomes de usuário), exceto
para os administradores e membros do grupo "suporte" que podem se conectar a
todos os bancos de dados. O arquivo $PGDATA/admins contém a lista de nomes de
usuários. A senha é requerida em todos os casos.
#

TYPE DATABASE USER CIDR-ADDRESS METHOD
local sameuser all md5
local all @admins md5
local all +suporte md5
As duas últimas linhas acima podem ser combinadas em uma única linha:
local all @admins,+suporte md5

Obs.: @admins - lista de usuários em arquivo
+suporte - grupo de usuários

Local é para conexão usando apenas Socket UNIX, local.
A coluna banco de dados também pode utilizar listas e nomes de arquivos,
mas não grupos:
local db1,db2,@demodbs all md5
Um arquivo pg_ident.conf que pode ser utilizado em conjunto com o arquivo pg_hba.conf do Exemplo 19-1 está mostrado no Exemplo 19-2. Nesta configuração de exemplo, qualquer usuário autenticado em uma máquina da rede 192.168 que não possua o nome de usuário Unix oliveira, lia ou andre não vai ter o acesso permitido. O usuário Unix andre somente poderá acessar quando tentar se conectar como o usuário do PostgreSQL pacheco, e não como andre ou algum outro. A usuária lia somente poderá se conectar como lia. O usuário oliveira poderá se conectar como o
próprio oliveira ou como guest1.

Exemplo 19-2. Arquivo pg_ident.conf de exemplo
MAPNAME IDENT-USERNAME PG-USERNAME
omicron oliveira oliveira
omicron lia lia

pacheco possui o nome de usuário andre nestas máquinas
omicron andre pacheco

oliveira também pode se conectar como guest1
omicron oliveira guest1
host – conexões remotas usando TCP/IP. Conexões host aceitam conexões SSL e não SSL, mas conexões hostssl somente aceitam conexões SSL
hostssl - via SSL em TCP/IP
IP address e IP MASK - do cliente
md5 - requer cliente com senha md5
password - requer senha mas texto claro

Se houver preocupação com relação aos ataques de "farejamento" (sniffing) de senhas, então md5 é o método preferido, com crypt como a segunda opção se for necessário suportar clientes pré-7.2. O método password deve ser evitado, especialmente em conexões pela Internet aberta (a menos que seja utilizado SSL, SSH ou outro método de segurança para proteger a conexão).

ident
Obtém o nome de usuário do sistema operacional do cliente (para conexões TCP/IP fazendo contato com o servidor de identificação no cliente, para conexões locais obtendo a partir do sistema operacional) e verifica se o usuário possui permissão para se conectar como o usuário de banco de dados solicitado consultando o mapa especificado após a palavra chave ident.

Mais detalhes sobre o pg_hba.conf em:
http://pgdocptbr.sourceforge.net/pg80/client-authentication.html
O postgresql.conf permite configurar as demais funcionalidades do PostgreSQL
Liberando acesso via rede TCP/IP na versão 7.4.x:
tcp_socket = true (default = false)

No 8.0.x:
listen_address = '10.0.0.16'

Alguns configurações do postgresql.conf:
Regra geral: os valores que vêm comentados com # são os valores default. Se formos alterar algum idealmente devemos fazer uma cópia da linha e descomentar, para sempre saber o valor default.

sameuser é o usuário padrão no ident.conf (significa o mesmo user do sistema operacional).

```
# FILE LOCATIONS
#hba_file = 'ConfigDir/pg_hba.conf'   # host-based authentication file

# CONNECTIONS AND AUTHENTICATION
# O parâmetro listen_address indica as máquinas que terão acesso via TCP/IP
# - Connection Settings – Aqui as máquinas que terão acesso via TCP/IP
#listen_addresses = 'localhost'         # Que IP ou nome para escutar;
                        # lista de endereços separados por vírgula;
                        # defaults para 'localhost', '*' = all              '*' permite acesso a todos
# Por default aceita somente conexões locais
#port = 5432
max_connections = 100 (duas são reservadas para o superusuário)
# note: increasing max_connections costs ~400 bytes of shared memory per
#superuser_reserved_connections = 2

# - Security & Authentication -
#authentication_timeout = 60           # 1-600, in seconds
#ssl = off
#password_encryption = on
# RESOURCE USAGE (except WAL)
# - Memory -
shared_buffers = 1000        # min 16 or max_connections*2, 8KB each
#temp_buffers = 1000         # min 100, 8KB each
#max_prepared_transactions = 5         # can be 0 or more
```

```
# note: increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
#work_mem = 1024                    # min 64, size in KB
#maintenance_work_mem = 16384          # min 1024, size in KB
#max_stack_depth = 2048              # min 100, size in KB
# - Free Space Map -
#max_fsm_pages = 20000     # min max_fsm_relations*16, 6 bytes each
#max_fsm_relations = 1000   # min 100, ~70 bytes each
```

Algumas Configurações no postgresql.conf

```
...
# AUTOVACUUM PARAMETERS
#autovacuum = off                  # enable autovacuum subprocess?
...
# - Locale and Formatting -

#datestyle = 'iso, mdy'  # Era o original
datestyle = 'sql, european'  #       Formato dd/mm/yyyy
...
#client_encoding = sql_ascii
#client_encoding = latin1            # Suporte à acentuação do Brasil
...
```

Consultando no psql:
SHOW DATESTYLE;

Retorna -> SQL, DMY

Ajustando o estilo da data no psql:
SET DATESTYLE TO SQL, DMY;

ALTER ROLE nomeuser SET datestyle TO SQL, DMY;

O caminho de entrada num banco do PostgreSQL:
-> postgresql.conf -> ph_hba.conf -> ident.conf (caso este exista e seja citado no pg_hba.conf)

O encoding e outros recursos podem ser passados para cada banco, no momento de sua criação, como por exemplo:

De fora do banco:
createdb -E LATIN1 nomebanco.

De dentro do banco (psql):
CREATE DATABASE nomebanco WITH ENCODING 'LATIN1';

Para a relação completa dos encoding suportados veja tabela 21-2.
Para visualizar a codificação no psql digite
\encoding.

Para mudar a codificação de um banco dinamicamente, estando nele utilize:
\encoding novoencoding

Como também podemos utilizar o comando SET:
SET CLIENT_ENCODING 'LATIN1';
Consultando o encoding existente
SHOW CLIENT_ENCODING;

PARA DESFAZER AS ALTERAÇÕES E VOLTAR À CODIFICAÇÃO PADRÃO:
RESET CLIENT_ENCODING;

Mais detalhes:
http://www.postgresql.org/docs/8.1/interactive/runtime-config.html#CONFIG-SETTING

Para saber os locales existentes execute de dentro do psql:
\l -- Exibe bancos, donos e locales (Codificação)

Em cada conexão com o PostgreSQL, somente se pode acessar um único banco.
No postgresql.conf podemos definir o encoding através da variável client_encoding.

12 Metadados (Catálogo)

Metadados são dados sobre dados.
Uma consulta normal retorna informações existentes em tabelas, já uma consulta sobre os metadados retorna informações sobre os bancos, os objetos dos bancos, os campos de tabelas, seus tipos de dados, seus atributos, suas constraints, etc.

Retornar Todas as Tabelas do banco e esquema atual

```
SELECT schemaname AS esquema, tablename AS tabela, tableowner AS dono
FROM pg_catalog.pg_tables
WHERE schemaname NOT IN ('pg_catalog', 'information_schema', 'pg_toast')
ORDER BY schemaname, tablename
```

Informações de Todos os Tablespaces

```
SELECT spcname, pg_catalog.pg_get_userbyid(spcowner) AS spcowner, spclocation
FROM pg_catalog.pg_tablespace
```

Retornar banco, dono, codificação, comentários e tablespace

```
SELECT pdb.datname AS banco,
pu.usename AS dono,
pg_encoding_to_char(encoding) AS codificacao,
(SELECT description FROM pg_description pd WHERE pdb.oid=pd.objoid) AS comentario,
(SELECT spcname FROM pg_catalog.pg_tablespace pt WHERE pt.oid=pdb.dattablespace)
AS tablespace
FROM pg_database pdb, pg_user pu WHERE pdb.datdba = pu.usesysid ORDER BY
pdb.datname
```

Tabelas, donos, comentários, registros e tablespaces de um schema

```
SELECT c.relname as tabela,
pg_catalog.pg_get_userbyid(c.relowner) AS dono,
pg_catalog.obj_description(c.oid, 'pg_class') AS comentario, reltuples::integer as registros,
(SELECT spcname FROM pg_catalog.pg_tablespace pt WHERE pt.oid=c.reltablespace) AS
tablespace
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind = 'r'
AND nspname='public'
ORDER BY c.relname
```

Mostrar Sequences de um Esquema

```
SELECT c.relname AS seqname, u.usename AS seqowner, pg_catalog.obj_description(c.oid,
'pg_class') AS seqcomment,
        (SELECT spcname FROM pg_catalog.pg_tablespace pt WHERE
pt.oid=c.reltablespace) AS tablespace
      FROM pg_catalog.pg_class c, pg_catalog.pg_user u, pg_catalog.pg_namespace n
       WHERE c.relowner=u.usesysid AND c.relnamespace=n.oid
      AND c.relkind = 'S' AND n.nspname='public' ORDER BY seqname
```

Mostrar Tablespaces

```
SELECT spcname, pg_catalog.pg_get_userbyid(spcowner) AS spcowner, spclocation
FROM pg_catalog.pg_tablespace
```

Mostrar detalhes de uma function

```
SELECT
pc.oid AS prooid,
proname,
lanname as prolanguage,
pg_catalog.format_type(prorettype, NULL) as proresult,
prosrc,
probin,
proretset,
proisstrict,
provolatile,
prosecdef,
pg_catalog.oidvectortypes(pc.proargtypes) AS proarguments,
proargnames AS proargnames,
pg_catalog.obj_description(pc.oid, 'pg_proc') AS procomment
FROM pg_catalog.pg_proc pc, pg_catalog.pg_language pl
WHERE pc.oid = 'oid_da_function'::oid
AND pc.prolang = pl.oid
```

Este exemplo mostra uma consulta que lista os nomes dos esquemas, tabelas, colunas e chaves das chaves estrangeiras, e os nomes dos esquemas, tabelas e colunas referenciadas. Exemplo tirado da lista de discussão pgsql-sql

```
 CREATE TEMPORARY TABLE t1 (id SERIAL PRIMARY KEY, nome TEXT);
 CREATE TEMPORARY TABLE t2 (id INT REFERENCES t1, nome TEXT);
 SELECT
   n.nspname AS esquema,
   cl.relname AS tabela,
   a.attname AS coluna,
   ct.conname AS chave,
   nf.nspname AS esquema_ref,
   clf.relname AS tabela_ref,
   af.attname AS coluna_ref,
   pg_get_constraintdef(ct.oid) AS criar_sql
 FROM pg_catalog.pg_attribute a
   JOIN pg_catalog.pg_class cl ON (a.attrelid = cl.oid AND cl.relkind = 'r')
   JOIN pg_catalog.pg_namespace n ON (n.oid = cl.relnamespace)
   JOIN pg_catalog.pg_constraint ct ON (a.attrelid = ct.conrelid AND
       ct.confrelid != 0 AND ct.conkey[1] = a.attnum)
   JOIN pg_catalog.pg_class clf ON (ct.confrelid = clf.oid AND clf.relkind = 'r')
   JOIN pg_catalog.pg_namespace nf ON (nf.oid = clf.relnamespace)
   JOIN pg_catalog.pg_attribute af ON (af.attrelid = ct.confrelid AND
       af.attnum = ct.confkey[1]);
```

Mostrar Esquemas e Tabelas

```
SELECT n.nspname as esquema, c.relname as tabela, a.attname as campo,
format_type(t.oid, null) as tipo_de_dado
  FROM pg_namespace n, pg_class c,
       pg_attribute a, pg_type t
  WHERE n.oid = c.relnamespace
    and c.relkind = 'r'     -- no indices
    and n.nspname not like 'pg\\_%' -- no catalogs
    and n.nspname != 'information_schema' -- no information_schema
    and a.attnum > 0        -- no system att's
    and not a.attisdropped   -- no dropped columns
    and a.attrelid = c.oid
    and a.atttypid = t.oid
  ORDER BY nspname, relname, attname;
```

Mostrar Esquemas e respectivas tabelas do Banco atual:

```
SELECT n.nspname as esquema, c.relname as tabela  FROM pg_namespace n, pg_class c
  WHERE n.oid = c.relnamespace
    and c.relkind = 'r'     -- no indices
    and n.nspname not like 'pg\\_%' -- no catalogs
    and n.nspname != 'information_schema' -- no information_schema
  ORDER BY nspname, relname
```

Contar Todos os Registros de todas as tabelas de todos os bancos:

```
<?php
$conexao=pg_connect("host=127.0.0.1 user=postgres password=postabir");

$sql="SELECT datname AS banco FROM pg_database ORDER BY datname";
$consulta=pg_query($conexao,$sql);

        $banco = array();
        $c=0;
        while ($data = @pg_fetch_object($consulta,$c)) {
                $cons=$data->banco;

                $banco[] .= $cons;
        $c++;
        }

  $sql2="SELECT n.nspname as esquema,c.relname as tabela  FROM pg_namespace n,
pg_class c
  WHERE n.oid = c.relnamespace
    and c.relkind = 'r'     -- no indices
    and n.nspname not like 'pg\\_%' -- no catalogs
    and n.nspname != 'information_schema' -- no information_schema
  ORDER BY nspname, relname";

for ($x=0; $x < count($banco);$x++){
        if ($banco[$x] !="template0" && $banco[$x] != "template1" && $banco[$x] !
="postgres"){
                $conexao2=pg_connect("host=127.0.0.1 dbname=$banco[$x] user=postgres
```

```
password=postabir");
		$consulta2=pg_query( $conexao2, $sql2 );


		while ($data = pg_fetch_object($consulta2)) {
			$esquematab=$data->esquema.'.'.$data->tabela;
			$sql3="SELECT count(*) FROM $esquematab";
			$consulta3=pg_query($conexao2,$sql3);
			$res=@pg_fetch_array($consulta3);

			print 'Banco.Esquema.Tabela -> '.$banco[$x].'.'.$data->esquema.'.'.$data-
>tabela.' - Registro(s) - '.$res[0].'<br>';
			$total += $res[0];
		}

	}
}
	print "Total de Registro de todas as tabelas de todos os bancos ". $total;
?>
```

Dado o banco de dados, qual o seu diretório:
select datname, oid from pg_database;

Dado a tabela, qual o seu arquivo:
select relname, relfilenode from pg_class;

Mostrar chaves primárias das tabelas do esquema public
select indexrelname as indice, relname as tabela from pg_catalog.pg_statio_user_indexes as A INNER JOIN pg_catalog.pg_index as B ON A.indexrelid=B.indexrelid WHERE A.schemaname='public' AND B.indisprimary = true;

Para visualizar como as consultas são feitas internamente via psql usamos o comando assim:
psql -U user banco -E

Vamos usar o banco municipios, criado com os municípios do Brasil. A tabela opt_cidades.

Veja Um Exemplo Que Retorna a Chave Primária da Tabela opt_cidades

```
		SELECT
			ic.relname AS index_name,
			bc.relname AS tab_name,
			ta.attname AS column_name,
			i.indisunique AS unique_key,
			i.indisprimary AS primary_key
		FROM
			pg_class bc,
			pg_class ic,
			pg_index i,
			pg_attribute ta,
			pg_attribute ia
		WHERE
			bc.oid = i.indrelid
			AND ic.oid = i.indexrelid
```

```
AND ia.attrelid = i.indexrelid
AND ta.attrelid = bc.oid
AND bc.relname = 'opt_cidades'
AND ta.attrelid = i.indrelid
AND ta.attnum = i.indkey[ia.attnum-1]
ORDER BY
index_name, tab_name, column_name;
```

Retornará:

```
index_name    |  tab_name   | column_name | unique_key | primary_key
 opt_cidades_pkey | opt_cidades | id          | t          | t
```

Retornando o Nome do Esquema

```
SELECT  n.nspname AS "Esquema"
 FROM  pg_catalog.pg_namespace AS n,
     pg_catalog.pg_class AS c
 WHERE c.relnamespace = n.oid AND c.relname='opt_cidades';
```

Retorno: Esquema

Retornar nomes de bancos:

```
SELECT datname AS banco FROM pg_database
WHERE datname != 'template0' and datname != 'template1' and datname != 'postgres'
ORDER BY datname
```

Retornar nomes e OIDs dos bancos:

```
SELECT oid, datname FROM pg_database;
```

Dado a tabela, qual o seu arquivo:

```
select relname, relfilenode from pg_class;
```

No Windows

Podemos passar parâmetros para as macros, por exemplo:

```
doskey /exename=psql.exe dbinfo=SELECT datname,pg_encoding_to_char(encoding) FROM
pg_database WHERE datname='$1';
```

E então apenas passar o parâmetro na linha de comando:

```
postgres=# dbinfo postgres
```

Listar tabelas, e dono do esquema atual:

```
SELECT n.nspname as "Schema",
c.relname as "Tabela",
CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'i' THEN
'index' WHEN 'S' THEN 'sequence' WHEN 's' THEN 'special' END as "Tipo",
u.usename as "Dono"
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_user u ON u.usesysid = c.relowner
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','')
AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
```

Listar Tabelas

```
select c.relname FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','') AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
AND pg_catalog.pg_table_is_visible(c.oid);

SELECT tablename FROM pg_tables WHERE tablename NOT LIKE 'pg%' AND tablename
NOT LIKE 'sql\_%'
```

Listar todas as tabelas, índices, tamanho em KB e OIDs:

```
VACUUM; --Executar antes este comando
SELECT c1.relname AS tabela, c2.relname AS indice,
c2.relpages * 8 AS tamanho_kb, c2.relfilenode AS arquivo
FROM pg_class c1, pg_class c2, pg_index i
WHERE c1.oid = i.indrelid AND i.indexrelid = c2.oid
UNION
SELECT relname, NULL, relpages * 8, relfilenode
FROM pg_class
WHERE relkind = 'r'
ORDER BY tabela, indice DESC, tamanho_kb;
```

Tabelas e Soma

```
SELECT tablename, SUM( size_kb )
FROM
( SELECT c1.relname AS "tablename",
c2.relpages * 8 AS "size_kb"
FROM pg_class c1, pg_class c2, pg_index i
WHERE c1.oid = i.indrelid
AND i.indexrelid = c2.oid
UNION
SELECT relname, relpages * 8
FROM pg_class
WHERE relkind = 'r' ) AS relations
GROUP BY tablename;

--	r = ordinary table, i = index, S = sequence, v = view, c = composite type,
--	s = special, t = TOAST table
```

Tamanho em bytes de um banco:

```
select pg_database_size('banco');
```

Tamanho em bytes de uma tabela:

```
pg_total_relation_size('tabela')
```

Tamanho em bytes de tabela ou índice:

```
pg_relation_size('tabelaouindice')
```

Lista donos e bancos:

```
SELECT rolname as dono, datname as banco
FROM pg_roles, pg_database
WHERE pg_roles.oid = datdba
ORDER BY rolname, datname;
```

Nomes de bancos:

```
select datname from pg_database where datname not in ('template0','template1') order by 1
```

Nomes e colunas:

```
select tablename,'T' from pg_tables where tablename not like 'pg\_%'
and tablename not in ('sql_features', 'sql_implementation_info', 'sql_languages',
'sql_packages', 'sql_sizing', 'sql_sizing_profiles')
union
select viewname,'V' from pg_views where viewname not like 'pg\_%'
```

Tamanho de esquema e índice:

```
SELECT nspname,
sum(relpages * cast( 8192 AS bigint )) as "table size",
sum( ( select sum(relpages)
     from pg_class i, pg_index idx
     where i.oid = idx.indexrelid
     and t.oid=idx.indrelid ) ) * cast( 8192 AS bigint ) as "index size",
sum ( relpages * cast( 8192 AS bigint ) + ( select sum(relpages)
     from pg_class i, pg_index idx
     where i.oid = idx.indexrelid
     and t.oid=idx.indrelid ) * cast( 8192 AS bigint ) ) as "size"
FROM pg_class t, pg_namespace
WHERE relnamespace = pg_namespace.oid
and pg_namespace.nspname not like 'pg_%'
and pg_namespace.nspname != 'information_schema'
and relkind = 'r' group by nspname;
```

Retornando Tabelas e Seus Donos de um Esquema

```
SELECT n.nspname as "public",
  c.relname as "opt_cidades",
  CASE c.relkind WHEN 'r' THEN 'tabela' WHEN 'v' THEN 'view' WHEN 'i' THEN 'índice'
WHEN 'S' THEN 'sequencia' WHEN 's' THEN 'especial' END as "Tipo",  u.usename as "Dono"
FROM pg_catalog.pg_class c
    LEFT JOIN pg_catalog.pg_user u ON u.usesysid = c.relowner
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','')
     AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
     AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
```

Retorno:

```
 public |   opt_cidades    | Tipo |   Dono
--------+------------------+--------+----------
 public | opt_cidades      | tabela | postgres
 public | opt_estado       | tabela | postgres
```

Retornando o OID e o Esquema de uma Tabela

```
SELECT c.oid AS "OID",
  n.nspname AS "Esquema",
  c.relname AS "Tabela"
FROM pg_catalog.pg_class c
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
```

```
WHERE pg_catalog.pg_table_is_visible(c.oid)
     AND c.relname ~ '^opt_cidades$'
ORDER BY 2, 3;
```

Retorno:
 OID | Esquema | Tabela

Este exemplo mostra uma consulta que lista os esquemas, nomes das tabelas e nomes das colunas das chaves primárias de um banco de dados. Exemplo tirado da lista de discussão pgsql-sql .

```
CREATE TEMP TABLE teste1 (id INT, texto TEXT, PRIMARY KEY (id));

CREATE TEMP TABLE teste2 (id1 INT, id2 INT, texto TEXT, PRIMARY KEY (id1,id2));
\dt

SELECT
   pg_namespace.nspname AS esquema,
   pg_class.relname AS tabela,
   pg_attribute.attname  AS coluna_pk
 FROM pg_class
   JOIN pg_namespace ON pg_namespace.oid=pg_class.relnamespace AND
       pg_namespace.nspname NOT LIKE 'pg_%'
   JOIN pg_attribute ON pg_attribute.attrelid=pg_class.oid AND
       pg_attribute.attisdropped='f'
   JOIN pg_index ON pg_index.indrelid=pg_class.oid AND
       pg_index.indisprimary='t' AND
       (
       pg_index.indkey[0]=pg_attribute.attnum OR
       pg_index.indkey[1]=pg_attribute.attnum OR
       pg_index.indkey[2]=pg_attribute.attnum OR
       pg_index.indkey[3]=pg_attribute.attnum OR
       pg_index.indkey[4]=pg_attribute.attnum OR
       pg_index.indkey[5]=pg_attribute.attnum OR
       pg_index.indkey[6]=pg_attribute.attnum OR
       pg_index.indkey[7]=pg_attribute.attnum OR
       pg_index.indkey[8]=pg_attribute.attnum OR
       pg_index.indkey[9]=pg_attribute.attnum
       )
 ORDER BY pg_namespace.nspname, pg_class.relname,pg_attribute.attname;
```

Este exemplo mostra uma consulta que lista os nomes dos esquemas, tabelas, colunas e chaves das chaves estrangeiras, e os nomes dos esquemas, tabelas e colunas referenciadas. Exemplo tirado da lista de discussão pgsql-sql

```
 CREATE TEMPORARY TABLE t1 (id SERIAL PRIMARY KEY, nome TEXT);
 CREATE TEMPORARY TABLE t2 (id INT REFERENCES t1, nome TEXT);
 SELECT
   n.nspname AS esquema,
   cl.relname AS tabela,
   a.attname AS coluna,
   ct.conname AS chave,
   nf.nspname AS esquema_ref,
   clf.relname AS tabela_ref,
   af.attname AS coluna_ref,
```

```
   pg_get_constraintdef(ct.oid) AS criar_sql
 FROM pg_catalog.pg_attribute a
   JOIN pg_catalog.pg_class cl ON (a.attrelid = cl.oid AND cl.relkind = 'r')
   JOIN pg_catalog.pg_namespace n ON (n.oid = cl.relnamespace)
   JOIN pg_catalog.pg_constraint ct ON (a.attrelid = ct.conrelid AND
       ct.confrelid != 0 AND ct.conkey[1] = a.attnum)
   JOIN pg_catalog.pg_class clf ON (ct.confrelid = clf.oid AND clf.relkind = 'r')
   JOIN pg_catalog.pg_namespace nf ON (nf.oid = clf.relnamespace)
   JOIN pg_catalog.pg_attribute af ON (af.attrelid = ct.confrelid AND
       af.attnum = ct.confkey[1]);
```

Retorno:

```
  esquema  | tabela | coluna |   chave    | esquema_ref | tabela_ref | coluna_ref |
criar_sql
 pg_temp_1 | t2     | id     | t2_id_fkey | pg_temp_1   | t1         | id | FOREIGN KEY (id)
REFERENCES t1(id)
```

```
SELECT a.attnum, a.attname AS field, t.typname as type, a.attlen AS length, a.atttypmod-4 as
lengthvar, a.attnotnull as notnull
       FROM pg_class c, pg_attribute a, pg_type t
       WHERE c.relname = 'apagar' AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid =
t.oid
       ORDER BY a.attnum;
```

Saída:
ID do campo, nomecampo, tipo, tamanho, nulo/nãonulo

Outros

```
SELECT ic.relname AS index_name, bc.relname AS tab_name, ta.attname AS column_name,
i.indisunique AS unique_key, i.indisprimary AS primary_key
        FROM pg_class bc, pg_class ic, pg_index i, pg_attribute ta, pg_attribute ia
        WHERE (bc.oid = i.indrelid)
           AND (ic.oid = i.indexrelid)
           AND (ia.attrelid = i.indexrelid)
           AND (ta.attrelid = bc.oid)
           AND (bc.relname = 'apagar')
           AND (ta.attrelid = i.indrelid)
           AND (ta.attnum = i.indkey[ia.attnum-1])
           ORDER BY index_name, tab_name, column_name
```

Saída:
nomeindex/chave, nometabela, nomecampo, unique(t/f), nomepk (t/f)

```
SELECT rcname as index_name, rcsrc
        FROM pg_relcheck, pg_class bc
        WHERE rcrelid = bc.oid
           AND bc.relname = 'apagar'
            AND NOT EXISTS (
               SELECT *
                  FROM pg_relcheck as c, pg_inherits as i
                  WHERE i.inhrelid = pg_relcheck.rcrelid
                     AND c.rcname = pg_relcheck.rcname
                     AND c.rcsrc = pg_relcheck.rcsrc
                     AND c.rcrelid = i.inhparent
            )
```

Saída: retorna as constraints check.
SELECT pg_class.relname, pg_attribute.attname, pg_type.typname,
pg_attribute.atttypmod-4
FROM pg_class, pg_attribute, pg_type
WHERE pg_attribute.attrelid = pg_class.oid
AND pg_attribute.atttypid = pg_type.oid
AND pg_class.relname = 'apagar'
AND pg_attribute.attname = 'descricao'

Saída: tabela, campo, tipo, tamanho (varchar)

Outros Exemplos
create table tabela_exemplo (
campo_1 integer default 5, campo_2 text default 'exemplo', campo_3 float(10),
campo_4 serial, campo_5 double precision, campo_6 int8, campo_7 Point,
campo_8 char(3), campo_9 varchar(17));

Depois de criada a tabela vamos criar a consulta que nos retornará as informações da tabela:

```
SELECT
        rel.nspname AS Esquema, rel.relname AS Tabela, attrs.attname AS Campo, "Type",
"Default",   attrs.attnotnull AS "NOT NULL"
FROM (
        SELECT c.oid, n.nspname, c.relname
        FROM pg_catalog.pg_class   c
        LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
        WHERE pg_catalog.pg_table_is_visible(c.oid)   ) rel
JOIN (
        SELECT a.attname, a.attrelid, pg_catalog.format_type(a.atttypid,   a.atttypmod) as
"Type",
        (SELECT substring(d.adsrc for 128) FROM pg_catalog.pg_attrdef   d
        WHERE d.adrelid = a.attrelid AND d.adnum = a.attnum AND a.atthasdef) as "Default",
a.attnotnull, a.attnum
        FROM pg_catalog.pg_attribute a WHERE a.attnum > 0   AND NOT a.attisdropped )
attrs
        ON (attrs.attrelid = rel.oid )
        WHERE relname = 'tabela_exemplo' ORDER BY attrs.attnum;
```

Retorno:

```
testes-#       WHERE relname = 'tabela_exemplo' ORDER BY attrs.attnum;
 esquema |    tabela    |  campo  |       Type        |                  Default                   | NOT
NULL
```

Antes de tudo devemos criar um novo tipo de dado relacionado ao retorno que obteremos da função:

CREATE TYPE tabela_estrutura AS (Esquema text, Tabela text, Campo text, Tipo text, Valor text, AutoIncremento bool);

A função abaixo é definida em PL/PgSQL, linguagem procedural muito semelhante ao PL/SQL do Oracle. A função foi criada nesta linguagem devido a certas limitações que as funções em SQL possuem.

```
CREATE OR REPLACE FUNCTION Dados_Tabela(varchar(30))
RETURNS SETOF tabela_estrutura AS '
 DECLARE
 r tabela_estrutura%ROWTYPE;
 rec RECORD;
 vTabela alias for $1;
 eSql TEXT;

 BEGIN
 eSql := ''SELECT
      CAST(rel.nspname as TEXT), CAST(rel.relname AS TEXT) , CAST(attrs.attname AS
TEXT), CAST("Type" AS TEXT), CAST("Default" AS TEXT), attrs.attnotnull
       FROM
            (SELECT c.oid, n.nspname, c.relname
            FROM pg_catalog.pg_class c
            LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
            WHERE pg_catalog.pg_table_is_visible(c.oid) ) rel
       JOIN
            (SELECT a.attname, a.attrelid,
            pg_catalog.format_type(a.atttypid, a.atttypmod) as "Type",
                  (SELECT substring(d.adsrc for 128) FROM pg_catalog.pg_attrdef d
                  WHERE d.adrelid = a.attrelid AND d.adnum = a.attnum AND a.atthasdef)
            as "Default", a.attnotnull, a.attnum
            FROM pg_catalog.pg_attribute a
            WHERE a.attnum > 0 AND NOT a.attisdropped ) attrs
       ON (attrs.attrelid = rel.oid )
       WHERE relname LIKE ''''%'''' || vTabela || ''''%''''
       ORDER BY attrs.attnum'';
FOR r IN EXECUTE eSql
 LOOP
 RETURN NEXT r;
END LOOP;
 IF NOT FOUND THEN
       RAISE EXCEPTION ''Tabela % não encontrada'', vTabela;
 END IF;
 RETURN;
 END
 '
 LANGUAGE 'plpgsql';
```

Para utilizar esta função, utilize o seguinte comando:

```
SELECT * FROM Dados_Tabela('tabela');
```

Retorno:

esquema | tabela | campo | tipo | valor | autoincremento

Exemplos contidos no arquivo:
/usr/local/src/postgresql-8.1.3/src/tutorial/syscat.sql

```
SELECT rolname as "Donos", datname as Bancos
  FROM pg_roles, pg_database
  WHERE pg_roles.oid = datdba
```

```
  ORDER BY rolname, datname;
```

Retorno: Donos e Bancos

```
SELECT n.nspname as esquema, c.relname as tabela
  FROM pg_class c, pg_namespace n
  WHERE c.relnamespace=n.oid
    and c.relkind = 'r'                -- not indices, views, etc
    and n.nspname not like 'pg\\_%'       -- not catalogs
    and n.nspname != 'information_schema' -- not information_schema
  ORDER BY nspname, relname;
```

Retorno: Esquemas e Tabelas

```
SELECT n.nspname as esquema, c.relname as tabela, a.attname as campo,
format_type(t.oid, null) as tipo_de_dado
  FROM pg_namespace n, pg_class c,
       pg_attribute a, pg_type t
  WHERE n.oid = c.relnamespace
    and c.relkind = 'r'     -- no indices
    and n.nspname not like 'pg\\_%' -- no catalogs
    and n.nspname != 'information_schema' -- no information_schema
    and a.attnum > 0        -- no system att's
    and not a.attisdropped   -- no dropped columns
    and a.attrelid = c.oid
    and a.atttypid = t.oid
  ORDER BY nspname, relname, attname;
```

Retorno: esquemas, tabelas, campos, tipos de dados

```
SELECT n.nspname, o.oprname AS binary_op,
       format_type(left_type.oid, null) AS left_opr,
       format_type(right_type.oid, null) AS right_opr,
       format_type(result.oid, null) AS return_type
  FROM pg_namespace n, pg_operator o, pg_type left_type,
       pg_type right_type, pg_type result
  WHERE o.oprnamespace = n.oid
    and o.oprkind = 'b'        -- binary
    and o.oprleft = left_type.oid
    and o.oprright = right_type.oid
    and o.oprresult = result.oid
  ORDER BY nspname, left_opr, right_opr;
```

Retorno: operadores binários

Baypassar os de sistema:

```
and n.nspname not like 'pg\\_%' -- no catalogs
```

```
SELECT n.nspname, p.proname, format_type(t.oid, null) as typname
  FROM pg_namespace n, pg_aggregate a,
       pg_proc p, pg_type t
  WHERE p.pronamespace = n.oid
    and a.aggfnoid = p.oid
    and p.proargtypes[0] = t.oid
  ORDER BY nspname, proname, typname;
```

Retorno: lista todas as funções agregadas e os tipos que podem ser aplicados

Dado o banco de dados, qual o seu diretório:

```
select datname, oid from pg_database;
```

Dado a tabela, qual o seu arquivo:

```
select relname, relfilenode from pg_class;
```

Exemplo que retorna índice, campo, tipo, comprimento, null, default:

```
SELECT pg_attribute.attnum AS index,
attname AS field,
typname AS type,
atttypmod-4 as length,
NOT attnotnull AS "null",
adsrc AS default
FROM pg_attribute,
pg_class,
pg_type,
pg_attrdef
WHERE pg_class.oid=attrelid
AND pg_type.oid=atttypid
AND attnum >0
AND pg_class.oid=adrelid
AND adnum=attnum
AND atthasdef='t'
AND lower(relname)='datas'
UNION
SELECT pg_attribute.attnum AS index,
attname AS field,
typname AS type,
atttypmod-4 as length,
NOT attnotnull AS "null",
'' AS default
FROM pg_attribute,
pg_class,
pg_type
WHERE pg_class.oid=attrelid
AND pg_type.oid=atttypid
AND attnum>0
AND atthasdef='f'
AND lower(relname)='datas';
```

13 –Conectividade

Vou mostrar a conectividade do PostgreSQL com o PHP, com o Java e com o Visual BASIC. Também mostrarei a conectividade através do ODBC com o Access.

Conectando com o PHP

Com o PHP existe uma conexão nativa. Veja um exemplo:

```
$conexao = pg_connect("host=127.0.0.1 dbname=testes user=postgres password=*******
port=5432");
if (!$conexao){
        echo "Falha na conexão com o banco. Veja detalhes técnicos: " .
pg_last_error($conexao);
}
```

Conexão com Java

A conexão do PostgreSQL com Java é utilizada por diversos clientes de gerenciamento ou modelagem do PostgreSQL. Neste caso utiliza-se o driver JDBC do PostgreSQL. Vide pasta \jdbc da instalação.

Baixar de acordo com sua versão do PostgreSQL, o driver JDBC para o PostgreSQL daqui:

http://jdbc.postgresql.org/download.html#jdbcselection

Aqui para o PostgreSQL versão 8.1.3 baixei o arquivo 8.1-405 JDBC 3.

VB Acessando PostgreSQL via ODBC

O PGODBC deve ser instalado no micro cliente e encontra-se em:
http://www.postgresql.org/ftp/odbc/versions/msi

Criar uma conexão ODBC ao banco do PostgreSQL e no código:

```
Global Conex As New ADODB.Connection
Global AccessConnect As String

Public Sub Conexao()
  AccessConnect =
"driver={PostgreSQL};server=10.10.10.10;database=maubanco;uid=postgres;pwd=postgres;"
  Conex.ConnectionString = AccessConnect
  Conex.Open AtivConex.ActiveConnection = Conex
End Sub
```

Exemplo Básico de Java Acessando PostgreSQL Via JDBC

Crie no PostgreSQL um pequeno banco de dados chamado agenda com uma única tabela chamada amigos.

Esta tabela contendo os campos nome e email apenas. Cadastre um ou mais registros para melhor visualização dos resultados.

```
import java.sql.*;
public class SQLStatement  {
    public static void main(String args[]) {
        //String url = "jdbc:postgresql://10.0.1.53:5432/agenda";
        String url = "jdbc:postgresql://localhost:5432/agenda";
```

```
        //String url = "jdbc:postgresql:agenda";    //Assim pega os defaults
        Connection con;
        String query = "select * from amigos”;
        Statement stmt;
        try {
           Class.forName("org.postgresql.Driver");
        } catch(java.lang.ClassNotFoundException e) {
           System.err.print("ClassNotFoundException: ");
           System.err.println(e.getMessage());
        }
        try {
           con = DriverManager.getConnection(url,"postgres", "postgres");
           stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery(query);
           ResultSetMetaData rsmd = rs.getMetaData();
           int numberOfColumns = rsmd.getColumnCount();
           int rowCount = 1;
           while (rs.next()) {
              System.out.println("Registro " + rowCount + ":  ");
              for (int i = 1; i <= numberOfColumns; i++) {
                 System.out.print("   Campo " + i + ":  ");
                 System.out.println(rs.getString(i));
              }
              System.out.println("");
              rowCount++;
           }
           stmt.close();
           con.close();
        } catch(SQLException ex) {
           System.err.print("SQLException: ");
           System.err.println(ex.getMessage());
        }
    }
}
```

Conexão Com o Visual BASIC

Podemos nos conectar a uma base de dados PostgreSQL usando o Visual Basic via ADO.
Para isto temos que usar um driver ODBC para a plataforma Windows.

Voce vai precisar ter o PostgreSQL instalado e o driver ODBC também.
Instala-se o psqlODBC e configura-se a conexão com o banco desejado.

If so then use something like
   CurrentProject.Connection.Execute StrSql2
If not linked tables then use something like
  Dim cnn as new ADODB.Connection
  cnn.Open "DSN=my_dbs_dsn_name" 'or a full PostgreSQL connection string to save a trip to the registry
  cnn.Execute StrSql2

Outro exemplo:
Criar um DSN ODBC "pgresearch" via ADO e use:

```
Dim gcnResearch As ADODB.Connection
Dim rsUId As ADODB.Recordset

' open the database
Set gcnResearch =3D New ADODB.Connection
With gcnResearch
.ConnectionString =3D "dsn=3Dpgresearch"
.Properties("User ID") =3D txtUsername
.Properties("Password") =3D txtPassword
.Open
End With
```

Acessando com o Visual C#.net, ver link:
http://www.linhadecodigo.com.br/artigos.asp?id_ac=355

14 - Ferramentas

14.1 – psql

A ferramenta básica de administração do PostgreSQL é o psql, mas é uma ferramenta de administração capaz de administrar praticamente tudo do PostgreSQL.

Para acessá-lo execute:
su – postgresql
psql –U nomeuser nomebanco (tanto no Linux quanto em outros SOs).

Geral:
psql -h host -P port -U user -W (perguntar pela senha)

Alguns comandos do PostgreSQL da linha de comando do SO:
Se num UNIX faça login como usuário do PostgreSQL, se no Windows execute passando -U nomeusuario.

Obtendo ajuda sobre um comando:
comando –help

Se num UNIX existem também as manpages (páginas do manual):
man comando

psql -l -> lista os bancos de dados
psql -U nomeusuario nomebanco -> conectar à console psql no banco de dados
psql banco -E -> (debug) mostra internamente como cada consulta é realizada
psql –version -> mostra versão do PostgreSQL

Outros comandos via linha de comando:
pg_dump, pg_dumpall, pg_restote, createdb, dropdb, createrole, droprole

Alguns Comandos do psql:

Para acessar, estando num UNIX:
su – nomeuserpg
psql -U nomeuserpg nomebanco

Estando no Windows
psql -U nomeuserpg nomebanco

O psql aceita quebra de linhas numa consulta.
O ponto e vírgula (ou <g) indica ordem de execução.

Observe atentamente o prompt e suas variações:
=# - este prompt indica um superusuário
=> - este indica um usuário comum
-# - indica comando não finalizado. Aguardando o ponto e vírgula
(# - aguardando o fecha parênteses)
'# - aguardando um fecha apóstrofo '

Obs.: Em caso de erro teclar Ctrl+C para encerrar. Lembrando que isso no Windows sai do psql.

\q - sair
\c nomebanco nomeuser – Conectar a outro banco
\i /path/script.sql -- importar script.sql
\timing -- iniciar/parar o cronômetro para atividades
\dT+ -- lista os tipos de dados do PG com detalhes
\cd -- mudar para outro diretório
\d – lista tabelas, índices, sequências ou views
\d nometabela – mostra estrutura da tabela
\dt – lista tabelas
\di – lista indices
\ds – lista sequências
\dv – lista views
\dS – lista tabelas do sistema
\dn – lista esquemas
\dp – lista privilégios
\du – lista usuários
\dg – lista grupos
\l - lista todos os bancos do servidor, juntamente com seus donos e codificações
\e - abre o editor vi com a última consulta
\o - inicia/termina a criação de arquivo. Ex.: \o arquivo.sql
\! comando_do_sistemaoperacional -- executa o arquivo do sistema operacional
\? - ajuda geral dos comandos do psql
\h * - exibe ajuda de todos os comandos
\h comandosql – ajuda específica sobre o comando SQL, ex.: \h alter table
\H – ativa/desativa saída em HTML
\encoding – exibe codificação atual

Boa sugestão:
\h CREATE DATABASE
\h CREATE ROLE

Exemplo de saída de consulta em HTML pelo PostgreSQL:

Gerando um relatório em HTML diretamente através do PostgreSQL
\o relatorio.html
SELECT * FROM cep_tabela WHERE uf='CE';

Obs.: Lembre que o PostgreSQL é case sensitive.

Com isso teremos um arquivo HTML contendo todos os registros retornados pela consulta em uma tabela HTML, como no exemplo abaixo:

```
<table border="1">
  <tr>
    <th align="center">cep</th>
    <th align="center">tipo</th>
    <th align="center">logradouro</th>
    <th align="center">bairro</th>
    <th align="center">municipio</th>
    <th align="center">uf</th>
  </tr>
  <tr valign="top">
    <td align="left">60420440</td>
    <td align="left">Rua
```

```
    <td align="left">Vasco da Gama
    <td align="left">Montese
    <td align="left">Fortaleza
    <td align="left">CE</td>
  </tr>
</table>
```

Console do psql

```
Interface de linha de comando - psql -U postgres
Pßgina de c¾digos ativa: 1252

C:\Arquivos de programas\PostgreSQL\8.1\bin>psql -U postgres
Password for user postgres:
Welcome to psql 8.1.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

postgres=# _
```

14.2 - phpPgAdmin

Baixar de - http://phppgadmin.sourceforge.net/

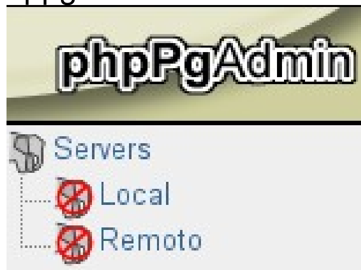
- Copiar para o diretório web
- Editar o arquivo conf/config.inc.php e alterar para dois servidores (um local e outro remoto):

```
...
        // Display name for the server on the login screen
        $conf['servers'][0]['desc'] = 'Local';

        // Hostname or IP address for server.  Use '' for UNIX domain socket.
        // use 'localhost' for TCP/IP connection on this computer
        $conf['servers'][0]['host'] = '127.0.0.1';
...
        // Example for a second server (PostgreSQL Remoto)
        $conf['servers'][1]['desc'] = 'Remoto';
        $conf['servers'][1]['host'] = '10.99.00.11';
        $conf['servers'][1]['port'] = 5432;
        $conf['servers'][1]['defaultdb'] = 'nomebancodefault';
...
        // If extra login security is true, then logins via phpPgAdmin with no
        // password or certain usernames (pgsql, postgres, root, administrator)
        // will be denied. Only set this false once you have read the FAQ and
        // understand how to change PostgreSQL's pg_hba.conf to enable
```

```
// passworded local connections.
$conf['extra_login_security'] = false;
```

Com isso teremos um login do phppgadmin assim:



14.3 – PgAdmin

PgAdmin
Site para download, caso sua distribuição não traga ou não tenha como instalar (apt, synaptic ou outro gerenciador de pacotes).
http://www.pgadmin.org/download/

É uma ferramenta gráfica desenvolvida pela equipe de desenvolvimento do PostgreSQL. Muitos recursos. Traz um help sobre si e a documentação do PostgreSQL. Tecle F1 para exibir.

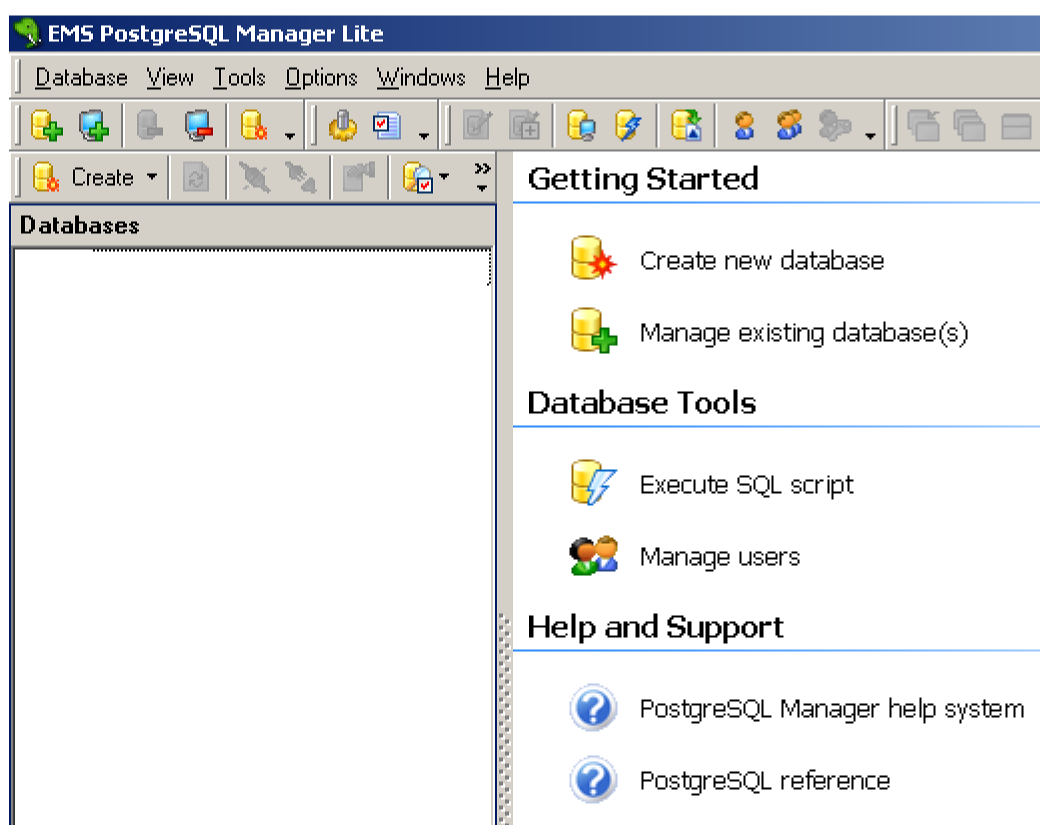
Ao executar consultas na ferramenta SQL, tecle F7 para visualizar graficamente a consulta na aba Explain.

14.4 - EMS PostgreSQL
O EMS é um ótimo gerenciador de diversos tipos de bancos, inclusive do PostgreSQL.

Download – http://www.sqlmanager.net/en/products/postgresql/manager (para Windows existe uma versão free, a lite)

Aqui vou abordar as atividades principais e básicas de uso do EMS:
- Abrir em banco
- Criar em novo banco
- Criar tabelas
- Criar campos
- Criar chave primária
- Criar chave estrangeira (relacionamento)
- Importar script .sql para um banco existente
- Exportar banco como script sql
- Executar consultas sql

Após executar aparece algo como (versão 3.1.5.2 lite for Windows):



CRIAR UM NOVO BANCO

- Em Getting Starting (acima e à direita) clique no botão Create new database
- Então digite o nome do novo banco:

Create Database Wizard

Create Database

Specify the name for a new database

Welcome to the Create Database Wizard!
This wizard allows you to create a new database and register it in the Database Explorer.

This wizard will generate the SQL statement for creating the database and execute it on PostgreSQL server.

Database name novoi_banco

Register after creating

- E clique no botão Next
- Então entre com os dados do servidor (como abaixo):

Create Database Wizard

Create Database

Set connection properties for a new database

Host name 127.0.0.1 Port 5432

User name ribafs

Password ********

Use HTTP tunneling

- Na próxima tela mude algo somente se tiver certeza:

Create Database Wizard

Create Database

Advanced database properties

Location

Template

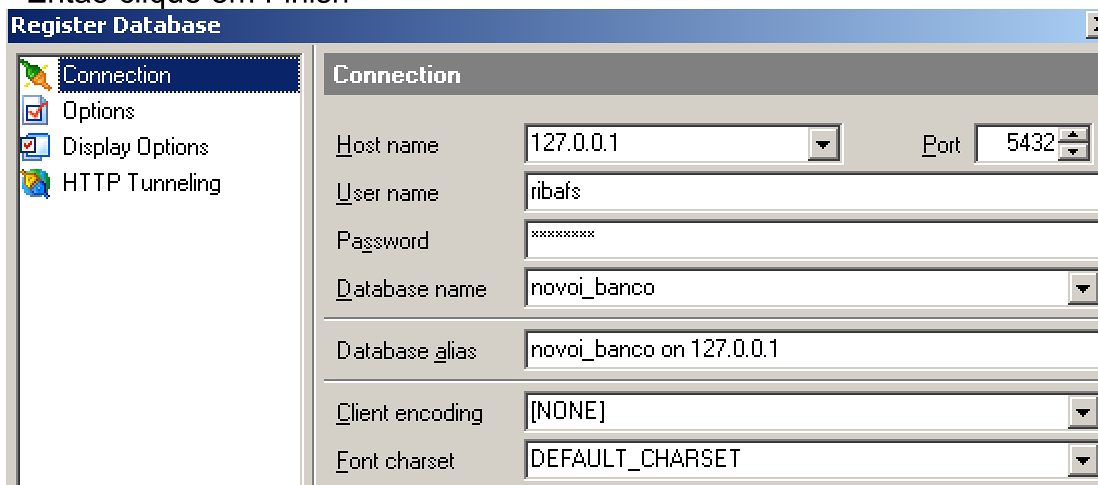
Encoding

Owner (7.3 or higher)

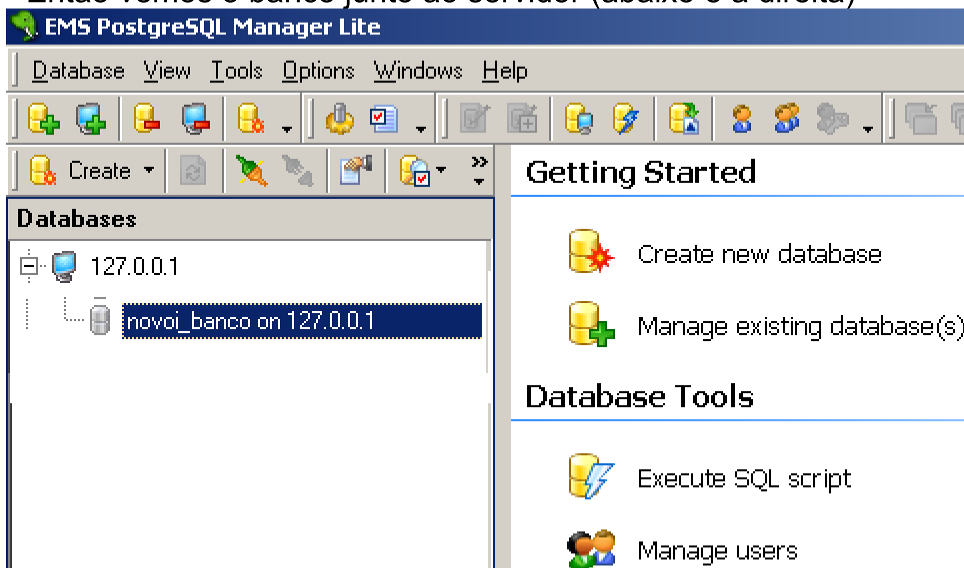
Default tablespace (8.0 or higher)

- Clique em Next

- Então clique em Finish



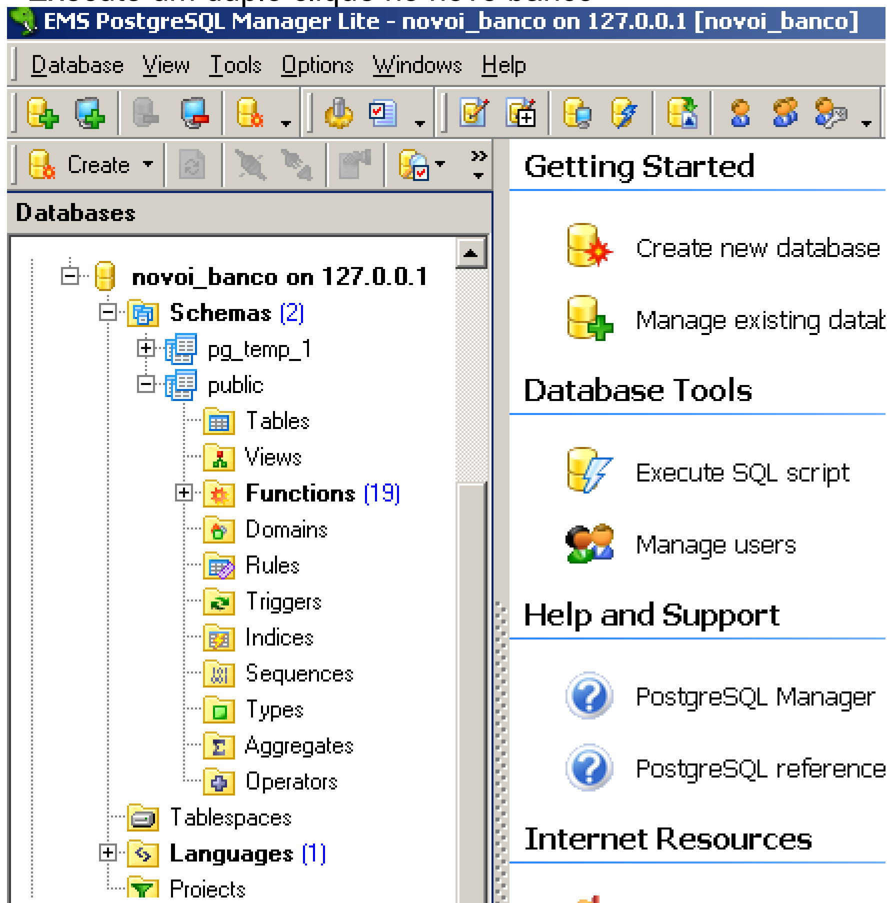
- Então clique em OK.
- Então vemos o banco junto ao servidor (abaixo e à direita)



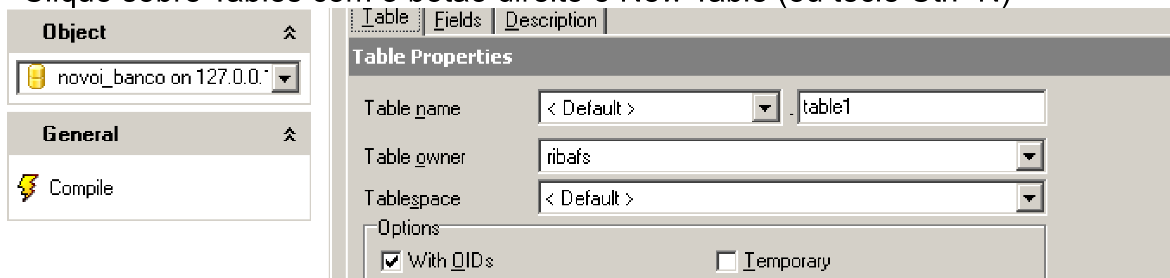
Para abri-lo e criar tabelas basta um duplo clique nele.

CRIAR TABELAS

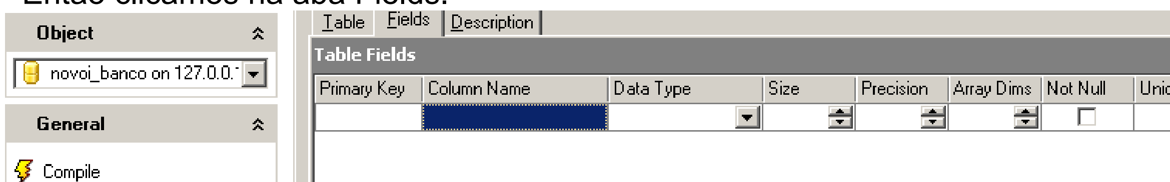
- Execute um duplo clique no novo banco



- Observe a estrutura criada para o novo banco:
- Clique sobre Tables com o botão direito e New Table (ou tecle Ctrl+N)



- Acima digitamos o nome da tabela onde existe table1
- Então clicamos na aba Fields.



- Mais um duplo clique, agora em Column Name, para que apareça o Wizard de Campos:

- Veja que o nome do campo é “codigo”. Que ele é do tipo BIGINT e também é chave primária.
- Veja agora como aparece nosso campo (com uma pequena chave à direita):

Isso mostra que este campo é nossa chave primária.

Add Chave Estrangeira

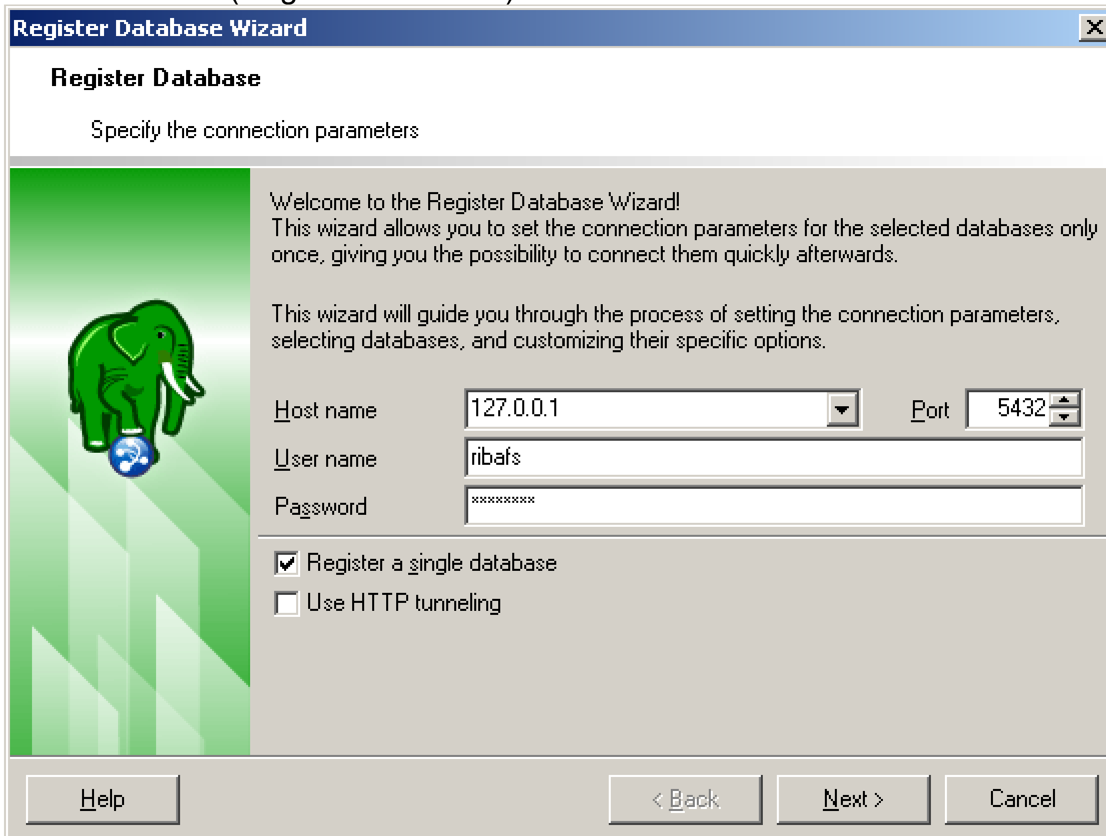
- Clique em Compile e veja como fica:

- Vamos adicionar mais um campo (nome varchar(40)):

- Adicione os demais campos de forma semelhante.
- Veja que sempre depois de um OK vem um botão de Commit, com a sintaxe SQL do comando que estamos executando no banco. Isso é um controle de transações do EMS através do recurso existente no PostgreSQL.

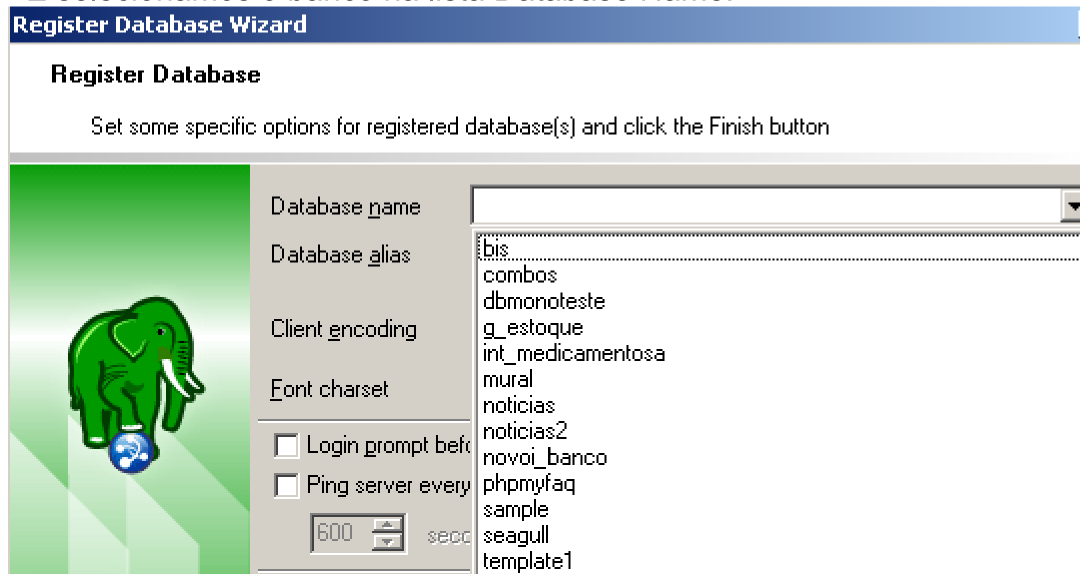
ABRIR UM BANCO EXISTENTE

Caso queiramos trabalhar em um banco que já exista no servidor, vamos apenas abri-lo:
- Após abrir o EMS apenas executamos um duplo clique sobre o nome do banco.
- Caso o nome do banco não esteja aparecendo no EMS clicamos no primeiro botão da barra de ferramentas (Register Database) e informamos os dados do servidor



- Clicamos em Next.

- E selecionamos o banco na lista Database Name:



- E clicamos em Finish

COMO CRIAR UMA CHAVE ESTRANGEIRA (FOREIGN KEY)

- Após criar a tabela e os campos, devemos criar a segunda tabela, que irá se relacionar com a primeira através de um campo (chave estrangeira).
- Vamos supor duas tabelas: pedidos e pedido_itens, que irão se relacionar através do campo código em pedido e cod_pedido em pedido_itens, como abaixo:

pedido (codigo, descricao, data, preco_unitario)

pedido_itens (codigo, cod_pedido, quantidade)

- Para que um campo de uma tabela se relacione com outro, ele deve ser do mesmo tipo que o outro.

- Abra a tabela pedido_itens

- Estando na aba Fields, clique em Foreign Key na coluna do meio com o botão direito e New Foreign Key. Veja o diálogo abaixo:

Add Foreign Key for [public.pedido_itens]

Foreign key name: pedido_itens_fk

Available Fields: codigo, cod_pedido, quantidade | Included Fields

Foreign table

Available Fields | Included Fields

On Delete action: No Action | On Update action: No Action

Match type: Simple, Full | Deferrable: Deferrable, Check Time: Immediate

OK | Cancel | Help

- Acima e à direita selecione o campo que irá se relacionar com a outra tabela (cod_pedido)
- Em Foreign Table selecione a tabela do relacionamento (pedidos)

- Então abaixo e à direita selecione o campo que vai se relacionar com este (codigo) e clique na seta para a direita. Então clique em OK. Veja que em OnDelete action e em On Update Action existem diversas opções. Veja meu tutorial sobre o assunto em:
http://ribafs.clanshosting.com

- Então clique em Commit.

Agora vejamos como fica o código SQL da nossa tabela pedido_itens. Clique na aba DDL e verá:

```
CREATE TABLE "public"."pedido_itens" (
  "codigo" BIGINT NOT NULL,
  "cod_pedido" BIGINT,
  "quantidade" INTEGER,
  CONSTRAINT "pedido_itens_pkey" PRIMARY KEY("codigo"),
  CONSTRAINT "pedido_itens_fk" FOREIGN KEY ("cod_pedido")
    REFERENCES "public"."pedidos"("codigo")
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE
) WITH OIDS;
```

EXPORTANDO UM BANCO COMO SCRIPT

Uma forma muito comum de se exportar um banco é na forma de script, especialmente para abrir num outro servidor do mesmo tipo:

- Clique no menu Tools – Extract Metadata
- Selecione o banco que deseja exportar e clique em Next
- Na combo File name selecione o diretório e nome de arquivo para onde deseja exportar e clique em Salvar. Então clique em Next.
- Escolha se quer exportar somente dados, somente estrutura ou ambos e clique em Next.
- Apenas clique em Finish e ao terminar em Close.

IMPORTANDO UM BANCO DE UM SCRIPT

Esta é a operação inversa da anterior mas com algumas diferenças. Se formos importar tudo, devemos ter aqui apenas um banco vazio.
- Abrir o banco no EMS
- Clicar em Tools – SQL Script
- Ao centro clique em Open script e indique onde está o script a ser importado.
- Se tudo for importado a contendo clique no botão Refresh Tables à direita do botão Create para visualizar a importação.

EXECUTANDO CONSULTAS SQL NO EMS

Uma boa utilidade para o gerenciador EMS é a de teste de consultas SQL.
- Abra o banco, abra o executor de script, digite a consulta em SQL e execute para saber os resultados.
- Sempre que tiver alguma dúvida sobre uma consulta execute aqui para testar antes.

14.5 - Azzurry Clay (modelagem)

Ferramenta de Modelagem Azzurry Clay:
http://www.azzurri.jp/en/software/clay/index.jsp

Visualizador de Objetos e gerador de Diagramas de Entidade Relacionamento (DER), além de fazer engenharia reversa nos bancos existentes.

Um ótimo tutorial online:
http://www.azzurri.jp/en/software/clay/quick_start_guide.jsp?print=on
Uma boa relação de ferramentas para o PostgreSQL pode ser encontrada no site do PostgreSQL Brasil:
https://wiki.postgresql.org.br/wiki/Ferramentas
Outra boa relação no site Data Modeling Tools:
http://www.databaseanswers.com/modelling_tools.htm

14.6 – DbVisualizer

Ótima ferramenta para visualizar bancos e montar o diagrama entidades-relacionamento.
http://www.dbvis.com/products/dbvis/download.html

14.7 – Openoffice2 Base

Usando o OpenOffice para abrir, editar bancos de dados PostgreSQL, como também criar consultas, formulários e relatórios.

Uma das formas de conectar o OpenOffice ao PostgreSQL é usando um driver JDBC do PostgreSQL.
- Antes devemos ter instalado o OpenOffice com suporte a Java
- Baixe daqui:

http://jdbc.postgresql.org/download.html#jars
Para o PostgreSQL 8.1 podemos pegar o JDBC3 -
http://jdbc.postgresql.org/download/postgresql-8.1-405.jdbc3.jar
- Abrir o OpenOffice, pode ser até o Writer – Ferramentas – Opções – Java – Class Path – Adicionar Arquivo (indicar o arquivo postgresql-8.0-313.jdbc2.jar baixado) e OK.
- Abrir o OOBase
- Conectar a um banco de dados existente
- Selecionar JDBC - Próximo
- URL da fonte de dados:

jdbc:postgresql://127.0.0.1:5432/bdteste

Classe do driver JDBC:
org.postgresql.Driver

Nome do usuário - postgres
password required (marque, caso use senha)

Concluir
Digitar um nome para o banco do OOBase
Pronto. Agora todas as tabelas do banco bdteste estão disponíveis no banco criado no OOBase.
Também podemos agora criar consulta com assistentes, criar formulários e relatórios com facilidade.

15 - Apêndices

15.1 – Planejamento e Projeto de Bancos de Dados

Projeto de bancos de dados é genérico e se aplica a qualquer SGBDR.
É com um bom planejamento do banco de dados que se determina o quão eficaz foi o processo de análise.

Introdução
O projeto do banco de dados e também os testes são muito importantes para a eficiência e consistência das informações e do aplicativo. É muito importante gastar algum tempo nesta etapa, pois depois de algum tempo de implantado fica muito trabalhoso alterar estruturas de bancos e aplicativos.

Projetos de banco de dados ineficazes geram consultas que retornam dados inesperados, relatórios que retornam valores sem sentido, etc. Um banco de dados bem projetado fornece um acesso conveniente às informações desejadas e resultados mais rápidos e precisos.

Exemplo de software de administração de SGBD para o PostgreSQL - PGAdmin
Informações de bancos de dados relacionais são armazenadas em tabelas ou entidades no Modelo Entidade Relacionamento (MER).

Dicas sobre Campos
• Não armazenar resultado de cálculos ou dados derivados de outros
• Armazenar todas as informações (campos) separadamente. Cuidado com campos que contém duas ou mais informações.

Selecionando o Campo para a Chave Primária
A chave primária é o campo ou campos que identificam de forma exclusiva cada registro.
• Não é permitido valores nulos na chave nem duplicados
• Caso a tabela não tenha um campo que a identifique, pode-se usar um campo que numere os registros seqüencialmente

Dica de Desempenho: O tamanho da chave primária afeta o desempenho das operações, portanto usar o menor tamanho que possa acomodar os dados do campo.

Exemplo
Tabela - Clientes
Campo - Nome (atributo)
Chave Primária (Primary-Key) - CPF

Todos os campos correspondentes a um único CPF juntamente com seus valores formam um Registro ou Linha (Row)

A correta determinação das tabelas, bem como dos campos é algo primordial no sucesso do projeto do banco de dados.
Chave Primária - obriga que todos os registros terão o campo correspondente à chave primária exclusivo (únicos - unique). Num cadastro de clientes, todos os clientes cadastrados terão um campo CPF exclisivo. Caso se tente inserir dois clientes com o mesmo CPF o banco não permitirá e emitirá uma mensagem de erro acusando tentativa de violação da chave primária.
Exemplos de Campos indicados para chave primária:
• CPF

• CNPJ
• Matrícula de aluno
• Matrícula de funcionário

Uma chave primária pode ser formada por mais de um campo, quando um único campo não é capaz de caracterizar a tabela.

Cada tabela somente pode conter uma única chave primária.

Relacionamentos - Um banco de dados é formado por várias tabelas. Idealmente essas tabelas devem ser relacionadas entre si para facilitar a troca de informações e garantir a integridade. Para relacionar tabelas usamos chaves existentes nas mesmas.

Tipos de Relacionamentos
• Um para um
• Um para vários
• Vários para vários

Relacionamento Um para Um
Aquele onde os campos que fazem o relacionamento são chaves primárias. Cada registro de uma tabela se relaciona com apenas um registro da outra tabela. Este relacionamento não é muito comum.

Exemplo: CorrentistaBanco - Conjuge

Relacionamento Um para Vários ou Vários para Um
Aquele onde uma tabela tem um campo chave primária que se relaciona com outra tabela através de um campo chave estrangeira. É o tipo de relacionamento mais utilizado.

Exemplos:
• Clientes - Pedidos
• Produtos - Itens
• Categorias - Itens
• Fornecedores - Produtos
• NotaFiscal - Produtos

Veja que cada um da esquerda se relaciona com vários do da direita.

Importante:
• O número de campos do relacionamento não precisa ser o mesmo
• O tipo de dados dos campos do relacionamento deve ser igual, assim como o tamanho dos campos e formatos

• Chave primária - Chave estrangeira (um - vários)
Relacionamento Vários para Vários
Este tipo de relacionamento não dá para ser implementado no modelo relacional, portanto sempre que nos deparamos com um deles devemos dividir em dois relacionamentos um para vários (criando uma terceira tabela, que armazenará o lado vários dos relacionamentos).

Exemplo:

Pedidos - Produtos
Cada pedido pode conter vários produtos, assim como cada produto pode estar em vários

pedidos. A saída é criar uma tabela que contenha os itens do pedido.

Pedidos - Pedidos_Itens - Produtos
Pedidos 1 - N Pedidos_Itens N - 1 Produtos

Integridade Referencial
Ela garante a integridade dos dados nas tabelas relacionadas. Um bom exemplo é quando o banco impede que se cadastre um pedido para um cliente inexistente, ou impede que se remova um cliente que tem pedidos em seu nome.

Também se pode criar o banco de forma que quando atualizamos o CPF de um cliente ele seja atualizado em todos os seus pedidos.

Normalização de Tabelas
Normalizar bancos tem o objetivo de tornar o banco mais eficiente.
Uma regra muito importante ao criar tabelas é atentar para que cada tabela contenha informações sobre um único assunto, de um único tipo.

1a Forma Normal
Os campos não devem conter grupos de campos que se repetem nos registros.

Exemplo:
Alunos: matricula, nome, data_nasc, serie, pai, mae

Se a escola tem vários filhos de um mesmo casal haverá repetição do nome dos pais. Estão para atender à primeira regra, criamos outra tabela com os nomes dos pais e a matrícula do aluno.

2ª Forma Normal
Quando a chave primária é composta por mais de um campo.

Devemos observar se todos os campos que não fazem parte da chave
dependem de todos os campos que fazem parte da chave.

Caso algum campo dependa somente de parte da chave, então devemos colocar este campo em outra tabela.

Exemplo:
TabelaAlunos

Chave (matricula, codigo_curso)

avaliacao descricao_curso
Neste caso o campo descricao_curso depende apenas do codigo_curso, ou seja, tendo o código do curso conseguimos sua descrição. Então esta tabela não está na 2ª Forma Normal.

Solução:
Dividir a tabela em duas (alunos e cursos):
TabelaAlunos
Chave (matricula, codigo_curso)

avaliacao

TabelaCursos
codigo_curso
descricao_curso

3ª Forma Normal
Quando um campo não é dependente diretamente da chave primária ou de parte dela, mas de outro campo da tabela que não pertence à chave primária. Quando isso ocorre esta tabela não está na terceira forma normal e a solução é dividir a tabela.

Lembrando: Engenharia Reversa (parte de um banco ou de um script sql e gera o modelo).

Projeto
Fases do Projeto do Banco de Dados
• Modelagem Conceitual
• Projeto Lógico

Observação.: Trataremos apenas de novos projetos.

Modelo Conceitual - Define apenas quais os dados que aparecerão no banco de dados, sem se importar com a implementação do banco. Para essa fase o que mais se utiliza é o DER (Diagrama Entidade-Relacionamento).

Modelo Lógico - Define quais as tabelas e os campos que formarão as tabelas, como também os campos-chave, mas ainda não se preocupa com detalhes como o tipo de dados dos campos, tamanho, etc.

Etapas na Estruturação e Projeto de um Banco de Dados
• Problemas a serem solucionados com o banco de dados
• Determinar o objetivo do banco de dados
• Determinar as tabelas necessárias (cada uma com um único assunto exclusivo)
• Determinar os campos de cada tabela
• Criar um DER
• Verificar a estimativa do crescimento do banco e preparar-se para isso
• Investigar como são armazenadas as informações atualmente e recolher a maior quantidade de informações para o projeto
• Adotar um modelo e justificá-lo
(Os itens acima fazem parte do Modelo Conceitual, abaixo do Lógico)
• Determinar a chave primária de cada tabela. Pode haver tabela sem chave primária.
• Determinar os relacionamentos e seus tipos

Obs.: Somente quando da implementação (modelo físico) serão tratados os detalhes internos de armazenamento. O modelo físico é a tradução do modelo lógico para a linguagem do SGBDR a ser utilizado no sistema.

15.2 – Implementação de Banco de Dados com o PostgreSQL - Modelo Físico

Softwares free de Modelagem e Gerenciamento do PostgreSQL
PGAdmin: (http://www.postgresql.org/ftp/pgadmin3/release/)
EMS: (http://www.sqlmanager.net/en/products/postgresql/manager/download)
DBDesigner: (http://fabforce.net/downloads.php)
DbVisualizer: http://www.dbvis.com/products/dbvis/
Em forma de Plug-ins para Eclipse
QuantumDB: (http://quantum.sourceforge.net/)
Azzurri/Clay: (http://www.azzurri.jp/en/software/clay/download.jsp)
SQLExplorer: (http://sourceforge.net/projects/eclipsesql)

Uma grande e boa relação de softwares de projeto, modelagem e gerenciamento para o PostgreSQL, free e comercial pode ser encontrada em no site oficial so PostgreSQL Brasil: https://wiki.postgresql.org.br/wiki/Ferramentas.

Suporte à Acentuação na Criação de Bancos no PostgreSQL
A codificação default do PG 7.X é a SQL_ASCII
A do PG 8.X é a UNICODE
Ambas tem suporte a acentuação, mas geram problemas no backup/importação.

Codificação
Para um suporte estável à acentuação em português do Brasil uma boa opção é criar o banco passando a codificação (Encoding) LATIN1
ENCODING = 'LATIN1'

Criação do Banco
Criaremos o banco do projeto de testes com o PGAdmin, contendo esquemas, tabelas, views, funções do tipo SQL e Pl/PgSQL, usuários, privilégios, consultas, etc. para ilustrar nosso projeto e servir de base para os testes (em seguida).

Analisar o modelo sugerido e detalhar o banco, tipos de dados de cada campo, tamanho, esquemas do banco, usuários e senhas, privilégios de cada um (cuidados com a segurança), etc.

Ativar o Suporte às Funções Pl/Pgsql (Stored Procedures)
Após ter criado o banco, podemos ativar o suporte a plpgsql.
Ativar suporte a Pl/PgSQL requer dois passos:
- instalar a biblioteca Pl/PgSQL, que é do tipo contrib
- definir a linguagem (como sugerido abaixo)
- Ativando na console do PG depois de conectar ao banco onde ficará o suporte:

CREATE LANGUAGE 'plpgsql' HANDLER plpgsql_call_handler LANCOMPILER 'PL/pgSQL'

- Ativando como superusuário na console (fora dos bancos)

su - postgres
$ createlang plpgsql -U nomesuperuser nomebanco
Ou simplesmente:
$ createlang plpgsql nomebanco

JDBC
Alguns programas em Java o utilizam, como o plugin QuantumDB.

O JDBC para o PostgreSQL encontra-se em:
http://jdbc.postgresql.org/download.html#jars

Veja que para selecionar o arquivo .jar correto, precisamos cruzar a versão do PostgreSQL à esquerda com a versão do JDBC desejado.

Exemplo: Para uso como cliente em sua máquina pelo Quantum DB (no Eclipse) e com PostgreSQL 8.1 baixar o arquivo: 8.1-405 JDBC 3

Esquemas

Definir os esquemas do banco.
Quando o cliente precisa de muitas tabelas, organizadas em várias áreas a saída imediata é a criação de vários bancos de dados. Mas quando da implementação do aplicativo que irá utilizar estes bancos os desenvolvedores se depararão com a dificuldade de comunicação e acesso entre os bancos, já que com uma única conexão terão acesso a todos os objetos do banco. É muito útil para estes casos criar um único banco e neste criar vários esquemas, organizados por áreas: pessoal, administracao, contabilidade, engenharia, etc.

Mas e quando uma destas áreas tem outras sub-áreas, como por exemplo a engenharia, que tem reservatórios, obras, custos e cada um destes tem diversas tabelas. O esquema engenharia ficará muito desorganizado. Em termos de organização o ideal seria criar um banco para cada área, engenharia, contabilidade, administração, etc. E para engenharia, por exemplo, criar esquemas para cada subarea, custos, obras, etc. Mas não o ideal em termos de comunicação e acesso entre todos os bancos.

Criar Esquema
Num gerenciador do PG entra-se no banco e nesse cria-se o esquema.

Ou

CREATE SCHEMA nomeesquema;

Acessando Objetos de Esquemas
Para acessar um esquema devemos passar seu caminho:

nomeesquema.nometabela

Ou

nomebanco. nomeesquema.nometabela

Criando Tabela em Esquema
CREATE TABLE nomeesquema.nometabela (
...
)

Criando Esquema e tornando um Usuário dono
CREATE SCHEMA nomeesquema AUTHORIZATION nomeusuario;
Removendo privilégios de acesso a usuário em esquema
REVOKE CREATE ON SCHEMA public FROM PUBLIC
Com isso estamos tirando o privilégio de todos os usuários acessarem o esquema public.

Acesso aos Esquemas
Quando se cria um banco no PostgreSQL, por default, ele cria um esquema público (public) no mesmo e é neste esquema que são criados todos os objetos quando não especificamos o esquema. A este esquema public todos os usuários do banco têm livre acesso, mas aos demais existe a necessidade de se dar permissão para que os mesmos acessem.

Tabelas
O PostgreSQL permite adicionar privilégios por objeto do banco: tabela, esquema, banco, etc. Em termos de segurança é importante, em geral, que os privilégios sejam adicionados ao usuário por tabela, cada tabela tendo um dono e cada dono tendo seus específicos privilégios.

Dica de Desempenho: Na criação das tabelas alertar para a criação de índices para os campos envolvidos na cláusula WHERE. Isso tornará essas consultas mais rápidas.

Views
Juntamente com as funções armazenadas (stored procedures) as views são boas alternativas para tornar o código mais simples e o aplicativo mais eficientes, já que parte do processamento feito pelo código agora já está pronto e debugado no banco, o que torna o código mais rápido e eficiente. O uso de views e de funções armazenadas em bancos é semelhante ao uso de funções e classes no código.

Dica: para uso de views, sintaxe de funções internas e uso de cláusulas SQL no PostgreSQL, tutoriais de EMS e vários outros sobre PostgreSQL, além de PHP, JavaScript, etc, confira o site abaixo:
http://ribafs.byethost2.com ou
http://ribafs.tk

Criação do Banco Tutorial sobre PGAdmin para criar o banco funcionarios.
Bem, de posse do script .sql acima, praticamente o que teremos de fazer é criar um banco vazio no PGAdmin.

Abrir o PGAdmin
Caso não tenha salvado a senha ele pedirá sempre que iniciar

Ao abrir clique com o botão direito à direita em Databases e em New Database.
- No diálogo New Database entre com o Name do banco (funcionarios), o Owner (postgres). Idealmente mudar o nome do superusuario default para um nome mais seguro, assim como a senha (mínimo de 8 caracteres, misturando letras e algarismos e idealmente com símbolos).

Também altere Encoding (codificação) para LATIN1.
- Então selecione o banco funcionarios e clique no botão SQL acima.
- Clique no botão open file para indicar o nosso script sql gerado anteriormente.
- Clique na setinha verde (Execute query)

Eventuais Correções:
Caso receba mensagens de erro sobre tipo UNSIGNED, verifique o script e remova todas as ocorrências de UNSIGNED e execute novamente. Como o DBDesigner foi projetado para o MySQL um outro erro que pode ocorrer é com a string AUTO_INCREMENT, que também deve ser removida e novamente devemos executar o script. Feitas estas correções o script executa normalmente e cria o nosso banco funcionarios.

Então verifique à esquerda que o banco já contém as 3 tabelas de acordo com o script.
Engenharia Reversa
Um ótimo software para conexão ao PostgreSQL, engenharia reversa (gera diagramas ER dos bancos existentes) e exporta os diagramas em forma de imagens: DbVisualizer.

15.3 - Integridade Referencial - Postgresql

Tradução livre do documentação "CBT Integrity Referential": http://techdocs.postgresql.org/college/002_referentialintegrity/.

Integridade Referencial (relacionamento) é onde uma informação em uma tabela se refere à informações em outra tabela e o banco de dados reforça a integridade.

Tabela1 ------------> Tabela2
Onde é Utilizado?

Onde pelo menos em uma tabela precisa se referir para informações em outra tabela e ambas precisam ter seus dados sincronizados.
Exemplo: uma tabela com uma lista de clientes e outra tabela com uma lista dos pedidos efetuados por eles.

Com integridade referencial devidamente implantada nestas tabelas, o banco irá garantir que você nunca irá cadastrar um pedido na tabela pedidos de um cliente que não exista na tabela clientes.

O banco pode ser instruído para automaticamente atualizar ou excluir entradas nas tabelas quando necessário.
Primary Key (Chave Primária) **-** é o campo de uma tabela criado para que as outras tabelas relacionadas se refiram a ela por este campo. Impede mais de um registro com valores iguais. É a combinação interna de UNIQUE e NOT NULL.

Qualquer campo em outra tabela do banco pode se referir ao campo chave primária, desde que tenham o mesmo tipo de dados e tamanho da chave primária.

Exemplo:
clientes (codigo INTEGER, nome_cliente VARCHAR(60))

codigo	nome_cliente
1	PostgreSQL inc.
2	RedHat inc.

pedidos (relaciona-se à Clientes pelo campo cod_cliente)
cod_pedido cod_cliente descricao

Caso tentemos cadastrar um pedido com cod_cliente 2 ele será aceito.
Mas caso tentemos cadastrar um pedido com cod_cliente 3 ele será recusado pelo banco.

Criando uma Chave Primária
Deve ser criada quando da criação da tabela, para garantir valores exclusivos no campo.

```
CREATE TABLE clientes(cod_cliente BIGINT, nome_cliente VARCHAR(60)
PRIMARY KEY (cod_cliente));
```

Criando uma Chave Estrangeira (Foreign Keys)
É o campo de uma tabela que se refere ao campo Primary Key de outra.
O campo pedidos.cod_cliente refere-se ao campo clientes.codigo, então pedidos.cod_cliente é uma chave estrangeira, que é o campo que liga esta tabela a uma outra.

```
CREATE TABLE pedidos(
cod_pedido BIGINT,
cod_cliente BIGINT REFERENCES clientes,
descricao VARCHAR(60)
);
```

Outro exemplo:

```
FOREIGN KEY (campoa, campob)
REFERENCES tabela1 (campoa, campob)
ON UPDATE CASCADE
ON DELETE CASCADE);
```

Cuidado com exclusão em cascata. Somente utilize com certeza do que faz.
Dica: Caso desejemos fazer o relacionamento com um campo que não seja a chave primária, devemos passar este campo entre parênteses após o nome da tabela e o mesmo deve obrigatoriamente ser UNIQUE.

```
...
cod_cliente BIGINT REFERENCES clientes(nomecampo),
...
```

Parâmetros Opcionais: ON UPDATE parametro e ON DELETE parametro.

ON UPDATE paramentros:

***NO ACTION (RESTRICT)* -** quando o campo chave primária está para ser atualizado a atualização é abortada caso um registro em uma tabela referenciada tenha um valor mais antigo. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO Ao tentar usar "UPDATE clientes SET codigo = 5 WHERE codigo = 2. Ele vai tentar atualizar o código para 5 mas como em pedidos existem registros do cliente 2 haverá o erro.

***CASCADE* (Em Cascata) -** Quando o campo da chave primária é atualizado, registros na tabela referenciada são atualizados.

Exemplo: Funciona: Ao tentar usar "UPDATE clientes SET codigo = 5 WHERE codigo = 2. Ele vai tentar atualizar o código para 5 e vai atualizar esta chave também na tabela pedidos.
***SET NULL (atribuir NULL)* -** Quando um registro na chave primária é atualizado, todos os campos dos registros referenciados a este são setados para NULL.

Exemplo: UPDATE clientes SET codigo = 9 WHERE codigo = 5;
Na clientes o codigo vai para 5 e em pedidos, todos os campos cod_cliente com valor 5 serão setados para NULL.

***SET DEFAULT (assumir o Default)* -** Quando um registro na chave primária é atualizado, todos os campos nos registros relacionados são setados para seu valor DEFAULT.

Exemplo: se o valor default do codigo de clientes é 999, então

UPDATE clientes SET codigo = 10 WHERE codigo = 2. Após esta consulta o campo código com valor 2 em clientes vai para 999 e também todos os campos cod_cliente em pedidos.

ON DELETE parametros:

***NO ACTION (RESTRICT)* - Quando um** campo de chave primária está para ser deletado, a exclusão será abortada caso o valor de um registro na tabela referenciada seja mais velho. Este parâmetro é o default quando esta cláusula não recebe nenhum parâmetro.

Exemplo: ERRO em DELETE FROM clientes WHERE codigo = 2. Não funcionará caso o cod_cliente em pedidos contenha um valor mais antigo que codigo em clientes.

***CASCADE* -** Quando um registro com a chave primária é excluído, todos os registros relacionados com aquela chave são excluídos.

***SET NULL* - Quando um** registro com a chave primária é excluído, os respectivos campos na tabela relacionada são setados para NULL.

***SET DEFAULT* -** Quando um registro com a chave primária é excluído, os campos respectivos da tabela relacionada são setados para seu valor DEFAULT.

Excluindo Tabelas Relacionadas

Para excluir tabelas relacionadas, antes devemos excluir a tabela com chave estrangeira.

Tudo isso está na documentação sobre CREATE TABLE:
http://www.postgresql.org/docs/8.0/interactive/sql-createtable.html

ALTER TABLE
http://www.postgresql.org/docs/8.0/interactive/sql-altertable.html

Chave Primária Composta (dois campos)
CREATE TABLE tabela (
codigo INTEGER,
data DATE,
nome VARCHAR(40),
PRIMARY KEY (codigo, data)
);

15.4 - Dicas Práticas de uso do SQL

Armazenar Arquivos Binários no Próprio Banco

Utilize a contrib LO para esta finalidade.

Lembre que como é uma contrib normalmente não vem ligada e temos que ligar especificamente ao banco onde queremos utilizar.

Ligando, de dentro do banco usar o comando \i:

Acesse o diretório lo das contribs do PostgreSQL:

/usr/local/src/postgresql-8.1.3/contrib/lo
Então execute o comando "make install".

Acesse o banco e:

\i /usr/local/src/postgresql-8.1.3/contrib/lo/lo.sql

Para usar veja o README.lo no diretório lo e também a documentação oficial do PostgreSQL:
Português do Brasil - Capítulo 28: http://pgdocptbr.sourceforge.net/pg80/largeobjects.html
Inglês - Capítulo 29: http://www.postgresql.org/docs/8.1/interactive/largeobjects.html

Nomes de Campos com espaço ou acento
Devem vir entre aspas duplas.

Comentários
Em SQL os comentários mais utilizados são da seguinte forma:
SELECT * FROM tabela; - - Este é um comentário
- - Este é outro comentário
Também são aceitos os comentários herdados do C:
/* Comentário herdado do C e válido em SQL */

Dicas Práticas de Uso do SQL

Testar se campo é de e-mail, ou seja, se contém um @:

SELECT POSITION('@' IN 'ribafs@gmail.com') > 0

select 'ribafs@gmail.com' ~ '@'
select 'ribafs@gmail.com' like '%@%'
select 'ribafs@gmail.com' similar to '%@%.%';

Alguns da lista de PHP (phpfortaleza@yahoogrupos.com.br - groups.yahoo.com).

Temos um campo (insumo) com valores = 1, 2, 3, ... 87
Queremos atualizar para 0001, 0002, 0003, ... 0087

```
UPDATE equipamentos SET insumo = '000' || insumo WHERE LENGTH(insumo) = 1;
UPDATE equipamentos SET insumo = '00' || insumo WHERE LENGTH(insumo) = 2;
```

Outra saída mais elegante ainda:

```
UPDATE equipamentos SET insumo = REPEAT('0', 4-LENGTH(insumo)) || insumo;
```

INSERINDO COM SELECT
Tendo uma tabela com registros e outra para onde desejo incluir registros daquela

```
INSERT INTO equipamentos2 SELECT grupo, insumo, descricao, unidade from equipamentos2;
insert into engenharia.precos (insumo_grupo,insumo) select grupo,insumo from engenharia;
```

Com CAST

```
insert into engenharia.insumos (grupo,insumo,descricao,unidade) select
grupo,insumo,descricao, CAST(unidade AS int2) AS "unidade" from engenharia.apagar

insert into engenharia.insumos (grupo,insumo,descricao,unidade) select
grupo,insumo,descricao, cast(unidade AS INT2) AS unidade from engenharia.apagar


select trim(length(bairro)) from cep_tabela where cep='60420440'; -- Montese, Retorna 7
```

Através do PHP

```
$conn = pg_connect("host=10.40.100.186 dbname=apoena user=_postgresql");
for($x=10;$x<=87;$x++){
	$sql="update engenharia.precos set custo_produtivo = (select custo_produtivo from
	engenharia.apagar where insumo='$x') where insumo='00' || '$x'";
	$ret=pg_query($conn,$sql);
}
```

Diferença em Dias entre duas Datas

```
SELECT DATE '2006-03-29' – DATE '2006-01-12';

SELECT (CAST('10/02/2005' AS DATE) - CAST('10/01/2006'));
```

POPULAR BANCO COM MASSA DE TESTES

Script el Perl

```
#!/usr/bin/perl
$count = 1;
$arquivosaida = "populate.sql";
@chars = ("A" .. "Z", "a" .. "z", 0 .. 9);
@numbers = (1 .. 9);
@single_chars = ("a" .. "e");
$totalrecords = 5000; # 5 milhoes

open(OUTPUT, "> $arquivosaida");
print OUTPUT "DROP TABLE index_teste;\n";
print OUTPUT "CREATE TABLE index_teste (";
print OUTPUT "codigo INT, nome VARCHAR(10), numero INT, letra CHAR(1)";
print OUTPUT ");\n";
print OUTPUT "COPY index_teste (codigo, nome, numero, letra) FROM stdin;\n";
while ($count <= $totalrecords){
   $randstring = join("", @chars [map{rand @chars} ( 1 .. 8 ) ]);
   $randnum = join("", @numbers [map{rand @numbers} ( 1 .. 8 ) ]);
   $randletter = join("", @single_chars [map{rand @single_chars} (1)]);
   print OUTPUT
   #print OUTPUT "INSERT INTO index_teste
VALUES($count,'$randstring',$randnum,'$randletter');\n";
   $count."\t".$randstring."\t".$randnum."\t".$randletter."\n";
   $count++;
};
#print OUTPUT "\n";
#print OUTPUT "\nCREATE INDEX indexteste_codigo_index ON index_teste(codigo);\n";
#print OUTPUT "CREATE INDEX indexteste_numero_index ON index_teste(numero);\n";
#print OUTPUT "VACUUM ANALYZE index_teste;\n";
close OUTPUT;
```

Via PHP

```
$con=pg_connect("host=127.0.0.1 user=postgres password=postgres");

function datediff($data_final, $data_inicial){
   global $con;
   $str="SELECT DATE '$data_final' - DATE '$data_inicial'";
   $recordset = pg_query($con, $str);
   $diferença=pg_fetch_array($recordset);
   return $diferença[0];
}

echo "Diferença: " . datediff("1969-01-08", "1968-10-16");
```

Ajustando o formato da Data do Sistema

SHOW DATESTYLE;
SET DATESTYLE TO ISO;   YYYY-MM-DD HH:MM:SS
SET DATESTYLE TO PostgreSQL;   Formato tradicional do PostgreSQL (
SET DATESTYLE TO US; MM/DD/YYYY
SET DATESTYLE TO NONEUROPEAN, GERMAN;   DD.MM.YYYY
SET DATESTYLE TO EUROPEAN;   DD/MM/YYYY
Obs.: De forma permanente ajustar o postgresql.conf.

Outros usos para SHOW:

SHOW server_version;
SHOW server_encoding; -- Idioma para ordenação do texto (definido pelo initdb)
SHOW lc_collate;   -- Idioma para classificação de caracteres (definido pelo initdb)
SHOW all;   -- Mostra todos os parâmetros

Também podemos setar o datestyle quando alteramos um banco:
ALTER DATABASE nomebanco SET DATESTYLE = SQL, DMY;

Também pode ser atribuído juntamente com o Usuário:
ALTER ROLE nomeuser SET DATESTYLE TO SQL, DMY;

Ajustando uma Faixa de Registros com LIMIT and OFFSET
SELECT isbn, title, publication FROM editions NATURAL JOIN books AS b (book_id)
ORDER BY publication DESC LIMIT 5;

SELECT isbn, title, publication FROM editions NATURAL JOIN books AS b (book_id)
ORDER BY publication DESC LIMIT 5 OFFSET 2;

Trará 5 registros, iniciando do segundo.

fsync - checa integridade dos dados gravados no banco, vindos dos logs. Vem ligado por padrão

Gargalo de SGBDs - leitura/gravação (I/O) de discos.
Ligar/Desligar fsync no:
postgresql.conf, setar para
fsync=true – Nunca deve ficar false

REORDENAR CAMPOS DE TABELA
Se você estiver falando da ordem dos campos na tabela não existe razão para isso no modelo relacional.

Você sempre pode especificar os campos desejados, e na ordem desejada,
no SELECT.
Se necessário você pode criar uma view:
CREATE VIEW nome_view AS SELECT id,cpf,nome FROM sua_tabela;

Se ainda não estiver satisfeito pois quer suas tabelas "bonitinhas" e organizadas:
1. CREATE TABLE novo_nome AS SELECT id,cpf,nome FROM sua_tabela;
2. DROP TABLE sua_tabela;
3. ALTER TABLE novo_nome RENAME TO sua_tabela;
Osvaldo (Na lista **PostgreSQL-Brasil**).

Calculando a Memória a ser usada pelo PostgreSQL

* Shared Buffers
Exemplo de 1GB RAM
A shared buffers será 25% da RAM
256 * 1024 / 8 = 32768
logo shared_buffers = 32768

* Shared Memory
A Shared Memory será igual a shared buffer + (de 10 a 20)%
Shared Memory = 256MB + 15%
256MB + 15% = 295 MB
295MB = 295 * 1024 * 1024 = 309329920

No Linux:
/etc/sysctl.conf
kernel.shmmax = 309329920
kernel.shmall = 309329920
kernel.shmmni = 1

Comando para alterar as variáveis do kernel sem re-iniciar o Linux:
sysctl -w kernel.shmmax=309329920
sysctl -w kernel.shmall=309329920
sysctl -w kernel.shmmni=1
Dicas de instalação do PostgreSQL em GNU/Linux.
* Utilizar HD do tipo SATA
* Criar uma partição exclusiva para os dados. Ex: /database
* Utilizar nesta partição o sistema de arquivos XFS
* Deixar nesta partição apenas os flags: RW,NOATIME

Do site: http://www.gescla.com.br/oficina_postgre.asp

Criação de Tipos de Dados

CREATE TYPE "img" (input = "int4in", output = "int4out", internallength = 4, externallength = 10, delimiter = ",", send = "int4out", receive = "int4in", passedbyvalue, alignment = int, storage = plain);

Uso:

create table imagens (codigo int8, descricao varchar(60), imagem img);

Construtor de Matriz

- Matriz unidimensional - array[2,4,6+2]
- SELECT array[2,4,6+2]; -- Retorna 2,4,8
- Multidimensional - composta por duas ou mais matrizes unidimensionais:
- Obs.: O índice do valor da matriz construído com ARRAY sempre começa com um.

- Ao criar uma tabela podemos usar matriz em seus tipos de dados, ao invés de tipos simples.

- Exemplo:

- CREATE TABLE testematriz (codigo INT [], nome char[30][30]);

- array[array[2,4,6],array[1,3,5]] ou
  array[[2,4,6],[1,3,5]]

- Com subconsultas. Entre parênteses e não concletes.

- select array(select oid from pg_proc where proname like 'bytea%');
  Retorna:
  1244,31,1948,1949,1950,1951,1952,1953,1954,2005,2006,2011,2412,2413,16823

ENCONTRAR REGISTROS DUPLICADOS

SELECT DISTINCT cep FROM cep_tabela
WHERE cep IN (SELECT cep FROM cep_tabela AS Tmp GROUP BY cep,tipo,logradouro, bairro, municipio,uf HAVING Count(*) >1) ORDER BY cep;
(Adaptação de consulta gerada pelo assistente Encontrar duplicadas do Access).

Ou:
select count(*) as quantos, cep from cep_tabela group by cep having count(*) > 1;

REMOVER DUPLICADOS
Para tabelas criadas WITH OIDS:
DELETE FROM cep_tabela2 WHERE oid NOT IN
(SELECT min(oid) FROM cep_tabela2 GROUP BY cep, tipo, logradouro, bairro, municipio, uf);

Do exemplo 8.10 do manual em português do Brasil.

Ou:

Criando uma segunda tabela que conterá somente os registros exclusivos e ainda guarda uma cópia da tabela original:

CREATE TABLE cep_tabela2 AS SELECT cep, tipo, logradouro, bairro, municipio, uf FROM cep_tabela GROUP BY cep, tipo, logradouro, bairro, municipio, uf ORDER BY cep;

Caso não importe qual das duplicatas irá permanecer:
CREATE TABLE tab_temp AS SELECT DISTINCT * FROM tabela;
DROP tabela;
ALTER TABLE tab_temp RENAME TO tabela;
(Dica de Osvaldo Rosario Kussama na lista de PostgreSQL Brasil)

Delimitadores
A maioria dos tipos de dados tem seus valores delimitados por apóstrofos (‘), a exemplo de:
- caracteres
- data/hora
- monetário
- boleanos
- binários
- geométricos
- arrays

A exceção é para os demais tipos numéricos: date ‘18/12/2005’ numeric 12345.45

Caracteres Especiais
Para poder escrever uma barra no valor de uma constante, usa-se duas barras:

```
SELECT '\\Barra';
```

Para escrever um apóstrofo usa-se dois apóstrofos:

```
SELECT 'Editora O''Reyle';
```

P PostgreSQL também permite o uso de caracteres de escape para escrever caracteres especiais:

```
SELECT 'Editora O\'Reyle';
```

Concatenação de expressões no terminal:

```
SELECT 'Concate'
'nação';
```

Equivale a:

```
SELECT 'Concatenação';
```

Quando resolvendo expressões matemáticas usar parênteses para tornar mais claras as precedências.

Convertendo para Números

```
SELECT TO_NUMBER('0' || '1,500.64',99999999.99);
```

Total de 8 dígitos com 2 decimais.

Variáveis no psql

```
\pset null '(nulo)'     -- traduzindo null por nulo

SELECT NULL;

\set variavel 14        -- Dando valor 14 à variável
SELECT :variavel;
```

phpPgGIS
http://www.geolivre.org.br/modules/news/
Em mais um grande lançamento, a OpenGEO coloca à disposição da comunidade uma ferramenta extremamente últil para gerência de dados geográficos no PostgreSQL. O phpPgGIS é mais um produto da OpenGEO que contempla uma demanda na área de Geotecnologias e visa atender usuários do mundo inteiro.

Desenvolvido com base no phpPgAdmin, o phpPgGIS utiliza o MapServer para visualizar o conteúdo espacial dos campos do PostGIS com muita simplicidade (um clique). Seqüências de códigos complexos (campo de geometria) agora podem ser vistos num mapa.

O OpenGEO tem atuado no mercado brasileiro de Geotecnologias com soluções inovadoras com base em software livre e já ganhou referência internacional com alguns importantes projetos como o Open 3D GIS e o GeoLivre Linux.

Este sistema vai integrar a solução de Hosting que a empresa deverá lançar nas próximas semanas.

Algumas Definições

Cursor
É um ponteiro para uma linha (registro).

Replicação
É a distribuição de dados corporativos para vários locais ou filiais de uma empresa, oferecendo confiabilidade, tolerância a falhas, melhor desempenho e capacidade de gerenciamento.

Criptografia
Seu objetivo é tornar os dados comuns em bits de aparência completamente aleatória.

MAIÚSCULAS E MINÚSCULAS NO PORTGRESQL

Ao digitar nomes de tabelas e campos em Maiúsculas eles serão convertidos automaticamente para minúsculas, a não ser que sejam digitados entre aspas duplas:

SELECT * FROM "CLIENTES";

Recomendação: evitar o uso de maiúsculas e de acentos em nomes de bancos, tabelas e campos.

POSTGRESQL NÃO CONECTA?

Do site do Rodrigo (HJort)

- Pingar no IP
- Verificar o pg_hba.conf - host, banco, usuário IP e senha
- Caso apareça "Is the server running on host.."
- Testar com telnet IP porta (Ctrl+C para sair)
- No postgresql.conf - listen_addresses = 'IP'
- Salvar e restartar o SGBD.

Contador de Resultados

Indicado para consultas e relatórios (não grava)

CREATE TEMP SEQUENCE seq;
SELECT nexval('seq'), * FROM esquema.tabela;
(Salvador S. Scardua na lista PostgreSQL Brasil)

LIMITES DO POSTGRESQL
Tamanho de um Banco de Dados - ilimitado
Tamanho de uma tabela - 32 TB
Quantidade de registros por tabela - ilimitados
Quantidade de campos por tabela - 250 a 1600 (depende do tipo)
Quantidade de índices por tabela - ilimitados

15.5 – Dicas sobre Desempenho e Otimizações do PostgreSQL
Existem duas principais formas de melhorar o desempenho de SGBDs: uma é melhorando o hardware, com CPUs, RAM, Discos mais novos, rápidos e confiáveis. A outra é otimizando as consultas realizadas nos bancos (usando VACUUM, VACUUM ANALYZE, EXPLAIN, criando CLUSTERS, entre outros).

Uma das medidas básicas adotada para melhorar o desempenho de tabelas com grandes quantidades de registros e especialmente com muitos acessos, é a inclusão de índices estratégicos. Além da chave primária é importante inserir índices em campos que compõem a cláusula WHERE, que fazem parte de cláusulas ORDER BY, GROUP BY entre outras. Em consultas com WHERE de vários campos usando OR, não adianta inserir índice, pois não será utilizado pelo PostgreSQL, somente usando AND.

Na criação do banco de dados e especialmente na criação das consultas é muito importante atentar para um bom planejamento, normalização, consultas otimizadas tendo em vista o planejador de consultas do PostgreSQL através do uso dos comandos EXPLAIN e ANALYZE.

A administração do PostgreSQL também é muito importante para tornar o SGBD mais eficiente e rápido. Desde a instalação e configuração temos cuidados que ajudam a otimizar o PostgreSQL.

Adaptação do Artigo sobre otimização do PostgreSQL do Diogo Biazus e do original do Bruce Momjian (http://www.ca.postgresql.org/docs/momjian/hw_performance).

Hardware
No computador as informações são manipuladas pelos registradores da CPU, pelo cache da CPU, pela memória RAM e pelos discos rígidos.

Na prática as informações utilizadas com mais freqüência são colocadas próximas à CPU.
Quem determina que informações devem ficar nos registradores são os compiladores.
Cache da CPU guarda ar informações utilizadas recentemente.
O Sistema Operacional controla o que está armazenado na RAM e o que mandar para o disco rígido.

Cache e Registradores da CPU não podem ser otimizados diretamente pelo administrador do SGBD. Efetivamente otimização em bancos de dados envolvem aumento da quantidade de informações úteis na RAM, prevenindo acesso a disco sempre que possível.

Não é tarefa simples de ser colocada em prática, pois a memória RAM guarda muitas outras informações: programas em execução, pilhas e dados de programas, memória cache compartilhada do PostgreSQL, cache do buffer de disco do kernel e kernel.

Otimização correta de bancos de dados procura manter a maior quantidade possível de informações do banco na memória RAM ao mesmo tempo que não afeta as demais áreas do sistema operacional.

Existem dois tipos de configuração de memória no PostgreSQL, a compartilhada e a individual. A compartilhada tem um tamanho fixo, ela é alocada sempre que o PostgreSQL inicializa e então é compartilhada por todos os clientes. Já a memória individual é tem um tamanho variável e é alocada separadamente para cada conexão feita ao SGBD.

Memória Cache Compartilhada do PostgreSQL

O PostgreSQL não altera as informações diretamente no disco. Ao invés disso ele solicita que os dados sejam lidos da memória cache compartilhada do PostgreSQL. O cliente PostgreSQL então lê e escreve os blocos e finalmente escreve no disco.

Clientes que precisam acessar tabelas primeiro procuram pelos blocos necessários no cache. Caso estejam aí então continuam processando normalmente. Caso contrário é feita uma solicitação ao sistema operacional para carregar os blocos. Os blocos são carregados do cache de buffer de disco do kernel ou diretamente do disco. Estas operações podem ser onerosas (lentas).

Na configuração default do PostgreSQL 8.1.3 ele aloca 1000 shared buffers. Cada buffer usa 8KB, o que soma 8MB. Aumentando o número de buffers fará com que os clientes encontrem as informações que procuram em cache e evita requisições onerosas ao sistema operacional. Mas cuidado, pois se aumentar muito a memória compartilhada (shared buffers) pode acarretar uso da memória virtual (swap). As alterações podem ser feitas através do comando postmaster na linha de comando ou através da configuração do valor do shared_buffers no postgresql.conf.

Que Porção da RAM Reservar para o PostgreSQL?

A maior porção útil que não atrapalhe os outros programas.
Nos sistemas UNIX as informações saem da RAM (quando insuficiente) para o swap. Ruim é quando as informações voltam do swap para a RAM, pois então os programas são suspensos até que as mesmas sejam carregadas.

Tamanho da Cache

Imaginemos que o PostgreSQL shared buffer cache seja suficiente para manipular uma tabela inteira. Repetidas buscas seqüenciais da tabela não devem necessitar de acesso ao disco já que todos os dados já estão em cache. Agora vamos imaginar que o cache é menor que a tabela, então neste caso as informações irão para o disco (swap) e terão um desempenho bem inferior.
Tamanho Adequado da Shared Buffer Cache

Idealmente a PostgreSQL shared buffer cache (*Memória Cache Compartilhada do PostgreSQL*) deve ser:

- Grande o suficiente para conseguir manipular as tabelas mais comumente acessadas.
- Pequena o bastante para evitar atividades de swap pagein.

Exemplo:
Por exemplo queremos x MB para memória compartilhada
(x / 8) * 1024 = Resultado a ser configurado em shared_buffer

Se x = 768 MB
(768 / 8) * 1024
Resultado a ser configurado em shared_buffer = 98304

Para informações sobre uma configuração do kernel para que vários sistemas operacionais trabalharem com o PostgreSQL:
http://developer.postgresql.org/docs/postgres/kernel-resources.html

Memória Individual (Sort Memory)
Principalmente utilizada em ordenações de registros das tabelas, em operações de criação de índices, ordenação (order by), merge join, etc.
Esta memória pode ser configurada através do parâmetro sort_mem do postgresql.conf.
Para a configuração leve em conta sua memória disponível (incluindo a memória já alocada para o shared buffers), também o número médio de conexões e o uso da memória virtual (swap).

Exemplo:

Considerando um servidor dedicado (rodando somente o servidor PostgreSQL), com memória RAM de 1,5GB e com até 10 conexões simultâneas com o SGBD:

```
shared_buffers = 80000    # 80.000 blocos de 8KB = 625 MB
sort_mem       = 64000    # tamanho em KB = 62,5 MB, para cada usuário com
                          # 10 usuários = 526 MB
vacuum_mem = 2000
```

Por exemplo: queremos x KB para memória individual sort_men
(x * 1024) = resultado para memória individual

x = 16

(16 * 1024) = sort_mem = 16384

Seria bom mudar também memória para vaccum
vacuum_mem = 131072 (mesmo cálculo do sort_mem)

Uso de Vários Discos
Em sistemas com mais de um disco podemos melhorar a performance do mesmo distribuindo algumas tarefas entre discos diferentes.
Supondo que temos dois HDs, hda e hdb:

Movendo os logs de transação para outro disco:
- Parar o PostgreSQL
- Montar hdb em /mnt/hdb
- Mover a pasta /usr/local/pgsql/data/pg_xlog para o /mnt/hdb
- Criar um link simbólico para o diretório original:
  ln -s /mnt/hdb/pg_xlog /usr/local/pgsql/data/pg_xlog
- Banco - /usr/local/pgsql/data (no hda)
- Logs - /usr/local/pgsql/data/pg_xlog (link simbólico para /mnt/hdb/pg_xlog).

Os logs de transação são os únicos registros que não podem ter o seu salvamento em disco adiado sem comprometer a segurança do sistema.

Mover os índices para um HD diferente de onde estão as tabelas:

- Parar PostgreSQL
- Mover os índices para o hdb
- Criar link simbólico para o local original

Para recriar os índices em outro Tablespace:
ALTER TABLE nometabela DROP CONSTRAINT nomeconstraint;

CREATE INDEX nome_idx ON nometabela (nomecampo) TABLESPACE nometablespace;
ALTER TABLE nometabela ADD CONSTRAINT nome_pk PRIMARY KEY (nomecampo);

ALTER INDEX nome_idx SET TABLESPACE nometablespace;

Ainda podemos separar astabelas mais utilizadas para o hdb, utilizando o comando tablespace no PostgreSQL 8.1.3 podemos fazer isso:

- Criar diretório /mnt/hdb/hotcluster e tornar postgres seu dono
CREATE TABLESPACE hotcluster OWNER postgres LOCATION '/mnt/hdb/hotcluster';

Criando um banco no novo cluster:
CREATE DATABASE hotbanco TABLESPACE = hotcluster;

Exportar as tabelas para este banco.

Uso de Mais de Um Processador

Atualmente o PostgreSQL está otimizado para uso de vários processadores, reforçando que cada conexão é gerenciada por um processo diferente.
Sistemas de Arquivos

Para sistemas BSD usa-se o tradicional UFS, que é robusto, rápido e tem a vantagem em relação ao PostgreSQL, de possuir os blocos de disco com um tamanho padrão de 8KB.

Para quem utiliza Linux as sugestões vão para EXT3 e ReiserFS.

Checkpoints

O wal_files é o parâmetro do postgresq.lconf que determina o número de arquivos usados pelo PostgreSQL para armazenar os logs de transação. Estes arquivos focam em pg_xlog, na pasta de dados.

Para que apareçam as datas e horas nos arquivos de logs usa-se no postgresql.conf:
log_timestamp = true

Para reduzir a freqüência dos checkpoints devemos aumentar o parâmetro do postgresql.conf:
checkpoint_segments = 3 (valor default)

O PostgreSQL não precisa de muito ajuste. Boa parte dos parâmetros é automaticamente ajustada para uma performance ótima. O cache size e sort size são dois parâmetros que o administrador pode controlar para ter um melhor uso da memória.

Tradução do Tutorial "Tuning PostgreSQL for Performance"
De Shridhar Daithankar e John Berkus

Shared Buffers
Definem um bloco de memória que o PostgreSQL usará para lidar com requisições que estão aguardando atenção no buffer do kernel e na CPU.

Deve ser manipulada com cuidado, pois simplesmente ampliada pode prejudicar a performance. Esta é a área que o PostgreSQL usa atualmente para trabalhar. Ela deve ser suficiente para controlar a carga do servidor do SGBD, do contrário o PostgreSQL irá iniciar empurrando dados para arquivos e isto irá prejudicar a performance geral. Esta é a principal configuração em termos de performance.

Seu valor deve ser configurado tendo em vista o tamanho do conjunto de bancos que se supões que no máximo o servidor irá carregar e da memória RAM (ter em mente que a memória RAM utilizada pelos demais aplicativos do servidor não estarão disponíveis).
Recomendações:
- Iniciar com 4MB (512) Workstation
- Médio tamanho do conjunto de bancos de dados e 256 a 512MB disponível de RAM:
  16-32MB (2948 a 4096)
- Grande conjunto de bancos de dados e muita memória RAM disponível (1 a 4GB):
  64 -256MB (8192 a 32768)

Obs.: Até para um conjunto de bancos de dados (dataset) que exceda 20GB, uma configuração de 128MB deve ser muito, caso você tenha apenas 1GB de RAM e um agressivo sistema de cache em Sistema Linux.
Sort Memory (Memória para Ordenação)
Limite máximo de memória que uma conexão pode usar para executar sort (ordenação).
Caso suas consultas usem as cláusulas ORDER BY ou GROUP BY que ordenem grandes conjuntos de dados, incrementar este parâmetro deverá ajudar.
Uma Recomendação:
Ajustar o parâmetro por conexão como e quando precisar: pouca para consultas mais simples e muita para consultas complexas e para dumps de dados.
Effective Cache Size (Tamanho do Cache Efetivo)
Permite ao PostgreSQL fazer melhor uso da RAM disponível no servidor.
Exemplo:
Caso exista 1,5GB de RAM na máquina, shared buffers deve ser ajustado para 32MB e

effective cache size para 800MB.

Fsync and the WAL files (Fsync e arquivos de WAL)
Caso não reste nenhuma opção, poderá usar a proteção do WAL e melhor performance. Simplesmente mova seus arquivos de WAL, montando outro dispositivo ou criando um link simbólico para o diretório **pg_xlog,** para um disco separado ou para o conjunto dos arquivos do seu cluster principal de arquivos de dados.

random_page_cost (custo de página aleatória)
Configura o custo para trazer um registro aleatório de um banco de dados, que influencia a escolha do planejador em usar index ou table scan.
Caso tenha um disco razoavelmente rápido como SCSI ou RAID, pode baixar o custo para 2.
Vacuum_mem
Configura a memória alocada para Vacuum. Deve acelerar permitindo que PostgreSQL copie grandes quantidades para a memória.
Entre 16-32MB é uma boa quantidade para muitos sistemas.

max_fsm_pages
PostgreSQL grava espaço livre em cada uma de suas páginas de dados.
Caso tenha um banco que usa muitos updates e deletes, que irá gerar registros mortos, devido ao sistema MVCC do PostgreSQL, então expanda o FSM para cobrir todos estes registros deads (mortos) e nunca mais precisará rodar vacuum full a não ser em feriados.
O mínimo FSM é *max_fsm_relations * 16.*
max_fsm_relations
Diz quantas tabelas devem ser localizadas no mapa de espaço livre.

wal_buffers
Esta configuração decide a quantidade de buffers WAL (Write Ahead Log) que pode ter.
Para chegar a uma quantidade ótima experimente e decida.
Um bom início está em torno de 32 a 64 correspondendo a 256-516 KB de memória.

Ativar o subprocesso do auto Vacuum
Vem desabilitado por defualt (autovacuum = off no 8.1.3). Para ativar edite o arquivo de configuração postgresq.conf e altere para autovacuum = on. Irá executar o vacuum quando necessário.

Melhor é executar o comando vacuum juntamente com o comando analyze:
vacuumdb -U postgres -a, caso seja executado na linha de comando.

Para adquirir informações sobre os índices (tornando a performance ainda melhor):
vacuumdb -U postgres -a -z

EXPLAIN

```
#!/usr/bin/perl
$count = 1;
$arquivosaida = "populate.sql";
@chars = ("A" .. "Z", "a" .. "z", 0 .. 9);
@numbers = (1 .. 9);
@single_chars = ("a" .. "e");
$totalrecords = 5000; # 5 milhões
open(OUTPUT, "> $arquivosaida");
print OUTPUT "DROP TABLE index_teste;\n";
print OUTPUT "CREATE TABLE index_teste (";
print OUTPUT "codigo INT, nome VARCHAR(10), numero INT, letra CHAR(1)";
print OUTPUT ");\n";
print OUTPUT "COPY index_teste (codigo, nome, numero, letra) FROM stdin;\n";
while ($count <= $totalrecords){
   $randstring = join("", @chars [map{rand @chars} ( 1 .. 8 ) ]);
   $randnum = join("", @numbers [map{rand @numbers} ( 1 .. 8 ) ]);
   $randletter = join("", @single_chars [map{rand @single_chars} (1)]);
   print OUTPUT
   #print OUTPUT "INSERT INTO index_teste
VALUES($count,'$randstring',$randnum,'$randletter');\n";
   $count."\t".$randstring."\t".$randnum."\t".$randletter."\n";
   $count++;
};
#print OUTPUT "\n";
#print OUTPUT "\nCREATE INDEX indexteste_codigo_index ON index_teste(codigo);\n";
#print OUTPUT "CREATE INDEX indexteste_numero_index ON index_teste(numero);\n";
#print OUTPUT "VACUUM ANALYZE index_teste;\n";
close OUTPUT;
```

Um bom artigo sobre performance no PostgreSQL “PostgreSQL 8.0 Checklist de Performance” encontra-se na revista eletrônica DBFree Magazine, número 02.

16 – Exercícios

Exemplo Prático

Vamos criar um banco (clientes_ex), contendo uma tabela (cliente) e um usuário (operador) que terá apenas alguns privilégios de acesso à tabela cliente (INSERT, SELECT, UPDATE) e será obrigado a utilizar senha. Veja que não terá privilégio DELETE. Então adicionar alguns registros e executar consultas dos quatro tipos: INSERT, SELECT, UPDATE e DELETE (este apenas para verificar se realmente ele não tem este privilégio).

1)
CREATE DATABASE clientes_ex WITH ENCODING 'latin1';
-- Para SGBDs que não estejam com esta configuração, pelo menos este banco a usará

Para Exibir a Codificação do lado do Cliente

SHOW CLIENT_ENCODING;

Para Voltar à Codificação Padrão

RESET CLIENT_ENCODING;

Alterando Banco para suportar Datas dd/mm/yyyy

ALTER DATABASE clientes_ex SET DATESTYLE = SQL, DMY;
-- No caso este banco apenas ficará com esta configuração de data
-- Para alteração definitiva para todos os bancos alterar o script "postgresql.conf".

Exibindo o DateStyle Atual

SHOW DATESTYLE;

2)
```
CREATE TABLE cliente (
        codigo INT PRIMARY KEY,
        nome VARCHAR(40) NOT NULL,
        data_nasc DATE NOT NULL,
        bonus NUMERIC(12,2),
        observacao TEXT
);
```

3)
CREATE ROLE operador WITH PASSWORD 'operador9128' VALID UNTIL '26/05/2007';
O usuário somente terá os privilégios até a data determinada.

REVOKE ALL ON cliente FROM operador;
GRANT SELECT,UPDATE,INSERT ON cliente TO operador;
Dica: Caso a tabela tenha campo tipo serial também devemos dar acesso ao objeto sequence gerado:
GRANT SELECT,UPDATE,INSERT ON cliente_codigo_seq TO operador;
-- Considerando que o nome da sequência seja cliente_codigo_seq.

Para permitir ao usuário operador que faça login, use:
ALTER ROLE operador WITH LOGIN;

Obs.: Veja como está aqui o pg_hba.conf:

```
host    all       all       127.0.0.1/32        md5
```

4)
Fazer o login como usuário operador para executar as consultas abaixo:

```
INSERT INTO cliente (codigo, nome, data_nasc, bonus, observacao) VALUES (1, 'João Pedro', '01/01/1967', 18.35, 'Apenas um texto de teste');
INSERT INTO cliente (codigo, nome, data_nasc, bonus, observacao) VALUES (2, 'Pedro Paulo Rosado', '04/11/1973', 25.35, '');
INSERT INTO cliente (codigo, nome, data_nasc, bonus, observacao) VALUES (3, 'José Roberto', '25/06/1938', 12.65, NULL);
```

Observe que para campos que não exigem NOT NULL, podemos entrar apenas '' ou NULL.

```
SELECT * FROM cliente;
SELECT codigo FROM cliente;
SELECT * FROM cliente WHERE codigo = 5;
SELECT * FROM cliente WHERE codigo = 5 AND nome='João de Brito Cunha';

UPDATE cliente SET nome = 'Roberval Taylor' WHERE codigo = 3;
UPDATE cliente SET nome = 'João Almeida' WHERE nome = 'Pedro Paulo';
-- Esta consulta não é eficiente, já que nomes podem se repetir, melhor seria pela chave
```

Observe ainda, que campos do tipo numérico não têm delimitador, mas os demais tem o delimitador apóstrofo, exceto palavras-chaves e funções como NULL, TRUE, NOW(), etc.

```
DELETE FROM cliente; -- Esta apaga todos os registros da tabela
DELETE FROM cliente WHERE codigo=1;
DELETE FROM cliente WHERE codigo=2 AND nome = 'Chico Manoel';
```

Veja as mensaens quando o user operador tenta excluir algum registro:

```
clientes_ex=> DELETE FROM cliente WHERE codigo=2 AND nome = 'Chico Manoel'
ERROR:  permission denied for relation cliente
```

Ou seja, falta privilégio para excluir e as regras funcionaram.
Um pequeno teste de conexão cia PHP:

```
<?php
$con=pg_connect('host=127.0.0.1 user=operador password=operador9128 dbname=clientes_ex');
if ($con){
	echo "OK";
}else{
	echo "NOK";
}
?>
```

EXERCÍCIO DE UM PEQUENO CONTROLE DE ESTOQUE

Utilizaremos somente minúsculas para os nomes dos objetos (bancos, esquemas, tabelas, campos, etc) e quando composto por duas ou mais palavras separar com sublinhado.

clientes
funcionarios
produtos
vendas
vendas_itens
bonus
comissoes

Por enquanto iremos criar apenas a tabela produtos, mais adiante criaremos as demais tabelas.

Obs.: A tabela de produtos irá guardar também uma informação sobre a posição do produto no local onde é estocado.

Esta posição conterá abscissa (x) e ordenada (y), ou seja a distância horizontal da esquerda e a distância vertical de baixo para cima. Exemplo simplificado da disposição dos produtos:

```
                        ProdA
-----------------x,y----------------x+10,y --------------x+20,y
                 x                |                             |
|
                                  |                             |
|
                                  |Y                         |Y
|Y
                                  |                             |
|
                                  |                             |
|
```

onde x=10cm e y=5cm

Existem tipos de dados geométricos no PostgreSQL, para pontos, linhas, polígonos, círculos, etc.
Iremos utilizar o ponto (point).

Vamos criar uma versão resumida da tabela Produtos:

CREATE TABLE produtos (codigo int, nome char(40), preco numeric(12,2));

Para excluir uma tabela:
DROP TABLE nometabela;

1 - Instalar o PostgreSQL (de acordo com seu sistema operacional) e realizar as configurações básicas nos arquivos pg_hba.conf e no postgresql.conf. Mude o estilo da data para um compatível com o brasileiro, mude os locales para pt_BR, mude a codificação para LATIN1 e permita conexão TCP/IP para uma máquina de IP 10.1.1.1.
Configure também a autenticação desta máquina para md5;

2 - Criar um banco com nome controle_estoque;

3 – Criar um esquema esq_estoque;

4 – Criar um grupo de usuários grupo_estoque;

5 – Criar dentro do esquema esq_estoque, tabelas, de acordo com as estruturas abaixo com os devidos atributos (campos), tipos de dados, tamanhos e constraints:

clientes (cpf, nome, endereco, cidade, uf, cep, telefone, data_cadastro, data_nascimento);

funcionarios (cpf, nome, endereco, cidade, uf, cep, telefone, data_admissao, data_nascimento);

produtos (codigo_produto, nome, unidade, quantidade, preco_unitario, estoque_minimo, estoque_maximo); -- nome deve ser UNIQUE

vendas (codigo_venda, data_venda, cpf_cliente, cpf_funcionario);

vendas_itens (codigo_item, codigo_venda, codigo_produto, quantidade_item);

bonus (codigo_bonus, cpf_cliente, codigo_venda, bonus);

comissoes (codigo_comissao, cpf_funcionario, codigo_venda, comissao);

6 – Criar as chaves estrangeiras que façam os devidos relacionamentos entre as tabelas;

7 – Remover somente a chave primária da tabela clientes e Adicionar novamente com nome clientes_pk;

8 – Adicionar a constraint NOT NULL no campo preco_unitário de produtos;

9 – Adicionar uma constraint CHECK que exija valores maiores que zero no estoque_minimo do produtos;

10 – Alterar o nome do campo nome da tabela produtos para descricao e o nome da tabela clientes para clientes2. Renomeie novamente para clientes;

11 – Alterar o tipo de dados do campo quantidade de produtos para NUMERIC(12,2);

12 – Criar três usuários user_cli, user_prod e user_adm, todos no grupo grupo_teste, com os seguintes privilégios:

- user_cli tem permissão de executar as consultas SELECT, UPDATE E INSERT na tabela clientes;
- user_pro tem permissão de executar a consulta SELECT na tabela produtos;
- user adm pode fazer o que bem entender em todos os bancos do servidor.

13 – Criar uma view que guarde a soma dos bonus por cliente. Receberá um cliente e retornará sua soma;

14 – Criar uma view que guarde a soma das comissões por funcionário. Receberá um funcionário e retornará sua soma;

15 – Criar uma transação com o bloco:
- Venda e Atualização do estoque,
- Atualização do bônus do cliente,
- Atualização da comissão do vendedor

16 – Cadastrar pelo menos três registros em cada tabela;

17 – Gerar um dump do banco e editar o script para ver seu conteúdo;

18 – Consultar qual o produto mais caro e o mais barato;

19 – Qual o cliente mais antigo;

20 – Atualize o preço de um produto, adicionando R$ 3.85 ao mesmo;

21 – Consulte qual o cliente que não tem bonus e o remova da tabela;

22 – Crie um banco chamado cep_brasil, com uma única tabela cep_tabela cuja estrutura deve ser:

create table cep_full (cep char(8), tipo char(72), logradouro char(70),bairro char(72), municipio char(60), uf char(2));
Importe o arquivo cep_brasil_unique.csv existente no CD ou no site: http://ribafs.byethost2.com seção downloads – PostgreSQL.
- Então execute \timing,
- Faça uma consulta que retorne apenas o seu CEP
- E anote o tempo gasto.

23 – Agora adicione uma chave primária na tabela. Então faça a mesma consulta anterior e veja a diferença de desempenho por conta do índice adicionado;

22 – Execute o PgAdmin, conecte ao banco controle_estoque para verificar o banco criado, esquemas, grupo de usuários e usuários, esquema, tabelas, fazer algumas consultas, visualizar os dados, a estrutura das tabelas e outras atividades;

23 – Faça o mesmo com o EMS PostgreSQL Manazer;

24 – Conecte ao banco com o DbVisualizer para verificar suas tabelas, esquema e veja o DER (Diagrama Entidade-Relacionamento) e salve como imagem uma cópia do DER.

25 – Criar uma tabela “site” contendo um campo com ip do visitante, do tipo inet.

26 – Criar uma tabela “geometria”, contendo campos do tipo ponto, polígono e círculo.

17 - Referências

Site Oficial
Site oficial – http://www.postgresql.org
Site da comunidade brasileira – http://www.postgresql.org.br

Documentação Oficial
Online - http://www.postgresql.org/docs/8.1/interactive/index.html (Com busca)
PDF - http://www.postgresql.org/files/documentation/pdf/8.1/postgresql-8.1-A4.pdf
Brasil - Online - http://pgdocptbr.sourceforge.net/pg80/index.html
Brasil - PDF - http://ufpr.dl.sourceforge.net/sourceforge/pgdocptbr/pgdocptbr800-pdf-1.1.zip
Brasil - PDF Tutorial -
http://www.pythonbrasil.com.br/moin.cgi/NabucodonosorCoutinho?action=AttachFile&do=get&target=tutorial_pg.pdf.tar.gz
PostgreSQL Technical Documentation - http://techdocs.postgresql.org/

Livros (E-books grátis)
- Practical PostgreSQL (inglês)
http://www.faqs.org/docs/ppbook/book1.htm
- PostgreSQL: Introduction and Concepts (inglês)
http://www.postgresql.org/files/documentation/books/aw_pgsql/index.html
- PostgreSQL: Das offizielle Handbuch (alemão)
http://www.postgresql.org/docs/books/pghandbuch.html.de
- Lista de Livros sobre o PostgreSQL
http://www.postgresql.org/docs/books/

Listas
Lista Oficial do PostgreSQL, com diversas categorias
- Lista de News (freqüência semanal)
http://www.postgresql.org/community/weeklynews/
Cadastro - http://www.postgresql.org/community/lists/subscribe

- Cadastro e Descadastro em Uma das Várias Listas
http://www.postgresql.org/community/lists/subscribe

Busca nos Arquivos das Listas do PostgreSQL
http://archives.postgresql.org/index.php?adv=1

- Lista da Comunidade Brasileira
http://pgfoundry.org/mailman/listinfo/brasil-usuarios/

Lista de Discussão no Yahoo
http://br.groups.yahoo.com/group/postgresql-br/
Para se cadastrar acesse o site acima e faça o cadastro.

PostgreSQL Users Groups Site
http://pugs.postgresql.org/

IRC
http://www.postgresql.org/community/irc
Existe um canal brasileiro

Sites do PostgreSQL em vários países
http://www.postgresql.org/community/international

Empresas que utilizam PostgreSQL
http://www.postgresql.org/about/casestudies/

Featured Users (Usuários Caracterizados)
Estão aqui algumas das centenas das companhias que construíram produtos, soluções, web sites e ferramentas usando o PostgreSQL
http://www.postgresql.org/about/users

Grandes Projetos do PostgreSQL
http://www.postgresql.org/community/resources

Projetos no PgFoundry
ftp://ftp2.br.postgresql.org/postgresql/projects/pgFoundry/

Projetos Gborg
ftp://ftp2.br.postgresql.org/postgresql/projects/gborg/

Análise de Diversas Ferramentas para PostgreSQL
https://wiki.postgresql.org.br/wiki/Ferramentas

Diversos Logos do PostgreSQL para divulgação em Sites
http://www.postgresql.org/community/propaganda

Comunicar e Existência de Bugs
http://www.postgresql.org/support/submitbug
Com formulário online de envio de relato de bugs.

Diversas Ferramentas para o PostgreSQL
Conversor de Script DDL para PostgreSQL
http://www.icewall.org/~hjort/conv2pg/
http://www.freedownloadscenter.com/Best/erd-postgresql.html
http://www.databaseanswers.com/modelling_tools.htm
http://top.softlandmark.com/Erd_postgresql.html
http://directory.fsf.org/autodia.html
http://www.datanamic.com/download/scripteditor.zip
http://tedia2sql.tigris.org/
http://tedia2sql.tigris.org/usingtedia2sql.html
http://www.fileboost.net/directory/development/databases_networks/cutesql/004405/review.html
http://www.fileboost.net/directory/development/databases_networks/case_studio_2_lite/013963/1/download.html
http://files.db3nf.com/download/DB3NF_Setup_1_4.exe
http://gborg.postgresql.org/project/pgxexplorer/download/download.php
http://gborg.postgresql.org/browse.php
http://gborg.postgresql.org/browse.php?83

Revistas
Revista Sobre Bancos de Dados Free (Português)

http://www.dbfreemagazine.com.br/index.php
Cadastre-se e faça o download. Já existem oito edições.

SQL Magazine (comercial)
http://www.sqlmagazine.com.br/revista.asp

Cursos
- Curso de PostgreSQL da dbExpert (São Paulo) – www.dbexpert.com.br
- Curso de PostgreSQL do Evolução (Fortaleza-CE) – www.evolucao.com.br

Modelagem e Normalização
- O Modelo Relacional de Dados (em cinco artigos, de Júlio Battisti)
http://www.imasters.com.br/artigo.php?cn=2419&cc=149
- Conceitos Fundamentais de Banco de Dados (de Ricardo Rezende)
http://www.sqlmagazine.com.br/Colunistas/RicardoRezende/02_ConceitosBD.asp

Outros:
- PostgreSQL no iMasters – http://www.imasters.com.br/secao.php?cs=35
- Lozano – http://www.lozano.eti.br
- Conversor de Script DDL para PostgreSQL - http://www.icewall.org/~hjort/conv2pg/
- Meu PostgreSQL não Conecta! - http://www.icewall.org/~hjort/pgsql/naoconecta.htm
- Junção entre Tabelas no Postgresql – **http://www.imasters.com.br/artigo/2867**
- Customize database queries using views in PostgreSQL - http://builder.com.com/5100-6388_14-6032031.html
- PostgreSQL Interagindo com Banco de dados - http://www.imasters.com.br/artigo/954
- O Tipo de Dados Serial – **http://www.imasters.com.br/artigo/1804**
- RunAs - Utilitário para rodar o PG no XP: http://www.softtreetech.com/24x7/archive/53.htm
- PostgreSQL com LDAP - http://itc.musc.edu/wiki/PostgreSQL
- FAQs - http://www.postgresql.org/docs/faqs.FAQ.html
- FAQs - http://wiki.ael.be/index.php/PostgresQL101
- Getting Started - http://postgresql.boeldt.net/getting_started.asp
- Down and Install - http://postgresql.boeldt.net/setup_postgresql.asp
- Microsoft SQL to PostgreSQL - http://postgresql.boeldt.net/mssql_to_postgresql.asp
- PG Configuration - http://postgresql.boeldt.net/postgres-linux-configuration.asp
- Muitos links - http://sql-info.de/postgresql/links.html
- General Bits - http://www.varlena.com/GeneralBits/
- Notes - http://www.archonet.com/pgdocs/pgnotes.html
- Presentations - http://candle.pha.pa.us/main/writings/computer.html
- EnterpriseDB - http://www.osdb.org/
- SQL-ish projects - http://docman.sourceforge.net/home_html/sql.html
- Quick Reference Material - http://techdocs.postgresql.org/#quickref
- Driver ODBC - http://www.postgresql.org/ftp/odbc/versions/msi/
- Replication Project - http://gborg.postgresql.org/project/pgreplication/download/download.php

Otimização
http://www.powerpostgresql.com/PerfList
http://www.powerpostgresql.com/Downloads/annotated_conf_80.html
http://www.varlena.com/GeneralBits/Tidbits/perf.html
https://wiki.postgresql.org.br/wiki/Otimiza%C3%A7%C3%A3o
http://www.revsys.com/writings/postgresql-performance.html
http://www.linuxjournal.com/article/4791
http://www.budget-ha.com/postgres/
http://archives.postgresql.org/pgsql-performance/