

Módulo II

- 1) Instalação do PostgreSQL-8.3.1 dos fontes no Linux Ubuntu 7.10 - 2**
- 2) Entendendo e trabalhando com CLUSTERS - 5**
- 3) Entendendo o Sistema de Autenticação - 15**
- 4) Administração de Grupos, Usuários e Privilégios - 27**
- 5) Manutenção do Servidor do PostgreSQL e dos Bancos - 41**
- 6) Monitorando as Atividades do Servidor do PostgreSQL - 62**
- 7) Catálogo do Sistema - 88**
- 8) Trabalhando com Esquemas - 114**
- 9) Segurança no SGBD - 120**
- 10) Melhorando a Performance do PostgreSQL - 131**
- 11) Entendendo o WAL (Write Ahead Log) e o PITR - 155**
- 12) Conectividade - 171**
- 13) Contribs - 185**
- 14) Guia de Replicação no PostgreSQL com Slony-I - 199**
- 15) Ferramentas - 216**
- 16) Módulo 2 - Aula Extra – BLOBs - 227**
- 17) Encoding - 231**
- 18) Usando o PostGIS - 233**
- 19) Introdução às Redes de Computadores - 242**

1) Instalação do PostgreSQL-8.3.1 dos fontes no Linux Ubuntu 7.10

Através dos Repositórios

Instalar

```
sudo apt-get install postgresql-8.3
```

Com isso teremos a versão 8.3 instalada em pouco tempo. O Ubuntu irá buscar o postgresql em seus servidores e o instalará e configurará para nós:

Detalhes

Diretório de dados - /var/lib/postgresql/8.3/main/base

Diretório do pg_hba.conf e do postgresql.conf - /etc/postgresql/8.3/main

autovacuum.conf - /etc/postgresql-common

pg_dump, pg_dumpall, psql e outros binários - /usr/bin

Acessando pelo prompt

Trocar a senha do super-usuário

```
sudo passwd postgres
```

Acessando a console psql

```
sudo -u postgres psql
```

Instalando através dos Fontes

Este método de instalação dá um pouco mais de trabalho mas em contrapartida é o que oferece um maior controle e uma maior possibilidade de customização.

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 (opcionais, para autenticação)

Veja como encontrar esses requisitos no Ubuntu:

```
sudo apt-get install build-essential libreadline5-dev zlib1g-dev gettext
```

Ou faça o download do site: <http://packages.ubuntu.com/> e instale com:

```
dpkg -i nome.deb
```

Ou então duplo clique no gerenciador de arquivos Nautilus.

Obs.: estes pacotes podem mudar de nome devido ao aparecimento de novas versões.
E use make ao invés de gmake.

Download - <http://www.postgresql.org/ftp/source/>

Descompacte em /usr/local/src e instale no diretório default, que é /usr/local/pgsql.

```
sudo tar xzpvf postgresql-8.3.1.tar.gz -C /usr/local/src  
cd /usr/local/src/postgresql-8.3.1
```

Caso já tenha o postgresql instalado no Ubuntu, pule as etapas de criação do grupo e do usuário na instalação abaixo, como também terá que alterar a porta no script postgresql.conf, logo após a criação do cluster com o comando initdb e, no caso, os comandos deverão passar também a porta, por exemplo:

```
bin/createdb -p 5433 bdteste  
bin/psql -p 5433 bdteste.
```

Configurar, Compilar e Instalar

```
sudo ./configure  
sudo make  
sudo make install  
sudo groupadd postgres  
sudo useradd -g postgres -d /usr/local/pgsql postgres  
sudo mkdir /usr/local/pgsql/data  
sudo chown postgres:postgres /usr/local/pgsql/data  
sudo passwd postgres  
su - postgres  
bin/initdb -D /usr/local/pgsql/data  
bin/pg_ctl -D /usr/local/pgsql/data start  
bin/createdb teste  
bin/psql teste
```

Isso irá criar um clustar com encoding UTF-8 (default do Ubuntu)

Caso prefira iso-8859-1, antes de executar o initdb, exporta a variável LANG:

```
export LANG=pt_BR.iso-8859-1  
bin/initdb --encoding latin1 -D /usr/local/pgsql/data
```

Copiar o script de inicialização "linux" para o /etc/init.d

```
sudo cp /usr/local/src/postgresql-8.3.0/contrib/start-script/linux /etc/init.d/postgresql-8.3.0
```

Dar permissão de execução ao script:

```
sudo chmod u+x /etc/init.d/postgresql-8.3.0
```

Adicionar ao Path

```
su - postgres
```

```
gedit /etc/bash.bashrc (e adicione a linha abaixo):
```

```
PATH=/usr/local/pgsql/bin:$PATH
```

Veja a instalação em outras distribuições e em FreeBSD em:

<http://postgresql.ribafs.net>

Módulo II

2) Entendendo e trabalhando com CLUSTERS

- 1 - Iniciando e parando o serviço do PostgreSQL
- 2 - Entendendo os Tablespace
- 3 - Criação de bancos de dados
- 4 - Removendo bancos de dados
- 5 - Bancos de dados de template/modelo
- 6 - Entendendo o layout físico dos bancos de dados

A palavra "cluster" pode significar várias coisas diferentes dependendo do contexto:

- 1) Existe o cluster de tabelas:

<http://www.postgresql.org/docs/8.3/static/sql-cluster.html> ou
<http://www.postgresql.org/docs/8.3/static/app-clusterdb.html>

- 2) Existe o cluster do postgresql, que consiste em todos os arquivos que compõem um conjunto de bancos de dados administrados por uma instância do postgresql que sobe em uma porta só dele. Este cluster é criado pelo initdb:

<http://www.postgresql.org/docs/8.3/static/app-initdb.html>

- 3) Existe o conceito de cluster de bancos de dados que são várias instâncias de bancos de dados rodando em máquinas diferentes e se comportando como se fossem uma só. O PostgreSQL não possui uma implantação deste tipo.

- 4) Existe o pg_cluster que é uma implementação de replicação e é parecido com um cluster, mas na verdade é uma replicação:

<http://pgfoundry.org/projects/pgcluster/>

Para ver a diferença entre cluster e replicação, veja:

<http://www.midstorm.org/~telles/2007/08/24/cluster-replicacao/>

Cluster de Versões

- 1) Os pacotes do Debian fazem isso automaticamente, de forma limpa.

Você instala o 8.1 e 8.2 na mesma máquina e ele sobe cada um em uma porta diferente. Vale a pena conferir para ver como ele organiza as coisas, particularmente a estrutura de diretórios é o init.d.

(Fábio Telles na lista pgbr-geral.)

Instalação do PostgreSQL-8.3 através dos repositórios do Ubuntu 7.10

Para instalar o Servidor, acesse um terminal e digite:

```
sudo apt-get install postgresql-8.3 (com este pacote serão instalados o server, o client e o common)
sudo apt-get install postgresql-contrib-8.3
sudo apt-get install postgresql-doc-8.3 (ficará em /usr/share/doc/postgresql-doc-8.3/html/index.html)
```

Em máquinas que somente precisam ter o cliente instalado use:

```
sudo apt-get install postgresql-client-8.3
sudo apt-get install postgresql-doc-8.3
sudo apt-get install pgadmin3
```

Após instalar o 8.3 repita os passos para instalar também a versão 8.2, de forma que fiquemos com dois clusters instalados, um da versão 8.3 e outro da versão 8.2. O Ubuntu gerencia bem isso e de forma transparente. Geralmente o primeiro instalado fica na porta 5432 e o segunda na 5433.

Após instalar o pgadmin execute o script abaixo para habilitar algumas funções úteis:

```
sudo -u postgres psql -d postgres < /usr/share/postgresql/8.3/contrib/adminpack.sql
```

Para acessar a console do psql:

Para acessar o 8.3 use

```
sudo -u postgres psql
```

Para acessar o 8.2 use

```
sudo -u postgres psql -p 5433
```

Ou mais adequadamente com:

```
sudo -u postgres /usr/lib/postgresql/8.2/bin/psql -p 5433
```

Observe que não será solicitada a senha do postgres. Isso devido ao método de autenticação usado por padrão no /etc/postgresql/8.3/main/pg_hba.conf, que identifica o usuário do sistema operacional, sem senha. Na primeira vez que acessar altere a senha do super-usuário:

```
ALTER ROLE postgres WITH ENCRYPTED PASSWORD 'postgres';
```

Para oferecer mais segurança no acesso remoto, já que localmente não requer senha.

Para constatar os dois serviços no ar abra um terminal execute:

```
ps ax | grep post
```

Diretórios quando instalado pelos repositórios:

- Diretório base (com bancos e outros) - /var/lib/postgresql/8.3/main
- Arquivos de configuração (pg_hba.conf, pg_ident.conf e postgresql.conf) - /etc/postgresql/8.3/main
- Diretório de gerenciamento dos clusters - /etc/postgresql-common
- Diretório dos binários - /usr/lib/postgresql/8.3/bin

Exercício

Importar o banco de dados de CEP existente no CD ou em:

http://tudoemum.ribafs.net/includes/cep_brasil.sql.bz2

Faça o download e copie para a pasta /home/aluno

Descompacte:

Abra o gerenciador de arquivos (locais – pasta pessoal). Clicando com o botão direito e extrair aqui.

Este banco de CEPs (do Brasil), não é completo nem está atualizado mas é uma boa base de testes e contém os CEPs das grandes cidades e capitais.

- Mude as permissões do arquivo cep_brasil_unique.csv para que o usuário postgres tenha permissão de acessá-lo:

```
sudo chown postgres:postgres cep_brasil_unique.csv
```

Experimente criar o banco na versão 8.3 com codificação latin1.

Ele não criará, informando incompatibilidade com as configurações do servidor/sistema operacional.

Para ter compatibilidade com latin1 no 8.3 devemos criar o cluster passando a codificação pelo comando initdb.

```
sudo -u postgres psql
create database cep_brasil with encoding 'latin1';
```

```
\c cep_brasil
create table cep_full (
    cep char(8),
    tipo char(72),
    logradouro char(70),
    bairro char(72),
    municipio char(60),
    uf char(2)
);
```

Execute os comandos abaixo e veja as informações:

```
\l
```

Depois:

```
\d
```

Importar o script de ceps para a tabela que acabamos de criar:

```
\copy cep_full from /home/ribafs/cep_brasil_unique.csv
```

Então, para ativar o cronômetro, execute:

```
\timing
```

Faça uma consulta que retorne apenas o registro com o CEP da sua rua:

```
select logradouro from cep_full where cep = '60420440';
```

Dual Core 2.2, 2GB RAM e Ubuntu 7.10 gastou 6711,204 ms.

Centrino 1.86 1GB de RAM e Ubuntu 7.10 gastou 8476.877 ms

Caso repita a consulta, levará apenas 514 ms na última máquina, pois está no cache.

Agora adicione uma chave primária na tabela:

```
ALTER TABLE cep_full add constraint cep_pk primary key (cep);
```

Então repita a mesma consulta anterior e veja a diferença de desempenho por conta do índice adicionado.

Tenha o cuidado de executar o comando ANALYZE antes da consulta.

Aqui gastou somente 1.135ms.

1 - Iniciando e parando o serviço do PostgreSQL

Parar:

```
sudo /etc/init.d/postgresql-8.3 stop
```

Iniciar:

```
sudo /etc/init.d/postgresql-8.3 start
```

Reiniciar:

```
sudo /etc/init.d/postgresql-8.3 restart
```

Para o 8.2 é de forma semelhante.

Observação

Estamos usando o script oferecido pela distribuição ou então aquele dos contribs que acompanha os fontes.

Manualmente:

```
sudo /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data start
```

```
sudo /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data stop
```



```
sudo /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data restart
```

2 - Entendendo os Tablespaces

Como o nome sugere, um espaço destinado a armazenar tabelas. Pode ser criado para proteção/segurança de algumas tabelas (que ficam em localização diferente do restante dos bancos) como para balancear carga e também para otimizar desempenho de apenas algumas tabelas ou bancos que requerem maior desempenho.

Utilizando espaços de tabelas, o administrador pode controlar a organização em disco da instalação do PostgreSQL. É útil pelo menos de duas maneiras:

Primeira: se a partição ou volume onde o agrupamento foi inicializado ficar sem espaço, e não puder ser estendido, pode ser criado um espaço de tabelas em uma partição diferente e utilizado até que o sistema possa ser reconfigurado.

Segunda: os espaços de tabelas permitem que o administrador utilize seu conhecimento do padrão de utilização dos objetos de banco de dados para otimizar o desempenho. Por exemplo, um índice muito utilizado pode ser colocado em um disco muito rápido com alta disponibilidade, como uma unidade de estado sólido. [4] Ao mesmo tempo, uma tabela armazenando dados históricos raramente utilizados, ou que seu desempenho não seja crítico, pode ser armazenada em um sistema de disco mais barato e mais lento.

Para definir um espaço de tabelas é utilizado o comando [CREATE TABLESPACE](#) como, por exemplo:

```
CREATE TABLESPACE nometablespace [ OWNER nomedono ] LOCATION 'diretório';
```

```
CREATE TABLESPACE area_veloz LOCATION '/mnt/sda1/postgresql/data';
```

Ao criar um banco, uma tabela ou um índice podemos escolher a qual tablespace pertencerá, usando o parâmetro tablespace.

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 é:

```
\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';
```

Importante: 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 Tablespace existentes:

```
\db
```

```
SELECT spcname FROM pg_tablespace;
```

Na documentação oficial em português do PG 8.0:

<http://pgdocptbr.sourceforge.net/pg80/manage-ag-tablespaces.html>

<http://pgdocptbr.sourceforge.net/pg80/sql-createtablespace.html>

E diversos outros capítulos fazem referência.

Na documentação do 8.3 em inglês:

<http://www.postgresql.org/docs/8.3/interactive/manage-ag-tablespaces.html>

<http://www.postgresql.org/docs/8.3/interactive/sql-createtablespace.html>

<http://www.postgresql.org/docs/8.3/interactive/sql-droptablespace.html>

<http://www.postgresql.org/docs/8.3/interactive/sql-altertablespace.html>

O Capítulo 5 do Livro Dominando o PostgreSQL aborda os tablespaces.

3 - Criação de bancos de dados

No prompt de comando:

```
createdb -p numeroporta -h nomehost -E LATIN1 -e nomebanco
```

Mais Detalhes: <http://www.postgresql.org/docs/8.3/interactive/app-createdb.html>

Como SQL dentro do psql ou em outro cliente:

```
CREATE DATABASE banco OWNER dono TABLESPACE nometablespace ENCODING 'LATIN1';
```

Mais Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/sql-createdatabase.html>

4 - Removendo bancos de dados

```
DROP DATABASE nomebanco;
```

Este comando não pode ser desfeito nem executado dentro de uma transação.

Mais detalhes: <http://www.postgresql.org/docs/8.3/interactive/sql-dropdatabase.html>

```
dropdb -U usuario nomebanco
```

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/app-dropdb.html>

5 - Bancos de dados de template/modelo

Na verdade o comando CREATE DATABASE funciona copiando um banco de dados existente. Por padrão, copia o banco de dados padrão do sistema chamado template1. Portanto, este banco de dados é o "modelo" a partir do qual os novos bancos de dados são criados. Se forem adicionados objetos ao template1, estes objetos serão copiados nos próximos bancos de dados de usuário criados. Este comportamento permite modificar localmente o conjunto padrão de objetos nos bancos de dados. Por exemplo, se for instalada a linguagem procedural PL/pgSQL em template1, esta se tornará automaticamente disponível em todos os próximos bancos de dados dos usuários sem que precise ser feito qualquer procedimento adicional na criação dos bancos de dados.

Existe um segundo banco de dados padrão do sistema chamado template0. Este banco de dados contém os mesmos dados contidos inicialmente em template1, ou seja, contém somente os objetos padrão pré-definidos pela versão do PostgreSQL. **O banco de dados template0 nunca deve ser modificado após a execução do utilitário [initdb](#). Instruindo o comando CREATE DATABASE para copiar template0 em vez de template1, pode ser criado um banco de dados de usuário "intacto", não contendo nenhuma adição feita ao**

banco de dados template1 da instalação local. É particularmente útil ao se restaurar uma cópia de segurança feita por [pg_dump](#): o script da cópia de segurança deve ser restaurado em um banco de dados intocado, para garantir a recriação do conteúdo correto da cópia de segurança do banco de dados, sem conflito com as adições que podem estar presentes em template1.

Para criar um banco de dados copiando template0 deve ser utilizado:

CREATE DATABASE nome_do_banco_de_dados TEMPLATE template0;
a partir do ambiente SQL, ou

createdb -T template0 nome_do_banco_de_dados
a partir do interpretador de comandos.

É possível criar bancos de dados modelo adicionais e, na verdade, pode ser copiado qualquer banco de dados do agrupamento especificando seu nome como modelo no comando CREATE DATABASE. Entretanto, é importante compreender que não há intenção (ainda) que este seja um mecanismo tipo "COPY DATABASE" de uso geral. Em particular, é essencial que o banco de dados de origem esteja inativo (nenhuma transação em andamento alterando dados) durante a operação de cópia. O comando CREATE DATABASE verifica se nenhuma sessão (além da própria) está conectada ao banco de dados de origem no início da operação, mas não garante que não possa haver alteração durante a execução da cópia, resultando em um banco de dados copiado inconsistente.

Portanto, recomenda-se que os bancos de dados utilizados como modelo sejam tratados como

somente para leitura.

Após preparar um banco de dados modelo, ou fazer alguma mudança em um deles, é recomendado executar o comando VACUUM FREEZE ou VACUUM FULL FREEZE neste banco de dados. Se for feito quando não houver nenhuma outra transação aberta no mesmo banco de dados, é garantido que todas as linhas no banco de dados serão "congeladas" e não estarão sujeitas a problemas de recomeço do ID de transação. Isto é particularmente importante em um banco de dados que terá datallowconn definido como falso, uma vez que não será possível executar a rotina de manutenção VACUUM neste banco de dados. Para obter informações adicionais deve ser consultada a [Seção 21.1.3](#) do manual oficial em português do Brasil em: <http://pgdocptbr.sourceforge.net/pg80/maintenance.html#VACUUM-FOR-WRAPAROUND> .

Nota: Os bancos de dados template1 e template0 não possuem qualquer status especial além do fato do nome template1 ser o nome padrão para banco de dados de origem do comando CREATE DATABASE, e além de ser o banco de dados padrão para se conectar utilizado por vários programas, como o [createdb](#). Por exemplo, template1 pode ser removido e recriado a partir de template0 sem qualquer efeito prejudicial. Esta forma de agir pode ser aconselhável se forem adicionadas, por descuido, coisas inúteis ao template1. Na versão 8 apareceu um novo template, que é uma cópia do template1, é o **postgres**, este a partir de então é o template default.

Recriação do banco de dados template1

Neste exemplo o banco de dados template1 é recriado. Deve ser observado na sequência de comandos utilizada que não é possível remover o banco de dados template1 conectado ao mesmo, e enquanto este banco de dados estiver marcado como modelo no catálogo do sistema pg_database.

Para recriar o banco de dados template1 é necessário: se conectar a outro banco de dados (teste neste exemplo); atualizar o catálogo pg_database para que o banco de dados template1 não fique marcado como um banco de dados modelo; remover e criar o banco de dados template1; conectar ao banco de dados template1; executar os comandos VACUUM FULL e VACUUM FREEZE; atualizar o catálogo do sistema pg_database para que o banco de dados template1 volte a ficar marcado como um banco de dados modelo.

Abaixo está mostrada a sequência de comandos utilizada:

```
template1=# DROP DATABASE template1;  
ERRO: não é possível remover o banco de dados aberto atualmente  
template1=# \c teste  
Conectado ao banco de dados "teste".
```

```
teste=# DROP DATABASE template1;
ERRO: não é possível remover um banco de dados modelo
teste=# UPDATE pg_database SET datistemplate=false WHERE datname='template1';
UPDATE 1
teste=# DROP DATABASE template1;
DROP DATABASE
teste=# CREATE DATABASE template1 TEMPLATE template0 ENCODING 'latin1';
CREATE DATABASE
teste=# \c template1
Conectado ao banco de dados "template1".
template1=# VACUUM FULL;
VACUUM
template1=# VACUUM FREEZE;
VACUUM
template1=# UPDATE pg_database SET datistemplate=true WHERE datname='template1';
UPDATE 1
Mais detalhes em: http://pgdocptbr.sourceforge.net/pg80/manage-ag-templatedbs.html
```

6 - Entendendo o layout físico dos bancos de dados

Formato de armazenamento no nível de arquivos e diretórios.

Todos os dados necessários para um agrupamento de bancos de dados são armazenados dentro do diretório de dados do agrupamento. Podem existir na mesma máquina vários agrupamentos, gerenciados por diferentes postmaster.

Quando instalamos pelos fontes e não alteramos o default ficam em /usr/local/pgsql/data.

O diretório data contém vários subdiretórios e arquivos de controle, conforme mostrado na tabela abaixo. Além destes itens requeridos, os arquivos de configuração do agrupamento postgresql.conf, pg_hba.conf e pg_ident.conf são tradicionalmente armazenados no data (embora a partir da versão 8.0 do PostgreSQL seja possível mantê-los em qualquer outro lugar).

Conteúdo de diretório data (lembre: algumas distribuições usam outro diretório)

Item	Descrição
PG_VERSION	Arquivo contendo o número de versão principal do PostgreSQL
base	Subdiretório contendo subdiretórios por banco de dados
global	Subdiretório contendo tabelas para todo o agrupamento, como pg_database
pg_clog	Subdiretório contendo dados sobre status de efetivação de transação
pg_subtrans	Subdiretório contendo dados sobre status de subtransação
pg_tblspc	Subdiretório contendo vínculos simbólicos para espaços de tabelas
pg_xlog	Subdiretório contendo os arquivos do WAL (registro prévio da escrita)
postmaster.opts	Arquivo contendo as opções de linha de comando com as quais o postmaster foi inicializado da última vez
postmaster.pid	Arquivo de bloqueio contendo o PID corrente do postmaster, e o ID do segmento de memória compartilhada (não mais presente após o

Item	Descrição
	postmaster ser parado)

Em caso de problema ao tentar iniciar o PostgreSQL remova o postmaster.pid.

Para cada banco de dados do agrupamento existe um subdiretório dentro de PGDATA/base, com nome correspondente ao OID do banco de dados em pg_database. Este subdiretório é o local padrão para os arquivos do banco de dados; em particular, os catálogos do sistema do banco de dados são armazenados neste subdiretório.

Cada tabela e índice é armazenado em um arquivo separado.

Mais detalhes em: <http://pgdocptbr.sourceforge.net/pg80/storage.html> e em <http://www.postgresql.org/docs/8.3/interactive/storage-file-layout.html>

3) Entendendo o Sistema de Autenticação

- 3.1) Conhecendo os Métodos de autenticação do pg_hba.conf
- 3.2) Arquivo de senhas pgpass.conf
- 3.3) Entendendo e manipulando o arquivo pg_ident.conf
- 3.4) Configurando o ambiente do PostgreSQL

Informações Básicas sobre Redes

IPs, Redes, Subredes e Máscaras

Sendo um IP - 10.0.0.1/32

Sendo uma sub-rede - 10.0.0.0/24

10.0.0.7/32 ou 10.0.0.7/255.255.255.255

189.41.12.0/24 ou 189.41.12.0/255.255.255.0

* Classe A (8 bits de rede) : 255.0.0.0

* Classe B (16 bits de rede): 255.255.0.0

* Classe C (24 bits de rede): 255.255.255.0

Tabela sub-rede IPv4

Notação CIDR	Máscara	Nº IPs
/0	0.0.0.0	4.294.967.296
/8	255.0.0.0	16.777.216
/16	255.255.0.0	65.536
/24	255.255.255.0	256
/25	255.255.255.128	128
/26	255.255.255.192	64
/27	255.255.255.224	32
/28	255.255.255.240	16
/29	255.255.255.248	8
/30	255.255.255.252	4
/32	255.255.255.255	1

3.1) pg_hba.conf no PostgreSQL 8.3

Quando um aplicativo cliente se conecta ao servidor de banco de dados especifica o nome de usuário do PostgreSQL a ser usado na conexão, de forma semelhante à feita pelo usuário para acessar o sistema operacional Unix. Dentro do ambiente SQL, o nome de usuário do banco de dados determina os privilégios de acesso aos objetos do banco de dados. Portanto, é essencial controlar como os usuários de banco de dados podem se conectar.

A *autenticação* é o processo pelo qual o servidor de banco de dados estabelece a identidade do

cliente e, por extensão, determina se o aplicativo cliente (ou o usuário executando o aplicativo cliente) tem permissão para se conectar com o nome de usuário que foi informado.

Observar que não se aborda autenticação para usuários locais, que no caso teria uma linha iniciando com 'local'.

A versão atual do PostgreSQL usa o arquivo:

C:\Documents and Settings\ribafs\Application Data\postgresql\pgpass.conf

Explicando os campos

TYPE pode ser conexão do tipo local ou host

DATABASE pode ser "all", "sameuser", "samerole", um nome de banco, ou uma lista separada por vírgula

USER pode ser "all", um nome de usuário, um nome de grupo prefixado com "+", ou uma lista separada por vírgula. Nos campos USER e DATABASE podemos escrever um nome de arquivo prefixado com "@" para incluir um arquivo.

CIDR-ADDRESS especifica os hosts, que podem ser um IP e a máscara CIDR, que é um inteiro entre 0 e 32 para IPv4 ou 128 para IPv6, que especifica o número de significância da máscara. Alternativamente podemos inserir um IP e a máscara em coluna separada para especificar um conjunto de hosts.

METHOD pode ser "trust", "reject", "md5", "crypt", "password", "gss", "sspi", "krb5", "ident", "pam" ou "ldap". Note que "password" envia senhas em texto claro; "md5" deve ser preferida desde que envia senhas criptografadas.

OPTION é o mapa ident ou o nome do serviço PAM, dependendo do METHOD.

Obs.: Nomes de bancos e de usuários contendo espaços, vírgulas, aspas e outros caracteres especiais devem vir entre aspas duplas.

Após alterar este arquivo pode usar o "pg_ctl -D data reload" para atualizar as alterações.

Trecho principal do arquivo pg_hba.conf padrão do PostgreSQL 8.3 no Windows

```
# TYPE DATABASE  USER      CIDR-ADDRESS      METHOD [OPTION]
# IPv4 local connections:

host      all       all       127.0.0.1/32      md5
# IPv6 local connections:
#host     all       all       ::1/128           md5
```

Trecho principal do arquivo pg_hba.conf padrão do PostgreSQL 8.3 no Linux/Ubuntu 7.10

```
# TYPE DATABASE  USER      CIDR-ADDRESS      METHOD [OPTION]
# "local" is for Unix domain socket connections only
```



```
local    all        all                                ident    sameuser
```

```
# IPv4 local connections:
```

```
host     all        all        127.0.0.1/32      md5
```

```
# IPv6 local connections:
```

```
host     all        all        ::1/128           md5
```

Alguns exemplos

```
# 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
```

A coluna banco de dados também pode utilizar listas e nomes de arquivos, mas não grupos:

```
local db1,db2,@demodbs all md5
```

Detalhes em: <http://pgdocptbr.sourceforge.net/pg80/client-authentication.html>

Criando uma lista de usuários para controlar o acesso ao banco de dados..

É possível definir no pg_hba.conf para ir buscar os usuarios que tem acesso ao PostgreSQL em algum arquivo.

Imagine tendo que criar centenas de usuários, então perceberá a utilidade deste recurso.

No caso segue os seguintes passos:

- Sugestão: Criar um arquivo "users" (sem extensão).
- Inserir alguns usuários de teste (Ex: alice)
- Salvar o arquivo no mesmo diretório do pg_hba.conf, no caso do ubuntu em /etc/postgresql/8.3/main ,

não esquecendo de fornecer permissões do arquivo para o usuário postgres

- Comentar as configurações originais (caso ocorra algum imprevisto, sempre é bom ter as configurações iniciais).
- E informar as configurações escolhidas. No caso foi:

```
# "local" is for Unix domain socket connections only
```

```
#local    all    all    ident sameuser
```

```
local     all    @usersmd5
```

- Salvar o pg_hba.conf.
- Reiniciar o serviço do PostgreSQL

Colaboração do Paulo Alceu (aluno da primeira turma do curso DBA PostgreSQL)

3.2) Arquivo de senhas pgpass.conf

O arquivo `.pgpass`, armazenado na pasta base (home) do usuário, é um arquivo que contém senhas a serem utilizadas se a conexão requisitar uma senha (e a senha não tiver sido especificada de outra maneira). No Microsoft Windows o arquivo se chama `%APPDATA%\postgresql\pgpass.conf` (onde `%APPDATA%` se refere ao subdiretório de dados do aplicativo no perfil do usuário).

Este arquivo deve conter linhas com o seguinte formato:

```
host:porta:banco:usuario:senha
```

Caso pretenda que o usuário 'copia' tenha acesso sem senha ao host '10.0.0.5' na porta 5432 e a todos os bancos, adicionar a linha abaixo ao arquivo `pgpass.conf` (windows) ou `.pgpass` (linux):

```
10.0.0.5:5432:*:copia:
```

Os quatro primeiros valores podem ser um literal, ou `*` para corresponder a qualquer coisa. É utilizada a senha da primeira linha que corresponder aos parâmetros da conexão corrente (portanto, as entradas mais específicas devem ser colocadas primeiro quando são utilizados curingas). Se a entrada precisar conter os caracteres `:` ou `\`, estes caracteres devem receber o escape de `\`.

O arquivo `.pgpass` não deve permitir o acesso por todos os usuários ou grupos; isto é conseguido pelo comando `chmod 0600 ~/.pgpass`. Se as permissões forem mais rígidas que esta, o arquivo será ignorado (Entretanto, atualmente as permissões não são verificadas no Microsoft Windows).

Detalhes em: <http://pgdocptbr.sourceforge.net/pg80/libpq-pgpass.html> e <http://www.postgresql.org/docs/8.3/interactive/libpq-pgpass.html>

3.3) Entendendo e manipulando o arquivo pg_ident.conf

O método de autenticação ident funciona obtendo o nome de usuário do sistema operacional do cliente, e determinando os nomes de usuário do PostgreSQL permitidos utilizando um arquivo de mapa que lista os pares de nomes de usuários correspondentes permitidos.

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

Este script de autenticação é utilizado nos UNIX, pois no Windows o usuário não é usuário com direito a login.

Exemplo de entrada no arquivo pg_hba.conf:

```
host    all    all    10.0.0.7/32 ident      mapateste
```

Todos os usuários da máquina 10.0.0.7 terão acesso a todos os bancos do PostgreSQL. Os usuários desta máquina serão mapeados pelo mapa ident mapateste. No caso teremos os seguintes registros no pg_ident.conf:

```
mapateste joao    joaozinho
mapateste mich    michael
mapateste dani    daniel
```

Quando o usuário joao (em 10.0.0.7) se conecta remotamente ao servidor do PostgreSQL ele será mapeado para o usuário joaozinho, que é uma conta do sistema no servidor. Os demais de forma similar.

Caso precise usar este arquivo e ele não exista, crie no diretório sugerido acima, com as entradas desejadas.

Mais detalhes em:

<http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=postgresql.conf.pdf>

<http://pgdocptbr.sourceforge.net/pg80/auth-methods.html#AUTH-IDENT>

<http://www.postgresql.org/docs/8.3/interactive/client-authentication.html>

Livro PostgreSQL 8 for Windows de Richard Blum, capítulo 3.

Alerta Ao consultar a documentação fique atento para a versão do PostgreSQL que estás utilizando. Fortemente recomendada a consulta ao script postgresql.conf e sua documentação (comentários).

Exercícios

1) Avaliando o desempenho de índices

Cenário:

Banco de dados de CEPs com 633.401 registros cep_brasil.

tabela cep_full sem índice

tabela cep_full_index com índice

Criando o banco com codificação win1252:

```
postgres=# create database cep_brasil encoding 'win1252';
```

```
postgres=# \c cep_brasil
```

```
create table cep_full (  
cep char(8),  
tipo char(72),  
logradouro char(70),  
bairro char(72),  
municipio char(60),  
uf char(2)  
);
```

Importando do CSV (http://tudoemum.ribafs.net/includes/cep_brasil.sql.bz2):

```
\copy cep_full from E:\Enviar\cep_brasil_unique.csv
```

Essa é interessante para quem não conhece. Crio uma segunda tabela tendo como base uma consulta sobre outra tabela, já importando todos os registros:

```
create table cep_full_index as select * from cep_full;
```

Alterar a tabela cep_full_index adicionando índice:

```
alter table cep_full_index add constraint cep_pk primary key (cep);
```

Atualizando as estatísticas:

```
vacuum analyze;
```

Pegando o plano de consulta da primeira consulta:

```
explain select logradouro from cep_full where cep='60420440';
```

QUERY PLAN

```
-----  
Seq Scan on cep_full (cost=0.00..33254.51 rows=1 width=71)  
  Filter: (cep = '60420440'::bpchar)  
(2 rows)
```

O plano da segunda:

```
explain select logradouro from cep_full_index where cep='60420440';
```

QUERY PLAN

```
-----  
Index Scan using cep_pk on cep_full_index (cost=0.00..8.37 rows=1 width=71)  
  Index Cond: (cep = '60420440'::bpchar)  
(2 rows)
```

Isso foi interessante, pois numa tabela grande gasta um tempo muito pequeno. Como eu estava fazendo, usando o \timing, a consulta demorava muito.

[1] – Adaptação de dica recebida do Roberto Mello (na lista pgbr-geral).

2) Acesso Remoto

- Visualizar os scripts originais: postgresql.conf, pg_hba.conf e o pg_ident.conf do micro servidor (do professor) do PostgreSQL
- Tentar conectar ao PostgreSQL numa máquina remota (outro micro da sala), usando o PGAdmin e o psql
- Observar as mensagens de erro
- Criar um novo usuário no micro cliente
- Liberar no postgresql.conf, no pg_hba.conf apenas para o usuário 'postgres', para o banco 'dbapg', para o IP 192.168.x.y
- Tentar novamente
- Liberar para toda a rede interna

Aproveitar para fazer um tour pelo PGAdmin.

Observe que as tablespaces e as roles ficam fora dos bancos.

3) Instalar o VirtualBox e nele o pg_live.

Observe que como no Windows, o pg_live traz a última versão do PGAdmin (1.82), como também traz o PostgreSQL 8.3 e o PHP rodando, além de outras ferramentas úteis.

Novo site de apoio ao curso

Para maior privacidade coloquei apenas para usuários registrados.

- Acesse <http://ribafs.net>
- Fazer seu registro, caso ainda não seja registrado e confirme o e-mail que receber.
- Faça login e acesse a seção Artigos – DBA PostgreSQL

Aqui estarei adicionando notícias, dicas e outros assuntos ligados direta ou indiretamente ao PostgreSQL.

3.4) Configurando o ambiente do PostgreSQL

Através do postgresql.conf e de comandos SQL 'SET'

Através do postgresql.conf

Lembrar que as linhas iniciadas com # estão comentadas e que todos os parâmetros comentados são o padrão. Caso queira alterar algum deixe comentado e copie a linha logo abaixo descomentada e com o valor desejado.

Alguns parâmetros importantes

#data_directory = 'ConfigDir'

Caso queira alterar o data_directory para outro diretório:

- copie o diretório atual ou mova-o para o novo diretório
- altere este parâmetro no postgresql.conf
- recarregue o serviço

#hba_file = 'ConfigDir/pg_hba.conf'

#listen_addresses = 'localhost'

Por default o PostgreSQL somente dá acesso para usuários locais, pela console.

Para dar permissão para acesso pelo PGAdmin, mesmo local e a outros hosts não locais devemos entrar com seu IP ou nome neste parâmetro.

Entrar com '*' para dar acesso a qualquer host.

Para uma lista de IPs separar por vírgula, assim: '10.0.0.7,10.0.0.8,10.0.0.9'.

Obs.: Mudanças requerem restartar serviço.

#port = 5432

#ssl = off

Uso de conexões seguras através do openssl. Mudanças requerem restartar serviço.

Para aumentar a segurança criptografando as comunicações cliente/servidor, o PostgreSQL possui suporte nativo para conexões SSL. É necessário que o OpenSSL esteja instalado tanto no cliente quanto no servidor, e que o suporte no PostgreSQL tenha sido ativado em tempo de instalação.

Detalhes em: <http://pgdocptbr.sourceforge.net/pg80/ssl-tcp.html> e <http://www.postgresql.org/docs/8.3/interactive/ssl-tcp.html>

#password_encryption = on

Quando é especificada uma senha em `CREATE USER` ou `ALTER USER`, sem que seja escrito `ENCRYPTED` ou `UNENCRYPTED`, este parâmetro determina se a senha deve ser criptografada. Atualmente recomendado o uso de senhas criptografadas.

max_connections = 100

Máximo de conexões simultâneas que serão atendidas pelo servidor. Manter o mais baixo possível, tendo em vista que cada conexão ativa requer recursos do hardware.

Mudanças requerem restartar serviço

autovacuum = on

habilita o subprocesso autovacuum

Para ativar, obrigatoriamente ative também *track_counts (na versão 8.3)*

#autovacuum_naptime = 1min

Tempo entre as execuções do vacuum

#datestyle = 'iso, mdy'

Formato/estilo da data

Para ter a data no formato do Brasil (dd/mm/yyyy) altere para: `datestyle = 'sql, dmy'`

#client_encoding = latin1

Na verdade, os padrões escolhidos pelo utilitário initdb são escritos no arquivo de configuração `postgresql.conf` apenas para servirem como padrão quando o servidor é inicializado. Se forem removidas as atribuições presentes no arquivo `postgresql.conf`, o servidor herdará as definições do ambiente de execução.

#lc_messages = 'C'

`lc_messages = 'pt_BR'` # Mensagens de erro em português do Brasil

Obs.: Alguns parâmetros do `postgresql.conf` podem ser alterados pelo comando "SET" do SQL e também pelo comando "ALTER".

Configurações Através do comando 'SET' do SQL

Através do comando SET podemos alterar as configurações do PostgreSQL.

O comando SET muda os parâmetros de configuração de tempo de execução. Muitos parâmetros de tempo de execução podem ser mudados dinamicamente pelo comando SET (Mas alguns requerem privilégio de superusuário para serem mudados, e outros não podem ser mudados após o servidor

ou a sessão iniciar). O comando SET afeta apenas os valores utilizados na sessão atual.

Para exibir o formato de data atual do PostgreSQL:

```
SHOW DATESTYLE;
```

Execute a consulta:

```
SELECT CURRENT_DATE;
```

Para alterar o estilo da data na seção atual para o formato dd-mm-yyyy:

```
SET DATESTYLE TO 'postgresql,dmy';
```

Execute novamente a consulta:

```
SELECT CURRENT_DATE;
```

Para alterar o estilo da data na seção atual para o formato dd/mm/yyyy:

```
SET DATESTYLE TO 'sql,dmy';
```

Execute novamente a consulta:

```
SELECT CURRENT_DATE;
```

```
SET DATESTYLE TO ISO;
```

```
SELECT CURRENT_TIMESTAMP;
```

```
SET DATESTYLE TO PostgreSQL;
```

```
SELECT CURRENT_DATE;
```

```
SET DATESTYLE TO US;
```

```
SELECT CURRENT_DATE;
```

```
SET DATESTYLE TO NONEUROPEAN, GERMAN;
```

```
SELECT CURRENT_DATE;
```

```
SET DATESTYLE TO EUROPEAN;
```

```
SELECT CURRENT_DATE;
```

Para exibir todas as configurações:

```
SHOW ALL;
```

Também podemos Configurar um banco para Aceitar datas no formato praticado no Brasil:

```
CREATE DATABASE bancobrasil;
```

```
ALTER DATABASE SET DATESTYLE TO 'SQL,DMY';
```

Também pode ser atribuído juntamente com o Usuário:

```
ALTER ROLE nomeuser SET DATESTYLE TO SQL, DMY;
```

Alerta: para bancos que já estejam em uso, geralmente fica inviável uma alteração permanente

(postgresql.conf), caso se esteja tratando as datas. Exemplo: ao consultar um campo data, a aplicação espera um formato e vai estar outro. A não ser que se altere consultas SQL existentes nos aplicativos.

```
ALTER DATABASE cep_brasil SET client_encoding=win1252;
```

ISO-8859-15 é o Latin9

```
SHOW ENABLE_SEQSCAN;  
SET ENABLE_SEQSCAN TO OFF;
```

Mais detalhes em:

<http://pgdocptbr.sourceforge.net/pg80/sql-set.html> e
<http://www.postgresql.org/docs/8.3/interactive/sql-set.html>

4) Administração de Grupos, Usuários e Privilégios

- 4.1) Gerenciando Grupos
- 4.2) Gerenciando Usuários
- 4.3) Gerenciando Permissões
- 4.4) Controle de acesso a Objetos
- 4.5) GRANT e REVOKE para Objetos
- 4.6) Acesso Remoto

DCL (Data Control Language) é formado por um grupo de comandos SQL, responsáveis pela administração dos usuários, dos grupos e dos privilégios.

Dois usos importantes deste assunto: organização (dividir os bancos e esquemas pelos setores de empresas e instituições) e segurança (cada usuário somente com suas permissões restritivas).

Garantindo (GRANT) Privilégios

<http://pgdocptbr.sourceforge.net/pg82/sql-grant.html>

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER }  
[,...] | ALL [ PRIVILEGES ] }  
ON [ TABLE ] nome_da_tabela [, ...]  
TO { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { USAGE | SELECT | UPDATE }  
[,...] | ALL [ PRIVILEGES ] }  
ON SEQUENCE nome_da_sequência [, ...]  
TO { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }  
ON DATABASE nome_do_banco_de_dados [, ...]  
TO { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

...

```
GRANT { CREATE | ALL [ PRIVILEGES ] }  
ON TABLESPACE nome_do_espaco_de_tabelas [, ...]  
TO { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

GRANT role [, ...] TO nome_do_usuario [, ...] [WITH ADMIN OPTION]

GRANT pode ser utilizado para conceder privilégios a um usuário sobre um objeto de banco de dados e para tornar um usuário membro de um papel (antigo grupo). A palavra chave PUBLIC indica que os privilégios devem ser concedido para todos os papéis, inclusive aos que vierem a ser criados posteriormente. PUBLIC pode ser considerado como um grupo definido implicitamente que sempre inclui todos os papéis.

Se for especificado WITH GRANT OPTION quem receber o privilégio poderá, por sua vez, conceder o privilégio a terceiros. Sem a opção de concessão, quem recebe não pode conceder o privilégio. A opção de concessão não pode ser concedida para PUBLIC.

Não é necessário conceder privilégios para o dono do objeto (geralmente o usuário que o criou), porque o dono possui todos os privilégios por padrão (Entretanto, o dono pode decidir revogar alguns de seus próprios privilégios por motivo de segurança). O direito de remover um objeto, ou de alterar a sua definição de alguma forma, não é descrito por um privilégio que possa ser concedido; é inerente ao dono e não pode ser concedido ou revogado.

Os privilégios possíveis são:

SELECT

Permite consultar (SELECT) qualquer coluna da tabela, visão ou sequência especificada. Também permite utilizar o comando COPY TO (exportar). Para as seqüências, este privilégio também permite o uso da função currval.

INSERT

Permite inserir (INSERT) novas linhas na tabela especificada. Também permite utilizar o comando COPY FROM (importar).

UPDATE

Permite modificar (UPDATE) os dados de qualquer coluna da tabela especificada. Os comandos SELECT ... FOR UPDATE e SELECT ... FOR SHARE também requerem este privilégio (além do privilégio SELECT). Para as seqüências, este privilégio permite o uso das funções nextval e setval.

DELETE

Permite excluir (DELETE) linhas da tabela especificada.

REFERENCES

Para criar uma restrição de chave estrangeira é necessário possuir este privilégio, tanto na tabela que faz referência quanto na tabela que é referenciada.

TRIGGER

Permite criar gatilhos na tabela especificada (Consulte o comando CREATE TRIGGER).

CREATE

Para bancos de dados, permite a criação de novos esquemas no banco de dados.

Para esquemas, permite a criação de novos objetos no esquema. Para mudar o nome de um objeto existente é necessário ser o dono do objeto e possuir este privilégio no esquema que o contém.

Para espaços de tabelas (tablespaces), permite a criação de tabelas e índices no espaço de tabelas, e permite a criação de bancos de dados possuindo este espaço de tabelas como seu espaço de tabelas padrão (Deve ser observado que revogar este privilégio não altera a colocação dos objetos existentes).

CONNECT

Permite ao usuário se conectar ao banco de dados especificado. Este privilégio é verificado no estabelecimento da conexão (além de serem verificadas as restrições impostas por pg_hba.conf).

TEMPORARY

TEMP

Permite a criação de tabelas temporárias ao usar o banco de dados.

EXECUTE

Permite utilizar a função especificada (internas) e qualquer operador implementado utilizando a função. Este é o único tipo de privilégio aplicável às funções (Esta sintaxe funciona para as funções de agregação também).

USAGE

Para as linguagens procedurais, permite o uso da linguagem especificada para criar funções nesta linguagem. Este é o único tipo de privilégio aplicável às linguagens procedurais.

Para os esquemas, permite acessar os objetos contidos no esquema especificado (assumindo que os privilégios requeridos para os próprios objetos estejam atendidos). Essencialmente, concede a quem recebe o direito de "procurar" por objetos dentro do esquema. Sem esta permissão ainda é possível ver os nomes dos objetos, por exemplo consultando as tabelas do sistema. Além disso, após esta permissão ter sido revogada os servidores existentes poderão conter comandos que realizaram anteriormente esta procura, portanto esta não é uma forma inteiramente segura de impedir o acesso aos objetos.

Para as seqüências este privilégio permite a utilização das funções currval e nextval.

ALL PRIVILEGES

Concede todos os privilégios disponíveis de uma só vez. A palavra chave PRIVILEGES é opcional no PostgreSQL, embora seja requerida pelo SQL estrito.

Revogando (REVOKE) Privilégios

<http://pgdocptbr.sourceforge.net/pg82/sql-revoke.html>

```
REVOKE [ GRANT OPTION FOR ]  
{ { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER }  
[,...] | ALL [ PRIVILEGES ] }  
ON [ TABLE ] nome_da_tabela [, ...]  
FROM { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]  
{ { USAGE | SELECT | UPDATE }  
[,...] | ALL [ PRIVILEGES ] }  
ON SEQUENCE nome_da_sequência [, ...]  
FROM { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]  
{ { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }  
ON DATABASE nome_do_banco_de_dados [, ...]  
FROM { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

...

```
REVOKE [ GRANT OPTION FOR ]  
{ { CREATE | USAGE } [,...] | ALL [ PRIVILEGES ] }  
ON SCHEMA nome_do_esquema [, ...]  
FROM { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]  
{ CREATE | ALL [ PRIVILEGES ] }  
ON TABLESPACE nome_do_espaco_de_tabelas [, ...]  
FROM { nome_do_usuario | GROUP nome_do_grupo | PUBLIC } [, ...]  
[ CASCADE | RESTRICT ]
```

```
REVOKE [ ADMIN OPTION FOR ]  
role [, ...] FROM nome_do_usuario [, ...]  
[ CASCADE | RESTRICT ]
```

O comando REVOKE revoga, de um ou mais papéis, privilégios concedidos anteriormente. A palavra chave PUBLIC se refere ao grupo contendo todos os usuários, definido implicitamente.

Se for especificado GRANT OPTION FOR somente a opção de concessão do privilégio é revogada, e não o próprio privilégio. Caso contrário, tanto o privilégio quanto a opção de concessão serão revogados.

Se, por exemplo, o usuário A concedeu um privilégio com opção de concessão para o usuário B, e o usuário B por sua vez concedeu o privilégio para o usuário C, então o usuário A não poderá revogar diretamente o privilégio de C. Em vez disso, o usuário A poderá revogar a opção de concessão do usuário B usando a opção CASCADE, para que o privilégio seja, por sua vez, revogado do usuário C.

Usuários, grupos e privilégios

De fora do psql (no prompt do SO) utiliza-se comandos sem espaços: createdb, dropdb, etc.
De dentro do psql os comandos são formados por duas palavras separadas por espaço:
CREATE DATABASE, DROP DATABASE, etc (sintaxe SQL).

Em SQL:

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 do script postgresql.conf.

Criar Usuário

```
CREATE ROLE nomeusuario;
```

Criar um Usuário local com Senha:

Entrar no pg_hba.conf:

```
local all all 127.0.0.1/32 md5
```

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

Criamos assim:

```
CREATE ROLE nomeuser WITH LOGIN ENCRYPTED PASSWORD '*****';
```

Ao se logar:

```
psql -U nomeuser nomebanco.
```

```
CREATE ROLE nomeusuario VALID UNTIL 'data'
```

Excluindo Usuário

```
DROP ROLE nomeusuario;
```

Como usuário, na linha de comando:

Criar Usuário:

```
createuser -U postgres nomeusuario
```

Excluindo Usuário

```
dropuser -U postgres nomeusuario;
```

Criando Superusuário

```
CREATE ROLE nomeuser WITH LOGIN SUPERUSER ENCRYPTED PASSWORD '*****';  
-- usuário com poderes de super usuário
```

Alterar Conta de Usuário

```
ALTER ROLE nomeuser ENCRYPTED PASSWORD '*****' CREATEUSER
```

```
-- permissão de criar usuários
```

```
ALTER ROLE nomeuser VALID UNTIL '12/05/2006';
```

```
ALTER ROLE fred VALID UNTIL 'infinity';
```

```
ALTER ROLE miriam CREATEROLE CREATEDB; -- poderes para criar bancos
```

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

Listando todos os usuários do SGBD:

```
SELECT username FROM pg_user;
```

A tabela pg_user é uma tabela de sistema (_pg) que guarda todos os usuários do PostgreSQL.

Também podemos utilizar:

\du ou \dg

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 de uma Tabela

```
REVOKE ALL ON nometabela FROM PUBLIC
```

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 ROLE dbas;
```

```
CREATE ROLE dba1 ENCRYPTED PASSWORD 'dba1' CREATEDB CREATEROLE;
```

-- Criação dos Usuários do Grupo adm

```
CREATE ROLE andre ENCRYPTED PASSWORD 'andre' CREATEDB IN ROLE adm;
```

```
CREATE ROLE michela ENCRYPTED PASSWORD 'michela' CREATEDB IN ROLE adm;
```

O usuário de sistema (super usuário) deve ser um usuário criado exclusivamente para o PostgreSQL. Nunca devemos torná-lo dono de nenhum executável, como também devemos evitar o uso do super-usuário para administrar o SGBD. Para isso é recomendado criar um usuário com algumas restrições e utilizar o super-usuário somente se estritamente necessário.

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.

DICA

Exemplo: para permitir a um usuário apenas os privilégios de INSERT, UPDATE e SELECT e não permitir o de DELETE em uma tabela, use:

- 1) REVOKE ALL ON tabela FROM usuario;
- 2) GRANT SELECT,UPDATE,INSERT ON tabela TO usuario;

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>

Privilégios com o PGAdmin

ALL

Concede todos os privilégios disponíveis de uma só vez. A palavra chave PRIVILEGES é opcional

no PostgreSQL, embora seja requerida pelo SQL estrito.

TEMPORARY TEMP

Permite a criação de tabelas temporárias ao usar o banco de dados.

CREATE

Para bancos de dados, permite a criação de novos esquemas no banco de dados.

Para esquemas, permite a criação de novos objetos no esquema. Para mudar o nome de um objeto existente é necessário ser o dono do objeto e possuir este privilégio no esquema que o contém.

Para espaços de tabelas, permite a criação de tabelas e índices no espaço de tabelas, e permite a criação de bancos de dados possuindo este espaço de tabelas como seu espaço de tabelas padrão (Deve ser observado que revogar este privilégio não altera a colocação dos objetos existentes).

CONNECT

Permite ao usuário se conectar ao banco de dados especificado. Este privilégio é verificado no estabelecimento da conexão (além de serem verificadas as restrições impostas por pg_hba.conf).

WITH GRANT OPTION

As concessões e revogações também podem ser realizadas por um papel que não é o dono do objeto afetado, mas é membro do papel que é dono do objeto, ou é um membro de um papel que possui privilégios com WITH GRANT OPTION no objeto. Nestes casos, os privilégios serão registrados como tendo sido concedidos pelo papel que realmente possui o objeto, ou possui os privilégios com WITH GRANT OPTION.

Exercícios – Gerenciando grupos e usuários (roles)

create group e create user ainda podem ser utilizados mas recomenda-se create role ao invés.

```
create role dbas;
```

Criar Superusuário:

```
create role super2 superuser login connection limit 40 password 'super'  
-- default é -1 (sem limites)
```

Estando usando o método ident e opção sameuser no pg_hba.conf, um usuário criado com login no psql, somente poderá acessar o postgresql se também for criado no sistema operacional.

```
sudo adduser super2  
su - super2  
psql -U super2 postgres
```

Criar usuário:

```
create role usuario2;
```

Criar Usuário

Na linha de comando do SO: `createuser -U dba usuario`

Comandos SQL:

```
create role usuario;  
create role with login;  
create role with login password 'usuario' valid until '2007-12-31';
```

Adicionar usuário ao grupo:

```
alter role adm role usuario2; // Adiciona o usuário2 no grupo adm  
alter role adm role usuario3;  
alter role adm role usuario4;
```

Excluir usuário:

```
drop role usuario2;
```

Remover apenas do Grupo, sem remover o usuário

```
revoke adm from usuario2;
```

Observação Importante

O comando "alter role ..." também pode ser utilizado para alterar praticamente todas as propriedades de usuários ou grupos, exceto para adicionar ou remover usuário de grupos, que agora ficam a cargo dos comandos GRANT e REVOKE.

Os usuários usuario2, usuario3 e usuario4 farão parte do grupo adm.

Listando todos os usuários do SGBD

```
SELECT username FROM pg_user;
```

\du - listar usuários

ou

\dg - lista roles e respectivos grupos (Membro de ...)

Listar os grupos

```
SELECT groname FROM pg_group;
```

Excluir Usuário ou Grupo:

```
drop role nomerole;
```

Concedendo (GRANT) e Revogando (REVOKE) Privilégios

Após a criação de uma tabela execute:

```
\z clientes
```

Verá:

Privilégios de acesso ao banco dados "teste2"

Esquema | Nome | Tipo | Privilégios de acesso

-----+-----+-----+-----

public | clientes | tabela |

(1 linha)

Podemos conceder/revogar Privilégios de

- esquemas

- funções

- procedures

- sequências

- tabelas (todas ou individualmente)

- views

Agora conceda alguns privilégios:

```
GRANT SELECT ON clientes TO PUBLIC; -- Todas as permissões
```

```
GRANT SELECT, UPDATE, INSERT ON clientes TO GROUP progs;
```

Execute novamente:

```
\z clientes
```

Veja agora o resultado:

Privilégios de acesso ao banco dados "teste2"

Esquema | Nome | Tipo | Privilégios de acesso

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
public | clientes | tabela | {usuario1=arwdRxt/miriam,=r/usuario1,"group todos=arw/usuario1"}
(1 linha)
```

As entradas mostradas pelo comando \z são interpretadas da seguinte forma:

```
=xxxx -- privilégios concedidos para PUBLIC
uname=xxxx -- privilégios concedidos para o usuário
group gname=xxxx -- privilégios concedidos para o grupo
```

```
r -- SELECT ("read")
w -- UPDATE ("write")
a -- INSERT ("append")
d -- DELETE
x -- REFERENCES
t -- TRIGGER
X -- EXECUTE
U -- USAGE
C -- CREATE
c -- CONNECT
T -- TEMPORARY
arwdxt -- ALL PRIVILEGES (para tabelas)
```

```
* -- opção de concessão para o privilégio precedente
```

```
/yyyy -- usuário que concedeu o privilégio
```

Se a coluna "Access privileges/Privilégios de acesso" estiver vazia para um determinado objeto, isto significa que o objeto possui os privilégios padrão. Os privilégios padrão sempre incluem todos os privilégios para o dono

Deve ser observado que as opções de concessão implícitas do dono não são marcadas na visualização dos privilégios de acesso. O * aparece somente quando as opções de concessão foram concedidas explicitamente para alguém.

Criar Usuário para Testes:

```
psql -U postgres
```

```
create role uteste;
```

```
alter role uteste with password 'uteste' login createdb createrole;
```

```
\c – uteste
```

```
create database bteste;
```

```
\c bteste
```

```
create table tabelat (codigo int primary key);
```

Quando um usuário cria um objeto ele é o dono deste objeto.

Se queremos ter segurança ao conceder privilégios a um usuário sobre um objeto, devemos antes remover todos os privilégios do mesmo sobre o objeto. Depois então conceder somente os privilégios desejados. Como no exemplo abaixo:

```
\c – postgres
```

```
revoke all privileges on tabelat from uteste;
```

```
\c bteste uteste
```

```
select * from tabelat; // receberemos um erro de permissão negada, já que removemos todos os  
privilégios do uteste sobre a tabela tabelat.
```

```
\c – postgres
```

```
grant select on tabelat to uteste;
```

```
\c – uteste
```

```
select * from tabelat; // Agora a consulta é permitida
```

Se quisermos conceder todos os privilégios existentes ao usuário usamos:

```
grant all privileges on tabelat to uteste;
```

Observar que quando existir um campo do tipo serial, temos que conceder também os privilégios para a sequência gerada pelo serial. Para saber o nome exato da sequência execute o comando \d. Geralmente o PG usa a regra *serial_nomecampo_seq*.

Conceder os privilégios de SELECT, UPDATE e INSERT na tabela tabelat ao usuário uteste:

```
\c - postgres
```

```
grant select, update, insert on tabelat to uteste;
```

```
\z
```

Agora teste os privilégios concedidos e também o delete, não concedido:

```
\c – uteste
```

```
select * from tabelat; // Ok
```

```
insert into tabelat values (1); // Ok
```

```
update tabelat set codigo = 5 where codigo =1; // Ok
```

```
delete tabelat where codigo = 5; // Permissão negada
```

Revogando (REVOKE) Privilégios

Revogar o privilégio de inserção na tabela filmes concedido para todos os usuários:

```
REVOKE INSERT ON filmes FROM PUBLIC;
```

Revogar todos os privilégios concedidos ao usuário manuel sobre a visão vis_tipos:

```
REVOKE ALL PRIVILEGES ON vis_tipos FROM manuel;
```

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.

```
-- Para que nenhum usuário tenha acesso à tabela use:
```

```
-- REVOKE ALL ON nometabela FROM PUBLIC;
```

Acesso Remoto

- Visualizar os scripts originais: postgresql.conf, pg_hba.conf e o pg_ident.conf do micro servidor (do professor) do PostgreSQL
- Tentar conectar ao PostgreSQL numa máquina remota (outro micro da sala), usando o PGAdmin e o psql
- Observar as mensagens de erro
- Criar um novo usuário no micro cliente
- Liberar no postgresql.conf, no pg_hba.conf apenas para o usuário 'postgres', para o banco 'dbapg', para o IP 192.168.x.y
- Tentar novamente
- Liberar para toda a rede interna

Aproveitar para fazer um tour pelo PGAdmin.

Observe que as tablespaces e as roles ficam fora dos bancos.

5) Manutenção do Servidor do PostgreSQL e dos Bancos

- 5.1) Instalando o PostgreSQL no Windows
- 5.2) Instalando o PostgreSQL no Linux
- 5.3) Backup Lógico e restauração de bancos de dados
- 5.4) Backup e restauração física offline
- 5.5) Usando o Vacuum e Analyze
- 5.6) Atualizando e migrando entre versões
- 5.7) Manutenção no arquivo de logs (registro)

5.1) Instalando o PostgreSQL no Windows

Abordado no módulo 1.

5.2) Instalando o PostgreSQL no Linux

Obs.: Veja a aula extra para instalação no Linux Mandriva e no FreeBSD.

Instalando pelos binários dos Repositórios da Distribuição

Atualmente todas as grandes distribuições Linux tem o PostgreSQL em seus repositórios e algumas trazem a última versão ou a penúltima, como o Slackware e o Ubuntu.

Instalaremos em uma distribuição filha da distribuição Debian, no nosso caso a Ubuntu. Caso queira instalar em uma distribuição Linux com outra origem procure pelo repositório da mesma e pelos programas de instalação.

Aqui adotarei o apt-get para a instalação (outra opção é o Synaptic).

Instalando pelo terminal:

```
sudo apt-get install postgresql-8.3
```

Com isso teremos a versão 8.3 instalada, configurada e o servidor no ar pronto para ser usado.

Onde ficam os arquivos:

Diretório de dados - /var/lib/postgresql/8.3/main/base
Diretório do pg_hba.conf e do postgresql.conf - /etc/postgresql/8.3/main
autovacuum.conf - /etc/postgresql-common
pg_dump, pg_dumpall, psql e outros binários - /usr/bin

Testando via console

```
sudo -u postgres psql
```

Caso queira ter acesso à linha de comando como usuário postgres, conceda uma senha:

```
sudo passwd postgres
```

Instalação Através dos Fontes

Numa instalação através dos fontes, fazemos o download do código fonte do PostgreSQL do site oficial, então configuramos e compilamos.

Faremos uma instalação do PostgreSQL 8.3.1 no Linux Ubuntu 7.10.

Pré-requisitos para instalação do PostgreSQL num UNIX em geral:

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

Pré-requisitos para a instalação com os fontes no Ubuntu:

```
sudo apt-get install build-essential  
sudo apt-get install libreadline5-dev  
sudo apt-get install zlib1g-dev  
sudo apt-get install gettext
```

Site de Pacotes do Ubuntu

Caso alguma não seja encontrada nos repositórios, faça o download do site de pacotes do Ubuntu, que fica aqui: <http://packages.ubuntu.com/>

Sobre os pacotes .deb

Os pacotes deste site devem ser rigorosamente para a versão do Ubuntu que você está usando (7.10 é a versão atual) e para a arquitetura do seu hardware (geralmente i386). São pacotes .deb para distribuições Debian ou oriundas, no caso, específicos para o Ubuntu.

Instalar pacotes .deb com:

```
sudo dpkg -i nomepacote.deb
```

Ou simplesmente abrindo o gerenciador de arquivos (nautilus) e um duplo clique sobre o arquivo.

Obs.: estes pacotes podem mudar de nome devido ao aparecimento de novas versões.

Download dos fontes do PostgreSQL

<http://www.postgresql.org/ftp/source/>

Faça o download da versão mais recente no formato tar.gz (fique à vontade para outro formato).

Descompactando

Acesse a console e descompacte em /usr/local/src com:

```
sudo tar xzpf postgresql-8.3.0.tar.gz -C /usr/local/src
```

Acessando os fontes

```
cd /usr/local/src/postgresql-8.3.0
```

Caso execute o comando 'ls' verá os arquivos fontes prontos para serem convertidos em binários.

Verificando se existem as bibliotecas necessárias e configurando para a compilação:

```
sudo ./configure
```

Compilando (este demora um pouco)

```
sudo make
```

Instalando o PostgreSQL

```
sudo make install
```

Criando o grupo (caso ainda não exista)

```
sudo groupadd postgres
```

Criando o usuário, adicionando ao grupo criado e setando sua home

```
sudo useradd -g postgres -d /usr/local/pgsql postgres
```

Criando o diretório do cluster padrão

```
sudo mkdir /usr/local/pgsql/data
```

Tornando o usuário postgres o dono do cluster

```
sudo chown postgres:postgres /usr/local/pgsql/data
```

Atribindo uma senha ao usuário (super-usuário) postgres

```
sudo passwd postgres
```

Alternando do usuário atual para o usuário postgres

```
su – postgres
```

Criando o cluster, os scripts de configuração, os logs, etc

```
bin/initdb -D /usr/local/pgsql/data
```

Iniciando manualmente o servidor

```
bin/pg_ctl -D /usr/local/pgsql/data start
```

Criando um banco de testes

```
bin/createdb teste
```

Acessando a console interativa do PostgreSQL (psql)

```
bin/psql teste
```

Obs.: Isso criará um cluster com os locais e codificação padrão do SO e da nossa versão atual do PostgreSQL.

No nosso caso, PostgreSQL 8.3.1 e Ubuntu 7.1 em português do Brasil teremos algo assim:

pt_BR.UTF-8, ou seja, locais pt_BR (português do Brasil) e codificação de caracteres UTF-8.

Confira executando o meta-comando, que Lista os bancos:

```
\l
```

Serão listados: template0, template1, postgres e teste. Todos com codificação UTF-8.

A codificação UTF-8 é uma codificação que atende as exigências de uma empresa ou instituição globalizada ou que tem possibilidades de crescer e manter alguma informação diferente das suportadas pelo nosso latin1 (que neste aspecto está obsoleto).

Além de trazer seus bancos, por default em UTF-8, esta versão do PostgreSQL não mais dá suporte à criação de bancos com a codificação LATIN1, que é o ISO-8859-1.

Mas se mesmo assim desejar criar bancos com codificação LATIN1. Nesta etapa da instalação do PostgreSQL podemos criar um segundo ou vários clusters com suporte a LATIN1.

Copiando o Script de Inicialização (torna mais prático iniciar e parar o servidor)

```
sudo cp /usr/local/src/postgresql-8.3.0/contrib/start-script/linux /etc/init.d/postgresql-8.3.0
```

Tornar executável

```
sudo chmod u+x /etc/init.d/postgresql-8.3.0
```

postmaster ou pg_ctl – iniciam o processo do servidor responsável por escutar por pedidos de conexão.

5.3) Backup Lógico e restauração de bancos de dados

pg_dump – faz o backup de um banco de dados do PostgreSQL em um arquivo de script (texto puro) ou de outro tipo (tar ou outro).

pg_dump opções banco

São feitas cópias de segurança consistentes, mesmo que o banco de dados esteja sendo utilizado ao mesmo tempo. O pg_dump não bloqueia os outros usuários que estão acessando o banco de dados (leitura ou escrita).

As cópias de segurança podem ser feitas no formato de *script (p)*, *customizado (c)* ou *tar (t)*.

O formato personalizado, por padrão é compactado.

Para restaurar a partir de scripts em texto devemos usar o comando psql.

Para restaurar scripts feitos com formato personalizado (-Fc) e tar (-Ft) devemos usar o pg_restore. Por default customizado é compactado.

Algumas Opções

- a (backup somente dos dados)
- b (incluir objetos grandes no backup)
- c (incluir comandos para remover – DROP, os objetos do banco antes de criá-lo)
- C (criar o banco e conectar ao mesmo)
- d (salvar os registros usando o comando INSERT ao invés do COPY). Reduz o desempenho. Somente se necessário (migração de SGBDs por exemplo).
- D (salvar os registros usando o comando INSERT ao invés do COPY explicitando os nomes das colunas).
- E codificação (passar uma codificação no backup)
- f arquivo (gerar backup para um arquivo)
- F formato (formato de saída. p – para texto plano, que é o padrão, c – para custom e t – para tar)
- i (ignorar a diferença de versão entre o pg_dump e o servidor)
- n esquema (backup apenas do esquema citado)
- N esquema (no backup NÃO incluir o esquema citado)
- s (backup somente a estrutura do banco, sem os dados)

- t tabela (salvar somente a tabela, sequência ou visão citada)
- T tabela (NÃO realizar o backup da referida tabela)
- disable-triggers (aplica-se somente quando salvando só os dados. Para desativar as triggers enquanto os dados são carregados).
- h host (host ou IP)
- p porta (porta)
- U usuário (usuário)
- W (forçar solicitação de senha)

Mais detalhes em:

<http://pgdocptbr.sourceforge.net/pg82/app-pgdump.html>

Observações

Se o agrupamento de bancos de dados tiver alguma adição local ao banco de dados template1, deve-se ter o cuidado de restaurar a saída do pg_dump em um banco de dados totalmente vazio; senão, poderão acontecer erros devido a definições duplicadas dos objetos adicionados. Para criar um banco de dados vazio, sem nenhuma adição local, deve-se fazê-lo a partir de template0, e não de template1 como, por exemplo:

```
CREATE DATABASE banco WITH TEMPLATE template0;
```

O backup no formato tar tem limitação para cada tabela, que devem ter no máximo 8GB no formato texto puro. O total dos objetos não tem limite de tamanho mas seus objetos sim.

É aconselhável executar o ANALYZE após restaurar de uma cópia de segurança para garantir um bom desempenho.

O pg_dump também pode ler bancos de dados de um PostgreSQL mais antigo, entretanto geralmente não consegue ler bancos de dados de um PostgreSQL mais recente.

Exemplos

Para salvar o banco de dados chamado meu_bd em um arquivo de script SQL:

```
pg_dump -U postgres meu_bd > bd.sql
```

Para restaurar este script no banco de dados (recém criado) chamado novo_bd:

```
dropdb novo_bd  
createdb novo_bd  
psql -U postgres -d novo_bd -f bd.sql
```

Para efetuar backup do banco de dados no script com formato customizado:

```
pg_dump -U postgres -Fc meu_bd > bd.dump
```

Para recarregar esta cópia de segurança no banco de dados (recém criado) chamado novo_bd:

```
pg_restore -U postgres -d novo_bd bd.dump
```

Para salvar uma única tabela chamada minha_tabela:

```
pg_dump -U postgres -t minha_tabela meu_bd > bd.sql
```

Para salvar todas as tabelas cujos nomes começam por emp no esquema detroit, exceto a tabela chamada empregado_log:

```
pg_dump -U postgres -t 'detroit.emp*' -T detroit.empregado_log meu_bd > bd.sql
```

Para salvar todos os esquemas cujos nomes começam por leste ou oeste e terminam por gsm, excluindo todos os esquemas cujos nomes contenham a palavra teste:

```
pg_dump -U postgres -n 'leste*gsm' -n 'oeste*gsm' -N '*teste*' meu_bd > bd.sql
```

A mesma coisa utilizando a notação de expressão regular para unificar as chaves:

```
pg_dump -U postgres -n '(leste|oeste)*gsm' -N '*teste*' meu_bd > bd.sql
```

Para salvar todos os objetos do banco de dados, exceto as tabelas cujos nomes começam por ts_:

```
pg_dump -U postgres -T 'ts_*' meu_bd > bd.sql
```

Backup dos usuários e grupos e depois de todos os bancos:

```
pg_dumpall -g  
pg_dump -Fc para cada banco
```

<http://www.pgadmin.org/docs/1.6/backup.html>

`pg_dumpall`

Salva todos os bancos de um agrupamento de bancos de dados do PostgreSQL em um único arquivo de script.

pg_dumpall opções

O `pg_dumpall` também salva os objetos globais, comuns a todos os bancos de dados (O `pg_dump` não salva estes objetos). Atualmente são incluídas informações sobre os usuários do banco de dados e grupos, e permissões de acesso aplicadas aos bancos de dados como um todo.

Necessário conectar como um superusuário para poder gerar uma cópia de segurança completa. Também serão necessários privilégios de superusuário para executar o *script* produzido, para poder adicionar usuários e grupos, e para poder criar os bancos de dados.

O `pg_dumpall` precisa conectar várias vezes ao servidor PostgreSQL (uma vez para cada banco de dados). Se for utilizada autenticação por senha, provavelmente será solicitada a senha cada uma destas vezes. Neste caso é conveniente existir o arquivo `~/.pgpass` ou `pgpass.conf` (win)

Algumas Opções

-a (realizar backup somente dos dados)
-c (insere comandos para remover os bancos de dados – DROP antes dos comandos para criá-los)
-d (realiza o backup inserindo comandos INSERT ao invés do COPY). Fica lenta.
-D (backup usando INSERT ao invés de COPY e explicitando os nomes de campos)
-g (salva somente os objetos globais do SGBD, ou seja, usuários e grupos)
e várias outras semelhante ao comando pg_dump.

Mais opções em: <http://pgdocptbr.sourceforge.net/pg82/app-pg-dumpall.html>

Exemplos

```
pg_dumpall -U postgres > todos.sql
```

Restaurando:

```
psql -U postgres -f todos.sql banco
```

Aqui o banco pode ser qualquer um, pois o script gerado pelo pg_dumpall contém os comandos para a criação dos bancos e conexão aos mesmos, de acordo com o original.

Backup Full Compactado (Linux)

```
pg_dumpall -U postgres | gzip > full.gz
```

```
cat full.gz | gunzip | psql template1
```

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 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
```

No Windows

```
pg_dump -f D:\MYDB_BCP -Fc -x -h localhost -U postgres MYDB
```

```
===== SCRIPT BAT =====
```

```
@echo off
rem (Nome do Usuário do banco para realizar o backup)
SET PGUSER=postgres
rem (Senha do usuário acima)
SET PGPASSWORD=1234
rem (Indo para a raiz do disco)
C:
rem (Selecionando a pasta onde será realizada o backup)
chdir C:\Sistemas\backup_postgresql
rem (banco.sql é o nome que definir para o meu backup)
rem (Deletando o backup existente, só por precaução)
del banco.sql
echo "Aguarde, realizando o backup do Banco de Dados"
pg_dump -i -U postgres -b -o -f "C:\Sistemas\backup_postgresql\banco%Date%.sql" banco
rem (sair da tela depois do backup)
exit
```

Por default o psql continua caso encontre um erro, mas podemos configurá-lo para parar caso encontre um:

```
\set ON_ERROR_STOP
```

Script de Backup no Windows

```
=====backup.bat=Formato: banco_dd-mm-aaaa_hh-mm=====
```

```
for /f "tokens=1,2,3,4 delims=/ " %%a in ('DATE /T') do set Data=%%b-%%c-%%d
for /f "tokens=1 delims=: " %%h in ('time /T') do set hour=%%h
```

```
for /f "tokens=2 delims=: " %%m in ('time /T') do set minutes=%%m
for /f "tokens=3 delims=: " %%a in ('time /T') do set ampm=%%a
set Hora=%hour%-%%minutes%-%%ampm%
```

```
pg_dump -U postgres controle_estoque -p 5222 -f "controle_estoque_%Data%_%Hora%.sql"
```

pg_restore - restaura um banco de dados do PostgreSQL a partir de um arquivo criado pelo pg_dump no formato custom (-Fc) ou tar (-Ft)

Importante: no Windows o pg_dumpall pedirá a senha para cada banco existente.

No pgpass.conf colocar * no campo bancodedados ou então faça uma cópia da linha para todos os bancos, inclusive tempalte0, template1 e postgres.

Ou não use pg_dumpall mas pg_dump apenas para os bancos desejados.

Na lista pgsql-general - <http://archives.postgresql.org/pgsql-general/2005-06/msg01279.php>

pg_restore

Objetivo - reconstruir o banco de dados no estado em que este se encontrava quando foi feito o backup. Para garantir a integridade dos dados, o banco a ser restaurado deve estar limpo, um banco criado recentemente e sem nenhum objeto ou registro.

- a (restaura somente os dados)
- c (exclui os objetos dos bancos antes de criar para restaurar)
- C (cria o banco antes de restaurar)
- d banco (restaura diretamente neste banco indicado). Caso esta opção não seja explicitada, a restauração será para o banco original do backup.
- e (encerra caso encontre erro. exit_on_error)
- f arquivo (restaura deste arquivo)
- i (ignorar diferença de versões entre o pg_restore e o servidor)
- L lista (restaura somente os objetos presentes no arquivo lista e na ordem)
- n esquema (restaura somente o esquema especificado)
- P função (restaura somente esta função)
- s (restaura somente o esquema do banco, não os dados)
- t tabela (restaura somente a definição e/ou dados da tabela)
- T gatilho (restaura somente este gatilho)
- W (força a solicitação da senha)
- l (a restauração ocorre em uma única transação. Tudo ocorrerá bem ou nada será salvo)

-h host
-p porta
-U usuário

Mais opções em: <http://pgdocptbr.sourceforge.net/pg82/app-pgrestore.html>

Para criar um banco de dados vazio, sem nenhuma adição local, deve-se fazê-lo partir de template0, e não de template1 como, por exemplo:

```
CREATE DATABASE banco WITH TEMPLATE template0;
```

Uma vez restaurado, é aconselhável executar o comando ANALYZE em todas as tabelas restauradas para que o otimizador possua estatísticas úteis.

Exemplos

Banco de dados meu_bd em um arquivo de cópia de segurança no formato personalizado:

```
pg_dump -U postgres -Fc meu_bd > db.dump
```

Para remover o banco de dados e recriá-lo a partir da cópia de segurança:

```
dropdb -U postgres meu_bd  
pg_restore -U postgres -C -d postgres db.dump
```

O banco de dados especificado na chave -d pode ser qualquer banco de dados existente no agrupamento; o pg_restore somente utiliza este banco de dados para emitir o comando CREATE DATABASE para meu_bd. Com a chave -C, os dados são sempre restaurados no banco de dados cujo nome aparece no arquivo de cópia de segurança.

Para recarregar a cópia de segurança em um banco de dados novo chamado bd_novo:

```
createdb -U postgres -T template0 bd_novo  
pg_restore -U postgres -d bd_novo db.dump
```

Deve ser observado que não foi utilizada a chave -C, e sim conectado diretamente ao banco de dados a ser restaurado. Também deve ser observado que o novo banco de dados foi clonado a partir de template0 e não de template1, para garantir que esteja inicialmente vazio.

pg_dumpall has lot of disadvantages compared to pg_dump -Fc, I don't see the point why one would want that.

What I'd recommend is to use pg_dumpall -g and pg_dump -Fc on each DB. Then get a copy of rdiff-backup for windows to create nice incremental backups. Wrap those 3 things in a cmd script and use the task scheduler to run it in a given interval.

Mais detalhes em:

<http://www.postgresql.org/docs/8.3/interactive/backup.html>

<http://pgdocptbr.sourceforge.net/pg80/backup.html>

5.4) Backup Físico offline (sistema de arquivos)

Uma estratégia alternativa para fazer cópia de segurança, é copiar diretamente os arquivos que o PostgreSQL usa para armazenar os dados dos bancos de dados.

Pode ser utilizada a forma preferida para fazer as cópias de segurança usuais dos arquivos do sistema como, por exemplo:

```
tar -cf copia_de_seguranca.tar /usr/local/pgsql/data
```

Entretanto, existem duas restrições que fazem com que este método seja impraticável ou, pelo menos, inferior ao `pg_dump`:

1. O servidor de banco de dados *deve* estar parado para que se possa obter uma cópia de segurança utilizável. Meias medidas, como impedir todas as conexões, não funcionam (principalmente porque o `tar`, e as ferramentas semelhantes, não capturam um instantâneo atômico do estado do sistema de arquivos em um determinado ponto no tempo). As informações sobre como parar o servidor podem ser encontradas na [Seção 16.6](#). É desnecessário dizer que também é necessário parar o servidor antes de restaurar os dados.
2. Caso tenha se aprofundado nos detalhes da organização do sistema de arquivos do banco de dados, poderá estar tentado a fazer cópias de segurança ou restauração de apenas algumas determinadas tabelas ou bancos de dados a partir de seus respectivos arquivos ou diretórios. Isto *não* funciona, porque as informações contidas nestes arquivos possuem apenas meia verdade. A outra metade está nos arquivos de registro de efetivação `pg_clog/*`, que contêm o status de efetivação de todas as transações. O arquivo da tabela somente possui utilidade com esta informação. É claro que também não é possível restaurar apenas uma tabela e seus dados associados em `pg_clog`, porque isto torna todas as outras tabelas do agrupamento de bancos de dados inúteis. Portanto, as cópias de segurança do sistema de arquivos somente funcionam para a restauração completa de todo o agrupamento de bancos de dados.

Deve ser observado que a cópia de segurança do sistema de arquivos não será necessariamente menor que a do Método SQL-dump. Ao contrário, é mais provável que seja maior; por exemplo, o `pg_dump` não necessita fazer cópia de segurança dos índices, mas apenas dos comandos para recriá-los.

Mais detalhes em:

<http://pgdocptbr.sourceforge.net/pg80/backup-file.html>

<http://www.postgresql.org/docs/8.3/interactive/backup-file.html>

5.5) Usando o Vacuum e Analyze

Uso do Vacuum

VACUUM

Vacuum - limpa e opcionalmente analisa um banco de dados. Recupera a área de armazenamento ocupada pelos registros excluídas. Na operação normal do PostgreSQL os registros excluídos, ou tornados obsoletos por causa de uma atualização, não são fisicamente removidos da tabela; permanecem presentes até o comando VACUUM ser executado. Portanto, é necessário executar o comando VACUUM periodicamente, especialmente em tabelas frequentemente atualizadas.

Sem nenhum parâmetro, o comando VACUUM processa todas as tabelas do banco de dados corrente. Com um parâmetro, o comando VACUUM processa somente esta tabela.

O comando VACUUM ANALYZE executa o VACUUM e depois o ANALYZE para cada tabela selecionada. Esta é uma forma de combinação útil para scripts de rotinas de manutenção. Para obter mais detalhes sobre o seu processamento deve ser consultado o comando ANALYZE.

O comando VACUUM simples (sem o FULL) apenas recupera o espaço, tornando-o disponível para ser reutilizado. Esta forma do comando pode operar em paralelo com a leitura e escrita normal da tabela, porque não é obtido um bloqueio exclusivo. O VACUUM FULL executa um processamento mais extenso, incluindo a movimentação das tuplas entre blocos para tentar compactar a tabela no menor número de blocos de disco possível. Esta forma é muito mais lenta, e requer o bloqueio exclusivo de cada tabela enquanto está sendo processada.

Parâmetros

FULL

Seleciona uma limpeza "completa", que pode recuperar mais espaço, mas é muito mais demorada e bloqueia a tabela no modo exclusivo.

FREEZE

Seleciona um "congelamento" agressivo das tuplas. Especificar FREEZE é equivalente a realizar o VACUUM com o parâmetro vacuum_freeze_min_age definido como zero. A opção FREEZE está em obsolescência e será removida em uma versão futura; em vez de utilizar esta opção deve ser definido o parâmetro vacuum_freeze_min_age. (adicionar ao postgresql.conf).
vacuum_freeze_min_age (integer)

Specifies the cutoff age (in transactions) that VACUUM should use to decide whether to replace transaction IDs with FrozenXID while scanning a table. The default is 100000000 (100 million). Although users can set this value anywhere from zero to 1000000000, VACUUM will silently limit the effective value to half the value of autovacuum_freeze_max_age, so that there is not an unreasonably short time between forced autovacuums. For more information see Seção 22.1.3.

A convenient way to examine this information is to execute queries such as SELECT relname, age(relfrozenxid) FROM pg_class WHERE relkind = 'r';
SELECT datname, age(datfrozenxid) FROM pg_database;

VERBOSE

Mostra, para cada tabela, um relatório detalhado da atividade de limpeza.

ANALYZE

Atualiza as estatísticas utilizadas pelo planejador para determinar o modo mais eficiente de executar um comando.

tabela

O nome (opcionalmente qualificado pelo esquema) da tabela específica a ser limpa. Por padrão todas as tabelas do banco de dados corrente.

coluna

O nome da coluna específica a ser analisada. Por padrão todas as colunas.

Executando o Vacuum Manualmente**Para uma tabela**

```
VACUUM ANALYZE tabela;  
VACUUM VERBOSE ANALYZE clientes;
```

Para todo um banco

```
\c nomebanco  
VACUUM FULL ANALYZE;
```

Encontrar as 5 maiores tabelas e índices

```
SELECT relname, relpages FROM pg_class ORDER BY relpages DESC LIMIT 5;
```

Ativando 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 checka o uso em baixo nível do coletor de estatísticas.

Só pode ser usado se stats_start_collector e stats_row_level forem alterados para true.

Veja detalhes na aula 5 do módulo 2 (Monitorando as Atividades do Servidor do PostgreSQL).

Por default será executado a cada 60 segundos. Para alterar descomente e mude a linha:

```
autovacuum_naptime = 60
```

Autovacuum

Assim que você entra em produção no 8.0, você vai querer fazer um plano de manutenção incluindo VACUUMs e ANALYZEs. Se seus bancos de dados envolvem um fluxo contínuo de escrita de dados, mas não requer a maciças cargas e apagamentos de dados ou freqüentes reinícios, isto significa que você deve configurar o pg_autovacuum. Isto é melhor que agendar vaccuns porque:

* Tabelas sofrem o vacuum baseados nas suas atividades, excluindo tabelas que apenas sofrem leituras.

* A frequência dos vacuums cresce automaticamente com o crescimento da atividade no banco de dados.

* É mais fácil calcular o mapa de espaço livre e evitar o inchaço do banco de dados.

Configurando o autovacuum requer a fácil compilação de um módulo do diretório contrib/pg_autovacuum da fonte do seu PostgreSQL (usuários Windows devem procurar o autovacuum incluído no pacote do instalador). Você liga as estatísticas de configuração detalhadas no README. Então você inicia o autovacuum depois de o PostgreSQL ser iniciado como um processo separado; ele será desligado automaticamente quando o PostgreSQL desligar.

As configurações padrões do autovacuum são muito conservadores, imagino, e são mais indicadas para bancos de dados muito pequenos. Eu geralmente uso algo mais agressivo como:

```
-D -v 400 -V 0.4 -a 100 -A 0.3
```

Isto irá rodar o vacuum nas tabelas após 400 linhas + 40% da tabela ser atualizada ou apagada e irá rodar o analyze após 100 linhas + 30% da tabelas sofres inserções, atualizações ou ser apagada. As configurações acima também me permitem configurar o meu max_fsm_pages para 50% das páginas de dados com a confiança de que este número não será subestimado gerando um inchaço no banco de dados. Nós atualmente estamos testando várias configurações na OSDL e teremos mais informações em breve.

Note que você também pode usar o autovacuum para configurar opções de atraso ao invés de configura-lo no PostgreSQL.conf. O atraso no Vacuum pode ser de vital importância em sistemas que tem tabelas e índices grandes; em último caso pode parar uma operação importante.

Existem infelizmente um par de limitações sérias para o autovacuum no 8.0 que serão eliminadas em versões futuras:

* Não tem memória de longa duração: autovacuum esquece toda a sua atividade quando você reinicia o banco de dados. Então se você reinicia regularmente, você deve realizar um VACUUM ANALYZE em todo o banco de dados imediatamente antes ou depois.

* Preste atenção em quanto o servidor está ocupado: há planos de checar a carga do servidor antes de realizar o vacuum, mas não é uma facilidade corrente. Então se você tem picos de carga extremos, o autovacuum não é para você.

Fábio Telles em - <http://www.midstorm.org/~telles/2006/10/>

5.6) Manutenção no arquivo de logs (registro)

Para saber a localização dos arquivos de logs confira o script postgresql.conf:

É uma boa prática, periodicamente, remover os arquivos mais antigos. Se será feito backup dos mesmos ou não, depende da importância dos mesmos.

O diretório pg_xlog contém os arquivos de logs das transações. Tamanhos fixos de 16MB.

O diretório pg_clog guarda o status das transações.

O diretório pg_subtrans contém o status das subtransações

No postgresql.conf temos:

```
#log_directory = 'pg_log'
```

- 1- Instalando o PostgreSQL no Windows
- 2- Instalando o PostgreSQL no Linux
- 3- Atualizando e migrando entre versões
- 4- Recuperando fisicamente os bancos

Instalações (itens 1 e 2), vide módulo 1 e aula 2 do módulo2.

5.7) – Atualizando e Migrando entre Versões

Existem dois tipos de atualização entre versões no PostgreSQL. Uma entre as versões menores e entre as maiores:

menores: com 3 dígitos (8.1.2, 8.1.4, etc)

maiores: com 2 dígitos (8.0, 8.1, 8.3, etc)

As menores sempre têm formato de armazenamento compatível e são chamadas compatíveis.

Exemplo: Tínhamos a versão 8.3.0. Foi lançada hoje a versão 8.3.1. Estas são compatíveis e permitem uma atualização física.

Vejamos o que diz o script upgrade.bat do PostgreSQL 8.3.1-1 no Windows:

"Você precisa ter o PostgreSQL 8.3.x instalado de um instalador MSI para usar este atualizador.

Nas "Release Notes":

<http://www.postgresql.org/docs/8.3/interactive/release-8-3-1.html>

"E.1.1. Migration to Version 8.3.1

A dump/restore is not required for those running 8.3.X. However, you might need to REINDEX indexes on textual columns after updating, if you are affected by the Windows locale issue described below."

Acostume-se a olhar pois eles sempre informam qual o procedimento a ser adotado para aquela versão.

Osvaldo na lista pgbr-geral.

Atualização entre versões menores de uma mesma versão maior:

No Windows, para atualizar da 8.1.2 para 8.1.3 podemos usar o upgrade.bat. Também alerta que se

o servidor ou qualquer cliente do PostgreSQL estiver rodando, será requerido um reboot.

Atualização entre versões maiores

Para atualizar entre versões 8.1 e 8.2 ou entre 8.2 e 8.3.

Por precaução pare o serviço antigo e mantenha o antigo em outro diretório.

Exportar todos os dados usando `pg_dumpall`

Reinstalar a nova versão

Importar os dados para a nova versão

Recuperando fisicamente o PostgreSQL

Em caso de falha do sistema e que não mais se tenha acesso ao servidor, podemos recuperar o sistema fisicamente, copiando todo o cluster.

- De um servidor com o mesmo SO, mesmo sistema de arquivos, mesma versão do PostgreSQL instalada, com os mesmos parâmetros de configuração.

- Substitua o cluster inteiro do novo servidor pelo cluster do seu backup. Verifique antes, é claro, se a estrutura de diretórios do seu backup está semelhante à do seu backup.

Obs.: Faça a cópia com o servidor parado!!!

Dica do Fábio Telles na lista pgbr-geral.

Migrações

É recomendada a utilização dos programas `pg_dump` e `pg_dumpall` da nova versão do PostgreSQL, para tirar vantagem das melhorias que podem ter sido introduzidas nestes programas.

O menor tempo de parada pode ser obtido instalando o novo servidor em um diretório diferente, e executando tanto o servidor novo quanto o antigo em paralelo, em portas diferentes. Depois pode ser executado algo como

```
pg_dumpall -p 5432 | psql -d template1 -p 6543
```

Exemplo através dos fontes (Linux):

```
pg_dumpall > backup
pg_ctl stop
mv /usr/local/pgsql /usr/local/pgsql.old
cd ~/postgresql-8.0.0
gmake install
initdb -D /usr/local/pgsql/data
postmaster -D /usr/local/pgsql/data
psql template1 < backup
```

Migrando 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 diretório data. Caso contrário faça os devidos 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 inicie 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 usando o novo servidor com:

/usr/local/pgsql/bin/psql -d postgres -f bancos.sql

Mais detalhes:

<http://pgdocptbr.sourceforge.net/pg80/install-upgrading.html>

<http://pgdocptbr.sourceforge.net/pg80/migration.html>

<http://pgdocptbr.sourceforge.net/pg80/migration.html>

<http://www.postgresql.org/docs/8.3/interactive/install-upgrading.html>

<http://www.postgresql.org/docs/8.3/interactive/migration.html>

Texto de discussão na lista pgbr-geral:

> Salve Pessoal,

Opa!

> Estou pensando em atualizar a versão do meu banco para a 8.3.

> Alguma recomendação especial? algum how to a indicar?

> Obrigado.

Kra, não há muitos mistérios. Siga:

1. Faça um backup (com o pg_dump)

2. pare o serviço da versão instalada e instale a nova (por enquanto não desinstale a antiga...)

3. restaure o backup na versão nova.
4. faça testes e tuning e todas as configurações que são necessárias
5. se tudo deu certo até aqui, desinstale a anterior e corra pro abraço!

Prezado Marco,

Recomendações:

* Revise todas as consultas que montam textos, seja unindo com constantes, seja concatenando campos texto:

*** Coloque um CAST explícito em todas elas. ***

* Revise todas as comparações entre campos e de campos com constantes.

Todas

as comparações devem ter rigorosamente o mesmo tipo de dado:

*** Coloque um CAST explícito em todas elas. ***

* Se tiver stored procedures, revise se os tipos de dados dos parâmetros batem

rigorosamente com o que estiver declarado na stored procedure:

*** Coloque um CAST explícito em todas elas. ***

* Se em algum lugar você vir uma mensagem de erro ao comparar alguma coisa:

*** Coloque um CAST explícito em todas elas. ***

* Cuidado com tabelas temporárias, o Postgres 8.3 não deixa você excluí-las. Caso tenha algum caso desses, você pode precisar usar tabelas reais e controlar sua limpeza manualmente.

* Prepare-se para sofrer muito e estourar seu cronograma, pois essa migração é cheia de surpresas. Teste cada consulta de cada script e cada programa com todas as alternativas imagináveis. Quanto mais cuidadoso for nisso, menos erros vão escapar.

* Quando seu chefe reclamar que está demorando para migrar, lembre-o de que a diferença de desempenho da versão 7 para a 8 é enorme e que, ao contrário da versão 7, a versão 8 não perde de lavada do SQL Server 2000 quando você começa a escrever consultas mais complexas em tabelas que não sejam pequenas. Não medi ainda para saber se empata ou ganha, mas pelo menos a diferença não é mais tão absurda.

Para finalizar: Boa sorte!

Mozart Hasse

Apenas para acrescentar, como se trata de uma versão '7' para a '8' é

importante que voce verifique as releases notes da 8.3, pois muitas coisas mudaram e isso pode gerar um impacto em determinados pontos de sua aplicação.

Conversões implícitas, tsearch2 entre outros, sofreram mudanças desde a versão 7.4...

Use sempre um ambiente de teste e realmente teste tudo o que puder, antes de colocar isso em produção.

Em muitas empresas existe a figura do Testador utilize-o para checar se sua aplicação não sofreu danos na lógica das regras de negócio.

--

[]s

Dickson S. Guedes

Tem mais uma:

Como as versões 8.X passaram a gravar usuários e grupos na mesma tabela. Elas não permitem mais a criação de um usuário com nome igual ao de um grupo.

Marco Aurélio

6) Monitorando as Atividades do Servidor do PostgreSQL

- 1 – Habilitando e monitorando o autovacuum
- 2 – Entendendo as estatísticas do PostgreSQL
- 3 – Rotina de reindexação
- 4 – Monitorando o espaço em disco e tamanho de objetos
- 5 – Monitorando o arquivo de registros (logs)

Estão disponíveis várias **ferramentas para monitorar** a atividade do banco de dados e analisar o desempenho. Não se deve desprezar os programas regulares de monitoramento do Unix, como **ps**, **top**, **iostat** e **vmstat**. Também, uma vez que tenha sido identificado um comando com baixo desempenho, podem ser necessárias outras investigações utilizando o comando EXPLAIN do PostgreSQL.

Das ferramentas citadas, apenas **iostat** não acompanha o Ubuntu.
Podemos instalar com:

```
sudo apt-get install sysstat
```

Para monitorar o PostgreSQL no Windows, como também no Linux, usar o PGAdmin, em Ferramentas – Status do Servidor.

1 – Habilitando e monitorando o autovacuum

A rotina de manutenção vacuum (limpeza do disco rígido) é algo tão importante, que passou a vir habilitado por default para rodar automaticamente na versão 8.3.

Na versão 8.2 para automatizar o vacuum habilitar os seguintes parâmetros no postgresq.conf:

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

Na versão 8.3:

```
autovacuum = on  
track_counts = on
```

Monitorando o autovacuum

No Linux, uma forma simples de monitorar o autovacuum é executando o comando:

```
ps ax | grep postgres (Linux XUbuntu 7.10 com PostgreSQL 8.3)
```

Com isso serão mostrados:

6881 ?	S	0:00 /usr/local/pgsql/bin/postmaster -D /var/lib/pgsql/data
6928 ?	Ss	0:00 postgres: writer process
6929 ?	Ss	0:00 postgres: wal writer process
6930 ?	Ss	0:00 postgres: autovacuum launcher process
6931 ?	Ss	0:00 postgres: stats collector process

Através dos logs (ao final) podemos ter acesso a mais informações sobre o autovacuum.

No Windows usar o PGAdmin.

2 – Entendendo as estatísticas do PostgreSQL

O coletor de estatísticas

O coletor de estatísticas do PostgreSQL é um subsistema de apoio a coleta e relatório de **informações sobre as atividades do servidor**. Atualmente, o coletor pode contar acessos a tabelas e índices em termos de blocos de disco e linhas individuais.

Configuração da coleta de estatísticas

Uma vez que a coleta de estatísticas adiciona alguma sobrecarga à execução das consultas, o sistema pode ser configurado para coletar informações, ou não. Isto é controlado por parâmetros de configuração, normalmente definidos no arquivo **postgresql.conf**

O parâmetro **track_counts** deve ser definido como true para que o coletor de estatísticas seja ativado. Esta é a definição padrão e recomendada, mas pode ser desativada se não houver interesse nas estatísticas e for desejado eliminar até a última gota de sobrecarga (Entretanto, provavelmente o ganho será pequeno e talvez não compense). Deve ser observado que esta opção não pode ser mudada enquanto o servidor está executando.

`track_activities (boolean)`

Habilita a coleta de informações das consultas em execução atualmente de cada seção, como também o tempo que cada consulta iniciou sua execução. Este parâmetro vem habilitado por default. Somente o superusuário e o usuário dono da sessão percebem essa informação e somente superusuários podem mudar essa configuração.

`track_counts (boolean)`

Habilita a coleta de informações das atividades dos bancos. Habilitado por default, pois o processo do autovacuum requer este parâmetro. Somente superusuário pode alterar.

`update_process_title (boolean)`

Habilita a atualização dos títulos dos processos cada vez que uma nova consulta SQL é recebida pelo servidor. O título do processo é tipicamente visualizado pelo comando **ps**, ou se em windows, pelo **Process Explorer**

[http://technet.microsoft.com/pt-br/sysinternals/bb896653\(en-us\).aspx](http://technet.microsoft.com/pt-br/sysinternals/bb896653(en-us).aspx)

Somente superusuário pode alterar esta configuração.

Monitorando Estatísticas

```
log_statement_stats (boolean)
log_parser_stats (boolean)
log_planner_stats (boolean)
log_executor_stats (boolean)
```

Para cada consulta, escreve as estatísticas de desempenho do respectivo módulo para o log (registro) do servidor. Este é um instrumento rústico. `log_statement_stats` reporta todas as declarações de estatística, enquanto os demais reportam as estatísticas por módulo.

`log_statement_stats` não pode ser habilitado para nenhum com a opção por módulo.

Todos estes parâmetros vêm desabilitados por default e somente o superusuário pode alterar seus status (através do comando SET).

Mostra o PID de cada consulta e a própria consulta em execução atualmente:

```
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;
```

Alerta: sempre ao trabalhar com estas funcionalidades e ainda outras é útil verificar a versão do PostgreSQL em uso para procurar a respectiva versão de documentação.

Utilização do comando EXPLAIN

EXPLAIN -- mostra o plano interno de execução de uma consulta.

Sintaxe:

```
EXPLAIN [ ANALYZE ] [ VERBOSE ] consulta
```

Este comando mostra o *plano de execução* gerado pelo planejador do PostgreSQL para a consulta fornecida. O plano de execução mostra como as tabelas referenciadas pelo comando serão varridas — por uma varredura seqüencial simples, varredura pelo índice, etc. — e, se forem referenciadas várias tabelas, quais algoritmos de junção serão utilizados para juntar as linhas requisitadas de cada uma das tabelas de entrada.

Usar o comando ANALYZE com o EXPLAIN para que assim o EXPLAIN tenha uma estimativa de custo mais realista.

Do que é mostrado, a parte mais importante é o custo estimado de execução do comando, que é a estimativa feita pelo planejador de quanto tempo vai demorar para executar o comando (medido em unidades de acesso às páginas do disco). Na verdade, são mostrados dois números: **o tempo inicial** para que a primeira linha possa ser retornada, e **o tempo total** para retornar todas as linhas. **Para a maior parte dos comandos o tempo total é o que importa**, mas em contextos como uma sub-seleção no EXISTS, o planejador escolhe o menor tempo inicial em vez do menor tempo total (porque o executor vai parar após ter obtido uma linha). Além disso, se for limitado o número de linhas retornadas usando a cláusula LIMIT, o planejador efetua uma interpolação apropriada entre

estes custos para estimar qual é realmente o plano de menor custo.

ANALYZE faz o comando ser realmente executado, e não apenas planejado. O tempo total decorrido gasto em cada nó do plano (em milissegundos) e o número total de linhas realmente retornadas são adicionados ao que é mostrado. Esta opção é útil para ver se as estimativas do planejador estão próximas da realidade.

Se for desejado utilizar EXPLAIN ANALYZE para um comando INSERT, UPDATE, DELETE ou EXECUTE sem deixar o comando afetar os dados, deve ser utilizado o seguinte procedimento (dentro de uma transação):

```
BEGIN;  
EXPLAIN ANALYZE INSERT INTO clientes (cod, nome) VALUES (1, 'João');  
ROLLBACK;
```

VERBOSE

Mostra a representação interna completa da árvore do plano, em vez de apenas um resumo. Geralmente esta opção é útil apenas para finalidades especiais de depuração.

Consulta - Qualquer comando SELECT, INSERT, UPDATE, DELETE, EXECUTE ou DECLARE, cujo plano de execução se deseja ver.

O PostgreSQL concebe um *plano de consulta* para cada consulta recebida. 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. A leitura do plano é uma arte que merece um estudo aprofundado. Aqui são fornecidas apenas algumas informações básicas.

Os números apresentados atualmente pelo EXPLAIN são:

- O **custo inicial** estimado (O tempo gasto para poder começar a varrer a saída como, por exemplo, o tempo para fazer a classificação em um único nó de classificação).
- O **custo total** estimado (Se todas as linhas fossem buscadas, o que pode não acontecer: uma consulta contendo a cláusula LIMIT para antes de gastar o custo total, por exemplo).
- **Número de linhas (rows)** de saída estimado para este nó do plano (Novamente, somente se for executado até o fim).
- **Largura média estimada (width)** (em bytes) das linhas de saída deste nó (fase) do plano.

Os custos são medidos em termos de **unidades de páginas de disco** buscadas (O esforço de CPU estimado é convertido em unidades de páginas de disco, utilizando fatores estipulados altamente arbitrários).

Linhas de saída é um pouco enganador, porque *não* é o número de linhas processadas/varridas pelo comando, geralmente é menos, refletindo a seletividade estimada de todas as condições da cláusula WHERE aplicadas a este nó. **Idealmente, a estimativa de linhas do nível superior estará próxima do número de linhas realmente retornadas, atualizadas ou excluídas pela consulta.**

Antes da realização de testes, devemos executar o comando:

VACUUM ANALYZE; -- Para atualizar as estatísticas internas (do banco atual)

Exemplos:

Testes no banco cep_brasil, após adicionar a chave primária na tabela cep_full.

Observe a saída destes comandos no psql ou através da query tool do PGAdmin.

```
EXPLAIN SELECT * FROM cep_full; (varredura sequencial, já que não usamos o índice e retornarão todos os registros).
```

```
EXPLAIN SELECT logradouro FROM cep_full WHERE cep = '60420440'; (varredura com o índice, pois retornamos apenas parte dos registros com a cláusula where).
```

Uma forma de ver outros planos é forçar o planejador a não considerar a estratégia que sairia vencedora, ativando e desativando sinalizadores de cada tipo de plano (Esta é uma forma deselegante, mas útil):

```
SET enable_nestloop = off;  
EXPLAIN SELECT * FROM cep_full t1, cep_full_index t2 WHERE t1.cep < '60000' AND  
t1.cep = t2.cep;
```

Examinando Índices

É muito ruim usar conjuntos de registros muito pequenos. Enquanto que selecionando 1000 de um total de 100.000 registros pode ser um candidato para um índice, selecionando 1 de 100 deve dificilmente ser, por causa de que 100 deve provavelmente caber em uma simples página de disco e não existe nenhum plano que pode ser sequencialmente coletado em uma única página de disco.

Também seja cuidadoso quando criando dados de teste, que não estarão disponíveis quando a aplicação estiver em produção. Valores que são muito similares, completamente aleatórios ou inseridos em alguma ordem devem atrapalhar as estatísticas diferente do que teriam uma distribuição real de dados.

Então uma cópia dos registros em produção deve ser mais adequada para testes.

Quando índices não são usados é útil fazer testes para forçar seu uso. Existem parâmetros para runtime que podem desativar varios tipos de planos. Por exemplo, desabilitando o sequential scan (enable_seqscan) com:

Via SQL:

```
set enable_seqscan to off;
```

No postgresql.conf:

```
enable_seqscan = off
```

e joins em loops aninhados (enable_nestloop), que são os planos mais básicos, irá forçar o sistema a usar planos diferentes. Caso o sistema continue selecionando um sequential scan ou nested-loop join (join com loops aninhados) então esta é provavelmente a mais fundamental razão de que o índice não é usado; por exemplo, a condição da consulta não corresponde ao índice.

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/runtime-config-query.html>

Controle do planejador com cláusulas JOIN explícitas

É possível ter algum controle sobre o planejador de comandos utilizando a sintaxe JOIN explícita. Para saber por que isto tem importância, primeiro é necessário ter algum conhecimento.

Em uma consulta de junção simples, como

```
SELECT * FROM a, b, c WHERE a.id = b.id AND b.ref = c.id;
```

o planejador está livre para fazer a junção das tabelas em qualquer ordem. Por exemplo, pode gerar um plano de comando que faz a junção de A com B utilizando a condição `a.id = b.id` do WHERE e, depois, fazer a junção de C com a tabela juntada utilizando a outra condição do WHERE. Poderia, também, fazer a junção de B com C e depois juntar A ao resultado. Ou, também, fazer a junção de A com C e depois juntar B, mas isto não seria eficiente, uma vez que deveria ser produzido o produto Cartesiano completo de A com C, porque não existe nenhuma condição aplicável na cláusula WHERE que permita a otimização desta junção (Todas as junções no executor do PostgreSQL acontecem entre duas tabelas de entrada sendo, portanto, necessário construir o resultado a partir de uma ou outra destas formas). **O ponto a ser destacado é que estas diferentes possibilidades de junção produzem resultados semanticamente equivalentes, mas podem ter custos de execução muito diferentes. Portanto, o planejador explora todas as possibilidades tentando encontrar o plano de consulta mais eficiente.**

Mais detalhes em: <http://pgdocptbr.sourceforge.net/pg80/explicit-joins.html> e <http://pgdocptbr.sourceforge.net/pg80/catalog-pg-statistic.html> <http://pgdocptbr.sourceforge.net/pg80/catalog-pg-class.html>

Estatísticas utilizadas pelo planejador

Conforme visto na seção anterior, o planejador de comandos precisa estimar o número de linhas buscadas pelo comando para poder fazer boas escolhas dos planos de comando.

Um dos componentes da estatística é o número total de entradas em cada tabela e índice, assim como o número de blocos de disco ocupados por cada tabela e índice. Esta informação é mantida nas colunas `reltuples` e `relpages` da tabela `pg_class`, podendo ser vista utilizando consultas semelhantes à mostrada abaixo:

```
SELECT relname, relkind, reltuples, relpages FROM pg_class WHERE relname LIKE 'cep%';
```

Em: <http://pgdocptbr.sourceforge.net/pg80/planner-stats.html>

Mais detalhes em: <http://pgdocptbr.sourceforge.net/pg80/monitoring-stats.html>, <http://www.postgresql.org/docs/8.3/interactive/monitoring-stats.html> e <http://www.postgresql.org/docs/8.3/interactive/runtime-config-statistics.html#GUC-TRACK-ACTIVITIES>.

Entendendo os Planos internos do PostgreSQL para a Execução das Consultas

Após o servidor do PostgreSQL receber uma consulta de uma aplicação cliente, o texto da consulta

é entregue ao parser (analisador). O analisador procura por toda a consulta e checa por erros de sintaxe. Caso a consulta esteja sintaticamente correta o parser deve transformar o texto da consulta em estrutura de árvore analisada. Um parse tree (gráfico em forma de árvore analisada) é uma estrutura de dados que representa o significado de sua consulta em um formulário formal e sem ambiguidades.

Tendo como base esta consulta:

```
SELECT cpf_cliente, valor FROM pedidos WHERE valor > 0 ORDER BY valor;
```

Após o parser ter completado a análise da consulta, o parse tree é entregue ao planejador/otimizador.

O planejador é responsável por percorrer o parse tree e procurar todos os possíveis planos de execução da consulta. O plano deve incluir um *sequential scan* através de toda a tabela e *index scan* se índices úteis forem definidos. Caso a consulta envolva duas ou mais tabelas o planejador pode sugerir um número de diferentes métodos para unir (join) as tabelas. Os planos de execução são desenvolvidos em termos de operadores de consulta. Cada operador de consulta transforma um ou mais conjuntos de entrada (input set) em um intermediário result set.

O operador seq scan, transforma um input set (tabela física) em um result set, filtrando qualquer registro que não satisfaça a constraint da consulta.

O operador sort produz um result set por ordenar o input set de acordo com uma ou mais chaves de ordenação.

Internamente complexas consultas são quebradas em consultas simples.

Quando todos os possíveis planos de execução forem gerados, o otimizador procurará pelo plano mais econômico. Para cada plano é atribuído um custo de execução. As estimativas de custo são medidos em unidades de disco I/O. **Um operador que lê um bloco simples de 8.192 bytes (8KB) de um disco tem o custo de uma unidade. O tempo de CPU também é medido em unidades de disco I/O mas usualmente como uma fração. Por exemplo, o total de tempo de CPU requerido para processar um simples registro é assumido como sendo 1% do simples disco I/O.** Podemos ajustar muitas das estimativas de custo. Cada operador de consulta tem uma diferente estimativa de custo.

Por exemplo, **o custo de um sequential scan de uma tabela completa é computado como o número de 8K blocos na tabela mais alguma sobrecarga de CPU.**

Após escolher o mais econômico (aparentemente) plano de execução o executor de consulta inicia no começo do plano e pede para o operador mais acima para produzir o result set. Cada operador transforma seu input set em um result set. O input set deve provir de outro operador mais abaixo na árvore/tree. Quando o operador mais acima completa sua transformação os results são retornados para a aplicação cliente.

Resumo das Operações Internas na Execução de uma Consulta

Cliente solicita uma consulta ->

Analizador recebe o texto da consulta ->

Analizador procura na consulta por erros de sintaxe ->

Analizador transforma texto da consulta (correta) em estrutura de árvore analisada (parse tree) ->

O planejador percorre o parse tree a procura todos os possíveis planos de execução ->

Internamente complexas consultas são quebradas em consultas simples ->

Para cada plano é atribuído um custo de execução ->

Após todos os possíveis planos de execução serem gerados, o otimizador procurará pelo mais econômico ->

Após escolher o plano mais econômico o executor de consulta vai ao começo do plano e pede para o operador mais acima para produzir o result set ->

Cada operador transforma seu input set em um result set ->

O input set deve provir de outro operador mais abaixo na árvore/tree ->

Quando o operador mais acima completa sua transformação os results são retornados para a aplicação cliente.

O explain é utilizado somente para analisar as consultas com SELECT, INSERT, DELETE, UPDATE e DECLARE... CURSOR.

Exemplo Simples de Uso do Explain:

```
cep_brasil=# explain analyze select * from cep_full_index;  
QUERY PLAN
```

```
-----  
Seq Scan on cep_full_index (cost=0.00..31671.01 rows=633401 width=290)  
    (actual time=0.032..10863.173 rows=633401 loops=1)
```

```
Total runtime: 12298.089 ms
```

```
(2 rows)
```

Vamos analisar o resultado.

Realmente o formato do plano de execução pode parecer um pouco misterioso no início.

Em cada passo no plano de execução o EXPLAIN mostra as seguintes informações:

- O tipo de operação requerida
- O custo estimado para a execução
- Caso se especifique EXPLAIN ANALYZE, a consulta será de fato executada e mostrado o custo atual.

Em se omitindo ANALYZE a consulta será apenas planejada e não executada e o custo atual não é mostrado.

- Tipo de operação - **Seq Scan**, foi o tipo de operação que o PostgreSQL elegeu para a consulta
- Custo estimado - (**cost=0.00..31671.01 rows=633401 width=290**). É composto de três partes: **cost=0.00..31671.01**, que nos informa quanto gastou a consulta. O gasto é medido em termos de leitura de disco. Dois números são mostrados:

0.00... – o primeiro nos informa o velocidade para que um registro do result set seja retornado pela

operação.

31671.01 – o segundo, que normalmente é mais importante, representa o gasto para a operação completa.

rows=633401 – Este nos informa quantos registros o PostgreSQL espera que retorne na operação. Não é exatamente o número real de registros.

width=290 – Esta última parte do custo estimado. É uma estimativa do tamanho, em bytes, do registro médio, no result set. Para comprovar experimente mudar o exemplo para a tabela clientes do dba_projeto e veja como não bate.

Lembrando que os operadores de consulta...

Exemplo sem o Analyze:

```
cep_brasil=# explain select * from cep_full_index;  
              QUERY PLAN
```

```
-----  
Seq Scan on cep_full_index (cost=0.00..31671.01 rows=633401 width=290)  
(1 row)
```

Outro Exemplo:

```
cep_brasil=# explain select * from cep_full;
               QUERY PLAN
```

```
-----
Seq Scan on cep_full (cost=0.00..31671.01 rows=633401 width=290)
(1 row)
```

Agora sem utilizar o Explain, ou seja, o atual (real):

```
\timing
select count(*) from cep_full_index;
count
-----
633401
Time: 10574,343 ms
```

Veja que o tempo é similar ao retornado no Exemplo Simples em: (actual time=0.032..10863.173.

Em consultas mais complexas o resultado é dividido em vários passos e os primeiros aparecem abaixo. Os últimos acima, como nestes exemplos:

Uma consulta contendo apenas 23 registros e ordenando por um campo que tem índice:

```
\c dba_projetodba_projeto=# explain analyze select * from clientes order by cpf;
               QUERY PLAN
```

```
-----
Sort (cost=1.75..1.81 rows=23 width=64) (actual time=0.571..0.616 rows=23 loop
s=1)
  Sort Key: cpf
  Sort Method: quicksort Memory: 20kB
-> Seq Scan on clientes (cost=0.00..1.23 rows=23 width=64) (actual time=0.0
11..0.137 rows=23 loops=1)
Total runtime: 0.865 ms
```

Agora uma tabela com uma grande quantidade de registros (mais de meio milhão), mas sem um índice no campo pelo qual está ordenando:

```
\c cep_brasil
cep_brasil=# explain analyze select * from cep_full order by cep;
               QUERY PLAN
```

```
-----
Sort (cost=352509.44..354092.94 rows=633401 width=290) (actual time=33675.802.
.53168.029 rows=633401 loops=1)
```

```
Sort Key: cep
Sort Method: external sort Disk: 189376kB
-> Seq Scan on cep_full (cost=0.00..31671.01 rows=633401 width=290) (actual
time=0.027..17997.280 rows=633401 loops=1)
Total runtime: 54728.569 ms
```

Em uma tabela com uma grande quantidade de registros (mais de meio milhão), mas agora com um índice no campo pelo qual está ordenando:

```
explain analyze select * from cep_full_index order by cep;
```

```
cep_brasil=# explain analyze select * from cep_full_index order by cep;
               QUERY PLAN
```

```
-----
Index Scan using cep_pk on cep_full_index
(cost=0.00..44605.36 rows=633401 width=290)
(actual time=90.159..18145.128 rows=633401 loops=1)
Total runtime: 19699.895 ms
```

Outro exemplo, agora usando a cláusula WHERE na PK:

```
cep_brasil=# explain select * from cep_full_index where cep='60420440';
               QUERY PLAN
```

```
-----
Index Scan using cep_pk on cep_full_index (cost=0.00..8.35 rows=1 width=290)
Index Cond: (cep = '60420440'::bpchar)
```

O Seq Scan ocorrerá primeiro e depois dele o Sort. ***Veja que quando temos um índice E uma grande quantidade de registros no campo pelo qual estamos ordenando, o planejador/otimizador do PostgreSQL decide usar o operador Index Scan.***

Após o Index Scan finalizar a construção do seu result set intermediário ele será então colocado no próximo passo do plano. O passo final deste plano será a operação Sort, que é requerida para satisfazer a cláusula order by. Este operador reordena o result set produzido pelo Seq Scan e retorna o final result set para a aplicação cliente. Observe que quando o Index Scan é eleito, a ordenação é feita pelo próprio índice e não pelo Sort.

Alguns dos Operadores de Consultas

Seq Scan – é o mais básico operador de consultas. O Seq Scan trabalha assim:

- iniciando no começo da tabela e
- varrendo até o final da tabela
- para cada linha da tabela, Seq Scan testa a constraint da consulta (no caso, where). Caso a constraint seja satisfeita, as colunas requeridas são retornadas para o result set.

Registros Mortos

Uma tabela pode ainda conter os registros excluídos, que não mais são visíveis. O Sec Scan não inclui registros mortos no result set, mas ele precisa ler esses registros o que pode ser custoso em uma tabela com pesada atualização.

Sec Scan não lerá a tabela completa antes de retornar o primeiro registro.

Já o operador Sort lerá todos os registros do input set antes de retornar o primeiro registro.

O planejador/otimizador escolherá o Sec Scan quando nenhum índice possa ser usado para satisfazer a consulta ou quando a consulta retorna todos os registros (neste caso o índice não será usado).

Index Scan – funciona percorrendo uma estrutura índice. Caso especifiquemos um valor inicial para uma coluna índice, como por exemplo, WHERE codigo >= 100, o Index Scan deve iniciar pelo valor apropriado. Caso especifiquemos um valor final, como WHERE codigo <= 200, então o Index Scan deverá terminar tão logo encontre um valor maior que 200.

O Index Scan tem duas vantagens sobre o operador Sec Scan: primeiro, o Sec Scan precisa ler todos os registros da tabela e somente pode remover registros do result set avaliando a cláusula WHERE para cada registro. Index Scan não precisa ler todos os registros se indicarmos um valor inicial ou final. Segundo: o Sec Scan retorna os registros na ordem da tabela e não em uma sorteada ordem. O Index Scan deve retornar os registros na ordem do índice.

Nem todos os tipos de índices podem ser utilizados no Index Scan, somente: *B-Tree*, *R-Tree* e *GiST* podem mas o tipo *Hash* não pode.

O planejador/otimizador usa um operador Index Scan quando ele pode reduzir o tamanho do result set percorrendo a faixa de valores do índice ou quando ele pode evitar uma ordenação porcauxa da implícita ordenação oferecida pelo índice.

Sort – impõe uma ordenação no result set. Podemos ajustar o PostgreSQL através do parâmetro de runtime *sort_mem*. Caso o result set caiba no *sort_mem* * 1024 bytes, o Sort será feito na memória, usando o algoritmo Qsort.

O Sort nunca reduz o tamanho do result set e também não remove registros ou campos.

Sort lerá todos os registros do input set antes de retornar o primeiro registro.

Obviamente um Sort pode ser utilizado pela cláusula ORDER BY.

Para os demais operadores veja este capítulo do Livro sugerido abaixo, que inclusive pode ser lido online: <http://www.iphelp.ru/faq/15/ch04lev1sec3.html>

Referências:

<http://www.postgresql.org/docs/8.3/interactive/using-explain.html>

<http://www.postgresql.org/docs/8.3/interactive/sql-explain.html>

<http://www.postgresql.org/docs/8.3/interactive/row-estimation-examples.html>

<http://pgdocptbr.sourceforge.net/pg80/sql-explain.html>

Livro: The comprehensive guide to building, programming, and administering PostgreSQL databases, Second Edition, Capítulo 4: Understanding How PostgreSQL Executes a Query, disponível como e-book em:

<http://www.iphelp.ru/faq/15/ch04lev1sec3.html>

Introdução ao otimizador de consultas do PostgreSQL

Walter Rodrigo de Sá Cruz Criado em Wed Aug 30 11:48:04 2006 em:

<http://artigos.waltercruz.com/postgresql/otimizador>

O caminho de uma consulta

Antes de entrarmos no assunto da otimização propriamente dito, precisamos rever qual é o caminho de uma consulta no banco de dados.

(A seção a seguir é adaptada da documentação do PostgreSQL)

O programa aplicativo transmite um comando para o servidor, e aguarda para receber de volta os resultados transmitidos pelo servidor.

1. O estágio de análise verifica o comando transmitido pelo programa aplicativo com relação à correção da sintaxe, e cria a árvore de comando.
2. O sistema de reescrita recebe a árvore de comando criada pelo estágio de análise, e procura por alguma regra (armazenada nos catálogos do sistema) a ser aplicada na árvore de comando. Realiza as transformações especificadas no corpo das regras. Uma das aplicações do sistema de reescrita é a criação de visões. Sempre que é executado um comando em uma visão (ou seja, uma tabela virtual), o sistema de reescrita reescreve o comando do usuário como um comando acessando as tabelas base especificadas na definição da visão, em vez da visão.
3. O planejador/otimizador recebe a árvore de comando (reescrita), e cria o plano de comando que será a entrada do executor. Isto é feito criando primeiro todos os caminhos possíveis que levam ao mesmo resultado. Por exemplo, se existe um índice em uma relação a ser varrido, existem dois caminhos para a varredura. Uma possibilidade é uma varredura sequencial simples, e a outra possibilidade é utilizar o índice. Em seguida é estimado o custo de execução de cada um dos caminhos, e escolhido o mais barato. O caminho mais barato é expandido em um plano completo para que o executor possa utilizá-lo.
4. O executor caminha recursivamente através da árvore do plano, e traz as linhas no caminho representado pelo plano. O executor faz uso do sistema de armazenamento ao varrer as relações, realiza classificações e junções, avalia as qualificações e, por fim, envia de volta as linhas derivadas.

O papel do otimizador

(Adaptado de http://en.wikipedia.org/wiki/Query_optimizer)

O otimizador de consultas é o componente do banco de dados que tenta determinar o modo mais eficiente de executar uma consulta.

O PostgreSQL usa algumas estatísticas mantidas em tabelas do sistema para direcionar o otimizador. Se essas estatísticas estiverem desatualizadas, você provavelmente não obterá o melhor plano para a consulta.

Para atualizar as estatísticas, devemos executar o comando ANALYZE.

É possível vermos o plano que o otimizador de consultas do PostgreSQL gerou, usando a cláusula EXPLAIN antes da consulta.

Adicionando a cláusula EXPLAIN ANALYZE antes da consulta, ela é de fato executada, de forma que possamos ter a medida do tempo.

Exemplos

Pagila

Vamos usar o banco de dados exemplo pagila, disponível em:

<http://pgfoundry.org/projects/dbsamples/>, que é um exemplo de banco de dados de uma locadora.

Vamos começar com a configuração padrão do otimizador do Postgres. Todas as variáveis no postgresql.conf estão comentadas, o que significa que elas usarão os valores padrão.

Como teste, criei dois bancos iguais, o pagila e o pagila2, com a diferença que não rodei o ANALYZE no pagila2.

A configuração do banco é a padrão do Debian.

A consulta a seguir retorna os filmes e os atores associados a ela:

```
SELECT f.title, a.first_name FROM film f
INNER JOIN film_actor ON film_actor.film_id=f.film_id
INNER JOIN actor a on film_actor.actor_id = a.actor_id
```

Aqui está o plano gerado para essa consulta no pagila2 (o que não recebeu ainda o ANALYZE):

```
Merge Join (cost=708.95..795.88 rows=5462 width=185) (actual
time=56.274..85.883 rows=5462 loops=1)
  Merge Cond: ("outer".film_id = "inner".film_id)
    -> Sort (cost=94.83..97.33 rows=1000 width=149) (actual time=8.489..9.926
rows=1000 loops=1)
      Sort Key: f.film_id
      -> Seq Scan on film f (cost=0.00..45.00 rows=1000 width=149) (actual
time=0.010..4.517 rows=1000 loops=1)
    -> Sort (cost=614.12..627.77 rows=5462 width=42) (actual time=47.775..56.152
rows=5462 loops=1)
      Sort Key: film_actor.film_id
      -> Merge Join (cost=0.00..275.06 rows=5462 width=42) (actual
time=0.027..33.019 rows=5462 loops=1)
        Merge Cond: ("outer".actor_id = "inner".actor_id)
        -> Index Scan using actor_pkey on actor a (cost=0.00..12.20
rows=200 width=44) (actual time=0.010..0.450 rows=200 loops=1)
        -> Index Scan using film_actor_pkey on film_actor
(cost=0.00..194.08 rows=5462 width=4) (actual time=0.006..13.315 rows=5462
```

```
loops=1)
Total runtime: 94.576 ms
```

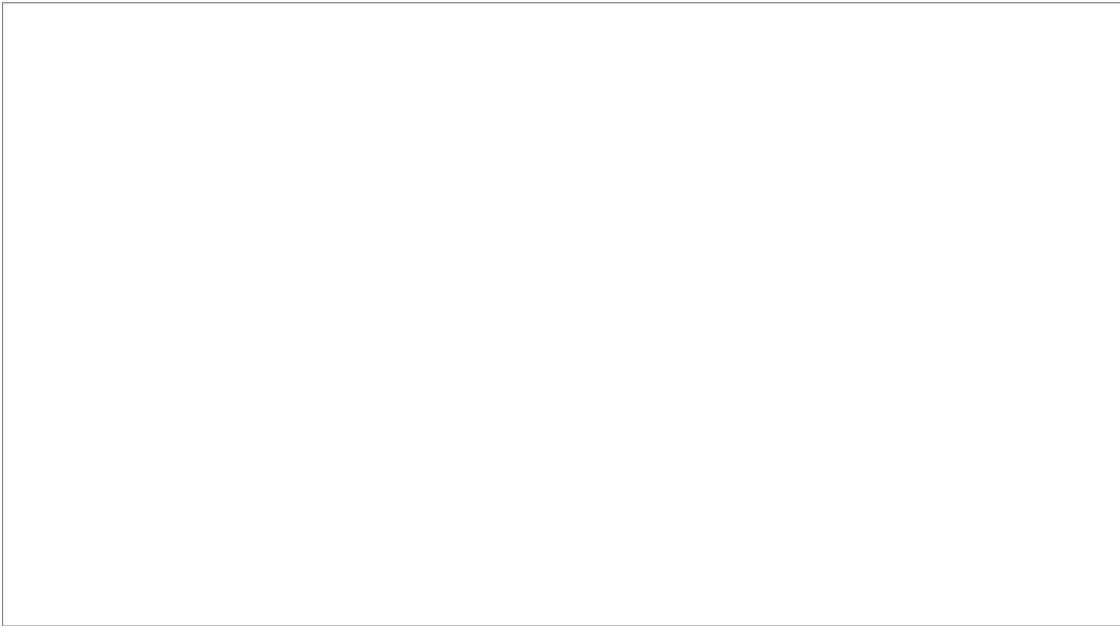


Sem entrar nos detalhes, vamos entender esse plano. Primeiro, temos que saber olhar de baixo pra cima.

1. Foi feito um index scan na tabela film_actor (a última linha)
2. Foi feito um index table scan na tabela actor
3. Essas duas tabelas foram unidas usando o algoritmo Merge Join
4. O resultado foi ordenado pela coluna film_actor.film_id
5. Foi feito um table scan na tabela film, e o resultado foi ordenado pela coluna film_id
6. O resultado do scan na tabela film foi juntado com o resultado anterior, usando o algoritmo Merge Join

E aqui está o plano gerado pelo pg planner (o banco no qual foi rodado o ANALYZE):

```
Merge Join  (cost=562.49..697.92 rows=5462 width=27) (actual time=47.691..78.064
rows=5462 loops=1)
  Merge Cond: ("outer".film_id = "inner".film_id)
    -> Index Scan using film_pkey on film f  (cost=0.00..51.00 rows=1000
width=22) (actual time=0.011..2.442 rows=1000 loops=1)
    -> Sort  (cost=562.49..576.15 rows=5462 width=11) (actual time=47.668..56.011
rows=5462 loops=1)
      Sort Key: film_actor.film_id
      -> Merge Join  (cost=0.00..223.43 rows=5462 width=11) (actual
time=0.025..32.304 rows=5462 loops=1)
        Merge Cond: ("outer".actor_id = "inner".actor_id)
          -> Index Scan using actor_pkey on actor a  (cost=0.00..6.20
rows=200 width=13) (actual time=0.007..0.459 rows=200 loops=1)
          -> Index Scan using film_actor_pkey on film_actor
(cost=0.00..148.46 rows=5462 width=4) (actual time=0.008..12.557 rows=5462
loops=1)
Total runtime: 86.610 ms
```



Vamos agora analisar esse plano:

1. Foi feito um scan na tabela film_actor pelo índice film_actor_pkey (a chave primária da tabela)
2. Agora, um scan na tabela actor pelo índice
3. As tabelas foram JOINadas pelo actor_id, e o resultado foi ordenado por film_actor.actor_id
4. A tabela film foi varrida pelo índice.
5. O resultado do JOIN anterior foi juntado com a tabela film.

Procurando clientes e filiais:

```
SELECT 'Filial ' || filial.store_id as filial, c.first_name || c.last_name as
client FROM customer c
INNER JOIN store filial on filial.store_id = c.store_id WHERE c.active = 1
```

Aqui, o explain do banco sem ANALYZE:

```
Nested Loop  (cost=1.02..17.65 rows=1 width=84) (actual time=0.042..15.491
rows=584 loops=1)
  Join Filter: ("inner".store_id = "outer".store_id)
    -> Seq Scan on customer c  (cost=0.00..16.49 rows=3 width=82) (actual
time=0.013..1.621 rows=584 loops=1)
      Filter: (active = 1)
    -> Materialize  (cost=1.02..1.04 rows=2 width=4) (actual time=0.002..0.006
rows=2 loops=584)
      -> Seq Scan on store filial  (cost=0.00..1.02 rows=2 width=4) (actual
time=0.004..0.011 rows=2 loops=1)
Total runtime: 16.480 ms
```

Repare que foi usado um materialize = um plano foi salvo num arquivo temporário.

E agora, o explain do banco com ANALYZE:

```
Nested Loop  (cost=4.05..46.50 rows=584 width=23) (actual time=0.175..6.097
rows=584 loops=1)
  -> Seq Scan on store filial  (cost=0.00..1.02 rows=2 width=4) (actual
time=0.008..0.015 rows=2 loops=1)
  -> Bitmap Heap Scan on customer c  (cost=4.05..16.80 rows=300 width=21)
```

```
(actual time=0.136..1.248 rows=292 loops=2)
  Recheck Cond: ("outer".store_id = c.store_id)
  Filter: (active = 1)
  -> Bitmap Index Scan on idx_fk_store_id (cost=0.00..4.05 rows=300
width=0) (actual time=0.122..0.122 rows=300 loops=2)
    Index Cond: ("outer".store_id = c.store_id)
Total runtime: 7.160 ms
```

Nesse caso, como as estatísticas já estavam atualizadas, o banco optou por uma otimização e usou o Bitmap Index Scan e o Bitmap Heap Scan.

O Bitmap Index Scan é usado em colunas que tem pouca variação (colunas de baixa cardinalidade). No caso, para cada cliente, eu tenho apenas duas opções de filias às quais ele pode estar associado (1 e 2). Então, o banco cria um mapa de bits para cada um desses valores. Esse mapa é carregado em memória, e é juntado com o resultado do table scan em store, que tem apenas dois valores possíveis.

No último select, ainda assim foi feito um Table Scan na tabela store, onde alguém poderia argumentar que seria melhor um index scan.

Random Page Cost

Existe um parâmetro de configuração, chamado **random_page_cost** que determina qual o peso que o PostgreSQL dá a leituras não sequencias no disco. Aumentar esse valor favorece o uso de table scans, abaixá-lo favorece o uso de índices. O valor padrão é 4.0.

Para um pequeno teste, vamos baixá-lo para 2.0 e executar a query no banco pagila.

```
Merge Join (cost=0.00..42.34 rows=584 width=23) (actual time=0.128..6.012
rows=584 loops=1)
  Merge Cond: ("outer".store_id = "inner".store_id)
  -> Index Scan using store_pkey on store filial (cost=0.00..3.02 rows=2
width=4) (actual time=0.092..0.098 rows=2 loops=1)
  -> Index Scan using idx_fk_store_id on customer c (cost=0.00..27.64 rows=584
width=21) (actual time=0.015..2.150 rows=584 loops=1)
    Filter: (active = 1)
Total runtime: 6.996 ms
```

Como o custo do índice estava mais baixo, o otimizador preferiu usar o índice do que gerar o índice bitmap em memória.

Agora, no pagila2, o plano gerado é semelhante ao plano do banco que está com as estatísticas atualizadas:

```
Nested Loop (cost=2.01..12.74 rows=1 width=84) (actual time=0.264..5.829
rows=584 loops=1)
  -> Seq Scan on store filial (cost=0.00..1.02 rows=2 width=4) (actual
time=0.012..0.019 rows=2 loops=1)
  -> Bitmap Heap Scan on customer c (cost=2.01..5.82 rows=3 width=82) (actual
time=0.182..1.236 rows=292 loops=2)
    Recheck Cond: ("outer".store_id = c.store_id)
    Filter: (active = 1)
  -> Bitmap Index Scan on idx_fk_store_id (cost=0.00..2.01 rows=3
width=0) (actual time=0.165..0.165 rows=300 loops=2)
    Index Cond: ("outer".store_id = c.store_id)
Total runtime: 6.861 ms
```

Executada no pagila2, a query gera o mesmo plano que havia sido gerado para `random_page_cost = 4.0`. E, com `random_page_cost = 2.0`, a primeira query do nosso teste passa a usar o índice, mesmo sem o `analyze`. Mas os custos não são os mesmos - isso porque o pagila2 ainda não teve as estatísticas atualizadas.

Configurações do otimizador

O arquivo de configuração do postgresql (`postgresql.conf`) contém várias opções que tratam da configuração do otimizador e dos vários métodos de consulta. São eles:

<code>enable_bitmapscan</code>
<code>enable_hashagg</code>
<code>enable_hashjoin</code>
<code>enable_indexscan</code>
<code>enable_mergejoin</code>
<code>enable_nestloop</code>
<code>enable_seqscan</code>
<code>enable_sort</code>
<code>enable_tidscan</code>

Todos eles podem ser configurados para `on` ou `off`, e por padrão todos vem habilitados. Alguns pontos a serem notados:

`enable_seqscan = off` não desabilita de fato o scan sequencial, já que essa pode ser a única forma de executar certas consultas. O que esse parâmetro faz na verdade é aumentar o custo de um table scan. O mesmo vale para `enable_nestloop` (algumas vezes não há forma de resolver uma consulta, exceto por esse tipo de loop) e `enable_sort`.

Esses valores podem ser alterados em tempo de execução para uma sessão em particular, usando o comando `SET`.

Esse é apenas um resumo. A descrição completa das opções relativas ao otimizador são encontradas em: <http://www.postgresql.org/docs/8.0/static/runtime-config.html>.

Listando atores e filmes

A primeira consulta, embora trouxesse a lista de filmes e atores, não trazia o resultado agrupado pelos filmes, com uma lista de todos os atores.

Vamos fazer essa consulta então.

Existem duas sintaxes que podem ser usadas. A primeira:

```
select
    film.title AS title,
    array_to_string(array_accum(actor.first_name || ' ' ||
```

```

actor.last_name),',') AS actors
from
    film
    inner join film_actor on film.film_id = film_actor.film_id
    inner join actor on film_actor.actor_id = actor.actor_id
GROUP BY film.title
ORDER BY film.title

```

E seu explain:

```

Sort  (cost=768.81..771.31 rows=1000 width=37) (actual time=151.996..153.650
rows=997 loops=1)
    Sort Key: film.title
    -> HashAggregate  (cost=698.98..718.98 rows=1000 width=37) (actual
time=132.534..142.902 rows=997 loops=1)
        -> Merge Join  (cost=536.24..671.67 rows=5462 width=37) (actual
time=53.739..97.676 rows=5462 loops=1)
            Merge Cond: ("outer".film_id = "inner".film_id)
            -> Index Scan using film_pkey on film  (cost=0.00..51.00
rows=1000 width=22) (actual time=0.012..3.780 rows=1000 loops=1)
            -> Sort  (cost=536.24..549.90 rows=5462 width=21) (actual
time=53.716..63.214 rows=5462 loops=1)
                Sort Key: film_actor.film_id
                -> Merge Join  (cost=0.00..197.18 rows=5462 width=21)
(actual time=0.026..36.636 rows=5462 loops=1)
                    Merge Cond: ("outer".actor_id = "inner".actor_id)
                    -> Index Scan using actor_pkey on actor
(cost=0.00..6.20 rows=200 width=23) (actual time=0.007..0.536 rows=200 loops=1)
                    -> Index Scan using film_actor_pkey on film_actor
(cost=0.00..122.21 rows=5462 width=4) (actual time=0.009..14.883 rows=5462
loops=1)
Total runtime: 159.766 ms

```

Uma outra forma possível de fazer essa consulta é essa:

```

SELECT f.title,
array_to_string(array(select first_name FROM film_actor fa
join actor a ON fa.actor_id = a.actor_id and fa.film_id = f.film_id
),', ' ) as film_actor
FROM film f

```

Será que essa forma, além de ser menos compacta, é mais rápida também?

```

Seq Scan on film f  (cost=0.00..16382.69 rows=1000 width=22) (actual
time=1.214..712.919 rows=1000 loops=1)
    SubPlan
        -> Merge Join  (cost=9.56..16.34 rows=5 width=9) (actual time=0.199..0.687
rows=5 loops=1000)
            Merge Cond: ("outer".actor_id = "inner".actor_id)
            -> Index Scan using actor_pkey on actor a  (cost=0.00..6.20 rows=200
width=13) (actual time=0.005..0.347 rows=165 loops=1000)
            -> Sort  (cost=9.56..9.57 rows=5 width=2) (actual time=0.056..0.069
rows=5 loops=1000)
                Sort Key: fa.actor_id
                -> Bitmap Heap Scan on film_actor fa  (cost=2.02..9.50 rows=5
width=2) (actual time=0.020..0.033 rows=5 loops=1000)
                    Recheck Cond: (film_id = $0)
                    -> Bitmap Index Scan on idx_fk_film_id  (cost=0.00..2.02
rows=5 width=0) (actual time=0.013..0.013 rows=5 loops=1000)

```


Index Cond: (film_id = \$0)

Total runtime: 714.655 ms

Parece que não.

Vemos que na consulta, o maior tempo gasto está no merge join (de 0.199 até 0.687). Vamos desabilitá-lo então. Isso é feito com o comando: `set enable_mergejoin = off`.

Vejamos o novo plano:

```
Seq Scan on film f (cost=0.00..24679.64 rows=1000 width=22) (actual
time=0.368..162.552 rows=1000 loops=1)
  SubPlan
    -> Nested Loop (cost=2.02..24.63 rows=5 width=9) (actual time=0.037..0.136
rows=5 loops=1000)
      -> Bitmap Heap Scan on film_actor fa (cost=2.02..9.50 rows=5
width=2) (actual time=0.020..0.039 rows=5 loops=1000)
        Recheck Cond: (film_id = $0)
        -> Bitmap Index Scan on idx_fk_film_id (cost=0.00..2.02 rows=5
width=0) (actual time=0.014..0.014 rows=5 loops=1000)
          Index Cond: (film_id = $0)
        -> Index Scan using actor_pkey on actor a (cost=0.00..3.01 rows=1
width=13) (actual time=0.007..0.009 rows=1 loops=5462)
          Index Cond: ("outer".actor_id = a.actor_id)
Total runtime: 164.212 ms
```

Fiz um pequeno teste com o beta do PostgreSQL 8.2, e para minha surpresa, o plano gerado para essa consulta é o mesmo que é gerado no 8.1 usando `enable_mergejoin = off` (com `random_page_cost = 2.0`).

Entendendo os ícones usados no pgadmin

Escrevi uma pequena descrição para os ícones usados no pgadmin, que está disponível em: <http://artigos.waltercruz.com/postgresql/explain/>

Outra coisa interessante pra se notar é que a grossura das setas é proporcional ao custo das operações.

3 – Rotina de reindexação

Em algumas situações vale a pena reconstruir índices periodicamente utilizando o comando REINDEX. Existe, também, o aplicativo `contrib/reindexdb` que pode reindexar todo o banco de dados. **Entretanto, a partir da versão 7.4 o PostgreSQL reduziu de forma substancial a necessidade desta atividade se comparado às versões anteriores.**

REINDEX -- reconstrói índices

Sintaxe

```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } nome [ FORCE ]
```

Exemplos

Reconstruir um único índice:

```
REINDEX INDEX meu_indice;
```

Reconstruir os índices da tabela `minha_tabela`:

```
REINDEX TABLE minha_tabela;
```

Reconstruir todos os índices de um determinado banco de dados, sem confiar que os índices do sistema estejam válidos:

```
$ export PGOPTIONS="-P"
$ psql bd_danificado
...
bd_danificado=> REINDEX DATABASE bd_danificado;
bd_danificado=> \q
```

Compatibilidade

Não existe o comando REINDEX no padrão SQL.

Notas

- [1] Oracle — O comando ALTER INDEX é utilizado para alterar ou reconstruir um índice existente. A cláusula de reconstrução (`rebuild_clause`) é utilizada para recriar um índice existente ou uma de suas partições ou sub-partições. Se o índice estiver marcado como UNUSABLE, uma reconstrução bem-sucedida irá marcá-lo como USABLE. Para um índice

baseado em função, esta cláusula também ativa o índice. Se a função sobre a qual o índice se baseia não existir, o comando de reconstrução irá falhar. [Oracle® Database SQL Reference 10g Release 1 \(10.1\) Part Number B10759-01](#) (N. do T.)

- [2] SQL Server — O comando ALTER INDEX modifica um índice de visão ou de tabela desativando, reconstruindo ou reorganizando o índice; ou definindo opções para o índice. *Reconstruindo índices* — A reconstrução do índice remove e recria o índice. Esta operação remove a fragmentação, recupera espaço em disco compactando as páginas com base na definição do fator de preenchimento especificado ou existente, e reordena as linhas do índice em páginas contíguas. Quando é especificada a opção ALL, todos os índices da tabela não removidos e reconstruídos em uma única transação. As restrições FOREIGN KEY não precisam ser previamente removidas. *Reorganizando índices* — A reorganização do índice utiliza recursos do sistema mínimos. Esta operação defragmenta o nível-folha (*leaf level*) dos índices agrupados (*clustered*) e não agrupados (*nonclustered*) de tabelas e visões, reordenando fisicamente as páginas do nível-folha para corresponderem a ordem lógica, esquerda para direita, dos nós folha. A reorganização também compacta as páginas de índice. A compactação é baseada no valor do fator de preenchimento existente. [SQL Server 2005 Books Online — ALTER INDEX \(Transact-SQL\)](#) (N. do T.)

Existe também o contrib reindexdb, que reindexa todo um banco de dados:

Exemplos:

Para reindexar todos os índices do banco de dados teste:

```
$ reindexdb -U postgres teste
```

Para reindexar o índice clientes_idx da tabela clientes do banco de dados dba_projeto:

```
$ reindexdb -U postgres --table clientes --index clientes_idx dba_projeto
```

Mais detalhes em: <http://pgdocptbr.sourceforge.net/pg82/sql-reindex.html>

4 – Monitorando o espaço em disco e tamanho de objetos

Cada tabela possui um arquivo em disco, onde a maior parte dos dados são armazenados. Se a tabela possuir alguma coluna com valor potencialmente longo, também existirá um arquivo TOAST associado à tabela, utilizado para armazenar os valores muito longos para caber confortavelmente na tabela principal. Haverá um índice para a tabela TOAST, caso esta esteja presente. Também podem haver índices associados à tabela base. **Cada tabela e índice é armazenado em um arquivo em disco separado — possivelmente mais de um arquivo, se o arquivo exceder um gigabyte.**

O espaço em disco pode ser monitorado a partir de três lugares: do psql utilizando as informações do VACUUM, do psql utilizando as ferramentas presentes em contrib/dbsize, e da linha de comando utilizando as ferramentas presentes em contrib/oid2name. Utilizando o psql em um banco de dados onde o comando VACUUM ou ANALYZE foi executado recentemente, podem ser efetuadas consultas para ver a utilização do espaço em disco de qualquer tabela:

```
\c cep_brasil
VACUUM ANALYZE;
```

```
SELECT relname, relfilenode, relpages
FROM pg_class
WHERE relname LIKE 'cep%'
ORDER BY relname;
```

```
Ou:
SELECT relfilenode, relpages FROM pg_class WHERE relname = 'cep_full_index';
```

```
Ou:
SELECT relfilenode as tabela_inode, relpages*8*1024/(1024*1024) AS "Espaço em
MB" FROM pg_class WHERE relname = 'cep_full_index';
```

```
Ou:
SELECT relfilenode as tabela_inode, relpages*8/1024 AS "Espaço em MB" FROM
pg_class WHERE relname = 'cep_full_index';
```

Cada página possui, normalmente, 8 kilobytes (Lembre-se, relpages somente é atualizado por VACUUM, ANALYZE e uns poucos comandos de DDL como CREATE INDEX). O valor de relfilenode possui interesse caso se deseje examinar diretamente o arquivo em disco da tabela.

Para ver o espaço utilizado pelas tabelas TOAST deve ser utilizada uma consulta como a mostrada abaixo:

```
-- Ver as tabelas TOAST da tabela pg_rewrite
SELECT relname, relpages
FROM pg_class,
     (SELECT reltoastrelid FROM pg_class
      WHERE relname = 'pg_rewrite') ss
WHERE oid = ss.reltoastrelid
      OR oid = (SELECT reltoastidxid FROM pg_class
               WHERE oid = ss.reltoastrelid)
ORDER BY relname;
```

Os tamanhos dos índices também podem ser facilmente exibidos:

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

Adaptação de: <http://pgdocptbr.sourceforge.net/pg80/diskusage.html>

Falha de disco cheio

A tarefa mais importante de monitoramento de disco do administrador de banco de dados é garantir que o disco não vai ficar cheio. Um disco de dados cheio não resulta em corrupção dos dados, mas pode impedir que ocorram atividades úteis. Se o disco que contém os arquivos do WAL ficar cheio, pode acontecer do servidor de banco de dados entrar na condição de pânico e encerrar a execução.

Se não for possível liberar espaço adicional no disco removendo outros arquivos, podem ser movidos alguns arquivos do banco de dados para outros sistemas de arquivos utilizando espaços de tabelas (tablespaces).

Dica: Alguns sistemas de arquivos têm desempenho degradado quando estão praticamente cheios, portanto não se deve esperar o disco ficar cheio para agir.

Se o sistema possuir cotas de disco por usuário, então naturalmente o banco de dados estará sujeito a cota atribuída para o usuário sob o qual o sistema de banco de dados executa. Exceder a cota ocasiona os mesmos malefícios de ficar totalmente sem espaço.

De: <http://pgdocptbr.sourceforge.net/pg80/disk-full.html>

5 – Monitorando o arquivo de registros (logs)

Estes arquivos guardam informações detalhadas de todas as operações do PostgreSQL, tanto as operações bem sucedidas quanto às más sucedidas, erros e alertas.

São arquivos com nomes no seguinte formato:

postgresql-2008-03-16_072122.log

postgresql – data – horada primeira ocorrência.log

E internamente exibe os registros em linhas assim:

2008-03-16 08:56:46 GMT ERROR: relation "wlw" does not exist

2008-03-16 08:56:46 GMT STATEMENT: select * from wlw;

Uma linha para o erro e a seguinte para a entrada que causou o erro.

Recomenda-se que o DBA mantenha sempre um olho nos logs.

Cada vez que o PostgreSQL é iniciado é criado um arquivo de log. Neste arquivo são registradas todas as ocorrências a partir daí e criado um novo arquivo na próxima vez que o servidor iniciar ou quando chegar a hora de rolar (configurado no postgresql.conf logrotate).

No postgresql.conf estão as regras:

#log_rotation_age = 1d

#log_rotation_size = 10MB

A cada dia ou quando chegar a 10MB um novo arquivo será criado.

Fique atento também para o volume que os logs ocupam em disco, pois podem comprometer o servidor.

No Windows localizam-se em:

C:\Program Files\PostgreSQL\8.3\data\pg_log

No Linux varia de acordo com a distribuição ou forma de instalação do PostgreSQL.

No postgresql.conf estão parâmetros que controlam e configuram os logs:

#log_directory = 'pg_log'

#log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'

E muitas outras configurações dos logs.

É uma boa prática, periodicamente, remover os arquivos mais antigos. Se será feito backup dos mesmos ou não, depende da importância dos mesmos.

O diretório pg_xlog contém os arquivos de logs das transações. Tamanhos fixos de 16MB.

O diretório pg_clog guarda o status das transações.

pg_subtrans - status das subtransações

pg_tblspc - links simbolicos de tablespace

Ativando os Logs no PostgreSQL 8.3 do Linux Ubuntu 7.10

Editar o script postgresql.conf

```
sudo gedit /etc/postgresql/8.3/main/postgresql.conf
```

E alterar a linha:

```
#logging_collector = off
```

Copiando-a para uma linha abaixo:

```
logging_collector = on
```

Restartar o servidor:

```
sudo /etc/init.d/postgresql-8.3 restart
```

Depois disso podemos acessar os logs como usuário postgres:

```
su – postgres
```

```
cd /var/lib/postgresql/8.3/main/pg_log
```

Sobre as configurações dos logs veja:

<http://pgdocptbr.sourceforge.net/pg80/runtime-config.html#RUNTIME-CONFIG-LOGGING-WHERE>

7) Catálogo do Sistema

7.1) Tabelas de Sistema

7.2) Informações sobre as tabelas de sistema

7.3) Exemplos práticos

Todo SGBD precisa ter seu catálogo de sistema (metadados, dicionário de dados), onde armazena pelo menos:

- Nomes das tabelas
- Nomes dos campos
- Tipos de dados de cada campo
- As constraints
- Informações sobre os índices
- Privilégios de acesso dos elementos

O Catálogo de sistema está disponível no PostgreSQL desde a versão 7.4.

7.1) Tabelas de Sistema do PostgreSQL 8.3

Catalog Name	Purpose
<u>pg_aggregate</u>	aggregate functions
<u>pg_am</u>	index access methods
<u>pg_amop</u>	access method operators
<u>pg_amproc</u>	access method support procedures
<u>pg_attrdef</u>	column default values
<u>pg_attribute</u>	table columns ("attributes")
<u>pg_authid</u>	authorization identifiers (roles)
<u>pg_auth_members</u>	authorization identifier membership relationships
<u>pg_autovacuum</u>	per-relation autovacuum configuration parameters
<u>pg_cast</u>	casts (data type conversions)
<u>pg_class</u>	tables, indexes, sequences, views ("relations")
<u>pg_constraint</u>	check constraints, unique constraints, primary key constraints, foreign key constraints
<u>pg_conversion</u>	encoding conversion information
<u>pg_database</u>	databases within this database cluster
<u>pg_depend</u>	dependencies between database objects
<u>pg_description</u>	descriptions or comments on database objects

Catalog Name	Purpose
pg_enum	enum label and value definitions
pg_index	additional index information
pg_inherits	table inheritance hierarchy
pg_language	languages for writing functions
pg_largeobject	large objects
pg_listener	asynchronous notification support
pg_namespace	schemas
pg_opclass	access method operator classes
pg_operator	operators
pg_opfamily	access method operator families
pg_pltemplate	template data for procedural languages
pg_proc	functions and procedures
pg_rewrite	query rewrite rules
pg_shdepend	dependencies on shared objects
pg_shdescription	comments on shared objects
pg_statistic	planner statistics
pg_tablespace	tablespaces within this database cluster
pg_trigger	triggers
pg_ts_config	text search configurations
pg_ts_config_map	text search configurations' token mappings
pg_ts_dict	text search dictionaries
pg_ts_parser	text search parsers
pg_ts_template	text search templates
pg_type	data types

Mais detalhes em:

<http://www.postgresql.org/docs/8.3/interactive/catalogs.html>

<http://pgdocptbr.sourceforge.net/pg80/catalogs.html>

http://www.cs.umu.se/kurser/TDBC86/H06/Slides/16_OH_catalog.pdf (trazendo informações sobre o catálogo do PostgreSQL, Oracle e ODBC).

7.2) Informações sobre as tabelas de sistema

Para visualizar a estrutura de cada uma destas relações, podemos usar o psql. Exemplo:

```
\d pg_database
```

Observar que existe um esquema ()

Para visualizar o catálogo completo, que está no esquema 'pg_catalog', usar:

```
\dS
```

Um total de 74 na versão 8.3 do PostgreSQL.

Para visualizar a estrutura de uma das relações do esquema pg_catalog:

```
\d pg_catalog.pg_table
```

serão listados os campos, seus tipos e abaixo uma definição de view.

Retornando todas as tabelas de sistema de um usuário:

```
select * from pg_catalog.pg_tables where tableowner='dba1';
```

```
select * from pg_catalog.pg_tables where tableowner='postgres';
```

7.3) Exemplos práticos e úteis

Uso do disco pela Tabela

```
\c dba_projeto
```

```
vacuum analyze clientes;
```

```
SELECT relfilenode, relpages FROM pg_class WHERE relname = 'clientes';
```

relfilenode é o arquivo, com nome usando números num diretório do 'base'.

relpages é o número de páginas ocupado pela relação (tabela). Lembrar que cada página ocupa 8KB. relpages somente é atualizado por VACUUM, ANALYZE e uns poucos comandos de DDL como CREATE INDEX). O valor de relfilenode possui interesse caso se deseje examinar diretamente o arquivo em disco da tabela.

Podemos então saber algo com mais detalhes assim:

```
\c dba_projeto
```

```
vacuum analyze clientes;
```

```
SELECT relfilenode AS arquivo, relpages*8 AS tamanho_em_kb FROM pg_class WHERE  
relname = 'clientes';
```

No diretório do tutorial existe o arquivo [syscat.sql](http://pgdocptbr.sourceforge.net/pg80/syscat.sql) contendo várias consultas interessantes aos catálogos do sistema:

<http://pgdocptbr.sourceforge.net/pg80/syscat.sql>

```
-- syscat.sql-
```

```
-- Exemplos de consultas aos catálogos do sistema
```

```
-- Portions Copyright (c) 1996-2003, PostgreSQL Global Development Group
```

```
-- Portions Copyright (c) 1994, Regents of the University of California
```

```
-- Primeiro definir o caminho de procura do esquema como pg_catalog,
```

```
-- para não ser necessário qualificar todo objeto do sistema.
```

```
--
```

```
SET SEARCH_PATH TO pg_catalog;
```

```
--
```

```
-- Listar o nome de todos os administradores de banco de dados e o seus bancos de dados.
```

```
--
```

```
SELECT username, datname  
FROM pg_user, pg_database  
WHERE usesysid = datdba  
ORDER BY username, datname;
```

```
--
-- Listar todas as classes definidas pelo usuário
--
SELECT n.nspname, c.relname
FROM pg_class c, pg_namespace n
WHERE c.relnamespace=n.oid
      AND c.relkind = 'r'           -- sem índices, visões, etc
      AND n.nspname not like 'pg\\_%' -- sem catálogos
      AND n.nspname != 'information_schema' -- sem information_schema
ORDER BY nspname, relname;

-- Listar todos os índices simples (ou seja, àqueles que são definidos
-- sobre uma referência de coluna simples)
--
SELECT n.nspname AS schema_name,
       bc.relname AS class_name,
       ic.relname AS index_name,
       a.attname
FROM pg_namespace n,
     pg_class bc,      -- classe base
     pg_class ic,      -- classe índice
     pg_index i,
     pg_attribute a     -- atributo na base
WHERE bc.relnamespace = n.oid
      AND i.indrelid = bc.oid
      AND i.indexrelid = ic.oid
      AND i.indkey[0] = a.attnum
      AND i.indnatts = 1
      AND a.attrelid = bc.oid
ORDER BY schema_name, class_name, index_name, attname;

--
-- Listar os atributos definidos pelo usuário e seus tipos
-- para todas as classes definidas pelo usuário
--
SELECT n.nspname, c.relname, a.attname, format_type(t.oid, null) AS typename
FROM pg_namespace n, pg_class c,
     pg_attribute a, pg_type t
WHERE n.oid = c.relnamespace
      AND c.relkind = 'r'           -- sem índices
      AND n.nspname not like 'pg\\_%' -- sem catálogos
      AND n.nspname != 'information_schema' -- sem information_schema
      AND a.attnum > 0               -- sem atributo de sistema
      AND not a.attisdropped         -- sem colunas removidas
      AND a.attrelid = c.oid
      AND a.atttypid = t.oid
ORDER BY nspname, relname, attname;
```

```
--  
-- Listar todos os tipos base definidos pelo usuário (sem incluir os tipos matriz)  
--
```

```
SELECT n.nspname, u.username, format_type(t.oid, null) AS typename  
FROM pg_type t, pg_user u, pg_namespace n  
WHERE u.usesysid = t.typowner  
      AND t.typnamespace = n.oid  
      AND t.typrelid = '0'::oid      -- sem tipos complexos  
      AND t.typelem = '0'::oid      -- sem matrizes  
      AND n.nspname not like 'pg\\_%' -- sem catálogos  
      AND n.nspname != 'information_schema' -- sem information_schema  
ORDER BY nspname, username, typename;
```

```
--  
-- Listar todas as funções de agregação e os tipos em que podem ser aplicadas  
--
```

```
SELECT n.nspname, p.proname, format_type(t.oid, null) AS typename  
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, typename;
```

```
-- Restaurar o caminho de procura
```

```
--  
RESET SEARCH_PATH;
```

Retornar o número de usuários conectados

```
select count(*) from pg_stat_activity;  
select count(*) from pg_stat_database;
```

pg_stat_database que apresenta para cada banco de dados o número de conexões.
Fica mais fácil de visualizar do que o pg_stat_activity quando se tem muitas conexões.

Mostrar uso dos índices:

```
select * from pg_statio_user_indexes;  
select * from pg_stat_user_indexes;
```

Mostra estatística de uso de todas as tabelas e manutenção:

```
select * from pg_stat_all_tables;
```

Mostra todas as tabelas e informações do atual esquema do atual banco:

```
select * from pg_stat_user_tables;
```

Veja só o retorno: relid | schemaname | relname | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | last_vacuum | last_autovacuum | last_analyze | last_autoanalyze

A função `pg_stat_get_backend_idset` provê uma conveniente maneira de gerar um registro/linha para cada processo ativo no servidor. Por exemplo, para **exibir os PIDs e as atuais consultas 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;
```

Visualizar os processos do postgresql num UNIX:

```
ps auxww | grep ^post
```

Mostrar todas as tabelas (inclusive de sistema) a quantidade de registros:

```
select relpages*8192 from pg_class;
```

Funções para Administração do Sistema

Definindo configurações

A função `current_setting` retorna o valor corrente da definição `nome_da_definição`. Corresponde ao comando SQL `SHOW`. Por exemplo:

```
SELECT current_setting('datestyle');
```

A função `set_config` define o parâmetro `nome_da_configuração` como `novo_valor`. Se o parâmetro `é_local` for `true`, então o novo valor se aplica somente à transação corrente. Se for desejado que o novo valor seja aplicado à sessão corrente, deve ser utilizado `false`. Esta função corresponde ao comando SQL `SET`. Por exemplo:

```
SELECT set_config('log_statement_stats', 'off', false);
```

Funções de Sinais para o Servidor

```
pg_cancel_backend(pid)  
pg_reload_conf()  
pg_rotate_logfile()
```

Se for bem-sucedida a função retorna 1, caso contrário retorna 0. O ID do processo (`pid`) de um servidor ativo pode ser encontrado a partir da coluna `procpid` da visão `pg_stat_activity`, ou listando os processos do postgres no servidor através do comando do Unix `ps`.

Funções que Retornam o Tamanho de Objetos

Name	Return Type	Description
<code>pg_column_size(any)</code>	int	Number of bytes used to store a particular value (possibly compressed)
<code>pg_tablespace_size(oid)</code>	bigint	Disk space used by the tablespace with the specified OID
<code>pg_tablespace_size(name)</code>	bigint	Disk space used by the tablespace with the specified name
<code>pg_database_size(oid)</code>	bigint	Disk space used by the database with the specified OID
<code>pg_database_size(name)</code>	bigint	Disk space used by the database with the specified name
<code>pg_relation_size(oid)</code>	bigint	Disk space used by the table or index with the specified OID
<code>pg_relation_size(text)</code>	bigint	Disk space used by the table or index with the specified name. The table name may be qualified with a schema name
<code>pg_total_relation_size(oid)</code>	bigint	Total disk space used by the table with the specified OID, including indexes and toasted data
<code>pg_total_relation_size(text)</code>	bigint	Total disk space used by the table with the specified name, including indexes and toasted data. The table name may be qualified with a schema name
<code>pg_size_pretty(bi gint)</code>	text	Converts a size in bytes into a human-readable format with size units

`pg_column_size` shows the space used to store any individual data value.

`pg_tablespace_size` and `pg_database_size` accept the OID or name of a tablespace or database, and return the total disk space used therein.

`pg_relation_size` accepts the OID or name of a table, index or toast table, and returns the size in bytes.

`pg_total_relation_size` accepts the OID or name of a table or toast table, and returns the size in bytes of the data and all associated indexes and toast tables.

`pg_size_pretty` can be used to format the result of one of the other functions in a human-readable way, using kB, MB, GB or TB as appropriate.

Funções de Acesso a Arquivos

Name	Return Type	Description
<code>pg_ls_dir(dirname text)</code>	setof text	List the contents of a directory
<code>pg_read_file(filename text, offset bigint, length bigint)</code>	text	Return the contents of a text file
<code>pg_stat_file(filename text)</code>	record	Return information about a file

`pg_ls_dir` returns all the names in the specified directory, except the special entries "." and "..".

`pg_read_file` returns part of a text file, starting at the given `offset`, returning at most `length` bytes (less if the end of file is reached first). If `offset` is negative, it is relative to the end of the file.

`pg_stat_file` returns a record containing the file size, last accessed time stamp, last modified time stamp, last file status change time stamp (Unix platforms only), file creation timestamp (Windows only), and a `boolean` indicating if it is a directory. Typical usages include:

```
SELECT * FROM pg_stat_file('filename');
SELECT (pg_stat_file('filename')).modification;
```

Mais detalhes em: <http://www.postgresql.org/docs/8.3/interactive/functions-admin.html>

Consultando a Estrutura de uma Tabela através do Catálogo

```
SELECT
    rel.nspname, rel.relname, attrs.attname, "Type", "Default",  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 = 'clientes' ORDER BY attrs.attnum;

```

Função em PLPGSQL:

```

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';

```

Usando:

```
SELECT * FROM Dados_Tabela('clientes');
```

Fonte: http://imasters.uol.com.br/artigo/2137/postgresql/consultando_a_estrutura_de_uma_tabela/

Retornando Informações sobre uma Tabela

```
SELECT pg_attribute.attnum AS index,  
attname AS field,  
typename AS type,  
atttypmod-4 as length,  
NOT attnotnull AS "null",  
adsrc AS def  
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)='clientes'  
UNION  
SELECT pg_attribute.attnum AS index,  
attname AS field,  
typename AS type,  
atttypmod-4 as length,  
NOT attnotnull AS "null",  
" AS def  
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)='clientes';
```

Use o comando \x no psql antes de executar a consulta caso queira exibir os registros em sequência, \x novamente para voltar ao normal.

Fonte:

http://imasters.uol.com.br/artigo/1283/postgresql/informacoes_atraves_do_catalogo_do_sistema/

Podemos listar tabelas, índices, sequências e vies usando os comandos do psql:

```
\d{t|i|s|v}
```

Abaixo veremos mais algumas funções que, ao contrário, usarão o SQL para receber essas informações.

Execute o trecho do script dba_projeto para criação do banco 'catalogo' da Aula 8.

Listando Tabelas do Banco Atual

```
SELECT relname
  FROM pg_class
 WHERE relname !~ '^(pg_|sql_)'
    AND relkind = 'r';

-- using INFORMATION_SCHEMA:

SELECT table_name
  FROM information_schema.tables
 WHERE table_type = 'BASE TABLE'
    AND table_schema NOT IN
      ('pg_catalog', 'information_schema');
```

Listando Views do Banco Atual

```
-- with postgresql 7.2:

SELECT viewname
  FROM pg_views
 WHERE viewname !~ '^pg_';

-- with postgresql 7.4 and later:

SELECT viewname
  FROM pg_views
 WHERE schemaname NOT IN
      ('pg_catalog', 'information_schema')
    AND viewname !~ '^pg_';

-- using INFORMATION_SCHEMA:

SELECT table_name
  FROM information_schema.tables
```

```
WHERE table_type = 'VIEW'
      AND table_schema NOT IN
          ('pg_catalog', 'information_schema')
      AND table_name !~ '^pg_';

-- or

SELECT table_name
      FROM information_schema.views
      WHERE table_schema NOT IN ('pg_catalog', 'information_schema')
          AND table_name !~ '^pg_';
```

Listando Todos os Usuários

```
SELECT username
      FROM pg_user;
```

Retornando os nomes dos Campos de uma Tabela

```
SELECT a.attname
      FROM pg_class c, pg_attribute a, pg_type t
      WHERE c.relname = 'test2'
          AND a.attnum > 0
          AND a.attrelid = c.oid
          AND a.atttypid = t.oid;
```

-- Usando INFORMATION_SCHEMA:

```
SELECT column_name
      FROM information_schema.columns
      WHERE table_name = 'test2';
```

Informações Detalhadas sobre os Campos de uma Tabela

```
SELECT a.attnum AS ordinal_position,
       a.attname AS column_name,
       t.typname AS data_type,
       a.attlen AS character_maximum_length,
       a.atttypmod AS modifier,
       a.attnotnull AS notnull,
       a.atthasdef AS hasdefault,
       col_description(a.attrelid, a.attnum) as field_comment
      FROM pg_class c,
           pg_attribute a,
           pg_type t
      WHERE c.relname = 'test2'
          AND a.attnum > 0
          AND a.attrelid = c.oid
          AND a.atttypid = t.oid
      ORDER BY a.attnum;
```

-- Usando INFORMATION_SCHEMA:

```
SELECT ordinal_position,
```

```

        column_name,
        data_type,
        column_default,
        is_nullable,
        character_maximum_length,
        numeric_precision
    FROM information_schema.columns
    WHERE table_name = 'test2'
    ORDER BY ordinal_position;

```

Retornando os Nomes de Índices de uma Tabela

```

SELECT relname
FROM pg_class
WHERE oid IN (
    SELECT indexrelid
    FROM pg_index, pg_class
    WHERE pg_class.relname='test2'
    AND pg_class.oid=pg_index.indrelid
    AND indisunique != 't'
    AND indisprimary != 't'
);

```

Retornando as Constraints de uma Tabela

```

SELECT conname
FROM pg_constraint, pg_class
WHERE pg_constraint.conrelid = pg_class.oid
AND relname = 'test2';

```

-- with INFORMATION_SCHEMA:

```

SELECT constraint_name, constraint_type
FROM information_schema.table_constraints
WHERE table_name = 'test2';

```

Recebendo Informações Detalhadas sobre as Constraints de uma Tabela

```

SELECT c.conname AS constraint_name,
       CASE c.contype
         WHEN 'c' THEN 'CHECK'
         WHEN 'f' THEN 'FOREIGN KEY'
         WHEN 'p' THEN 'PRIMARY KEY'
         WHEN 'u' THEN 'UNIQUE'
       END AS "constraint_type",
       CASE WHEN c.condeferrable = 'f' THEN 0 ELSE 1 END AS is_deferrable,
       CASE WHEN c.condeferred = 'f' THEN 0 ELSE 1 END AS is_deferred,
       t.relname AS table_name,
       array_to_string(c.conkey, ' ') AS constraint_key,
       CASE confupdtype
         WHEN 'a' THEN 'NO ACTION'
         WHEN 'r' THEN 'RESTRICT'
         WHEN 'c' THEN 'CASCADE'
         WHEN 'n' THEN 'SET NULL'
         WHEN 'd' THEN 'SET DEFAULT'

```

```

END AS on_update,
CASE confdeltype
  WHEN 'a' THEN 'NO ACTION'
  WHEN 'r' THEN 'RESTRICT'
  WHEN 'c' THEN 'CASCADE'
  WHEN 'n' THEN 'SET NULL'
  WHEN 'd' THEN 'SET DEFAULT'
END AS on_delete,
CASE confmatchtype
  WHEN 'u' THEN 'UNSPECIFIED'
  WHEN 'f' THEN 'FULL'
  WHEN 'p' THEN 'PARTIAL'
END AS match_type,
t2.relname AS references_table,
array_to_string(c.confkey, ' ') AS fk_constraint_key
FROM pg_constraint c
LEFT JOIN pg_class t ON c.conrelid = t.oid
LEFT JOIN pg_class t2 ON c.confrelid = t2.oid
WHERE t.relname = 'testconstraints2'
AND c.conname = 'testconstraints_id_fk';

-- with INFORMATION_SCHEMA:

SELECT tc.constraint_name,
       tc.constraint_type,
       tc.table_name,
       kcu.column_name,
       tc.is_deferrable,
       tc.initially_deferred,
       rc.match_option AS match_type,
       rc.update_rule AS on_update,
       rc.delete_rule AS on_delete,
       ccu.table_name AS references_table,
       ccu.column_name AS references_field
FROM information_schema.table_constraints tc
LEFT JOIN information_schema.key_column_usage kcu
  ON tc.constraint_catalog = kcu.constraint_catalog
  AND tc.constraint_schema = kcu.constraint_schema
  AND tc.constraint_name = kcu.constraint_name
LEFT JOIN information_schema.referential_constraints rc
  ON tc.constraint_catalog = rc.constraint_catalog
  AND tc.constraint_schema = rc.constraint_schema
  AND tc.constraint_name = rc.constraint_name
LEFT JOIN information_schema.constraint_column_usage ccu
  ON rc.unique_constraint_catalog = ccu.constraint_catalog
  AND rc.unique_constraint_schema = ccu.constraint_schema
  AND rc.unique_constraint_name = ccu.constraint_name
WHERE tc.table_name = 'testconstraints2'
AND tc.constraint_name = 'testconstraints_id_fk';

```

Listando as Sequências

```

SELECT relname
FROM pg_class
WHERE relkind = 'S'
AND relnamespace IN (
  SELECT oid
  FROM pg_namespace

```

```
WHERE nspname NOT LIKE 'pg_%'
      AND nspname != 'information_schema'
);
```

Listando Todas as Triggers

```
SELECT trg.tgname AS trigger_name
      FROM pg_trigger trg, pg_class tbl
      WHERE trg.tgrelid = tbl.oid
            AND tbl.relname !~ '^pg_';
-- or
SELECT tgname AS trigger_name
      FROM pg_trigger
      WHERE tgname !~ '^pg_';

-- with INFORMATION_SCHEMA:

SELECT DISTINCT trigger_name
      FROM information_schema.triggers
      WHERE trigger_schema NOT IN
            ('pg_catalog', 'information_schema');
```

Listando Somente as Triggers de uma Tabela

```
SELECT trg.tgname AS trigger_name
      FROM pg_trigger trg, pg_class tbl
      WHERE trg.tgrelid = tbl.oid
            AND tbl.relname = 'newtable';

-- with INFORMATION_SCHEMA:

SELECT DISTINCT trigger_name
      FROM information_schema.triggers
      WHERE event_object_table = 'newtable'
            AND trigger_schema NOT IN
            ('pg_catalog', 'information_schema');
```

Recebendo Informações Detalhadas sobre as Triggers

```
SELECT trg.tgname AS trigger_name,
      tbl.relname AS table_name,
      p.proname AS function_name,
      CASE trg.tgtype & cast(2 as int2)
        WHEN 0 THEN 'AFTER'
        ELSE 'BEFORE'
      END AS trigger_type,
      CASE trg.tgtype & cast(28 as int2)
        WHEN 16 THEN 'UPDATE'
        WHEN 8 THEN 'DELETE'
        WHEN 4 THEN 'INSERT'
        WHEN 20 THEN 'INSERT, UPDATE'
        WHEN 28 THEN 'INSERT, UPDATE, DELETE'
        WHEN 24 THEN 'UPDATE, DELETE'
```

```

        WHEN 12 THEN 'INSERT, DELETE'
    END AS trigger_event
FROM pg_trigger trg,
     pg_class tbl,
     pg_proc p
WHERE trg.tgrelid = tbl.oid
     AND trg.tgfoid = p.oid
     AND tbl.relname !~ '^pg_';

-- with INFORMATION_SCHEMA:

SELECT *
FROM information_schema.triggers
WHERE trigger_schema NOT IN
      ('pg_catalog', 'information_schema');
```

Listar as Funções

```

SELECT proname
FROM pg_proc pr,
     pg_type tp
WHERE tp.oid = pr.prorettype
     AND pr.proisagg = FALSE
     AND tp.typname <> 'trigger'
     AND pr.pronamespace IN (
        SELECT oid
        FROM pg_namespace
        WHERE nspname NOT LIKE 'pg_%'
            AND nspname != 'information_schema'
    );

-- with INFORMATION_SCHEMA:

SELECT routine_name
FROM information_schema.routines
WHERE specific_schema NOT IN
      ('pg_catalog', 'information_schema')
     AND type_udt_name != 'trigger';
```

Outra mais detalhada:

```

CREATE OR REPLACE FUNCTION public.function_args(
    IN funcname character varying,
    IN schema character varying,
    OUT pos integer,
    OUT direction character,
    OUT argname character varying,
    OUT datatype character varying)
RETURNS SETOF RECORD AS $$DECLARE
    rettype character varying;
    argtypes oidvector;
    allargtypes oid[];
    argmodes "char"[];
    argnames text[];
    mini integer;
```



```

maxi integer;
BEGIN
/* get object ID of function */
SELECT INTO rettype, argtypes, allargtypes, argmodes, argnames
CASE
WHEN pg_proc.proretset
THEN 'setof ' || pg_catalog.format_type(pg_proc.prorettype, NULL)
ELSE pg_catalog.format_type(pg_proc.prorettype, NULL) END,
pg_proc.proargtypes,
pg_proc.proallargtypes,
pg_proc.proargmodes,
pg_proc.proargnames
FROM pg_catalog.pg_proc
JOIN pg_catalog.pg_namespace
ON (pg_proc.pronamespace = pg_namespace.oid)
WHERE pg_proc.prorettype <> 'pg_catalog.cstring'::pg_catalog.regtype
AND (pg_proc.proargtypes[0] IS NULL
OR pg_proc.proargtypes[0] <> 'pg_catalog.cstring'::pg_catalog.regtype)
AND NOT pg_proc.proisagg
AND pg_proc.proname = funcname
AND pg_namespace.nspname = schema
AND pg_catalog.pg_function_is_visible(pg_proc.oid);

/* bail out if not found */
IF NOT FOUND THEN
RETURN;
END IF;

/* return a row for the return value */
pos = 0;
direction = 'o'::char;
argname = 'RETURN VALUE';
datatype = rettype;
RETURN NEXT;

/* unfortunately allargtypes is NULL if there are no OUT parameters */
IF allargtypes IS NULL THEN
mini = array_lower(argtypes, 1); maxi = array_upper(argtypes, 1);
ELSE
mini = array_lower(allargtypes, 1); maxi = array_upper(allargtypes, 1);
END IF;
IF maxi < mini THEN RETURN; END IF;

/* loop all the arguments */
FOR i IN mini .. maxi LOOP
pos = i - mini + 1;
IF argnames IS NULL THEN
argname = NULL;
ELSE
argname = argnames[i];
END IF;
IF allargtypes IS NULL THEN
direction = 'i'::char;
datatype = pg_catalog.format_type(argtypes[i], NULL);
ELSE
direction = argmodes[i];
datatype = pg_catalog.format_type(allargtypes[i], NULL);
END IF;
RETURN NEXT;

```

```
END LOOP;

RETURN;
END;$$ LANGUAGE plpgsql STABLE STRICT SECURITY INVOKER;
COMMENT ON FUNCTION public.function_args(character varying, character
varying)
IS $$For a function name and schema, this procedure selects for each
argument the following data:
- position in the argument list (0 for the return value)
- direction 'i', 'o', or 'b'
- name (NULL if not defined)
- data type$$;
```

Do artigo de Lorenzo Alberton. Extracting metadata from PostgreSQL using INFORMATION_SCHEMA :

http://www.alberton.info/postgresql_meta_info.html

Que também traz artigos sobre os catálogos do Firebird, Oracle e SQL Server.

Saber quantidade de registros no banco inteiro:

```
SELECT sum(C.reltuples)::int FROM pg_class C WHERE c.relkind = 'r':"char";
```

Fonte: Blog da Kenia Milene

<http://keniamilene.wordpress.com/2007/09/18/contar-registros-em-banco-postgresql/>

Listando os bancos e sua codificação

```
SELECT datname, pg_encoding_to_char(encoding) FROM pg_database;
```

Exibindo codificações e versões

```
SELECT name, setting FROM pg_settings
WHERE name ~ 'encoding|^lc_|version';
```

Mostrar data da última execução do Vacuum e do Autovacuum

```
select relname, last_vacuum, last_autovacuum from
pg_stat_all_tables where schemaname like 'public';
```

Vide o capítulo "Table Statistics" do Livro PostgreSQL The Comprehensive Guide, da Sam's em:

<http://www.iphelp.ru/faq/15/ch04lev1sec4.html>

Exibir Todas as Tabelas e seus Registros

```
select
    n.nspname as esquema,
    c.relname as tabela, c.reltuples::int as registros
from pg_class c
    left join pg_namespace n on n.oid = c.relnamespace
    left join pg_tablespace t on t.oid = c.reltablespace
where c.relkind = 'r'::char
    and nspname not in('information_schema','pg_catalog', 'pg_toast')
order by n.nspname,registros;
```

Outra:

```
select relname as tabelas, reltuples
from pg_class
where relkind = 'r'::char and relname not like 'pg_%' and relname not like 'sql_%' order by reltuples;
```

Outra solução:

VACUUM ANALYZE (em todo o banco)

```
SELECT sum(reltuples) as qtd_linhas
FROM pg_class
WHERE relkind = 'r';
```

criar uma view com a instrução SQL

Rodrigo Hjort - <http://icewall.org/~hjort> – na lista pgbr-geral

Boa fonte de leitura sobre o Catálogo do Sistema do PostgreSQL:

System Tables no capítulo 6 do livro: PostgreSQL Developer's Handbook de Ewald Geschwinde, Hans-Jürgen Schönig.

System Administration Functions

[Table 9-51](#) shows the functions available to query and alter run-time configuration parameters.

Table 9-51. Configuration Settings Functions

Name	Return Type	Description
<code>current_setting(setting_name)</code>	text	current value of setting
<code>set_config(setting_name, new_value, is_local)</code>	text	set parameter and return new value

The function `current_setting` yields the current value of the setting `setting_name`. It corresponds to the SQL command `SHOW`. An example:

```
SELECT current_setting('datestyle');

current_setting
-----
ISO, MDY
(1 row)
```

`set_config` sets the parameter `setting_name` to `new_value`. If `is_local` is `true`, the new value will only apply to the current transaction. If you want the new value to apply for the current session, use `false` instead. The function corresponds to the SQL command `SET`. An example:

```
SELECT set_config('log_statement_stats', 'off', false);

set_config
-----
off
(1 row)
```

The functions shown in [Table 9-52](#) send control signals to other server processes. Use of these functions is restricted to superusers.

Table 9-52. Server Signalling Functions

Name	Return Type	Description
<code>pg_cancel_backend(pid int)</code>	boolean	Cancel a backend's current query
<code>pg_reload_conf()</code>	boolean	Cause server processes to reload their configuration files
<code>pg_rotate_logfile()</code>	boolean	Rotate server's log file

Each of these functions returns `true` if successful and `false` otherwise.

`pg_cancel_backend` sends a query cancel (SIGINT) signal to a backend process identified by process ID. The process ID of an active backend can be found from the `procpid` column in the

`pg_stat_activity` view, or by listing the postgres processes on the server with `ps`.

`pg_reload_conf` sends a SIGHUP signal to the server, causing the configuration files to be reloaded by all server processes.

`pg_rotate_logfile` signals the log-file manager to switch to a new output file immediately. This works only when the built-in log collector is running, since otherwise there is no log-file manager subprocess.

The functions shown in [Table 9-53](#) assist in making on-line backups. Use of the first three functions is restricted to superusers.

Table 9-53. Backup Control Functions

Name	Return Type	Description
<code>pg_start_backup(label text)</code>	text	Set up for performing on-line backup
<code>pg_stop_backup()</code>	text	Finish performing on-line backup
<code>pg_switch_xlog()</code>	text	Force switch to a new transaction log file
<code>pg_current_xlog_location()</code>	text	Get current transaction log write location
<code>pg_current_xlog_insert_location()</code>	text	Get current transaction log insert location
<code>pg_xlogfile_name_offset(location text)</code>	text, integer	Convert transaction log location string to file name and decimal byte offset within file
<code>pg_xlogfile_name(location text)</code>	text	Convert transaction log location string to file name

`pg_start_backup` accepts a single parameter which is an arbitrary user-defined label for the backup. (Typically this would be the name under which the backup dump file will be stored.) The function writes a backup label file into the database cluster's data directory, and then returns the backup's starting transaction log location as text. The user need not pay any attention to this result value, but it is provided in case it is of use.

```
postgres=# select pg_start_backup('label_goes_here');
pg_start_backup
-----
0/D4445B8
(1 row)
```

`pg_stop_backup` removes the label file created by `pg_start_backup`, and instead creates a backup history file in the transaction log archive area. The history file includes the label given to `pg_start_backup`, the starting and ending transaction log locations for the backup, and the starting and ending times of the backup. The return value is the backup's ending transaction log location (which again might be of little interest). After noting the ending location, the current transaction log insertion point is automatically advanced to the next transaction log file, so that the ending transaction log file can be archived immediately to complete the backup.

`pg_switch_xlog` moves to the next transaction log file, allowing the current file to be archived (assuming you are using continuous archiving). The result is the ending transaction log location

within the just-completed transaction log file. If there has been no transaction log activity since the last transaction log switch, `pg_switch_xlog` does nothing and returns the end location of the previous transaction log file.

`pg_current_xlog_location` displays the current transaction log write location in the same format used by the above functions. Similarly, `pg_current_xlog_insert_location` displays the current transaction log insertion point. The insertion point is the "logical" end of the transaction log at any instant, while the write location is the end of what has actually been written out from the server's internal buffers. The write location is the end of what can be examined from outside the server, and is usually what you want if you are interested in archiving partially-complete transaction log files. The insertion point is made available primarily for server debugging purposes. These are both read-only operations and do not require superuser permissions.

You can use `pg_xlogfile_name_offset` to extract the corresponding transaction log file name and byte offset from the results of any of the above functions. For example:

```
postgres=# select * from pg_xlogfile_name_offset(pg_stop_backup());
      file_name      | file_offset
-----+-----
 000000010000000000000000D |      4039624
(1 row)
```

Similarly, `pg_xlogfile_name` extracts just the transaction log file name. When the given transaction log location is exactly at a transaction log file boundary, both these functions return the name of the preceding transaction log file. This is usually the desired behavior for managing transaction log archiving behavior, since the preceding file is the last one that currently needs to be archived.

For details about proper usage of these functions, see [Section 24.3](#).

The functions shown in [Table 9-54](#) calculate the actual disk space usage of database objects.

Table 9-54. Database Object Size Functions

Name	Return Type	Description
<code>pg_column_size(any)</code>	int	Number of bytes used to store a particular value (possibly compressed)
<code>pg_database_size(oid)</code>	bigint	Disk space used by the database with the specified OID
<code>pg_database_size(name)</code>	bigint	Disk space used by the database with the specified name
<code>pg_relation_size(oid)</code>	bigint	Disk space used by the table or index with the specified OID
<code>pg_relation_size(text)</code>	bigint	Disk space used by the table or index with the specified name. The table name can be qualified with a schema name
<code>pg_size_pretty(bigint)</code>	text	Converts a size in bytes into a human-readable format with size units
<code>pg_tablespace_size(oid)</code>	bigint	Disk space used by the tablespace with the specified OID

Name	Return Type	Description
<code>e(oid)</code>		
<code>pg_tablespace_size(name)</code>	<code>bigint</code>	Disk space used by the tablespace with the specified name
<code>pg_total_relation_size(oid)</code>	<code>bigint</code>	Total disk space used by the table with the specified OID, including indexes and toasted data
<code>pg_total_relation_size(text)</code>	<code>bigint</code>	Total disk space used by the table with the specified name, including indexes and toasted data. The table name can be qualified with a schema name

`pg_column_size` shows the space used to store any individual data value.

`pg_database_size` and `pg_tablespace_size` accept the OID or name of a database or tablespace, and return the total disk space used therein.

`pg_relation_size` accepts the OID or name of a table, index or toast table, and returns the size in bytes.

`pg_size_pretty` can be used to format the result of one of the other functions in a human-readable way, using kB, MB, GB or TB as appropriate.

`pg_total_relation_size` accepts the OID or name of a table or toast table, and returns the size in bytes of the data and all associated indexes and toast tables.

The functions shown in [Table 9-55](#) provide native file access to files on the machine hosting the server. Only files within the database cluster directory and the `log_directory` can be accessed. Use a relative path for files within the cluster directory, and a path matching the `log_directory` configuration setting for log files. Use of these functions is restricted to superusers.

Table 9-55. Generic File Access Functions

Name	Return Type	Description
<code>pg_ls_dir(dirname text)</code>	<code>setof text</code>	List the contents of a directory
<code>pg_read_file(filename text, offset bigint, length bigint)</code>	<code>text</code>	Return the contents of a text file
<code>pg_stat_file(filename text)</code>	<code>record</code>	Return information about a file

`pg_ls_dir` returns all the names in the specified directory, except the special entries `"."` and `".."`.

`pg_read_file` returns part of a text file, starting at the given `offset`, returning at most `length` bytes (less if the end of file is reached first). If `offset` is negative, it is relative to the end of the file.

`pg_stat_file` returns a record containing the file size, last accessed time stamp, last modified time stamp, last file status change time stamp (Unix platforms only), file creation time stamp (Windows only), and a `boolean` indicating if it is a directory. Typical usages include:

```
SELECT * FROM pg_stat_file('filename');
SELECT (pg_stat_file('filename')).modification;
```

The functions shown in [Table 9-56](#) manage advisory locks. For details about proper usage of these functions, see [Section 13.3.4](#).

Table 9-56. Advisory Lock Functions

Name	Return Type	Description
<code>pg_advisory_lock(key bigint)</code>	void	Obtain exclusive advisory lock
<code>pg_advisory_lock(key1 int, key2 int)</code>	void	Obtain exclusive advisory lock
<code>pg_advisory_lock_shared(key bigint)</code>	void	Obtain shared advisory lock
<code>pg_advisory_lock_shared(key1 int, key2 int)</code>	void	Obtain shared advisory lock
<code>pg_try_advisory_lock(key bigint)</code>	boolean	Obtain exclusive advisory lock if available
<code>pg_try_advisory_lock(key1 int, key2 int)</code>	boolean	Obtain exclusive advisory lock if available
<code>pg_try_advisory_lock_shared(key bigint)</code>	boolean	Obtain shared advisory lock if available
<code>pg_try_advisory_lock_shared(key1 int, key2 int)</code>	boolean	Obtain shared advisory lock if available
<code>pg_advisory_unlock(key bigint)</code>	boolean	Release an exclusive advisory lock
<code>pg_advisory_unlock(key1 int, key2 int)</code>	boolean	Release an exclusive advisory lock
<code>pg_advisory_unlock_shared(key bigint)</code>	boolean	Release a shared advisory lock
<code>pg_advisory_unlock_shared(key1 int, key2 int)</code>	boolean	Release a shared advisory lock
<code>pg_advisory_unlock_all()</code>	void	Release all advisory locks held by the current session

`pg_advisory_lock` locks an application-defined resource, which can be identified either by a single 64-bit key value or two 32-bit key values (note that these two key spaces do not overlap). If another session already holds a lock on the same resource, the function will wait until the resource becomes available. The lock is exclusive. Multiple lock requests stack, so that if the same resource is locked three times it must be also unlocked three times to be released for other sessions' use.

`pg_advisory_lock_shared` works the same as `pg_advisory_lock`, except the lock can be shared with other sessions requesting shared locks. Only would-be exclusive lockers are locked out.

`pg_try_advisory_lock` is similar to `pg_advisory_lock`, except the function will not

wait for the lock to become available. It will either obtain the lock immediately and return `true`, or return `false` if the lock cannot be acquired now.

`pg_try_advisory_lock_shared` works the same as `pg_try_advisory_lock`, except it attempts to acquire shared rather than exclusive lock.

`pg_advisory_unlock` will release a previously-acquired exclusive advisory lock. It will return `true` if the lock is successfully released. If the lock was in fact not held, it will return `false`, and in addition, an SQL warning will be raised by the server.

`pg_advisory_unlock_shared` works the same as `pg_advisory_unlock`, except to release a shared advisory lock.

`pg_advisory_unlock_all` will release all advisory locks held by the current session. (This function is implicitly invoked at session end, even if the client disconnects ungracefully.)

Fonte: <http://www.postgresql.org/docs/8.3/static/functions-admin.html>

Finding the size of your biggest tables

Note this this only shows you the totals for the database you're currently connected to:

```
SELECT nspname || '.' || relname AS "relation",
       pg_size_pretty(pg_relation_size(nspname || '.' || relname)) AS "size"
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
WHERE nspname NOT IN ('pg_catalog', 'information_schema')
      AND nspname !~ '^pg_toast'
      AND pg_relation_size(nspname || '.' || relname) > 0
ORDER BY pg_relation_size(nspname || '.' || relname) DESC
LIMIT 20;
```

Example output (from a database created with `pgbench`, `scale=25`):

relation	size
public.accounts	326 MB
public.accounts_pkey	44 MB
public.history	592 kB
public.tellers_pkey	16 kB
public.branches_pkey	16 kB
public.tellers	16 kB
public.branches	8192 bytes

Retrieved from "http://wiki.postgresql.org/wiki/Disk_Usage"

8) Trabalhando com Esquemas

8.1) O que são esquemas e sua importância

8.2) Gerenciando esquemas

8.1) O que são esquemas e sua importância

Um agrupamento de bancos de dados do PostgreSQL contém um ou mais bancos de dados com nome. Os usuários e os grupos de usuários são compartilhados por todo o agrupamento, mas nenhum outro dado é compartilhado entre os bancos de dados. Todas as conexões dos clientes com o servidor podem acessar somente os dados de um único banco de dados, àquele que foi especificado no pedido de conexão.

Nota: Os usuários de um agrupamento de bancos de dados não possuem, necessariamente, o privilégio de acessar todos os bancos de dados do agrupamento. O compartilhamento de nomes de usuários significa que não pode haver, em dois bancos de dados do mesmo agrupamento, mais de um usuário com o mesmo nome como, por exemplo, joel; mas o sistema pode ser configurado para permitir que o usuário joel acesse apenas determinados bancos de dados.

Um banco de dados contém um ou mais *esquemas* com nome, os quais por sua vez contêm tabelas. Os esquemas também contêm outros tipos de objetos com nome, incluindo tipos de dado, funções e operadores. O mesmo nome de objeto pode ser utilizado em esquemas diferentes sem conflito; por exemplo, tanto o esquema_1 quanto o meu_esquema podem conter uma tabela chamada minha_tabela. Diferentemente dos bancos de dados, os esquemas não são separados rigidamente: um usuário pode acessar objetos de vários esquemas no banco de dados em que está conectado, caso possua os privilégios necessários para fazê-lo.

Existem diversas razões pelas quais pode-se desejar utilizar esquemas:

- Para permitir vários usuários utilizarem o mesmo banco de dados sem que um interfira com o outro.
- Para organizar objetos do banco de dados em grupos lógicos tornando-os mais gerenciáveis.
- Os aplicativos desenvolvidos por terceiros podem ser colocados em esquemas separados, para não haver colisão com nomes de outros objetos.

Os esquemas são análogos a diretórios no nível do sistema operacional, exceto que os esquemas não podem ser aninhados.

Mais detalhes em: <http://pgdocptbr.sourceforge.net/pg80/ddl-schemas.html>

<http://www.postgresql.org/docs/8.3/interactive/ddl-schemas.html>

\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 portátil devemos excluir este esquema public e criar outros. Por default todos os usuários criados tem privilégio CREATE e USAGE para o esquema public.

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.

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.

"Esquemas são partições lógicas de um banco de dados. São formas de se organizar logicamente um banco de dados em conjuntos de objetos com características em comum sem criar bancos de dados distintos. Por exemplo, podem ser criados esquemas diferentes para dados dos níveis operacional, tático e estratégico de uma empresa, ou ainda esquemas para dados de cada mês de um ano.

Podem ser utilizados também como meios de se estabelecer melhores procedimentos de segurança, com autorizações de acesso feitas por esquema ao invés de serem definidas por objeto do banco de dados.

O PostgreSQL possui um conjunto de esquemas padrão, sendo que a inclusão de objetos do usuário por padrão é feita no esquema PUBLIC. Ao se criar um banco de dados, o banco apresentará inicialmente os seguintes esquemas:

- information_schema – informações sobre funções suportadas pelo banco. Armazena informações

sobre o suporte a SQL, linguagens suportadas e tamanho máximo de variáveis como nome de tabela, identificadores, nomes de colunas, etc.

- pg_catalog – possui centenas de funções e dezenas de tabelas com os metadados do sistema. Guarda informações sobre as tabelas, suas colunas, índices, estatísticas, tablespaces, triggers e demais objetos.

- pg_toast – Informações relativas ao uso de TOAST (The Oversized-Attribute Storage Technique).

- public – Esquema com as tabelas e objetos do usuário.

Exibir a ordem de busca dos Esquemas:

```
SHOW SEARCH_PATH;
```

```
postgres=# SHOW SEARCH_PATH;
search_path
-----
"$user",public
```

\$user – mostra que será procurado o esquema com o mesmo nome do usuário atual.

public – que será procurado no esquema public.

Alterando a ordem de busca dos Esquemas adicionando o esquema esquema1:

```
SET SEARCH_PATH TO public, esquema1;
```

Fonte: <http://postgresqlbr.blogspot.com/2007/06/esquemas-no-postgresql.html>

Criando Um Esquema

```
CREATE SCHEMA nomeesquema;
```

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.

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
```

```
SELECT current_schema();  
SELECT current_schemas(true);  
SELECT current_schemas(false);
```

Schemas ou Databases?

Fabio Telles - na lista de postgresql-br

Realmente é difícil justificar o uso de bancos de dados separados. Em geral, utilizar schemas é sempre mais indicado. Vale a pena lembrar que existem 3 níveis de compartilhamento num mesmo servidor:

* Cluster, que compartilham a mesma porta, os mesmos processos e a mesma configuração global (postgresql.conf e pg_hba.conf);

* Bancos de Dados, que compartilham o mesmo cluster mas podem ter codificação de caracteres distintos e dependendo da configuração, podem ter usuários distintos (se 'db_user_namespace' for configurado como ON);

* Shemas, que compartilham o mesmo banco de dados;

Em geral, utilizar mais de um banco de dados pode ser uma boa se:

- Você realmente precisa ter usuários com poderes plenos num banco de dados sem afetar outras aplicações. Isto é muito incomum, pois é normal você dar permissões para um usuário em um schema específico.

Mas se você quiser soltar seu estagiário para brincar no seu servidor... esta pode ser uma opção (mas eu instalaria o PostgreSQL localmente na máquina dele)

- Você precisa de ambientes de teste, homologação e/ou produção na mesma máquina. Em geral é melhor deixar seu ambiente de teste em outro servidor. Você vai descobrir que uma única consulta mal feita no ambiente de teste pode sentar todo o servidor... melhor evitar isso a todo o custo! Você também pode criar ambientes isolados em clusters separados e garantir mais recursos para o ambiente de testes ao ambiente de produção e ainda deixar os ambientes mais isolados, utilizando portas distintas.

- Você precisa de bancos de dados com codificações de caracteres diferentes. Bom... seria ideal você pensar em utilizar utf-8 para todas as bases se você está nesta situação. Mas isto é uma loooooonga conversa, para lá de complicada.

- Você precisa de configurações de desempenho diferentes para diferentes aplicações (OLTP x BI por exemplo). Bom, você pode fazer isso com schemas também... especificando parâmetros por usuário, ao invés de parâmetros por banco de dados, mas se você quer levar isto realmente a sério, é melhor criar clusters distintos e melhor ainda, utilizar servidores distintos.

Conclusão:

Se mais de uma aplicação podem conviver no mesmo servidor e no mesmo cluster, então elas podem estar no mesmo banco de dados. Existem raros casos em que isto não se aplique... mas são exceções e não a regra.

Criei um pequeno tutorial explicando como unificar vários bancos de dados distintos utilizando schemas... vide:

<http://www.midstorm.org/~telles/2006/09/28/unificando-bases-de-dados-com-schemas/>

Atenciosamente,
Fábio Telles

O ideal seria realmente o uso de schemas e centralização de informação. Se você mantiver diversos banco de dados, para cada bando terá que abrir uma conexão, ou então ficar mudando o tempo todo de banco em uso nas suas queries, o que pode causar facilmente erros. Se você tem tabelas compartilhadas entre os sistemas, melhor centralizar tudo mesmo. Aí, você pode aumentar o número de conexões simultâneas sem maiores problemas.

Hoje, onde trabalho, tenho os banco de dados divididos em schemas.

Por exemplo.

O schema security tem tabelas relacionados com segurança como menus dos usuários, direito de

cada módulo para cada usuário.

No schema scp temos o sistema de controle de processos.

No schema comum, temos tabelas que são utilizados por qualquer sistema ou schema, como por exemplo o cadastro de usuários que acessam o sistema.

Quando você coloca em banco de dados separado, você pode fazer views para fazer select com junções entre tabelas de banco de dados distintos, mas para isso vc tem que compilar e instalar o módulo dblink e utilizá-lo sempre que efetuar consultas com bancos distintos.

O schema facilita muito a criação de views e selects de tabelas distribuídas.

Quanto ao desempenho, bem, não notei redução de desempenho em nenhum serviço rodando nas 350 estações.

Uma dica. O banco de dados depende muito das operações de leitura e escrita. O mais importante no servidor é ter um bom disco e a configuração do postgresql.conf bem afinada.

Se você ver que o seu disco está sendo muito utilizado e o servidor está meio lento, você pode dividir as tabelas ou até mesmo os schemas em tablespaces. Isso permite vc colocar tabelas em um segundo disco. Quando você faz isso, o planejador pode trabalhar de forma a dividir o trabalho dos discos. Por exemplo:

Você tem os schemas A, B e C. Cada schema tem seu próprio sistema e cada sistema tem em média 100 usuários conectado simultaneamente.

Se você colocar em 3 discos e colocar cada schema em um disco, quando o usuário do sistema A, B e C gravarem dados simultaneamente o postgres vai gravar esses dados mais rápido. Isso porque cada transação é uma thread e as threads vão trabalhar em paralelo gravando cada um em um disco ao invés de uma thread esperar os dados do usuário A gravar para começar a gravar os dados do usuário B.

Cleber Costa Silva.

Se você tem cópias de tabelas mantidas em seus diversos bancos de dados e necessita manter a integridade entre elas então é melhor utilizar esquemas. Se seus bancos de dados são totalmente independentes então é melhor mantê-los separados.

Análise sob o aspecto da integridade dos dados e seu peso nos sistemas.

9) Segurança no SGBD

9.1) Segurança Física

9.1.1) Segurança do Hardware

9.1.2) Segurança no Acesso Físico ao Servidor

9.2) Segurança Lógica do SGBD

9.2.1) Segurança relativa aos grupos, usuários e privilégios (Capítulo 4)

9.2.2) Segurança do projeto dos bancos dados e seus objetos e consultas e código SQL

9.2.3) Segurança dos aplicativos (senhas, usuários, etc)

9.1) Segurança Física

9.1.1) Segurança do Hardware

Atualmente, segurança para servidores de bancos de dados não é uma opção mas um requisito, especialmente se o servidor estiver numa rede. Conhecer como permitir aos clientes chegarem ao SGBD e como evitar que outros não cheguem é uma tarefa importante para o DBA.

É importante que o hardware onde encontra-se o servidor do PostgreSQL, seja um hardware além de robusto fisicamente, seguro. Para isso diversos cuidados devem ser tomados:

- Fontes redundantes
- RAID com HDs

Muitos outros cuidados semelhantes, em especial que a origem do hardware seja confiável e tenha uma boa garantia.

Importante uma leitura do **PostgreSQL Hardware Performance Tuning**:

http://www.postgresql.org/files/documentation/books/aw_pgsql/hw_performance/

Traduzido: <http://www.vivaolinux.com.br/artigos/impressora.php?codigo=5930>

9.1.2) Segurança no Acesso Físico ao Servidor

Sem garantir segurança física ao servidor de nada vai adiantar os cuidados com a administração do mesmo, com usuários, senhas, etc. Primeiro garantir boa segurança no hardware, depois garantir que somente pessoal de confiança tenha acesso ao servidor. Finalmente a segurança lógica.

A segurança física, de acesso ao servidor, deve ser levada em conta e receber a devida atenção para que as informações possam de fato ser protegidas.

Por padrão, o PostgreSQL é instalado no Windows em: C:\Program Files\PostgreSQL\8.3 e no Linux, quando através dos fontes em: /usr/local/pgsql

Para saber o nome dos arquivos dos respectivos bancos de dados podemos usa a consulta:

```
select oid, datname from pg_database;
```

Uma forma de esconder um pouco alguns bancos é criando um Tablespace e os armazenando em outro diretório. Vide capítulo 2 do módulo 2.

Em termos de Sistema Operacional, tudo indica que um Linux seja mais seguro para um servidor do SGBD PostgreSQL, devido ao seu sistema de permissões e controle de usuários, que é mais exigente que o de um Windows. Sem contar que existem indícios de ser um ambiente para maior desempenho.

A Segurança da Informação Pessoal e Corporativa

O conceito de segurança da informação não está ligado somente à computadores e seus sistemas, este é um termo muito mais amplo utilizado para dar a idéia de segurança de dados pessoais ou corporativos.

O termo sempre é associado à segurança de informações digitais, mas devemos nos lembrar que a informação pode estar em qualquer mídia, um cd-rom, um pendrive, uma folha de papel, um bloco de notas, uma agenda...

Imagine que você tem uma agenda, dessas de papel (lembra que isso existe?), e nesta agenda você tem todos os seus dados pessoais, compromissos, telefones de amigos e familiares, contatos profissionais, informações bancárias e etc. Agora imagine que você perde essa agenda... O que fazer? Toda a sua vida está nela, será muito fácil alguém se passar por você de posse te todas essas informações.

Da mesma forma funcionam seus dados digitais, praticamente todos que usam computadores mantém algum tipo de dado pessoal armazenado nele. E a segurança disso? A diferença da a boa e velha agenda de papel é que os computadores podem ser acessados à distância, sem você saber. Porque é mais fácil conseguir informações no meio digital? Ora, encontrar uma agenda na rua não é algo tão comum, mas encontrar sistemas de informação sem proteção ou com proteção fraca é. Vamos pegar como exemplo uma rede corporativa com, por exemplo, um sistema acessado por meio de autenticação com senha. Esse sistema foi projetado e concebido com a idéia de ser seguro, ele usa conexão criptografada e ainda pede para a pessoa digitar algo aleatório exibido em uma imagem (Captcha) para evitar ataques de força bruta. Certo, até aqui temos um bom sistema, um sistema seguro e com uma bela muralha na frente. Qual seria o principal ponto fraco desse sistema supondo que não há bugs ou falhas exploráveis na autenticação dos usuários? Quer uma dica? Começa com 'Usu' e termina com 'ários'. É isso mesmo, de que adianta o alto investimento em um sistema seguro se o usuário anota a senha em um bloco de notas e o deixa em cima da mesa à vista de qualquer um? O que acontece quando o usuário usa como senha o próprio nome? Ou palavras simples? Ou mesmo seqüências de teclas no teclado como 'qwerty' e 'asdfg'? Ou mesmo o famoso '123'?

Pronto a segurança foi para o espaço, pouco adiantou investir alto no desenvolvimento de algo

seguro.

Todas as empresas, usando computadores ou não, deveriam ter um mínimo de preocupação com a segurança de informações. Estabelecer regras de comportamento para os funcionários, treiná-los para seguir essas regras, monitorar constantemente se estão seguindo tudo de conforme definido.

A falta de regras e treinamento pode levar os usuários a ter um ‘comportamento de risco’.

Políticas de boa utilização de dados e de segurança devem fazer parte da vida de todos, não apenas de grandes corporações, as micro e pequenas empresas estão sujeitas a ataques também, aliás os usuários domésticos correm riscos igualmente!

É claro que o pensamento ‘quem vai querer me invadir?’ existe. Afinal de contas quem vai querer invadir o meu pc? Eu não tenho informações de grande valor mesmo. É, mas com várias pequenas informações e muitos ‘bots’ se pode fazer muita coisa ilegal. Imagine a Polícia Federal chegando em sua casa com a acusação de que centenas de e-mails SPAM partem da sua conexão diariamente. Isso sim é uma situação desagradável, como provar que não é você o responsável se o programa que envia as mensagens está instalado no seu computador e disparando o tempo todo? Pois é meu amigo, ou você acha que para esse tipo de coisa os cybercriminosos invadem apenas os servidores da Nasa?

Como você pode proteger suas informações e seu computador de possíveis ataques?

- Mantenha um bom firewall ativo

Ele bloqueia conexões indesejadas.

- Não clique em links recebidos por e-mail

Mesmo que você tenha, supostamente, ganhado \$ 1mi.

- Não divulgue seu e-mail de forma irresponsável

Bots de rede varem sites em busca de e-mails

- Tente não seguir as malditas correntes de e-mail

Você sabia que seu IP vai no e-mail enviado?

- Nunca responda e-mails de desconhecidos

- Não aceite programas que lhe enviam por e-mail ou softwares de mensagens instantâneas

Para isso vá ao site do desenvolvedor e baixe o software ou compre

- Não utilize software pirata

Eles vem com cracks que, além de quebrar a segurança do programa, contém vírus, spyware e malware em geral.

- Não utilize senhas simples!

Entrar no e-mail de alguém que deixa a senha ‘qwerty’ não pode nem ser considerado crime, já que essa senha e senha nenhuma nenhuma senha são praticamente equivalentes.

- Mantenha um bom anti-vírus

Sim, esse é um dos fatores principais (Ainda mais se você usa o sistema número um em quantidade de vírus). Preciso dizer que anti-vírus ‘crackeado’ é inútil?

- Proteja seu computador com senha

O atacante pode ser uma visita que liga o pc e pega as informações.

- Não acesse o site do banco em lan-houses!

Nestes lugares podem estar ativos os chamados Keyloggers que guardam todas as teclas digitadas.

Ou seja, a grande sacada é agir com bom senso e evitar armadilhas.

Nunca dê seus dados à pessoas que ligam oferecendo serviços e produtos ou mesmo dizendo ser do banco, da Receita Federal, etc... Sempre que precisa ou desconfiar de algo vá ao banco ou ligue, vá ao posto de atendimento da Receita, mas não acredite em ligações!

A segurança da informação é algo que é praticamente impossível de ser 100% efetiva, sempre existem falhas humanas, falhas nos softwares, falhas nos cadeados, agendas se perdem...

Mas agindo com bom senso, guardando bem os dados pessoais, protegendo as informações nos computadores e identificando possíveis golpes é possível tornar seu dia-a-dia bem melhor.

Não pense que ninguém quer suas informações porque elas não valem nada, elas valem sim, mas eu tenho certeza que para você e sua empresa elas não têm preço.

<http://infog.casoft.info/?p=32>

9.2) Segurança Lógica do SGBD

Jamais aconselharia deixar dessa maneira para um banco em produção. Com isso o usuário postgres poderia ter acesso (mesmo que com senha) de qualquer máquina da rede. Aconselho a permitir conexões com o usuário postgres apenas para os endereços 127.0.0.1/32 e o ip do servidor/32. E não use o usuário postgres como usuário de conexão em nenhum dos casos. Tenha um usuário apenas com os GRANTS necessários para não deixar a segurança da sua base de dados vulnerável.

Fernando Brombatti na lista pgbr-geral.

Cada aplicação deve ter um usuário com acesso apenas ao necessário para usar a aplicação.

As configurações default do postgresql rejeitam toda conexão de outros computadores e usam a autenticação do tipo ident para gerenciar o acesso de usuários com mesmo nome no sistema operacional (isso em Linux/UNIX/BSD, não no Windows).

As versões atuais também suportam autenticação tipo LDAP.

Segurança no Sistema de Arquivos

Por default quando instalamos o PostgreSQL no Linux através dos fontes ele é instalado em:

/usr/local/pgsql

No Windows uma instalação com o instalador ele fica em:

[C:\Arquivos](#) de Programas\PostgreSQL

Os programas executáveis, como o pg_dump, o psql e outros ficam num subdiretório bin desse diretório de instalação.

Os bancos, logs e outros arquivos ficam no subdiretório data. Dentro desse subdiretório data ficam alguns diretórios importantes como o base (que abriga os bancos de dados), o global (que guarda informações de todos os bancos) e o pg_xlog (guarda os logs de transações).

Os bancos que ficam no diretório data/base são gravados com números dos OIDs. Para saber que banco corresponde aos OIDs podemos usar uma consulta à tabela de sistema pg_database:

```
dnocs=# select oid, datname from pg_database;
```

```
 oid | datname
-----+-----
10819 | postgres
16400 | template_postgis
16816 | iniciante
16821 | dnocs
      | 1 | template1
10818 | template0
16835 | biblioteca
(7 rows)
```

Uma boa prática é monitorar as informações detalhadas sobre os arquivos e diretórios do PostgreSQL: permissões, data de criação e modificação, dono, etc.

Lembrando que no Linux a permissão de execução em diretórios permite listar o diretório e o acesso ao diretório.

Conexões Remotas

Também devemos tomar precauções ao conectar remotamente ao servidor, usando conexões seguras como SSH.

Veja outras recomendações sobre Conexões TCP/IP seguras por túneis SSH em:

<http://pgdocptbr.sourceforge.net/pg80/ssh-tunnels.html>

E Conexões TCP/IP seguras com SSL em:

<http://pgdocptbr.sourceforge.net/pg80/ssl-tcp.html>

9.2.1) Segurança relativa aos usuários

Em relação aos usuários, grupos e privilégios, já vimos na aula 4: 4) Administração de Grupos, Usuários e Privilégios.

Esta segurança é interna do SGBD. Mas antes que chegue ao SGBD deve passar por uma barreira de segurança chamada Firewall, depois de passar pelo Firewall deve ainda passar pelos scripts postgresql.conf, pelo pg_hba.conf e pg_ident.conf (caso esteja num Linux).

Firewall – esta é a mais básica barreira de segurança de um servidor que encontra-se numa rede. Ele nos permite filtrar que clientes irão passar pelo firewall para que aplicações. Enquanto podemos ter um único firewall protegendo toda uma rede, também podemos ter um firewall protegendo um único servidor. Quando habilitamos um firewall, por default, está tudo bloqueado. A partir daí precisamos desbloquear cada cliente que deve ter acesso.

postgresql.conf - Este é o primeiro script que o cliente remoto encontra quando pretende conectar com o servidor do PostgreSQL. O acesso remoto e muitas configurações importantes começam neste script. Para que clientes remotos tenham acesso precisamos alterar a configuração padrão de localhost para *:

```
listen_addresses = '*'
```

pg_hba.conf - O próximo passo é a configuração do pg_hba.conf, que define que usuários podem conectar a que bancos, usando que IP ou rede com respectivas máscaras e com que sistema de autenticação. Cada linha define uma individual regra de acesso. Somente terá acesso se satisfazer a todas as colunas.

O pg_hba.conf permite linhas de três tipos: comentários, em branco e registros (linhas válidas ao final). Os registros podem ser separados por tabulação ou espaços. Espaços no início e ao final são ignorados. Um registro obrigatoriamente deve ser contido em uma única linha.

As colunas de cada linha do script são:

conexão-tipo	banco	usuário	end_rede	método-login	opções
--------------	-------	---------	----------	--------------	--------

Exemplo:

hostssl	all	all	10.0.1.0/28	password	
host	teste	joao	192.168.120.5	md5	
host	teste2	joao	192.168.120.5	md5	
host	teste3	all	10.0.2.0/28	md5	

O tipo de conexão pode ser:

local – conexão local ao SGBD,

host – conexão permitida somente quando existe o suporte à conexões tipo TCP/IP.

Hostssl – deve estar habilitado o suporte a SSL.

Para testar uma conexão remota podemos usar um cliente como o psql:

```
psql -h 10.0.1.33 teste joao
```

Habilitando SSL no PostgreSQL

A instalação for Windows já suporta SSL por default. No Linux devemos verificar ou habilitar.

Habilitar no postgresql.conf:

```
ssl = on
```

Após restartar o servidor ele estará escutado por conexões normais (TCP) e conexões SSL (SSL TCP) na mesma porta. Logo que conecta com suporte a SSL o PostgreSQL procura a chave de criptografia e os arquivos do certificado no diretório 'data' e não iniciará até que encontre os mesmos. Vide capítulo 3 do Livro "PostgreSQL 8 for Windows".

Obs.: Nunca use o tipo de autenticação **trust** em redes que não ofereçam uma boa segurança.

O tipo de autenticação **ident** deixa a cargo do cliente a segurança.

Antes de efetuar alterações no tipo de autenticação do pg_hba.conf, por exemplo, para md5, conceda senha ou as altere para todos os usuários, como a seguir:

```
CREATE ROLE usuario WITH ENCRYPTED PASSWORD 'senha';
```

```
ALTER ROLE usuario WITH ENCRYPTED PASSWORD 'senha';
```

Obs.: Se a senha for em texto claro, do tipo **password**, não use a palavra ENCRYPTED

Após isso altere o pg_hba.conf e restart o servidor.

Rejeitando Conexões

Um dos métodos de autenticação é o **reject**, que pode ser aplicado em caso de suspeita ou certeza de conexão não desejada.

Exemplo:

```
host all          192.168.0.15          255.255.255.255 reject
```

Monitorando Usuários

Através do PGAdmin podemos monitorar muito bem todos os usuários e suas ações.

Após conectar clique em Ferramentas – Status do Servidor.

Aí podemos monitorar o PID, usuário, banco, IP, início da conexão, consultas, etc.

Como também os bloqueios, as transações e o arquivo de Log. Até rotacionar o arquivo de Log,

caso necessário e caso o usuário conectado tenha este privilégio.

Referência: Capítulo 10 do Livro PostgreSQL 8 for Windows.

9.2.2) Segurança do projeto dos bancos dados e seus objetos e consultas e código SQL

Esta parte diz respeito ao código SQL das consultas, que deve contemplar pelo menos as 3 formas normais, sempre usar chaves primárias nas tabelas, índices em campos muito consultados na cláusula WHERE e boas constraints que venham a melhorar a segurança das consultas.

Também vale lembrar que é útil o uso de VIEWS, de funções e procedures.

Usuários autorizados podem se aproveitar da estrutura dos objetos do SGBD para ter acesso ao que não devia.

9.2.3) Segurança dos aplicativos (senhas, usuários, etc)

Este tópico deve fazer parte da política de segurança da empresa/instituição para suas informações.

Aplicativos devem usar senhas fortes e renovar as mesmas com certa periodicidade.

Veja alguns bons artigos sobre segurança com senhas:

<http://www.microsoft.com/brasil/technet/Colunas/DiogoHenrique/SenhasAltaSeguranca2.msp>

Recomendações e Procedimentos de Segurança :

<http://www2.iq.usp.br/sti/index.dhtml?pagina=754&chave=89N>

Recomendações para Política de Senhas:

http://www.brasilacademico.com/maxpt/article_read.asp?id=190&ARTICLE_TITLE=Recomenda%E7%F5es+para+Pol%EDtica+de+Senhas&CATEG_Title=Dicas%3A+Seguran%E7a

Desafios da Segurança de Informação

Autor: Anderson de Sousa Pereira <link.twister at gmail.com>

Desafios

O maior desafio para as empresas é manter a segurança de seus dados, e quando falamos em segurança de dados não devemos ter o luxo de ignorar qualquer tipo de risco que possa comprometer a integridade dos mesmos. E esses riscos envolvem corrompimentos de dados, perda, roubo, entre outros fatores... Isso pode ser ocasionado devido a fenômenos naturais como, furacões, inundações, terremotos... E também podem ocorrer devido á uma má administração.

O maior perigo para uma empresa é ter a falsa sensação de segurança. Afinal, pior do que não se preocupar com invasões, ataques, vírus, tragédias entre outras ameaças é confiar cegamente em uma estrutura que não esteja preparada para impedir o surgimento de problemas.

Por isso, não devemos confiar apenas em software, ou hardware. Quando falamos em segurança da

informação, é necessário considerar quatro obstáculos que têm a mesma importância: a natureza, as pessoas, a tecnologia e os processos.

Não adianta investir apenas em um deles e deixar os outros de lado, eles devem ter a mesma atenção. E os quatro devem estar intimamente ligados, afinal um necessita do outro. Quanto a tecnologia, natureza e processos até que não é um desafio tão gritante, o maior desafio mesmo são as pessoas.

Quanto a natureza, nós podemos nos prevenir com um estudo do local onde a empresa será implantada e mantendo cópias dos dados em outros locais, afinal até alguns anos atrás aqui no Brasil eu não me preocuparia com furacões e terremotos, mas ultimamente com o aquecimento global tudo é possível, só do ano de 2007 até hoje já tiveram vários tremores em algumas cidades do nosso país, claro que isso é só a ponta do iceberg...

Quanto à tecnologia e processos, desde que sejam bem administrados e usados corretamente se tornam ferramentas essenciais para o crescimento e evolução do negócio. Parece fácil, mas embora a teoria seja lógica, fazer com que sejam bem implantados é uma difícil e árdua missão. Afinal, é aí que entram as pessoas, cada um usando a tecnologia a seu favor e muitas vezes para fazer um mal uso. E quando eu falo de pessoas eu estou me referindo aos usuários (meros mortais) que dedicam suas vidas à atormentar o CSO (Chefe...xD). Grande parte dos incidentes de segurança contam com o apoio, intencional ou não, do inimigo interno. As pessoas estão em toda a parte da empresa.

Os cuidados básicos com as atitudes das pessoas, muitas vezes são esquecidos ou ignorados. Encontrar senhas escritas e coladas no monitor, bem como o repasse de informações sigilosas em ambientes não seguros, como parada de ônibus, reuniões informais são situações muito comuns. Contar com a colaboração das pessoas é simplesmente fundamental para a empresa. Mas tudo deve ser auditado pelo CSO. Mas o ponto principal da preocupação com as pessoas é que os fraudadores irão em busca delas para perpetuar seus crimes.

Mas nem sempre as pessoas tem 100% de culpa, afinal, no mundo cibernético, tem muito lixo e armadilhas que foram projetadas justamente para enganar pessoas, aí fica muitas vezes difícil de controlar os processos que são confiáveis e não confiáveis, então cabe ao CSO orientar e auditar toda e qualquer mudança que possa colocar em risco os dados da corporação. Antivírus, anti-spywares, um bom proxy e um firewall são essenciais, mas nada melhor do que orientar e auxiliar os usuários no uso da tecnologia a favor da corporação..

Postado no vivaolinux - <http://www.vivaolinux.com.br/artigos/impressora.php?codigo=8133>
Recomendo a leitura do post:

- <http://marcelokalib.blogspot.com/2008/04/procura-se-segurana-particular.html>

Referências

Livros:

- PostgreSQL – Korry Douglas, Susan Douglas, SAMS, Capítulo 21
- PostgreSQL Developer's Handbook de Ewald Geschwinde, Hans-Jürgen Schöning, SAMS, Capítulo 6
- Boas fontes de informações: Capítulo 23 do livro PostgreSQL The Comprehensive Guide de Korry Douglas e Susan Douglas.

e Security and Access Restrictions no capítulo 6 do livro PostgreSQL Developer's Handbook de Ewald Geschwinde, Hans-Jürgen Schöning.

Riscos Envolvendo Informações

- Concentração das informações
- Permitir acesso indiscriminado
- Obscuridade
- Concentração de funções
- Falta de controle
- Retenção duradoura
- Relacionamento e Combinação de Informações
- Introdução de erros
- Lealdade
- Acesso não autorizado
- Perda da integridade

Principais Fatores de Segurança da Informação de Clare Less, Telecommucations, fev. 1989

Infra-estrutura

Separação de ambientes

Energia e ar-condicionado
Radiação eletro-magnética

Proteção física

Recursos Técnicos - Integridade de dados - Confiabilidade - Integridade de programas - Gerenciam.de Recursos de PD	Segurança da Informação	Recursos Humanos: - Controle de acesso - Autorização de acesso - Identificação de usuários
	Manutenção - Proteção da privacidade dos dados - Salvaguardas legais - Organização	

Políticas de Segurança

Definir os Objetivos

Que deve constar?

- objetivos
- destino
- propriedade dos recursos
- responsabilidades
- requisitos de acesso
- responsabilização

- generalizações

Procedimentos

Softwares de segurança: firewalls, anti-virus, antispy, etc.

Backups, mídias, local de armazenamento e acesso

Política de distribuição de e-mails

Logins e senha de usuários locais e de e-mails

Referência: Livro Segurança em Informática e de Informações, de Carso & Steffen

10) Melhorando a Performance do PostgreSQL

O Que O Que Esperar do Tunning do SGBD

Tunning de banco de dados não é nenhuma panacéia.

Antes que o Tunning do SGBD possa trazer resultados concretos, é necessário que haja cuidado na:

Modelagem conceitual e física das tabelas

Escrita dos comandos SQL enviados pelas aplicações

Definição de índices, clustering de tabelas, etc

Dimensionamento da rede de comunicação entre clientes e o servidor

Metodologia para Tunning

Defina requisitos de desempenho

Isole medições sobre o SO, SGBD, rede e clientes

Realize bechmarks em um sistema sem carga, para obter um parâmetro de "melhor possível"

Realize benchmarks continuamente, para medir a melhora (ou piorar) a cada etapa de tuning

Pare quando atingir os requisitos!

Tunning da Aplicação

Modelagem física

Otimização de comandos SQL

Modelagem Física

Utilize tabelas em CLUSTER caso sejam frequentemente acessadas segundo uma única ordenação

Evite a denormalização de dados; em geral, o custo de manutenção da redundância (fora o risco de perda da integridade dos dados) não compensa os pequenos ganhos de performance obtidos

Utilize bancos replicados para aplicações com padrões de acesso muito diferenciados, ex: aplicação OLAP de linha-de-negócios e aplicação OLTP, gerencial

Otimização Otimização de Comandos SQL

Utilize o comando EXPLAIN para visualizar o plano de acesso dos comandos SQL utilizados pela aplicação

Experimente criar índices para otimizar junções, filtros, agrupamento ou ordenação, e verifique os resultados com o comando EXPLAIN

Execute o comando VACUUM ANALYSE regularmente, para atualizar as estatísticas de tabelas e índices

Verifique periodicamente mudanças nos planos de acesso – pode ser necessario mudar os índices!

Arquitetura do PostgreSQL

Processos

Memória

Disco

Processos Processos do PostgreSQL

Um único processo postmaster escuta por conexões TCP ou Unix Sockets, e inicia um processo postgres para cada conexão

Assim sendo, o PostgreSQL utiliza naturalmente múltiplos processadores, mas para usuários simultâneos, não para acelerar um único comando SQL

Dois processos postgres estão sempre ativos, mas não atendem a conexões. Eles cuidam da gravação física de blocos de log ou tabelas, e da manutenção de estatísticas

Memória Memória e o PostgreSQL

Todos os processos postgres compartilha duas áreas de memória:

O buffer cache armazena blocos lidos (ou modificados) de tabelas e índices

O write-ahead log armazena temporariamente o log de transações, até que ele possa ser armazenado em disco

Além disso, cada processo postgres tem uma área individual para operações de ordenação (sort buffer)

Disco e o PostgreSQL

Bancos de dados são nada mais do que diretórios

Tabelas e índices são arquivos dentro destes diretórios

Um diretório em separado armazena um ou mais segmentos de log de transações

Não há no PostgreSQL estruturas similares a tablespaces, log groups, etc

Conta-se com a capacidade do SO em alocar espaço em disco de forma eficiente, e com espelhamento pelo HW

Tunning do SO

O que medir

Como medir

Tunning do SO

Um SGBD, como qualquer outra aplicação, utiliza processos, memória, disco e conexões de rede. Portanto, obtenha do seu SO medidas de cada uma dessas áreas!

Está havendo muito swap?

A fila de requisições ao disco está crescendo?

A fila de requisições à placa de rede está crescendo?

O processador está ocioso?

Tunning do SO

Muitos contadores do Monitor de Performance do Windows NT/2000 apresentam valores incorretos, especialmente em discos IDE.

Sistemas Unix (Linux, FreeBSD, etc) são ricos em ferramentas de monitoração – aprenda a usa-las!

vmstat

netstat

ps

/proc

Espalhando Arquivos pelos Discos

Para obter máxima performance de E/S, utilize discos dedicados para:

Log de transações

Índices de tabelas

Tabelas muito/pouco consultadas

Tabelas muito/pouco atualizadas

Utilize links simbólicos para mover tabelas, índices e etc para os discos dedicados

Objetos x Arquivos

Dado o banco de dados, qual o seu diretório:

```
select datname, oid from pg_database;
```

```
teste | 16976
```

Dado a tabela, qual o seu arquivo:

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

```
contato_pkey | 16979
```

```
contato | 16977
```

Então os dados da tabela "contato" do banco "teste"

estão no arquivo base/16976/16977

Recomendações

25% da RAM para shared buffer cache

2-4% da RAM para sort buffers

Pode ser necessário recompilar o kernel (do Linux)

para sistemas com mais do que 2Gb de RAM

Algumas versões do Linux não suportam mais do que 2Gb de memória compartilhada (shared buffer cache)

Apresentação de: **Fernando Lozano** www.lozano.eti.br

PostgreSQL Tuning: O elefante mais rápido que um leopardo

O Banco está Lento – Problemas Comuns

- 60% dos problemas são relacionados ao mau uso da linguagem SQL;
- 20% dos problemas são relacionados a má modelagem do banco de dados;
- 10% dos problemas são relacionados a má configuração do SGDB;
- 10% dos problemas são relacionados a má configuração do SO.

O Banco está Lento – Decisões erradas

Concentração de regras de negócio na aplicação para processos em lote;

- Integridade referencial na aplicação
- Mal dimensionamento de I/O (CPU, Plataforma, Disco)
- Ambientes virtualizados (Vmware, XEN, etc..) em AMD64/EMT64
- Uso de configurações padrões do SO e/ou do PostgreSQL

Melhor Hardware

Servidores dedicados para o PostgreSQL

- Storage com Fiber Channel e iSCSI: Grupos de RAID dedicados
- RAID 5 ou 10: por Hardware
- Mais memória! (Até 4GB em 32 bits)
- Processadores de 64 bits: Performance até 3 vezes do que os 32 bits (AMD64 e EMT64 - Intel)

Melhor SO

Sistemas Operacionais *nix: Linux (Debian, Gentoo), FreeBSD, Solaris, etc

- Em Linux: use Sistemas de arquivos XFS (noatime), Ext3 (writeback, noatime), Ext2
- Instale a última versão do PostgreSQL (atualmente 8.2) e à partir do código-fonte
- Não usar serviços concorrentes (Apache, MySQL, SAMBA...) em discos, semáforos e shared memory
- Usar, se possível, um kernel (linux) mais recente (e estável)

Parâmetros do SO – Modificando o *nix

```
echo "2" > /proc/sys/vm/overcommit_memory
```

```
echo "25%" > /proc/sys/kernel/shmmax
```

```
echo "25%/64" > /proc/sys/kernel/shmall
```

```
echo "deadline" > /sys/block/sda/queue/scheduler
echo "250 32000 100 128" > /proc/sys/kernel/sem
echo "65536" > /proc/sys/fs/file-max
ethtool -s eth0 speed 1000 duplex full autoneg off
echo "16777216" > /proc/sys/net/core/rmem_default
echo "16777216" > /proc/sys/net/core/wmem_default
echo "16777216" > /proc/sys/net/core/wmem_max
echo "16777216" > /proc/sys/net/core/rmem_max
pmanson:~# su - postgres
postgres@pmanson:~$ ulimit 65535
```

```
/etc/security/limits.conf
postgres soft nofile 4096
postgres hard nofile 65536
```

Como Organizar os Discos – O Melhor I/O

Discos ou partições distintos para:

- Logs de transações (WAL)
 - Índices: Ext2
 - Tabelas (particionar tabelas grandes)
 - Tablespace temporário (em ambiente BI)*
 - Archives
 - SO + PostgreSQL
 - Log de Sistema
- * Novo no PostgreSQL 8.3

postgresql.conf – Memória

- max_connections: O menor número possível
- shared_buffers: 33% do total -> Para operações em execução
- temp_buffers: Acesso às tabelas temporárias
- work_mem: Para agregação, ordenação, consultas complexas
- maintenance_work_mem: 75% da maior tabela ou índice
- max_fsm_pages: Máximo de páginas necessárias p/ mapear espaço livre. Importante para operações de UPDATE/DELETE.

postgresql.conf – Disco e WAL

- wal_sync_method: open_sync, fdatasync, open_datasync
- wal_buffers: tamanho do cache para gravação do WAL
- commit_delay: Permite efetivar várias transações na mesma chamada de fsync
- checkpoint_segments: tamanho do cache em disco para operações de escrita
- checkpoint_timeout: intervalo entre os checkpoints
- wal_buffers: 8192kB -> 16GB
- bgwriter: ????
- join_collapse_limit = > 8

Tuning de SQL

- Analyze:
test_base=# EXPLAIN ANALYZE SELECT foo FROM bar;
- Ferramentas:
 - Pgfoine;
 - Pgadmin3;
 - PhpPgAdmin;

Manutenção

- Autovacuum X Vacuum: Depende do uso (Aplicações Web, OLT, BI)
 - Vacuum:
- vacuum_cost_delay: tempo de atraso para vacuum executar automaticamente nas tabelas grandes
- Autovacuum (ativado por padrão a partir da versão 8.3):
- autovacuum_naptime: tempo de espera para execução do autovacuum.

Ferramentas de Stress

- Pgbench: no diretório do contrib do PostgreSQL, padrão de transações do tipo TPC-B.
- DBT-2: Ferramenta da OSDL, padrão de transações do tipo TPC-C.
- BenchmarkSQL: Ferramenta Java para benchmark em SQL para vários banco de dados (JDBC), padrão de transações do tipo TPC-C.
- Jmeter: Ferramenta Java genérica para testes de stress, usado para aplicações (Web, ...) e também pode ser direto para um banco de dados.

Quando o tuning não resolve

- Escalabilidade vertical:
 - Mais e melhores discos;
 - Mais memória;
 - Melhor processador (quad core, 64bits)
 - Escalabilidade horizontal:
 - Pgpool I (distribuição de carga de leitura e pool de conexões)
 - PgPool II (PgPool I + paralelização de grandes consultas)
 - Slony I (Replicação Multi-Master Assíncrona)
 - Warm Stand By
 - PgBouncer + PL/Proxy + Slony
- Apresentação de: Fernando Ike e Fábio Telles

Checklist de performance do PostgreSQL 8.0

Autor: Fábio Telles Rodriguez

Tradução livre do texto "PostgreSQL 8.0 Performance Checklist", publicado por Josh Berkus em 12/01/2005 em:

- <http://www.powerpostgresql.com/PerfList>

Copyright (c) 2005 by Josh Berkus and Joe Conway. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Aqui está um conjunto de regras para configurar seu servidor PostgreSQL 8.0. Muito do que está abaixo é baseado em evidências ou testes de escalabilidade práticos; há muito sobre performance de bancos de dados que nós e a OSDL, ainda estamos trabalhando. Contudo, isto deve ser um início. Todas as informações abaixo são úteis a partir de 12 de janeiro de 2005 e serão atualizadas depois. Discussões sobre configurações abaixo superam as que eu realizei no General Bits.

Cinco Princípios de Hardware para Configurar o seu Servidor PostgreSQL

1. Discos > RAM > CPU

Se você vai gastar dinheiro em um servidor PostgreSQL, gaste em arranjos de discos de alta performance e tenha processadores medianos e uma memória adequada. Se você tiver um pouco mais de dinheiro, adquira mais RAM. PostgreSQL, como outros SGDBs que suportam ACID, utilizam E/S muito intensamente e é raro uma aplicação utilizar mais a CPU do que a placa SCSI (com algumas exceções, claro). Isto se aplica tanto a pequenos como grandes servidores; obtenha uma CPU com custo baixo se isso permitir você comprar uma placa RAID de alta performance ou vários discos.

2. Mais unidades de discos == Melhor

Tendo múltiplos discos, o PostgreSQL e a maioria dos sistemas operacionais irão paralelizar as requisições de leitura e gravação no banco de dados. Isto faz uma enorme diferença em sistemas transacionais, e uma significativa melhoria em aplicações onde o banco de dados inteiro não cabe na RAM. Com os tamanhos mínimos de discos (72GB) você será tentado a utilizar apenas um disco ou um único par espelhado em RAID 1; contudo, você verá que utilizando 4, 6 ou até 14 discos irá render um impulso na performance. Ah, e SCSI é ainda significativamente melhor em fluxo de dados em BD que um IDE ou mesmo um Serial ATA.

3. Separe o Log de Transações do Banco de Dados

Assumindo que você já investiu dinheiro num arranjo com tamanho decente num conjunto de discos, existe um monte de opções mais inteligentes do que jogar tudo num único RAID. De início coloque o log de transações (pg_xlog) no seu próprio recurso de discos (um arranjo ou um disco), o que causa uma diferença de cerca de 12% na performance de bancos de dados com grande volume de gravações. Isto é especialmente vital em pequenos sistemas com discos SCSI ou IDE lentos: mesmo em um servidor com dois discos, você pode colocar o log de transações sobre o disco do sistema operacional e tirar algum benefício.

4. RAID 1+0/0+1 > RAID 5

RAID 5 com 3 discos tem sido um desafortunado padrão entre vendedores de servidores econômicos. Isto possibilita a mais lenta configuração de discos possível para o PostgreSQL; você pode esperar pelo menos 50% a menos de velocidade nas consultas em relação ao obtido com discos SCSI normais. Por outro lado, foque em RAID 1 ou 1+0 para um conjunto de 2, 4 ou 6

discos. Acima de 6 discos, o RAID 5 começa a ter uma performance aceitável novamente, e a comparação tende a ser bem melhor com base na sua controladora individual. No entanto, o mais importante, usar uma placa RAID barata pode ser um risco; é sempre melhor usar RAID por software do que um incorporado numa placa Adaptec que vem com seu servidor.

5. Aplicações devem rodar bem junto

Outro grande erro que eu vejo em muitas organizações é colocar o PostgreSQL em um servidor com várias outras aplicações competindo pelos mesmos recursos. O pior caso é colocar o PostgreSQL junto com outros SGDBs na mesma máquina; ambos bancos de dados irão lutar pela banda de acesso aos discos e o cache de disco do SO, e ambos vão ter uma performance pobre. Servidores de arquivo e programas de log de segurança também são ruins. O PostgreSQL pode compartilhar a mesma máquina com aplicações que utilizam principalmente CPU e RAM intensamente, como o Apache, garantindo que exista RAM suficiente.

Doze Ajustes que Você Irá Querer Fazer no Seu Arquivo PostgreSQL.Conf

Existem um monte de novas opções verdadeiramente assustadoras no arquivo PostgreSQL.conf. Mesmo as já familiares opções das 5 últimas versões mudaram de nomes e formato dos parâmetros. Elas tem a intenção de dar ao administrador de banco de dados mais controle, mas podem levar algum tempo para serem usados.

O que segue são configurações que a maioria dos DBAs vão querer alterar, focado no aumento de performance acima de qualquer outra coisa. Existem algumas poucas configurações que particularmente a maioria dos usuários não querem mexer, mas quem o fizer irá descobri-las indispensáveis. Para estes, vocês terão de aguardar pelo livro.

Lembre-se: as configurações no PostgreSQL.conf precisam ser descomentadas para fazerem efeito, mas recommentá-las não restaurará necessariamente o valor padrão!

Conexão

`listen_addresses`:

Substitui as configurações `tcp_ip` e o `virtual_hosts` do 7.4. O padrão é `localhost` na maioria das instalações, habilitando apenas conexões pelo console. A maioria dos DBAs irá querer mudar isto para `"*"`, significando que todas as interfaces avaliáveis, após configurar as permissões em `hba.conf` apropriadamente, irão tornar o PostgreSQL acessível pela rede. Como uma melhoria sobre a versão anterior, o `"localhost"` permite conexões pela interface de `"loopback"`, `127.0.0.1`, habilitando vários utilitários baseados em servidores web.

`max_connections`:

Exatamente como na versão anterior, isto precisa ser configurado para o atual número de conexões simultâneas que você espera precisar. Configurações altas vão requerer mais memória compartilhada (`shared_buffers`). Como o overhead por conexão, tanto do PostgreSQL como do SO do host podem ser bem altos, é importante utilizar um pool de conexões se você precisar servir um número grande de usuários. Por exemplo, 150 conexões ativas em um servidor Linux com um processador médio de 32 bits consumirá recursos significativos, e o limite deste hardware é de 600. Claro que um hardware mais robusto irá permitir mais conexões.

Memória

`shared_buffers`:

Como um lembrete: Este não é a memória total do com o qual o PostgreSQL irá trabalhar. Este é o bloco de memória dedicado ao PostgreSQL utilizado para as operações ativas, e deve ser a menor parte da RAM total na máquina, uma vez que o PostgreSQL usa o cache de disco também. Infelizmente, o montante exato de `shared buffers` requer um complexo cálculo do total de RAM, tamanho do banco de dados, número de conexões e complexidade das consultas. Assim, é melhor seguir algumas regras na alocação, e monitorar o servidor (particularmente as visões `pg_statio`) para determinar ajustes.

Em servidores dedicados, valores úteis costumam ficar entre 8MB e 400MB (entre 1000 e 50.000 para páginas de 8K). Fatores que aumentam a quantidade de `shared buffers` são grandes porções ativas do banco de dados, consultas grandes e complexas, grande número de conexões simultâneas,

longos procedimentos e transações, maior quantidade de RAM disponível, CPUs mais rápidas ou em maior quantidade obviamente, outras aplicações na mesma máquina. Contrário a muitas expectativas, alocando, muita, demasiadamente `shared_buffers` pode até diminuir a performance, aumentando o tempo requerido para explorá-la. Aqui estão alguns exemplos baseados em experiências e testes TPC em máquinas Linux:

- Laptop, processador Celeron, 384MB RAM, banco de dados de 25MB: 12MB/1500;
- Servidor Athlon, 1GB RAM, banco de dados de 10GB para suporte a decisão: 120MB/15000;
- Servidor Quad PIII, 4GB RAM, banco de dados de 40GB, com 150 conexões e processamento pesado de transações: 240MB/30000;
- Servidor Quad Xeon, 8GB RAM, banco de dados de 200GB, com 300 conexões e processamento pesado de transações: 400MB/50000.

Favor notar que incrementando `shared_buffer`, e alguns outros parâmetros de memória, vão requerer que você modifique o System V do seu sistema operacional. Veja a documentação principal do PostgreSQL para instruções nisto.

`work_mem`:

Costuma ser chamado de `sort_mem`, mas foi renomeado uma vez que ele agora cobre ordenações, agregações e mais algumas operações. Esta memória não é compartilhada, sendo alocada para cada operação (uma a várias vezes por consulta); esta configuração está aqui para colocar um teto na quantidade de memória que uma única operação ocupar antes de ser forçada para o disco. Este deve ser calculado dividindo a RAM disponível (depois das aplicações e do `shared_buffers`) pela expectativa de máximo de consultas concorrentes vezes o número de memória utilizada por conexão. Considerações devem ser tomadas sobre o montante de `work_mem` por consulta; processando grandes conjuntos de dados requisitará mais. Bancos de dados de aplicações Web geralmente utilizam números baixos, com numerosas conexões mas consultas simples, 512K a 2048K geralmente é suficiente. Contrariamente, aplicações de apoio a decisão com suas consultas de 160 linhas e agregados de 10 milhões de linhas precisam de muito, chegando a 500MB em um servidor com muita memória. Para bancos de dados de uso misto, este parâmetro pode ser ajustado por conexão, em tempo de execução, nesta ordem, para dar mais RAM para consultas específicas.

`maintenance_work_mem`:

Formalmente chamada de `vacuum_mem`, esta quantidade de RAM é utilizada pelo PostgreSQL para o VACUUM, ANALYZE, CREATE INDEX, e adição de chaves estrangeiras. Você deve aumentar quanto maior forem suas tabelas do banco de dados e quanto mais memória RAM você tiver de reserva, para fazer estas operações o mais rápidas possível. Uma configuração com 50% a 75% da sua maior tabela ou índice em disco é uma boa regra, ou 32MB a 256MB onde isto não pode ser determinado.

Disco e WAL

`checkpoint_segments`:

Define o tamanho do cache de disco do log de transações para operações de escrita. Você pode ignorar isto na maioria dos bancos de dados web com a maioria das operações em leitura, mas para bancos de dados de processamento de transações ou para bancos de dados envolvendo grandes cargas de dados, o aumento dele é crítico para a performance. Dependendo do volume de dados, aumente ele para algo entre 12 e 256 segmentos, começando conservadoramente e aumentando se você ver mensagens de aviso no log. O espaço requerido no disco é igual a $(\text{checkpoint_segments} * 2 + 1) * 16\text{MB}$, então tenha certeza de ter espaço em disco suficiente (32 significa mais de 1GB).

`max_fsm_pages`:

Dimensiona o registro que rastreia as páginas de dados parcialmente vazias para popular com novos dados; se configurado corretamente, torna o VACUUM mais rápido e remove a necessidade do VACUUM FULL ou REINDEX. Deve ser um pouco maior que o total de número páginas de dados que serão tocados por atualizações e remoções entre vacuums. Os dois modos de determinar este número são rodar o VACUUM VERBOSE ANALYZE, ou se estiver utilizando autovacuum (veja abaixo) configure este de acordo com o parâmetro -V como uma porcentagem do total de páginas de dados utilizado por seu banco de dados. fsm_pages requer muito pouco memória, então é melhor ser generoso aqui.

vacuum_cost_delay:

Se você tiver tabelas grandes e um significativo montante de atividades de gravações concorrentes, você deve querer fazer uso deste novo recurso que diminui a carga de I/O do VACUUM sobre o custo de fazê-las mais longas. Como este é um novo recurso, é um complexo de 5 configurações dependentes para o qual nós temos apenas poucos testes de performance. Aumentando o vacuum_cost_delay para um valor não zero ativa este recurso; use um atraso razoável, algo entre 50 e 200ms. Para um ajuste fino, aumente o vacuum_cost_page_hit e diminua o vacuum_cost_page_limit irá diminuir o impacto dos vacuums e tornará eles mais longos; em testes de Jan Wieck's num teste de processamento de transações, um delay de 200, page_hit de 6 e limit de 100 diminuiu o impacto do vacuum em mais de 80% enquanto triplicou o tempo de execução dele.

Planejador de Consultas

Estas configurações permitem o planejador de consultas fazer estimativas mais precisas dos custos de operação e assim escolher o melhor plano de execução. Os dois valores de configurações para se preocupar são:

effective_cache_size:

Diz ao planejador de consultas o mais largo objeto do banco de dados que pode se esperar ser cacheado. Geralmente ele deve ser configurado em cerca de 2/3 da RAM, se estiver num servidor dedicado. Num servidor de uso misto, você deve estimar quanto de RAM e cache de disco outras aplicações estarão utilizando e subtrair eles.

random_page_cost:

Uma variável que estima o custo médio em buscas por páginas de dados indexados. Em máquinas mais rápidas, com arranjos de discos velozes ele deve ser reduzido para 3.0, 2.5 ou até mesmo 2.0. Contudo, se a porção ativa do seu banco de dados é muitas vezes maior que a sua RAM, você vai querer aumentar o fator de volta para o valor padrão de 4.0. Alternativamente, você pode basear seus ajustes na performance. Se o planejador injustamente a favor de buscas sequenciais sobre buscas em índices, diminua-o. Se ele estiver utilizando índices lentos quando não deveria, aumente-o. Tenha certeza de testar uma variedade de consultas. Não abaixe ele para menos de 2.0; se isto parecer necessário, você precisa de ajustes em outras áreas, como as estatísticas do planejador.

Logging

log_destination:

Isto substitui o intuitivo a configuração syslog em versões anteriores. Suas escolhas são usar o log

administrativo do SO (syslog ou eventlog) ou usar um log separado para o PostgreSQL (stderr). O primeiro é melhor para monitorar o sistema; o último é melhor para encontrar problemas no banco de dados e para o tuning.

redirect_stderr:

Se você decidir usar um log separado para o PostgreSQL, esta configuração permitirá registrar num arquivo utilizando uma ferramenta nativa do PostgreSQL ao invés do redirecionamento em linha de comando, permitindo a rotação do log. Ajuste para True, e então ajuste o log_directory para dizer onde colocar os logs. A configuração padrão para o log_filename, log_rotation_size e log_rotation_age são bons para a maioria das pessoas.

Autovacuum e você

Assim que você entra em produção no 8.0, você vai querer fazer um plano de manutenção incluindo VACUUMs e ANALYZEs. Se seus bancos de dados envolvem um fluxo contínuo de escrita de dados, mas não requer a maciças cargas e apagamentos de dados ou frequentes reinícios, isto significa que você deve configurar o pg_autovacuum. Isto é melhor que agendar vacuums porque:

- Tabelas sofrem o vacuum baseados nas suas atividades, excluindo tabelas que apenas sofrem leituras.
- A frequência dos vacuums cresce automaticamente com o crescimento da atividade no banco de dados.
- É mais fácil calcular o mapa de espaço livre e evitar o inchaço do banco de dados.

Carga de dados no banco: <http://pgdocptbr.sourceforge.net/pg80/populate.html>

Dicas de Performance em aplicações com PostgreSQL

Tradução do texto original de Josh Berkus

O que se segue é a versão editada de um conjunto de conselhos que eu tenho dado ao time da Sun no redesenho de uma aplicação em C++ que foi construída para MySQL, portado para o PostgreSQL, e nunca otimizado para performance. Ocorreu que estes conselhos podem ser geralmente úteis para a comunidade, então aí vão eles.

Projeto de aplicações para performance no PostgreSQL

Escrevendo regras de consultas

Para todos os sistemas gerenciadores de bancos de dados (SGDBs), o tempo por rodada significativo. Este é o tempo que leva para uma consulta passar pelo analisador de sintaxe da linguagem, o driver, a interface da rede, o analisador de sintaxe do banco de dados, o planejador, o executor, o analisador de sintaxe novamente, voltar pela interface de rede, passar pelo manipulador de dados do driver e para o cliente da aplicação. SGDBs variam na quantidade de tempo e CPU que elas levam para processar este ciclo, e por uma variedade de razões, o PostgreSQL possui um alto consumo de tempo e recursos do sistema por rodada.

Contudo, o PostgreSQL tem um overhead significativo por transação, incluindo o log de saída e as

regras de acesso que precisam ser ajustadas em cada transação. Enquanto você pode pensar que não está utilizando transações para um simples comando de leitura SELECT, de fato, cada simples comando no PostgreSQL é uma transação. Na ausência de uma transação explícita, o comando é por si mesmo implicitamente uma transação.

Passando por isto, o PostgreSQL é claramente o segundo depois do Oracle em processamento de consultas complexas e longas com transações com vários comandos com fácil resolução de conflitos de concorrência. Ele também suporta cursores, tanto rolável quanto não rolável.

Dica 1: Nunca use várias consultas pequenas quando uma grande consulta pode fazer o trabalho.

É comum em aplicações MySQL lidar com joins no código da aplicação, ou seja, consultando o ID de um registro relacionado e então iterando através dos registros filhos com aquele ID manualmente. Isto pode resultar em rodar centenas de consultas por tela de interface com o usuário. Cada uma destas consultas levam 2 a 6 milissegundos por rodada, o que significa que se você executar cerca de 1000 consultas, neste ponto você estará perdendo de 3 a 5 segundos. Comparativamente, solicitando estes registros numa única consulta levará apenas algumas centenas de milissegundos, economizando cerca de 80% do tempo.

Dica 2: Agrupe vários pequenos UPDATES, INSERTs ou DELETEs em um único comando ou, se não for possível, em uma longa transação.

Antes, a falta de subselects nas versões anteriores do MySQL fizeram com que os desenvolvedores de aplicação projetassem seus comandos de modificação de dados (DML) da mesma forma que as junções em middleware. Esta é uma má idéia para o PostgreSQL. Ao invés, você irá tirar vantagem de subselects e joins no seu comando UPDATE, INSERT, e DELETE para tentar realizar modificações em lote com um único comando. Isto reduz o tempo da rodada e o overhead da transação.

Em alguns casos, contudo, não há uma única consulta que consiga alterar todas as linhas que você deseja e você irá usar um grupo de comandos em série. Neste caso, você irá querer se assegurar de envolver a sua série de comandos DML em uma transação explícita (ex. BEGIN; UPDATE; UPDATE; UPDATE; COMMIT;). Isto reduz o overhead de transação e corta o tempo de execução em até 50%.

Dica 3: Considere realizar cargas em lotes ao invés de INSERTs seriais.

O PostgreSQL prove um mecanismo de carga em lote chamado COPY, que pode pegar uma entrada de um arquivo ou pipe delimitado por tabulações ou CSV. Quando o COPY pode ser usado no lugar de centenas de INSERTs, ele pode cortar o tempo de execução em até 75%.

Dica 4: O DELETE é caro

É comum para um desenvolvedor de aplicação pensar que o comando DELETE é praticamente não tem custo. Você está apenas desligando alguns nós, correto? Errado. SGDBs não são sistemas de arquivo; quando você apaga uma linha, índices precisam ser atualizados, o espaço liberado precisa ser limpo, fazendo a exclusão de fato mais cara que a inserção. Assim, aplicações que habitualmente apagam todas as linhas de detalhe e repõe elas com novas toda vez que é realizada qualquer alteração estão economizando esforço no lado da aplicação e empurrando este dentro do banco de dados. Quando possível, isto deve ser substituído pela substituição mais discriminada das linhas, como atualizar apenas as linhas modificadas.

Além disso, quando for limpar toda uma tabela, sempre use o comando TRUNCATE TABLE ao

invés de DELETE FROM TABLE. A primeira forma é até 100 vezes mais rápida que a posterior devido ao processamento da tabela como um todo ao invés de uma linha por vez.

Dica 5: Utilize o PREPARE/EXECUTE para iterações em consultas

Algumas vezes, mesmo tentando consolidar iterações de consultas semelhantes em um comando mais longo, nem sempre isto é possível de estruturar na sua aplicação. É para isto que o PREPARE ... EXECUTE serve; ele permite que o motor do banco de dados pule o analisador de sintaxe e o planejador para cada iteração da consulta. Por exemplo:

Preparar:

```
query_handle = query('SELECT * FROM TABLE WHERE id = ?')  
(parameter_type = INTEGER)
```

Então inicie as suas iterações:

```
for 1..100  
query_handle.execute(i);  
end
```

Classes para a preparação de comandos no C++ são explicadas na documentação do libpqxx.

Isto irá reduzir o tempo de execução na direta proporção do tamanho do número de iterações.

Dica 6: Use pool de conexões efetivamente

Para uma aplicação web, você irá perceber que até 50% do seu potencial de performance pode ser controlado através do uso, e configuração apropriada de um pool de conexões. Isto é porque criar e destruir conexões no banco de dados leva um tempo significativo no sistema, e um excesso de conexões inativas continuarão a requerer RAM e recursos do sistema.

Há um número de ferramentas que você pode utilizar para fazer um pool de conexões no PostgreSQL. Uma ferramenta de terceiros de código aberto é o pgPool. Contudo, para uma aplicação em C++ com requisitos de alta disponibilidade, é provavelmente melhor utilizar a técnica de pseudo-pooling nativa do libpqxx chamada de "conexões preguiçosas". Eu sugiro contatar a lista de e-mail para mais informações sobre como utilizar isto.

Com o PostgreSQL, você irá querer ter quantas conexões persistentes (ou objetos de conexão) forem definidas no seu pico normal de uso de conexões concorrentes. Então, se o uso máximo normal (no início da manhã, digamos) é de 200 conexões concorrentes de agentes, usuários e componentes, então você irá querer que ter esta quantidade definida para que sua aplicação não tenha que esperar por novas conexões durante o pico onde será lenta na criação.

Fábio Telles em:

<http://www.midstorm.org/~telles/2007/01/05/dicas-de-performance-em-aplicacoes-com-postgresql/>

Veja Melhorando a Performance do seu HD:

Piter Punk em: <http://piterpunk.info02.com.br/artigos/hdparm.html>

Gerenciando Recursos do Kernel:

<http://www.postgresql.org/docs/8.3/interactive/kernel-resources.html>

Escolhendo o sistema Operacional para o servidor do SGBD PostgreSQL

Sem discussão, o Linux/Unix ganha 25% a 50% de performance em cima do Windows. Isso se dá ao melhor gerenciamento de recursos.

<http://vivaolinux.com.br/dicas/verDica.php?codigo=9847>

Melhorando a performance do PostgreSQL com o comando VACUUM em:

http://imasters.uol.com.br/artigo/2421/postgresql/melhorando_a_performance_do_postgresql_com_o_comando_vacuum/

Otimizando bancos PostgreSQL - Parte 01

Olá! Neste meu primeiro artigo gostaria de descrever alguns ajustes finos do PostgreSQL que ajudam a melhorar a performance geral do banco de dados. Estas configurações são válidas para a versão 8.1. Caso você possua uma versão anterior, recomendo fortemente a atualização para a **8.1**, pois esta última é **muito** mais rápida que as anteriores, mesmo utilizando as configurações padrão. Infelizmente, a maior parte da degradação de performance de um banco de dados está na estrutura e/ou nos comandos SELECT mal elaborados. Neste artigo, será feita uma abordagem única e específica nos ajustes de configuração do SGBD, fazendo-o usufruir do máximo dos recursos de hardware onde está instalado o serviço do PostgreSQL.

É importante salientar também que não existe fórmula mágica para as configurações, sendo que as opções de um servidor pode não ser a melhor opção para outro.

Edição do arquivo postgresql.conf

Para começar, localize o arquivo chamado **postgresql.conf**. Este arquivo encontra-se no diretório de dados do cluster (ou agrupamento de bancos de dados) o qual você está inicializando. Em instalações normais, você pode encontrá-lo em:

- Microsoft Windows:

- Pasta "Arquivos de Configuração" no Menu Iniciar/Programas
- ou
- C:\Arquivos de Programas\PostgreSQL\8.1\data\

- Linux (Red Hat e Fedore)

/var/lib/pgsql/data

Após encontrá-lo, certifique-se de criar uma cópia de backup do arquivo antes de alterá-lo, pois erros na configuração podem fazer com que o PostgreSQL não seja inicializado.

Feito o backup, abra o arquivo em modo de edição. Se estiver no Linux, certifique-se de estar logado com o usuário **root** ou **postgres**.

Os parâmetros devem ser preenchidos seguindo o padrão **nome_do_parametro = valor**.

Lembrando que para valores do tipo data e texto deve-se envolver o valor com aspas simples, e para os valores numéricos não inserir separador de milhar. Não esqueça de remover o caracter cerquilha "#" do início das opções que deseja ativar. Todas as alterações serão efetivadas após a reinicialização do serviço PostgreSQL.

shared_buffers

Define o espaço de memória alocado para o PostgreSQL armazenar as consultas SQL antes de devolvê-las ao buffer do sistema operacional. Esta opção pode solicitar que parâmetros do Kernel sejam modificados para liberar mais memória compartilhada do sistema operacional, pois esta passa a ser utilizada também pelo Postgre, em maior quantidade. O valor desta configuração está expresso em blocos de 8 Kbytes (128 representa 1.024 Kbytes ou 1 Mb).

Uma boa pedida é utilizar valores de 8% a 12% do total de RAM do servidor para esta configuração. Caso, após mudar o valor deste parâmetro, o PostgreSQL não inicializar o cluster em questão, altere o sistema operacional para liberar mais memória compartilhada. Consulte o manual ou equivalentes do seu sistema operacional para obter instruções de como aumentar a memória compartilhada (Shared Memory) disponível para os programas.

Exemplo:

```
shared_buffers = 2048 # Seta a memória compartilhada para 16 Mbytes
```

work_mem

Configura o espaço reservado de memória para operações de ordem e manipulação/consulta de índices. Este parâmetro configura o tamanho em KBytes utilizado no servidor para cada conexão efetivada ao SGBD, portanto esteja ciente que o espaço total da RAM utilizado (valor da opção multiplicado pelo número de conexões simultâneas) não deve ultrapassar 20% do total disponível (valor aproximado).

Exemplo:

```
work_mem = 2048 # Configura 2 Mbytes de RAM do servidor para operações de ORDER BY, CREATE INDEX e JOIN disponíveis para cada conexão ao banco.
```

maintenance_work_mem

Expressa em KBytes o valor de memória reservado para operações de manutenção (como VACUUM e COPY). Se o seu processo de VACUUM está muito custoso, tente aumentar o valor deste parâmetro.

Nota: O total de memória configurada neste parâmetro é utilizado somente durante as operações de manutenção do banco de dados, sendo liberada durante o seu uso normal.

Exemplo:

```
maintenance_work_mem = 16384 # 16 Mbytes reservados para operações de manutenção.
```

max_fsm_pages

Em bancos de dados grandes, é ideal que a cada execução do VACUUM mais páginas "suja" sejam removidas do banco de dados do que a quantidade padrão, principalmente por questões de espaço em disco, e claro, performance. Porém, a configuração padrão traz apenas 20000 páginas. Em bancos transacionais, com no mínimo 10 usuários, o número mensal de páginas sujas pode exceder este valor.

Para realizar uma limpeza maior, aumente o valor desta configuração. Nota: incrementando o valor deste parâmetro pode resultar em aumento do tempo para execução do VACUUM, e cada página ocupa em média 6 bytes de RAM constantemente. O comando VACUUM pode ser comparado ao comando PACK do DBASE (DBFs), porém possui mais funcionalidades.

Exemplo:

```
max_fsm_pages = 120000 # Realiza a procura por até 120.000 páginas sujas na limpeza pelo VACUUM utilizando cerca de 71 Kb de RAM para isto.
```

wal_buffers

Número de buffers utilizados pelo WAL (Write Ahead Log). O WAL garante que os registros sejam gravados em LOG para possível recuperação antes de fechar uma transação. Porém, para bancos transacionais com muitas operações de escrita, o valor padrão pode diminuir a performance. Entretanto, é importante ressaltar que aumentar muito o valor deste parâmetro pode resultar em perda de dados durante uma possível queda de energia, pois os dados mantêm-se . Cada unidade representa 8 Kbytes de uso na RAM. Valores ideais estão entre 32 e 64. Se o disco conjunto do servidor (HW & SW) são quase infalíveis, tente usar 128 ou 256.

Exemplo:

`wal_buffers = 64` # Seta para 512 Kbytes a memória destinada ao buffer de escrita no WAL.

effective_cache_size

Esta configuração dita o quanto de memória RAM será utilizada para cache efetivo (ou cache de dados) do banco de dados. Na prática, é esta a configuração que dá mais fôlego ao SGBD, evitando a constante leitura das tabelas e índices ,dos arquivos do disco rígido.

Como exemplo, se uma tabela do banco de dados possui 20 Mbytes, e o tamanho para esta configuração limita-se aos quase 8 Mbytes padrão, o otimizador de queries irá carregar a tabela por etapas, em partes, até que ela toda seja vasculhada em busca dos registros. Esta opção impacta diretamente aumentando a performance do banco de dados, principalmente quando há concorrência, pois diminui consideravelmente as operações de I/O de disco.

Em consultas pela internet, encontrei várias referências para utilizar no máximo 25% da RAM total. Porém, se o servidor for dedicado, ou dispôr de uma grande quantidade de memória (512 Mbytes ou mais), recomendo o uso de 50% da RAM para esta configuração. Cada unidade corresponde a 8 Kbytes de RAM.

Exemplo:

`effective_cache_size = 32768` # Seta o cache de dados do PostgreSQL para 256 Mbytes de RAM.

random_page_cost

Define o custo (em tempo) para a seleção do plano de acesso aos dados do banco. Se você possui discos rígidos velozes (SCSI, por exemplo), tente utilizar valores como 1 ou 2 para esta configuração. Para os demais casos, limite-se a 3 ou 4. Isto impacta no plano estabelecido pelo otimizador interno, que pode realizar um Index_Scan ou Table_Scan, etc, conforme aquilo que ele determinar ser mais otimizado.

Exemplo:

`random_page_cost = 2` # Diminui o tempo para seleção aleatória de páginas do otimizador de consultas.

Notas gerais

Caso os valores inseridos possuam algum erro, é possível que o PostgreSQL não consiga inicializar o agrupamento de banco de dados no qual o arquivo postgresql.conf foi modificado. Se isto ocorrer, retorne o backup do mesmo e certifique-se de que as modificações estão corretamente setadas.

É possível obter ganhos de performance significativos com a correta seleção dos valores para as configurações acima. Tente várias combinações dependendo da capacidade e do uso do servidor

(hardware) onde está instalado o PostgreSQL 8.1.

Otimizando bancos PostgreSQL - Parte 02

Criação de Índices Parciais (Partial Indexes)

Em tabelas com muitos registros, a utilização de índices normais pode causar um desempenho insatisfatório, principalmente quando se trata de colunas que representam abstrações de dados com pouca variação. É o caso de colunas representando tempo (DATE, TIME e TIMESTAMP), ou colunas numéricas representando tipos pré-definidos (Ex.: Regiões, Sexo, Faixa Salarial, etc.). Tabelas de movimentação analítica com estes tipos de colunas podem conter milhares, ou até milhões de registros. Entretanto, em consultas SQL específicas por um determinado valor, um índice normal completo (Full Index) irá considerar na consulta todos os registros da tabela, organizados na ordem do índice.

O PostgreSQL possui um fantástico recurso para criação de índices que permite delimitar os registros que este irá considerar. Isto representa um enorme ganho de desempenho, especialmente em consultas SQL que utilizam filtros complexos no WHERE.

Vamos exemplificar este caso. Considere uma tabela de movimentação analítica de estoque de uma empresa de comércio comum. Vamos usar um modelo simples, apenas para demonstrar o caso. Utilize o código SQL abaixo para criar a tabela:

```
-- Criação da Tabela
CREATE TABLE estoque(
  ID_Empresa INT2 NOT NULL,
  ID_Produto INT4 NOT NULL,
  ID_Local_Estoque INT2 NOT NULL,
  TIPO_Entrada BOOLEAN NOT NULL DEFAULT false,
  QTD_Quantidade NUMERIC(12,6) DEFAULT 0.000000,
  VAL_Unitario NUMERIC(15,3) DEFAULT 0.000,
  DT_Movimento DATE NOT NULL
);

-- Definição da chave primária
ALTER TABLE estoque ADD PRIMARY KEY(ID_Empresa, ID_Produto, ID_Local_Estoque);

-- Criação de Índice sobre o campo Data
CREATE INDEX idx_DATA ON estoque (DT_Movimento, ID_Empresa);
```

Imagine esta tabela com mais de 2.000.000 registros. O departamento de gerência de estoque emite relatórios mensais sobre a movimentação de estoque dos produtos para conferência. Um exemplo de relatório é de Entrada e Saída Consolidada, que considera os valores de entrada e saída por período. Um SQL típico para demonstrar as informações do mês de Dezembro de 2006 utilizaria o filtro no WHERE mencionando o campo DT_Movimento da seguinte forma: (...) WHERE DT_Movimento BETWEEN 2006-12-01 AND 2006-12-31.

Considere o volume de dados caso a empresa possua um movimento de mais de 50.000 registros por mês, mantendo esta marca desde 01/01/2000. Ao utilizar o WHERE acima, uma varredura completa no índice idx_DATA seria feita, considerando a massa completa de dados no índice.

Dependendo de condições de uso dos registros estes podem estar na memória cache, então o resultado seria rapidamente apresentado. Entretanto, caso uma pesquisa aleatória não armazenada em cache for executada, o custo de IO do gerenciador de banco de dados seria problemático.

A solução neste caso - e uma medida muito satisfatória - é a criação dos índices parciais sobre o campo data, combinando-os com um índice normal completo. É possível criar os índices parciais para datas muito além das atuais, para prever a população de registros na tabela no futuro, de modo a garantir o desempenho. Lembre-se de que se não existirem registros com uma data prevista no índice, este não terá tamanho, portanto não será prejudicial em nenhum aspecto (espaço ou IO).

Para aperfeiçoar o acesso a dados nestas condições, os índices parciais consideram a cláusula SQL WHERE:

```
-- Criação de Índice sobre o campo Data - Janeiro de 2006
CREATE INDEX idx_DATA_0106 ON estoque (DT_Movimento, ID_Empresa) WHERE
(DT_Movimento BETWEEN 2006-01-01 AND 2006-01-31);

-- Criação de Índice sobre o campo Data - Fevereiro de 2006
CREATE INDEX idx_DATA_0206 ON estoque (DT_Movimento, ID_Empresa) WHERE
(DT_Movimento BETWEEN 2006-02-01 AND 2006-02-28);

(...)

-- Criação de Índice sobre o campo Data - Dezembro de 2006
CREATE INDEX idx_DATA_1206 ON estoque (DT_Movimento, ID_Empresa) WHERE
(DT_Movimento BETWEEN 2006-12-01 AND 2006-12-31);

(...)
```

Desta forma, a todo SQL onde for utilizado a condição DT_Movimento BETWEEN 2006-12-01 AND 2006-12-31 ou sua equivalente DT_Movimento 2006-12-01 AND DT_Movimento 2006-12-31, o índice idx_DATA_0106 será apresentado para o otimizador interno como o mais eficaz, e portanto será usado.

Uma aplicação muito boa para os índices parciais é a utilização deste em tabelas que fazem parte de VIEWS (Visões) complexas. Todo o WHERE fixo da VIEW pode ser considerado em um índice parcial, o que resulta na diminuição considerável do tempo de resposta.

Tiago Adami em:

http://imasters.uol.com.br/artigo/4406/postgresql/otimizando_bancos_postgresql_-_parte_01/
http://imasters.uol.com.br/artigo/5328/postgresql/otimizando_bancos_postgresql_-_parte_02/

Otimizando operações de Entrada e Saída (I/O) em:

http://imasters.uol.com.br/artigo/4020/bancodedados/otimizando_operacoes_de_entrada_e_saida_io

Otimizar consultas SQL

Diferentes formas de otimizar as consultas realizadas em SQL.

A linguagem SQL é não procedimental, ou seja, nas sentenças se indica o que queremos conseguir e não como tem que fazer o intérprete para consegui-lo. Isto é pura teoria, pois na prática todos os gerenciadores de SQL têm que especificar seus próprios truques para otimizar o rendimento.

Portanto, muitas vezes não basta com especificar uma sentença SQL correta, e sim que além disso,

há que indicar como tem que fazer se quisermos que o tempo de resposta seja o mínimo. Nesta seção, veremos como melhorar o tempo de resposta de nosso intérprete ante umas determinadas situações:

Design de tabelas

- * Normalize as tabelas, pelo menos até a terceira forma normal, para garantir que não haja duplicidade de dados e aproveitar o máximo de armazenamento nas tabelas. Se tiver que desnormalizar alguma tabela pense na ocupação e no rendimento antes de proceder.

- * Os primeiros campos de cada tabela devem ser aqueles campos requeridos e dentro dos requeridos primeiro se definem os de longitude fixa e depois os de longitude variável.

- * Ajuste ao máximo o tamanho dos campos para não desperdiçar espaço.

- * É normal deixar um campo de texto para observações nas tabelas. Se este campo for utilizado com pouca frequência ou se for definido com grande tamanho, por via das dúvidas, é melhor criar uma nova tabela que contenha a chave primária da primeira e o campo para observações.

Gerenciamento e escolha dos índices

Os índices são campos escolhidos arbitrariamente pelo construtor do banco de dados que permitem a busca a partir de tal campo a uma velocidade notavelmente superior. Entretanto, esta vantagem se vê contra-arrestada pelo fato de ocupar muito mais memória (o dobro mais ou menos) e de requerer para sua inserção e atualização um tempo de processo superior.

Evidentemente, não podemos indexar todos os campos de uma tabela extensa já que dobramos o tamanho do banco de dados. Igualmente, tampouco serve muito indexar todos os campos em uma tabela pequena já que as seleções podem se efetuar rapidamente de qualquer forma.

Um caso em que os índices podem ser muito úteis é quando realizamos petições simultâneas sobre várias tabelas. Neste caso, o processo de seleção pode se acelerar sensivelmente se indexamos os campos que servem denexo entre as duas tabelas.

Os índices podem ser contraproducentes se os introduzimos sobre campos triviais a partir dos quais não se realiza nenhum tipo de petição já que, além do problema de memória já mencionado, estamos lentificando outras tarefas do banco de dados como são a edição, inserção e eliminação. É por isso que vale a pena pensar duas vezes antes de indexar um campo que não serve de critério para buscas ou que é usado com muita frequência por razões de manutenção.

Campos a Selecionar

- * Na medida do possível há que evitar que as sentenças SQL estejam embebidas dentro do código da aplicação. É muito mais eficaz usar vistas ou procedimentos armazenados por que o gerenciador os salva compilados. Se se trata de uma sentença embebida o gerenciador deve compila-la antes de executa-la.

- * Selecionar exclusivamente aqueles que se necessitem

- * Não utilizar nunca SELECT * porque o gerenciador deve ler primeiro a estrutura da tabela antes de executar a sentença

- * Se utilizar várias tabelas na consulta, especifique sempre a que tabela pertence cada campo, isso economizará tempo ao gerenciador de localizar a que tabela pertence o campo. Ao invés de

SELECT Nome, Fatura FROM Clientes, Faturamento WHERE IdCliente = IdClienteFaturado, use:
SELECT Clientes.Nome, Faturamento.Fatura WHERE Clientes.IdCliente =
Faturamento.IdClienteFaturado.

Campos de Filtro

* Procuraremos escolher na cláusula WHERE aqueles campos que fazem parte da chave do arquivo pelo qual interrogamos. Ademais se especificarão na mesma ordem na qual estiverem definidas na chave.

* Interrogar sempre por campos que sejam chave.

* Se desejarmos interrogar por campos pertencentes a índices compostos é melhor utilizar todos os campos de todos os índices. Suponhamos que temos um índice formado pelo campo NOME e o campo SOBRENOME e outro índice formado pelo campo IDADE. A sentença WHERE NOME='Jose' AND SOBRENOME Like '%' AND IDADE = 20 seria melhor que WHERE NOME = 'Jose' AND IDADE = 20 porque o gerenciador, neste segundo caso, não pode usar o primeiro índice e ambas sentenças são equivalentes porque a condição SOBRENOME Like '%' devolveria todos os registros.

Ordem das Tabelas

Quando se utilizam várias tabelas dentro da consulta há que ter cuidado com a ordem empregada na cláusula FROM. Se desejarmos saber quantos alunos se matricularam no ano 1996 e escrevermos:

```
FROM Alunos, Matriculas WHERE Aluno.IdAluno = Matriculas.IdAluno AND Matriculas.Ano = 1996
```

o gerenciador percorrerá todos os alunos para buscar suas matrículas e devolver as correspondentes. Se escrevermos

```
FROM Matriculas, Alunos WHERE Matriculas.Ano = 1996 AND Matriculas.IdAluno = Alunos.IdAlunos
```

o gerenciador filtra as matrículas e depois seleciona os alunos, desta forma tem que percorrer menos registros.

De Cláudio em:

<http://www.criarweb.com/artigos/otimizar-consultas-sql.html>

Dicas de Desempenho em:

<http://pgdocptbr.sourceforge.net/pg80/performance-tips.html>

Diferença de Desempenho entre alguns Sistemas Operacionais

Fiz alguns teste com mesma máquina e configuração com windows 2000 server, Debian e FreeBSD 6.2, perdi um tempão instalado os tres SO (testei um de cada vez seperado) na máquina todos com configuração minima para não dizer que o debian e o FreeBSD pode ser instalado sem interface ai

consumiria menos memória instalei o debian e o FreeBSD com KDE para ficar semelhante ao windows.

Uma consulta pesada que demorava 30s no Free e uns 33s no debian demora 1,5min em windows a diferença é grande.

Leandro DUTRA na lista pebr-geral

Melhor performance em instruções SQL

Artigo de Mauro Pichiliani no iMasters:

http://imasters.uol.com.br/artigo/222/sql_server/melhor_performance_em_instrucoes_sql/

Oi pessoal, nesta primeira coluna vamos falar um pouco sobre melhora de performance em instruções SQL. Para começar podemos entender instruções SQL por comandos do tipo SELECT, UPDATE e DELETE.

Cada comando possui uma série de detalhes, e no SQL Server, estes detalhes podem impactar muito na performance. No artigo desta semana, vou falar um pouco sobre talvez o mais utilizado deles: o SELECT.

Primeiro vamos à sintaxe básica do SELECT

```
SELECT <lista_de_campos> [ FROM <lista_de_tabelas> [ WHERE <lista_de_condições>]]  
[ GROUP BY <lista_de_campos> ]
```

No lugar de <lista_de_campos> devemos colocar uma expressão que seja passível de entendimento ao SQL Server. Por exemplo:

```
SELECT GETDATE() – Retorna a data atual do sistema
```

E na lista de tabelas os campos da tabela. Exemplo:

```
SELECT CAMPO FROM TABELA1
```

Com isso , já temos uma dica de performance: procura NÃO utilizar o ‘*’ para retornar todos os campos , pois isso traz para a estação cliente dados as vezes desnecessários.Exemplo:

```
SELECT * FROM TABELA1 - Procure não fazer isso.... e sim:
```

```
SELECT CAMPO1 , CAMPO2 , CAMPO3 FROM TABELA1
```

Bom , quanto à lista de filtros (cláusula where) devemos tomar muito cuidado , pois é ai que geralmente temos problemas ou perda de performance. Para começar , evite utilizar os operadores IN() e subquery's , pois eles oneram a performance do banco.Evite também instruções muito grandes. Procure quebrá-las em várias instruções e ligue os resultados com o comando UNION.

Cuidado com os operadores lógicos AND na cláusula WHERE pois para cada AND a mais que é colocado , todo conjunto de dados que será retornado tem que ser filtrado. Isto consome muito processamento as vezes desnecessário.

Sempre que possível procure utilizar a cláusula TOP n para indicar qual a quantidade de registro. Saber de antemão quantos registros a query tem que retornar ajuda o SQL Server a fazer um plano de execução da instrução menos e isso diminui o tempo de resposta.Por exemplo , se quisermos somente os 5 primeiros registros que atendem a uma condição:

```
SELECT TOP 5 FLAG1 FROM TABELA1 WHERE FLAG1 = 2
```

Não abuso muito do operador LIKE. Ele é ótimo para consultas, mais devemos procurar não colocar % antes e depois:

- Se houver como, evite isto

```
SELECT NOME FROM TABELA1 WHERE NOME LIKE '%A%'
```

Outra dica interessante é não colocar muitos campos na cláusula ORDER BY (ordem da consulta), pois para cada campo adicional, temos uma re-ordenação interna do conjunto de dados retornado. Por exemplo :

```
SELECT CAMPO1 , CAMPO2 , CAMPO3 FROM TABELA1  
ORDER BY CAMPO1 , CAMPO2 , CAMPO3
```

procure usar (quando possível):

```
SELECT CAMPO1 , CAMPO2 , CAMPO3 FROM TABELA1  
ORDER BY CAMPO1
```

Um ponto a ser levantado é o uso de joins. A ordem que se relaciona as tabelas na instrução SELECT não importa. Isto por que internamente o SQL Server vai escolher a ordem de acesso às tabelas. Por exemplo:

```
SELECT A.CAMPO1 , B.CAMPO2 FROM TABELA1 A , TABELA2 B  
WHERE  
A.COD = B.COD
```

Tem um performance igual se a tabela A possuir mais registros, mesmo que não satisfaçam a condição do join, do que a ordem inversa :

```
SELECT A.CAMPO1 , B.CAMPO2 FROM TABELA1 A , TABELA2 B  
WHERE  
B.COD = A.COD
```

Lembrando que as duas instruções anteriores retornam SEMPRE o mesmo resultado.

Otimização do PostgreSQL - Introdução em:

http://www.sqlmagazine.com.br/Artigos/Postgre/02_Otimizacao.asp

PostgreSQL Leopardo em:

http://www.midstorm.org/~telles/uploads/postgresql_leopardo_conisli_2007.odp

Introdução ao Tuning do SGBD PostgreSQL em:

<http://www.lozano.eti.br/palestras/tuning-pgsql.pdf>

Tutorial completo sobre RAID 0, RAID 1, RAID 0+1 e RAID 5 em:

<http://www.guiadohardware.net/comunidade/raid-tutorial/665151/>

11) Entendendo o WAL (Write Ahead Log) e o PITR

O *registro prévio da escrita* (WAL = *write ahead logging*) é uma abordagem padrão para registrar transações. A descrição detalhada pode ser encontrada na maioria (se não em todos) os livros sobre processamento de transação. Em poucas palavras, o conceito central do WAL é que as alterações nos arquivos de dados (onde as tabelas e os índices residem) devem ser escritas somente após estas alterações terem sido registradas, ou seja, quando os registros que descrevem as alterações tiverem sido descarregados em um meio de armazenamento permanente. Se este procedimento for seguido, não será necessário descarregar as páginas de dados no disco a cada efetivação de transação, porque se sabe que no evento de uma queda será possível recuperar o banco de dados utilizando o registro: todas as alterações que não foram aplicadas às páginas de dados são refeitas a partir dos registros (isto é a recuperação de rolar para a frente, *roll-forward*, também conhecida como *REDO*),

Benefícios do WAL

O primeiro grande benefício da utilização do WAL é a redução significativa do número de escritas em disco, uma vez que na hora em que a transação é efetivada somente precisa ser descarregado em disco o arquivo de registro, em vez de todos os arquivos de dados modificados pela transação. Em ambiente multiusuário, a efetivação de várias transações pode ser feita através de um único `fsync()` do arquivo de registro. Além disso, o arquivo de registro é escrito sequencialmente e, portanto, o custo de sincronizar o registro é muito menor do que o custo de descarregar as páginas de dados. Isto é especialmente verdade em servidores tratando muitas transações pequenas afetando partes diferentes do armazenamento de dados.

O benefício seguinte é a consistência das páginas de dados. A verdade é que antes do WAL o PostgreSQL nunca foi capaz de garantir a consistência no caso de uma queda. Antes do WAL, qualquer queda durante a escrita poderia resultar em:

1. linhas de índice apontando para linhas inexistentes da tabela
2. perda de linhas de índice nas operações de quebra de página (*split*)
3. conteúdo da página da tabela ou do índice totalmente danificado, por causa das páginas de dados parcialmente escritas

Os problemas com os índices (problemas 1 e 2) possivelmente poderiam ter sido resolvidos através de chamadas adicionais à função `fsync()`, mas não é óbvio como tratar o último caso sem o WAL; se for necessário, o WAL salva todo o conteúdo da página de dados no registro, para garantir a consistência da página na recuperação após a queda.

Por fim, o WAL permite que seja feita cópia de segurança em linha e recuperação para um ponto no tempo, conforme descrito na [Seção 22.3](#). Fazendo cópia dos arquivos de segmento do WAL pode-se retornar para qualquer instante no tempo coberto pelos registros do WAL: simplesmente se instala uma versão anterior da cópia de segurança física do banco de dados, e se refaz o WAL até o ponto desejado no tempo. Além disso, a cópia de segurança física não precisa ser um instantâneo do estado do banco de dados — se a cópia for realizada durante um período de tempo, quando o WAL for refeito para este período de tempo da cópia serão corrigidas todas as inconsistências internas.

Detalhes em: <http://pgdocptbr.sourceforge.net/pg80/wal.html>

Internamente

O WAL é ativado automaticamente; não é requerida nenhuma ação por parte do administrador, exceto garantir que o espaço em disco adicional necessário para o WAL seja atendido, e que seja feito qualquer ajuste necessário (consulte a [Seção 25.2](#)).

O WAL é armazenado no diretório `pg_xlog`, sob o diretório de dados, como um conjunto de arquivos de segmento, normalmente com o tamanho de 16 MB cada. Cada segmento é dividido em páginas, normalmente de 8 kB cada. Os cabeçalhos dos registros estão descritos em `access/xlog.h`; o conteúdo do registro depende do tipo de evento que está sendo registrado. São atribuídos para nomes dos arquivos de segmento números que sempre aumentam, começando por `000000010000000000000000`. Atualmente os números não recomeçam, mas deve demorar muito tempo até que seja exaurido o estoque de números disponíveis.

Os *buffers* do WAL e estruturas de controle ficam na memória compartilhada e são tratados pelos processos servidor filhos; são protegidos por bloqueios de peso leve. A demanda por memória compartilhada é dependente do número de *buffers*. O tamanho padrão dos *buffers* do WAL é 8 *buffers* de 8 kB cada um, ou um total de 64 kB.

É vantajoso o WAL ficar localizado em um disco diferente do que ficam os arquivos de banco de dados principais. Isto pode ser obtido movendo o diretório `pg_xlog` para outro local (enquanto o servidor estiver parado, é óbvio), e criando um vínculo simbólico do local original no diretório de dados principal para o novo local.

A finalidade do WAL, garantir que a alteração seja registrada antes que as linhas do banco de dados sejam alteradas, pode ser subvertida pelos controladores de disco (*drives*) que informam ao núcleo uma escrita bem-sucedida falsa, e na verdade apenas colocam os dados no *cache* sem armazenar no disco. Numa situação como esta a queda de energia pode conduzir a uma corrupção dos dados não recuperável. Os administradores devem tentar garantir que os discos que armazenam os arquivos de segmento do WAL do PostgreSQL não fazem estes falsos relatos.

Após um ponto de verificação ter sido feito e o registro descarregado, a posição do ponto de verificação é salva no arquivo `pg_control`. Portanto, quando uma recuperação vai ser feita o servidor lê primeiro `pg_control`, e depois o registro de ponto de verificação; em seguida realiza a operação de REDO varrendo para frente a partir da posição indicada pelo registro de ponto de verificação. Como, após o ponto de verificação, na primeira modificação feita em uma página de dados é salvo todo o conteúdo desta página, todas as páginas modificadas desde o último ponto de verificação serão restauradas para um estado consistente.

Para tratar o caso em que o arquivo `pg_control` foi danificado, é necessário haver suporte para a possibilidade de varrer os arquivos de segmento do WAL em sentido contrário — mais novo para o mais antigo — para encontrar o último ponto de verificação. Isto ainda não foi implementado. O arquivo `pg_control` é pequeno o suficiente (menos que uma página de disco) para não estar sujeito a problemas de escrita parcial, e até o momento em que esta documentação foi escrita não haviam relatos de falhas do banco de dados devido unicamente a incapacidade de ler o arquivo `pg_control`. Portanto, embora este seja teoricamente um ponto fraco, na prática o arquivo `pg_control` não parece ser um problema.

PIRT

Esse documento e uma traducao de um capitulo de um livro no qual a fonte foi citada no final do documento. Vejo que muitas pessoas detem duvidas em relacao a Point-in-time recovery , essa e minha forma de contribuicao com a comunidade Postgresql. Desculpem pela falta de acentuacao no documento(traduzi o documento ontem as 11/04/2006 01:30 AM ,nao me preocupei muito com a estetica :P) por qualquer falha de traducao, acho que nao sao muitas e nao poe em risco o bom entendimento.

Point-in-time recovery

Quando voce faz uma mudanca em algum banco de dados do Postgresql, Postgresql grava suas mudancas no shared-buffer pool, o write ahead log(WAL) e eventualmente no arquivo que voce mudou. O WAL contem registros completos das mudancas que voce fez. O mecanismo de Point-in-time recovery usa um historico de modificacoes gravados nos arquivos WAL. Imagine o PITR como um esquema de backup incremental. Voce comeca com um backup completo e depois, arquiva as mudancas realizadas. Quando ocorrer um crash no seu server, voce restaura o backup completo(full backup) e aplica as mudancas, em sequencia, ate que se recupere todos os dados que desejas recuperar.

Point-in-time recovery(PITR) pode parecer muito intimidador se voce comeca a ler a documentacao. Para voce ter uma boa visao sobre o sistema de PITR, iremos criar uma nova base, da um crash nela e recuperar os dados usando o mecanismo de PITR. Voce pode seguir se quiser, mas voce ira precisar de um spaco em disco extra.

Iremos comecar criando um novo cluster.

```
$export PGDATA=/usr/local/pgPITR
initdb
```

The files belonging to this database system will be owned by user "pg".
This user must also own the server process.

...

Success. You can now start the database server using:

```
postmaster -D /usr/local/pgPITR/data
or
pg_ctl -D /usr/local/pgPITR/data -l logfile start
```

Agora iremos mudar o arquivo \$PGDATA/postgresql.conf para acionar o mecanismo de PITR. A unica mudanca que voce deve fazer e definir o parametro archive_command. O parametro archive_command diz ao Postgresql como os arquivos de WAL(write-ahead-log) irao ser gerados pelo servidor. Tomemos como exemplo a seguinte configuracao:

```
archive_command='cp %p /tmp/wals/%f '
```

Postgresql ira executar o archive_command ao inves de simplesmente deletar os arquivos WAL como ele normalmente faz. %p significa o caminho completo do WAL e o %f e o nome do arquivo.

Agora iremos criar o diretorio /tmp/wals, startar o processo postmaster e criar um banco de dados para que possamos trabalhar.

```
$mkdir /tmp/wals
$pg_ctl -l /tmp/pg.log start
$createdb teste
CREATE DATABASE
```

Nesse momento eu tenho um cluster completo , os dados relativos ao cluster estao em \$PGDATA, e um banco de dados chamado teste. Quando eu realizo mudancas no meu banco de dados teste, essas mudancas sao gravados nos arquivos WAL que estao em \$PGDATA/pg_xlog(como em qualquer outro cluster). Quando um arquivo WAL cresce, Postgresql ira copiar ele para o diretorio /tmp/wals para mante-los sao e salvos. A unica diferenca entre um cluster convencional e um cluster que esteja com o mecanismo de PITR acionado e que o servidor Postgresql ira arquivar os arquivos WAL ao inves de deleta-los.

Como o mecanismo de PITR trabalha com as mudancas efetuadas no banco de dados e gravadas nos arquivos WAL, iremos gerar alguns arquivos WAL criando algumas tabelas. Nao interessa o que dados irao

conter as tabelas, nos queremos somente gerar os arquivos WAL, ate que eles crescam(cada arquivo WAL contem 16 megas).

```
$psql
```

```
Welcome to psql 8.0.0, the PostgreSQL interactive terminal. ...
```

```
teste=# BEGIN WORK; /* aqui comecemos a nossa transacao*/
```

```
teste=# create table teste1 as select * from pg_class,pg_attribute;
```

```
SELECT
```

```
teste=# COMMIT; executed at 12:30:00 pm /*terminamos a nossa transacao*/
```

```
teste=#\q
```

O commando create table produz uma tabela que contem mais de 245000 registros e produz bastante arquivos WAL capas de chegar aos 16 megas cada um. Voce pode ver os segmentos WAL olhando o diretorio /tmp/wals

```
$ls /tmp/wals
```

```
00000001000000000000000000000000
```

```
00000001000000000000000000000001
```

```
00000001000000000000000000000002
```

```
00000001000000000000000000000003
```

```
00000001000000000000000000000004
```

Agora iremos criar um backup completo de todo os meus dados do meu cluster. Para simplificar o exemplo irei criar um arquivo .tar.gz e salva-lo no diretorio /tmp. Antes de eu comecar o meu backup, iremos dizer ao Postgresql o que estamos prestes a fazer chamando a funcao pg_start_backup()

```
$psql teste
```

```
Welcome to psql 8.0.0, the PostgreSQL interactive terminal. ...
```

```
teste=# select pg_start_backup('full bacup-Segunda');
```

```
pg_start_backup
```

```
-----
```

```
0/52EA2B8 (1 row)
```

```
teste=# \q
```

Agora iremos criar o backup dos nossos dados do cluster

```
tar czvf /tmp/pgdata.tar.gz $PGDATA
```

O argumento que demos ao funcao pg_start_backup() e simplesmente um rotulo que ajuda a lembrarmos de onde os arquivos vieram. Voce ire achar o seguinte arquivo \$PGDATA/backup_label depois de chamar a funcao, e o rotulo que passamos como parametro estara dentro desse arquivo.

Quando terminarmos de criar o nosso backup, no caso o nosso tarball, iremos dizer ao Postgresql que terminamos chamando a funcao pg_stop_backup()

```
$psql teste
```

```
Welcome to psql 8.0.0, the PostgreSQL interactive terminal. ...
```

```
teste=# select pg_stop_backup();
```

```
pg_stop_backup
```

```
-----
```

```
0/52EA2F4 (1 row)
```

```
teste=# \q
```

Nesse ponto temos um backup completo de todo o nosso cluster, Postgresql continuar rodando(nao paramos o processo postmaster), qualquer mudanca sera gravada nos arquivos WAL, e eles continuam a serem gravados no diretorio /tmp/wals.

****Note que temos um backup completo e no nosso backup temos somente 1 tabela criada.**

Iremos gerar mais arquivos WAL, irei criar outra tabela demonstrativa.

```
$ psql teste
```

```
Welcome to psql 8.0.0, the PostgreSQL interactive terminal.
```

```
...
```

```
teste=# BEGIN WORK;
```

```
BEGIN
```

```
teste=# CREATE TABLE teste2 AS SELECT * FROM pg_class, pg_attribute;
```

```
SELECT
```

```
teste=# COMMIT; executed at 12:38:00pm
```

Agora, so para as coisas ficarem mais interessantes criaremos uma terceira tabela e droparemos ela.

Esperamos que ela desapareca quando tentarmos recuperar os dados.

```
$ psql teste
Welcome to psql 8.0.0, the PostgreSQL interactive terminal.
...
teste=# BEGIN WORK;
BEGIN
test=# CREATE TABLE teste3 AS SELECT * FROM pg_class, pg_attribute;
SELECT
test=# COMMIT; executed at 12:38:00pm
teste=#BEGIN WORK;
BEGIN
teste=# DROP TABLE teste3;
DROP TABLE
teste=# COMMIT; executed at 12:40:00 pm
COMMIT
teste=#\q
```

Como esperavamos, Postgresql copiou varios arquivos WAL para o diretorio /tmp/wals

```
$ls /tmp/wals
00000001000000000000000000
00000001000000000000000001
00000001000000000000000002
00000001000000000000000003
00000001000000000000000004
00000001000000000000000005
00000001000000000000000005.002EA2B8.backup
00000001000000000000000006
00000001000000000000000007
00000001000000000000000008
00000001000000000000000009
0000000100000000000000000A
0000000100000000000000000B
0000000100000000000000000C
0000000100000000000000000D
0000000100000000000000000E
```

Nesse ponto acontece um disastre, a energia cai, um furacao inesperado, ou meu computador pega fogo(tudo acontece, arrastao, mensalao e tals menos o meu diretorio /tmp e destruido) para simular um disastre iremos da um kill no processo postmaster.

```
$KILL -9 (head -1 $PGDATA/postmaster.pid)
```

Agora e hora de recuperarmos todo o nosso cluster. Comecaremos por renomear o nosso cluster detonado.

```
$ mv $PGDATA $PGDATA.old
```

Agora iremos restorar o backup da nossa tarball

```
$ cp /tmp/pgdata.tar.gz $PGDATA
```

```
$ tar -xzf pgdata.tar.gz
```

agora temos o nosso \$PGDATA.old(que e o diretorio dos arquivos do nosso cluster detonado) e o backup do nosso cluster antigo no \$PGDATA. Iremos da uma limpada no cluster restaurado do backup antigo removendo arquivos WAL antigos e o aquivo postmaster.pid

```
$ rm -rf $PGDATA/pg_xlog/0*
```

```
$ rm -rf $PGDATA/postmaster.pid
```

Para garantir que posso recuperar a maior quantidade de dados possiveis irei copiar os arquivos WAL do cluster danificado dentro do diretorio do cluster restaurado

```
$cp %PGDATA.old/pg_xlog/0* $PGDATA/pg_xlog/
```

Se os arquivos do cluster danificado nao estiverem ao nosso alcance(podem ter queimado quando o computador pegou fogo), podemos recuperar todas as transacoes comitadas antes dos mais recentes arquivos WALs que foram criados. Em um banco de dados com muitas transacoes, perderiamos apenas alguns minutos, cerca de 16 megas.

Agora iremos comecar o processo de recuperacao , mas antes devemos nos lembrar do que aconteceu...

12:30:00pm criamos a tabela teste1 and commitamos as mudancas
 12:38:00pm criamos a tabela teste2 and commitamos as mudancas
 12:39:00pm criamos a tabela teste3 and commitamos as mudancas
 12:40:00pm dropamos a tabela teste3 and commitamos as mudancas

Algum tempo entre 12:30 e 12:38 nos realizamos o backup de todo o banco. (lembre-se que nosso backup fisico so continha a tabela teste1).

Quando iniciamos o processo de recuperacao, podemos recuperar todas as mudancas ou podemos dizer ao Postgresql para parar em certo ponto do tempo(Point in time). Se a recuperacao parar antes de 12:38, so teremos a tabela teste1, nao teremos a tabela teste2 nem a teste3. Se a recuperacao parar antes de 12:39, deveremos ter as tabelas teste1, teste3, mas nao teremos a teste2. Se a recuperacao parar antes de 12:40 teremos as 3 tabelas. Se deixarmos o Postgresql recuperar todas as mudancas a tabela teste3 deve desaparecer, porque dropamos ela.

para controlarmos o processo de recuperacao , criaremos um arquivo chamado \$PGDATA/recovery.conf que diz ao Postgresql como proceder. O arquivo recovery.conf se parece com isso:

```
$ cat $PGDATA/recovery.conf
```

```
restore_command= 'cp /tmp/wals/%f %p'
```

```
recovery_target_time='2005-06-22 12:39:01 EST'
```

O parametro restore_command diz ao postgresql como recuperar os arquivos WAL que estao armazenados em /tmp/wals. O parametro recovery_target_time diz ao Postgresql quando parar. Se voce quer recuperar todas as mudancas simplesmente omite esse parametro. Como dizemos ao postgresql para parar em 12:39 irmos encontrar as tabelas teste1, teste2, teste3.

Postgresql comeca o processo de recuperacao logo que o processo postmaster e startado(postmaster sabe que tem algo a fazer porque ele acha o arquivo \$PGDATA/recovery.conf

```
$pg_ctl -l /tmp/pg.log start
```

```
postmaster started
```

Se voce estiver rodando linux/Unix voce pode observar o arquivo de log do servidor

```
$ tail -f /tmp/pg.log
```

```
LOG: starting archive recovery
```

```
LOG: restore_command = "cp /tmp/wals/%f %p"
```

```
LOG: recovery_target_time = 2005-06-22 13:05:00-05
```

```
...
```

```
LOG: restored log file "00000001000000000000000006" from archive
```

```
LOG: restored log file "00000001000000000000000007" from archive
```

```
LOG: restored log file "00000001000000000000000008" from archive
```

```
LOG: restored log file "00000001000000000000000009" from archive
```

```
...
```

```
LOG: archive recovery complete
```

```
LOG: database system is ready
```

Quando o processo de recuperacao se completa, Postgresql renomeia o arquivo recovery.conf para recovery.done, para evitar que quando o processo postmaster seja iniciado novamente ocorra de novo a restauracao.

Agora posso me conectar ao meu servidor e ver as tabelas teste1, teste2, teste3.

```
$psql teste
```

```
Welcome to psql 8.0.0, the PostgreSQL interactive terminal. ...
```

```
test=# \d
```

Como voce pode ver PITR e facil de configurar(somente definir o archive_command no postgresql.conf.

Recapitulando todo o processo de configuracao:

- 1- configure o PITR definindo o archive_command no postgresql.conf
- 2- Restart o postmaster para habilitar o processamento do arquivo WAL
- 3- conecte em uma database e chame a funcao pg_start_backup(rotulo)
- 4- Faca o backup do cluster(Voce nao precisa parar o servidor)

Recapitulando o processo de recuperacao

- 1- Se possivel renomeie o cluster danificado para que possa copiar a maior quantidade de arquivos wal possiveis
- 2- Descompacte o cluster do arquivo que foi criado o backup

- 3- No diretorio do cluster restaurado remova os seguintes arquivos \$PG_DATA/pg_xlog/0* e o \$PGDATA/postmaster.pid
- 4- Se possivel copie os arquivos WAL do cluster danificado para o cluster restaurado para garantir que as transacoes mais recentes sejam tambem recuperadas
- 5- Crie o arquivo recovery.conf, com no minimo o seguinte parametro restore_command
- 6- Start o postmaster
- 7- Verifique se os dados que vc esperavam se encontram no lugar

Fonte: **PostgreSQL: The comprehensive guide to building, programming, and administering PostgreSQL databases, Second Edition** By Korrry Douglas, Susan Douglas

Traducao: Joao Cosme de Oliveira Junior : joaocosme@pop.com.br

Cópia de segurança em-linha (PITR)

Durante todo o tempo, o PostgreSQL mantém o *registro de escrita prévia* (WAL = *write ahead log*) no subdiretório `pg_xlog` do diretório de dados do agrupamento. O WAL contém todas as alterações realizadas nos arquivos de dados do banco de dados. O WAL existe, principalmente, com a finalidade de fornecer segurança contra quedas: se o sistema cair, o banco de dados pode retornar a um estado consistente "refazendo" as entradas gravadas desde o último ponto de verificação. Entretanto, a existência do WAL torna possível uma terceira estratégia para fazer cópia de segurança de banco de dados: pode ser combinada a cópia de segurança do banco de dados no nível de sistema de arquivos, com cópia dos arquivos de segmento do WAL. Se for necessário fazer a recuperação, pode ser feita a recuperação da cópia de segurança do banco de dados no nível de sistema de arquivos e, depois, refeitas as alterações a partir da cópia dos arquivos de segmento do WAL, para trazer a restauração para o tempo presente. A administração desta abordagem é mais complexa que a administração das abordagens anteriores, mas existem alguns benefícios significativos:

- O ponto de partida não precisa ser uma cópia de segurança totalmente consistente. Toda inconsistência interna na cópia de segurança é corrigida quando o WAL é refeito (o que não é muito diferente do que acontece durante a recuperação de uma queda). Portanto, não é necessário um sistema operacional com capacidade de tirar instantâneos, basta apenas o tar, ou outra ferramenta semelhante.
- Como pode ser reunida uma sequência indefinidamente longa de arquivos de segmento do WAL para serem refeitos, pode ser obtida uma cópia de segurança contínua simplesmente continuando a fazer cópias dos arquivos de segmento do WAL. Isto é particularmente útil para bancos de dados grandes, onde pode não ser conveniente fazer cópias de segurança completas regularmente.
- Não existe nada que diga que as entradas do WAL devem ser refeitas até o fim. Pode-se parar de refazer em qualquer ponto, e obter um instantâneo consistente do banco de dados como se tivesse sido tirado no instante da parada. Portanto, esta técnica suporta a *recuperação para um determinado ponto no tempo*: é possível restaurar voltando o banco de dados para o estado em que se encontrava a qualquer instante posterior ao da realização da cópia de segurança base.
- Se outra máquina, carregada com a mesma cópia de segurança base do banco de dados, for

alimentada continuamente com a série de arquivos de segmento do WAL, será criado um sistema reserva à quente (*hot standby*): a qualquer instante esta outra máquina pode ser ativada com uma cópia quase atual do banco de dados.

Da mesma forma que o método de cópia de segurança no nível de sistema de arquivos simples, este método suporta apenas a restauração de todo o agrupamento de bancos de dados, e não a restauração de apenas um subconjunto deste. Requer, também, grande volume de armazenamento de arquivos: a cópia de segurança base pode ser grande, e um sistema carregado gera vários megabytes de tráfego para o WAL que precisam ser guardados. Ainda assim, é o método de cópia de segurança preferido para muitas situações onde é necessária uma alta confiabilidade.

Para fazer uma recuperação bem-sucedida utilizando cópia de segurança em-linha, é necessária uma seqüência contínua de arquivos de segmento do WAL guardados, que venha desde, pelo menos, o instante em que foi feita a cópia de segurança base do banco de dados. Para começar, deve ser configurado e testado o procedimento para fazer cópia dos arquivos de segmento do WAL, *antes* de ser feita a cópia de segurança base do banco de dados. Assim sendo, primeiro será explicada a mecânica para fazer cópia dos arquivos de segmento do WAL.

Cópia dos arquivos de segmento do WAL

Em um sentido abstrato, a execução do sistema PostgreSQL produz uma seqüência indefinidamente longa de entradas no WAL. O sistema divide fisicamente esta seqüência em *arquivos de segmento* do WAL, normalmente com 16 MB cada (embora o tamanho possa ser alterado durante a construção do PostgreSQL). São atribuídos nomes numéricos aos arquivos de segmento para refletir sua posição na seqüência abstrata do WAL. Quando não é feita cópia dos arquivos de segmento do WAL, normalmente o sistema cria apenas uns poucos arquivos de segmento e, depois, "recicla-os" renomeando os arquivos que não são mais de interesse com número de segmento mais alto. Assume-se não existir mais interesse em um arquivo de segmento cujo conteúdo preceda o ponto de verificação anterior ao último, podendo, portanto, ser reciclado.

Quando é feita a cópia dos arquivos de segmento do WAL, deseja-se capturar o conteúdo de cada arquivo quando este é completado, guardando os dados em algum lugar antes do arquivo de segmento ser reciclado para ser reutilizado. Dependendo da aplicação e dos periféricos disponíveis, podem haver muitas maneiras de "guardar os dados em algum lugar": os arquivos de segmento podem ser copiados para outra máquina usando um diretório NFS montado, podem ser escritos em uma unidade de fita (havendo garantia que os arquivos poderão ser restaurados com seus nomes originais), podem ser agrupados e gravados em CD, ou de alguma outra forma. Para que o administrador de banco de dados tenha a máxima flexibilidade possível, o PostgreSQL tenta não assumir nada sobre como as cópias serão feitas. Em vez disso, o PostgreSQL deixa o administrador escolher o comando a ser executado para copiar o arquivo de segmento completado para o local de destino. O comando pode ser tão simples como `cp`, ou pode envolver um script complexo para o interpretador de comandos — tudo depende do administrador.

O comando a ser executado é especificado através do parâmetro de configuração [archive_command](#), que na prática é sempre colocado no arquivo `postgresql.conf`. Na cadeia de caracteres do comando, todo `%p` é substituído pelo caminho absoluto do arquivo a ser copiado, enquanto todo `%f` é substituído pelo nome do arquivo apenas. Se for necessário incorporar o caractere `%` ao comando, deve ser escrito `%%`. A forma mais simples de um comando útil é algo como

```
archive_command = 'cp -i %p /mnt/servidor/dir_copias/%f </dev/null'
```

que irá copiar os arquivos de segmento do WAL, prontos para serem copiados, para o diretório

/mnt/servidor/dir_copias (Isto é um exemplo, e não uma recomendação, e pode não funcionar em todas as plataformas).

O comando para realizar a cópia é executado sob a propriedade do mesmo usuário que está executando o servidor PostgreSQL. Como a série de arquivos do WAL contém efetivamente tudo que está no banco de dados, deve haver certeza que a cópia está protegida contra olhos curiosos; por exemplo, colocando a cópia em diretório sem acesso para grupo ou para todos.

É importante que o comando para realizar a cópia retorne o status de saída zero se, e somente se, for bem-sucedido. Ao receber o resultado zero, o PostgreSQL assume que a cópia do arquivo de segmento do WAL foi bem-sucedida, e remove ou recicla o arquivo de segmento. Entretanto, um status diferente de zero informa ao PostgreSQL que o arquivo não foi copiado; serão feitas tentativas periódicas até ser bem-sucedida.

Geralmente o comando de cópia deve ser projetado de tal forma que não sobrescreva algum arquivo de cópia pré-existente. Esta é uma característica de segurança importante para preservar a integridade da cópia no caso de um erro do administrador (tal como enviar a saída de dois servidores diferentes para o mesmo diretório de cópias). Aconselha-se a testar o comando de cópia proposto para ter certeza que não sobrescreve um arquivo existente, *e que retorna um status diferente de zero neste caso*. Tem sido observado que `cp -i` faz isto corretamente em algumas plataformas, mas não em outras. Se o comando escolhido não tratar este caso corretamente por conta própria, deve ser adicionado um comando para testar a existência do arquivo de cópia. Por exemplo, algo como

```
archive_command = 'test ! -f .../%f && cp %p .../%f'
```

funciona corretamente na maioria das variantes do Unix.

Ao projetar a configuração de cópia deve ser considerado o que vai acontecer quando o comando de cópia falhar repetidas vezes, seja porque alguma funcionalidade requer intervenção do operador, ou porque não há espaço para armazenar a cópia. Esta situação pode ocorrer, por exemplo, quando a cópia é escrita em fita e não há um sistema automático para troca de fitas: quando a fita ficar cheia, não será possível fazer outras cópias enquanto a fita não for trocada. Deve-se garantir que qualquer condição de erro, ou solicitação feita a um operador humano, seja relatada de forma apropriada para que a situação possa ser resolvida o mais rápido possível. Enquanto a situação não for resolvida, continuarão sendo criados novos arquivos de segmento do WAL no diretório `pg_xlog`.

A velocidade do comando de cópia não é importante, desde que possa acompanhar a taxa média de geração de dados para o WAL. A operação normal prossegue mesmo que o processo de cópia fique um pouco atrasado. Se o processo de cópia ficar muito atrasado, vai aumentar a quantidade de dados perdidos caso ocorra um desastre. Significa, também, que o diretório `pg_xlog` vai conter um número grande de arquivos de segmento que ainda não foram copiados, podendo, inclusive, exceder o espaço livre em disco. Aconselha-se que o processo de cópia seja monitorado para garantir que esteja funcionando da forma planejada.

Havendo preocupação em se poder recuperar até o presente instante, devem ser efetuados passos adicionais para garantir que o arquivo de segmento do WAL corrente, parcialmente preenchido, também seja copiado para algum lugar. Isto é particularmente importante no caso do servidor gerar pouco tráfego para o WAL (ou tiver períodos ociosos onde isto acontece), uma vez que pode levar muito tempo até que o arquivo de segmento fique totalmente preenchido e pronto para ser copiado. Uma forma possível de tratar esta situação é definir uma entrada no cron [1] que periodicamente, talvez uma vez por minuto, identifique o arquivo de segmento do WAL corrente e o guarde em

algum lugar seguro. Então, a combinação dos arquivos de segmento do WAL guardados, com o arquivo de segmento do WAL corrente guardado, será suficiente para garantir que o banco de dados pode ser restaurado até um minuto, ou menos, antes do presente instante. Atualmente este comportamento não está presente no PostgreSQL, porque não se deseja complicar a definição de [archive_command](#) requerendo que este acompanhe cópias bem-sucedidas, mas diferentes, do mesmo arquivo do WAL. O [archive_command](#) é chamado apenas para segmentos do WAL completados. Exceto no caso de novas tentativas devido a falha, só é chamado uma vez para um determinado nome de arquivo.

Ao escrever o comando de cópia, deve ser assumido que os nomes dos arquivos a serem copiados podem ter comprimento de até 64 caracteres, e que podem conter qualquer combinação de letras ASCII, dígitos e pontos. Não é necessário recordar o caminho original completo (%p), mas é necessário recordar o nome do arquivo (%f).

Deve ser lembrado que embora a cópia do WAL permita restaurar toda modificação feita nos dados dos bancos de dados do PostgreSQL, não restaura as alterações feitas nos arquivos de configuração (ou seja, nos arquivos `postgresql.conf`, `pg_hba.conf` e `pg_ident.conf`), uma vez que estes arquivos são editados manualmente, em vez de através de operações SQL. Aconselha-se a manter os arquivos de configuração em um local onde são feitas cópias de segurança regulares do sistema de arquivos. Para mudar os arquivos de configuração de lugar, deve ser consultada a [Seção 16.4.1](#).

Criação da cópia de segurança base

O procedimento para fazer a cópia de segurança base é relativamente simples:

1. Garantir que a cópia dos arquivos de segmento do WAL esteja ativada e funcionando.
2. Conectar ao banco de dados como um superusuário e executar o comando

```
SELECT pg_start_backup('rótulo');
```

onde rótulo é qualquer cadeia de caracteres que se deseje usar para identificar unicamente esta operação de cópia de segurança (Uma boa prática é utilizar o caminho completo de onde se deseja colocar o arquivo de cópia de segurança). A função `pg_start_backup` cria o arquivo *rótulo da cópia de segurança*, chamado `backup_label`, com informações sobre a cópia de segurança, no diretório do agrupamento.

Para executar este comando, não importa qual banco de dados do agrupamento é usado para fazer a conexão. O resultado retornado pela função pode ser ignorado; mas se for relatado um erro, este deve ser tratado antes de prosseguir.

3. Realizar a cópia de segurança utilizando qualquer ferramenta conveniente para cópia de segurança do sistema de arquivos, como `tar` ou `cpio`. Não é necessário, nem desejado, parar a operação normal do banco de dados enquanto a cópia é feita.
4. Conectar novamente ao banco de dados como um superusuário e executar o comando:

```
SELECT pg_stop_backup();
```

Se a execução for bem sucedida, está terminado.

Não é necessário ficar muito preocupado com o tempo decorrido entre a execução de `pg_start_backup` e o início da realização da cópia de segurança, nem entre o fim da realização da cópia de segurança e a execução de `pg_stop_backup`; uns poucos minutos de atraso não vão criar nenhum problema. Entretanto, deve haver certeza que as operações são realizadas sequencialmente,

sem que haja sobreposição.

Deve haver certeza que a cópia de segurança inclui todos os arquivos sob o diretório do agrupamento de bancos de dados (por exemplo, `/usr/local/pgsql/data`). Se estiverem sendo utilizados espaços de tabelas que não residem sob este diretório, deve-se ter o cuidado de incluí-los também (e ter certeza que a cópia de segurança guarda vínculos simbólicos como vínculos, senão a restauração vai danificar os espaços de tabelas).

Entretanto, podem ser omitidos da cópia de segurança os arquivos sob o subdiretório `pg_xlog` do diretório do agrupamento. Esta pequena complicação a mais vale a pena ser feita porque reduz o risco de erros na restauração. É fácil de ser feito se `pg_xlog` for um vínculo simbólico apontando para algum lugar fora do agrupamento, o que é uma configuração comum por razões de desempenho.

Para poder utilizar esta cópia de segurança base, devem ser mantidas por perto todas as cópias dos arquivos de segmento do WAL gerados no momento ou após o início da mesma. Para ajudar a realizar esta tarefa, a função `pg_stop_backup` cria o *arquivo de histórico de cópia de segurança*, que é armazenado imediatamente na área de cópia do WAL. Este arquivo recebe um nome derivado do primeiro arquivo de segmento do WAL, que é necessário possuir para fazer uso da cópia de segurança. Por exemplo, se o arquivo do WAL tiver o nome `0000000100001234000055CD`, o arquivo de histórico de cópia de segurança vai ter um nome parecido com `0000000100001234000055CD.007C9330.backup` (A segunda parte do nome do arquivo representa a posição exata dentro do arquivo do WAL, podendo normalmente ser ignorada). Uma vez que o arquivo contendo a cópia de segurança base tenha sido guardado em local seguro, podem ser apagados todos os arquivos de segmento do WAL com nomes numericamente precedentes a este número. O arquivo de histórico de cópia de segurança é apenas um pequeno arquivo texto. Contém a cadeia de caracteres rótulo fornecida à função `pg_start_backup`, assim como as horas de início e fim da cópia de segurança. Se o rótulo for utilizado para identificar onde está armazenada a cópia de segurança base do banco de dados, então basta o arquivo de histórico de cópia de segurança para se saber qual é o arquivo de cópia de segurança a ser restaurado, no caso disto precisar ser feito.

Uma vez que é necessário manter por perto todos os arquivos de segmento do WAL copiados desde a última cópia de segurança base, o intervalo entre estas cópias de segurança geralmente deve ser escolhido tendo por base quanto armazenamento se deseja consumir para os arquivos do WAL guardados. Também deve ser considerado quanto tempo se está preparado para despendar com a restauração, no caso de ser necessário fazer uma restauração — o sistema terá que refazer todos os segmentos do WAL, o que pode ser muito demorado se tiver sido decorrido muito tempo desde a última cópia de segurança base.

Também vale a pena notar que a função `pg_start_backup` cria no diretório do agrupamento de bancos de dados um arquivo chamado `backup_label`, que depois é removido pela função `pg_stop_backup`. Este arquivo fica guardado como parte do arquivo de cópia de segurança base. O arquivo rótulo de cópia de segurança inclui a cadeia de caracteres rótulo fornecida para a função `pg_start_backup`, assim como a hora em que `pg_start_backup` foi executada, e o nome do arquivo de segmento inicial do WAL. Em caso de dúvida, é possível olhar dentro do arquivo de cópia de segurança base e determinar com exatidão de qual sessão de cópia de segurança este arquivo provém.

Também é possível fazer a cópia de segurança base enquanto o postmaster está parado. Neste caso, obviamente não podem ser utilizadas as funções `pg_start_backup` e `pg_stop_backup`, sendo responsabilidade do administrador controlar a que cópia de segurança cada arquivo pertence, e até quanto tempo atrás os arquivos de segmento do WAL associados vão. Geralmente é melhor seguir

os procedimentos para cópia de segurança mostrados acima.

Recuperação a partir de cópia de segurança em-linha

Certo, aconteceu o pior e é necessário recuperar a partir da cópia de segurança. O procedimento está mostrado abaixo:

1. Parar o postmaster, se estiver executando.
2. Havendo espaço para isso, copiar todo o diretório de dados do agrupamento, e todos os espaços de tabelas, para um lugar temporário, para o caso de necessidade. Deve ser observado que esta medida de precaução requer a existência de espaço no sistema suficiente para manter duas cópias do banco de dados existente. Se não houver espaço suficiente, é necessário pelo menos uma cópia do conteúdo do subdiretório `pg_xlog` do diretório de dados do agrupamento, porque pode conter arquivos de segmento do WAL que não foram copiados quando o sistema parou.
3. Apagar todos os arquivos e subdiretórios existentes sob o diretório de dados do agrupamento, e sob os diretórios raiz dos espaços de tabelas em uso.
4. Restaurar os arquivos do banco de dados a partir da cópia de segurança base. Deve-se tomar cuidado para que sejam restaurados com o dono correto (o usuário do sistema de banco de dados, e não o usuário root), e com as permissões corretas. Se estiverem sendo utilizados espaços de tabelas, deve ser verificado se foram restaurados corretamente os vínculos simbólicos no subdiretório `pg_tblspc/`.
5. Remover todos os arquivos presentes no subdiretório `pg_xlog`; porque estes vêm da cópia de segurança base e, portanto, provavelmente estão obsoletos. Se o subdiretório `pg_xlog` não fizer parte da cópia de segurança base, então este subdiretório deve ser criado, assim como o subdiretório `pg_xlog/archive_status`.
6. Se existirem arquivos de segmento do WAL que não foram copiados para o diretório de cópias, mas que foram salvos no passo 2, estes devem ser copiados para o diretório `pg_xlog`; é melhor copiá-los em vez de movê-los, para que ainda existam arquivos não modificados caso ocorra algum problema e o processo tenha de ser recomeçado.
7. Criar o arquivo de comando de recuperação `recovery.conf` no diretório de dados do agrupamento (consulte [Recovery Settings](#)). Também pode ser útil modificar temporariamente o arquivo `pg_hba.conf`, para impedir que os usuários comuns se conectem até que se tenha certeza que a recuperação foi bem-sucedida.
8. Iniciar o postmaster. O postmaster vai entrar no modo de recuperação e prosseguir lendo os arquivos do WAL necessários. Após o término do processo de recuperação, o postmaster muda o nome do arquivo `recovery.conf` para `recovery.done` (para impedir que entre novamente no modo de recuperação no caso de uma queda posterior), e depois começa as operações normais de banco de dados.
9. Deve ser feita a inspeção do conteúdo do banco de dados para garantir que a recuperação foi feita até onde deveria ser feita. Caso contrário, deve-se retornar ao passo 1. Se tudo correu bem, liberar o acesso aos usuários retornando `pg_hba.conf` à sua condição normal.

A parte chave de todo este procedimento é a definição do arquivo contendo o comando de recuperação, que descreve como se deseja fazer a recuperação, e até onde a recuperação deve ir. Pode ser utilizado o arquivo `recovery.conf.sample` (geralmente presente no diretório de

instalação share) na forma de um protótipo [2]. O único parâmetro requerido no arquivo `recovery.conf` é `restore_command`, que informa ao PostgreSQL como trazer de volta os arquivos de segmento do WAL copiados. Como no `archive_command`, este parâmetro é uma cadeia de caracteres para o interpretador de comandos. Pode conter `%f`, que é substituído pelo nome do arquivo do WAL a ser trazido de volta, e `%p`, que é substituído pelo caminho absoluto para onde o arquivo do WAL será copiado. Se for necessário incorporar o caractere `%` ao comando, deve ser escrito `%%`. A forma mais simples de um comando útil é algo como

```
restore_command = 'cp /mnt/servidor/dir_copias/%f %p'
```

que irá copiar os arquivos de segmento do WAL previamente guardados a partir do diretório `/mnt/servidor/dir_copias`. É claro que pode ser utilizado algo muito mais complicado, talvez um script que solicite ao operador a montagem da fita apropriada.

É importante que o comando retorne um status de saída diferente de zero em caso de falha. Será solicitado ao comando os arquivos do WAL cujos nomes não estejam presente entre as cópias; deve retornar um status diferente de zero quando for feita a solicitação. Esta não é uma condição de erro. Deve-se tomar cuidado para que o nome base do caminho `%p` seja diferente de `%f`; não deve ser esperado que sejam intercambiáveis.

Os arquivos de segmento do WAL que não puderem ser encontrados entre as cópias, serão procurados no diretório `pg_xlog/`; isto permite que os arquivos de segmento recentes, ainda não copiados, sejam utilizados. Entretanto, os arquivos de segmento que estiverem entre as cópias terão preferência sobre os arquivos em `pg_xlog`. O sistema não sobrescreve os arquivos presentes em `pg_xlog` quando busca os arquivos guardados.

Normalmente a recuperação prossegue através de todos os arquivos de segmento do WAL, portanto restaurando o banco de dados até o presente momento (ou tão próximo quanto se pode chegar utilizando os segmentos do WAL). Mas se for necessário recuperar até algum ponto anterior no tempo (digamos, logo antes do DBA júnior ter apagado a tabela principal de transação de alguém), deve-se simplesmente especificar no arquivo `recovery.conf` o ponto de parada requerido. O ponto de parada, conhecido como "destino da recuperação", pode ser especificado tanto pela data e hora quanto pelo término de um ID de transação específico. Até o momento em que este manual foi escrito, somente podia ser utilizada a opção data e hora, pela falta de ferramenta para ajudar a descobrir com precisão o identificador de transação a ser utilizado.

Nota: O ponto de parada deve estar situado após o momento de término da cópia de segurança base (o momento em que foi executada a função `pg_stop_backup`). A cópia de segurança não pode ser utilizada para recuperar até um momento em que a cópia de segurança base estava em andamento (Para recuperar até este ponto, deve-se retornar para uma cópia de segurança base anterior e refazer a partir desta cópia).

Definições de recuperação

Estas definições somente podem ser feitas no arquivo `recovery.conf`, e são aplicadas apenas durante a recuperação. As definições deverão ser definidas novamente nas próximas recuperações que se desejar realizar. As definições não podem ser alteradas após a recuperação ter começado.

`restore_command` (string)

O comando, para o interpretador de comandos, a ser executado para trazer de volta os segmentos da série de arquivos do WAL guardados. Este parâmetro é requerido. Todo `%f`

presente na cadeia de caracteres é substituído pelo nome do arquivo a ser trazido de volta das cópias, e todo %p é substituído pelo caminho absoluto para onde o arquivo será copiado no servidor. Se for necessário incorporar o caractere % ao comando, deve ser escrito %%.

É importante que o comando retorne o status de saída zero se, e somente se, for bem-sucedido. Será solicitado ao comando os arquivos cujos nomes não estejam presentes entre as cópias; deve retornar um status diferente de zero quando for feita a solicitação. Exemplos:

```
restore_command = 'cp /mnt/servidor/dir_copias/%f "%p"'
restore_command = 'copy /mnt/servidor/dir_copias/%f "%p"' # Windows
```

recovery_target_time (timestamp)

Este parâmetro especifica o carimbo do tempo até onde a recuperação deve prosseguir. Somente pode ser especificado um entre `recovery_target_time` e [recovery_target_xid](#). O padrão é recuperar até o fim do WAL. O ponto de parada preciso também é influenciado por [recovery_target_inclusive](#).

recovery_target_xid (string)

Este parâmetro especifica o identificador de transação até onde a recuperação deve prosseguir. Deve-se ter em mente que enquanto os identificadores são atribuídos sequencialmente no início da transação, as transações podem ficar completas em uma ordem numérica diferente. As transações que serão recuperadas são aquelas que foram efetivadas antes (e opcionalmente incluindo) a transação especificada. Somente pode ser especificado um entre `recovery_target_xid` e [recovery_target_time](#). O padrão é recuperar até o fim do WAL. O ponto de parada preciso também é influenciado por [recovery_target_inclusive](#).

recovery_target_inclusive (boolean)

Especifica se a parada deve acontecer logo após o destino de recuperação especificado (`true`), ou logo antes do destino de recuperação especificado (`false`). Aplica-se tanto a [recovery_target_time](#) quanto a [recovery_target_xid](#), o que for especificado para esta recuperação. Indica se as transações que possuem exatamente a hora de efetivação ou o identificador de destino, respectivamente, serão incluídas na recuperação. O valor padrão é `true`.

recovery_target_timeline (string)

Especifica a recuperação de uma determinada cronologia. O padrão é recuperar ao longo da cronologia que era a cronologia corrente quando foi feita a cópia de segurança base. Somente será necessário definir este parâmetro em situações de re-recuperações complexas, onde é necessário retornar para um estado a que se chegou após uma recuperação para um ponto no tempo. Consulte a explicação na [Seção 22.3.4](#).

Cronologias

A capacidade de restaurar um banco de dados para um determinado ponto anterior no tempo cria algumas complexidades que são semelhantes às das narrativas de ficção científica sobre viagem no tempo e universos paralelos. Por exemplo, na história original do banco de dados talvez tenha sido removida uma tabela importante às 5:15 da tarde de terça-feira. Imperturbável, o administrador pega a cópia de segurança e faz uma restauração para o ponto no tempo 5:14 da tarde de terça-feira, e o sistema volta a funcionar. *Nesta* história do universo do banco de dados, a tabela nunca foi removida. Mas suponha que mais tarde seja descoberto que esta não foi uma boa idéia, e que se deseja voltar para algum ponto posterior da história original. Mas isto não poderá ser feito, porque quando o banco de dados foi posto em atividade este sobrescreveu alguns dos arquivos de segmento do WAL que levariam ao ponto no tempo onde agora quer se chegar. Portanto, realmente é necessário fazer distinção entre a série de entradas no WAL geradas após a recuperação para um ponto no tempo, e àquelas geradas durante a história original do banco de dados.

Para lidar com estes problemas, o PostgreSQL possui a noção de *cronologias (timelines)*. Cada vez que é feita uma recuperação no tempo anterior ao fim da seqüência do WAL, é criada uma nova cronologia para identificar a série de registros do WAL geradas após a recuperação (entretanto, se a recuperação prosseguir até o final do WAL, não é criada uma nova cronologia: apenas se estende a cronologia existente). O número identificador da cronologia é parte dos nomes dos arquivos de segmento do WAL e, portanto, uma nova cronologia não sobrescreve os dados do WAL gerados pelas cronologias anteriores. É possível, na verdade, guardar muitas cronologias diferentes. Embora possa parecer uma funcionalidade sem utilidade, muitas vezes é de grande valia. Considere a situação onde não se tem certeza absoluta de até que ponto no tempo deve ser feita a recuperação e, portanto, devem ser feitas muitas tentativas de recuperação até ser encontrado o melhor lugar para se desviar da história antiga. Sem as cronologias este processo em pouco tempo cria uma confusão impossível de ser gerenciada. Com as cronologias, pode ser feita a recuperação até *qualquer* estado anterior, inclusive os estados no desvio de cronologia abandonados posteriormente.

Toda vez que é criada uma nova cronologia, o PostgreSQL cria um arquivo de "história da cronologia" que mostra de que cronologia foi feito o desvio, e quando. Os arquivos de cronologia são necessários para permitir o sistema buscar os arquivos de segmento do WAL corretos ao fazer a recuperação a partir de uma área de cópias que contém várias cronologias. Portanto, estes arquivos são guardados na área de cópias como qualquer arquivo de segmento do WAL. Os arquivos de cronologia são apenas pequenos arquivos texto sendo, portanto, barato e apropriado mantê-los guardados indefinidamente (ao contrário dos arquivos de segmento que são grandes). É possível, caso se deseje fazê-lo, adicionar comentários aos arquivos de cronologia para fazer anotações personalizadas sobre como e porque foi criada uma determinada cronologia. Estes comentários são muito úteis quando há um grande número de cronologias criadas como resultado de experiências.

O comportamento padrão de recuperação é recuperar ao longo da cronologia que era a cronologia corrente quando foi feita a cópia de segurança base. Se for desejado fazer a recuperação utilizando uma cronologia filha (ou seja, deseja-se retornar para algum estado que foi gerado após uma tentativa de recuperação), é necessário especificar o identificador da cronologia de destino no arquivo `recovery.conf`. Não é possível fazer a recuperação para um estado que foi um desvio anterior à cópia de segurança base.

Cuidado

No momento em que esta documentação foi escrita, haviam várias limitações para o método de cópia de segurança em-linha, que provavelmente serão corrigidas nas versões futuras:

- Atualmente não são gravadas no WAL as operações em índices que não são B-tree (índices hash, R-tree e GiST), portanto quando o WAL é refeito os índices destes tipos não são atualizados. A prática recomendada para contornar este problema é reindexar manualmente todos os índices destes tipos após o término da operação de recuperação.

Também deve ser notado que o formato atual do WAL é muito volumoso, uma vez que inclui muitos instantâneos de páginas de disco. Isto é apropriado para a finalidade de recuperação de quedas, uma vez que pode ser necessário corrigir páginas de disco parcialmente preenchidas. Entretanto, não é necessário armazenar tantas cópias de páginas para operações de recuperação para um determinado ponto no tempo. Uma área para desenvolvimento futuro é a compressão dos dados do WAL copiados, pela remoção das cópias de página desnecessárias.

Notas

[1] cron — processo (*daemon*) para executar comandos agendados. (N. do T.)

[2] O arquivo `recovery.conf.sample` está presente no diretório `/src/backend/access/transam` da distribuição do código fonte e do CVS. (N. do T.)

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/continuous-archiving.html>

12) Conectividade

- 12.1) Integração com o PHP
- 12.2) Integração com Java (jdbc)
- 12.3) Integração com MS Access via ODBC
- 12.4) Integração com Visual C++
- 12.5) Integração com .NET

O PostgreSQL oferece conectividade com os principais linguagens e ferramentas que utilizam SGBDs.

Existem somente duas interfaces clientes incluídas na distribuição do PostgreSQL:

- libpq – é a interface primária para a linguagem C. Muitas outras interfaces são construídas sobre esta (<http://www.postgresql.org/docs/8.3/interactive/libpq.html>).
- ecpg – SQL incorporado ao PostgreSQL.

Artigo sobre ecpg - <http://www.vivaolinux.com.br/artigos/impressora.php?codigo=4652>

Todas as outras interfaces são projetos externos e são distribuídos separadamente.

Interfaces Clientes Mantidos Externos

Name	Language	Comments	Website
DBD::Pg	Perl	Perl DBI driver	http://search.cpan.org/dist/DBD-Pg/
JDBC	JDBC	Type 4 JDBC driver	http://jdbc.postgresql.org/
libpqxx	C++	New-style C++ interface	http://pqxx.org/
Npgsql	.NET	.NET data provider	http://npgsql.projects.postgresql.org/
ODBCng	ODBC	An alternative ODBC driver	http://projects.commandprompt.com/public/odbcng/
pgtclng	Tcl		http://pgfoundry.org/projects/pgtclng/
psqlODBC	ODBC	The most commonly-used ODBC driver	http://psqlodbc.projects.postgresql.org/
psycopg	Python	DB API 2.0-compliant	http://www.initd.org/

12.1) Integração com o PHP

Para a integração do PHP com o PostgreSQL não há necessidade de se instalar nenhum driver externo, ela acontece usando a libpq do PostgreSQL. A não ser que estejamos usando no Windows poderá acontecer da conexão não estar habilitada, então basta descomentar no php.ini a linha:

```
extension=php_pgsql.dll
```

A conexão da linguagem PHP com o SGBD é efetuada através da função:

pg_connect() - http://www.php.net/manual/pt_BR/function.pg-connect.php

Exemplo:

```
$con = pg_connect("host=127.0.0.1 dbname=teste user=postgres password=postgres port=5432");  
if (!$con){  
    echo "Falha na conexão com o banco. Veja detalhes técnicos: " . pg_last_error($con);  
}
```

Obs.: Lembrando que uma conexão ao PostgreSQL é realizada com um único banco.

Após a conexão podemos manipular a codificação de caracteres com as funções:

```
pg_set_client_encoding("UNICODE");  
echo pg_client_encoding();  
echo pg_set_client_encoding();
```

http://www.php.net/manual/pt_BR/function.pg-set-client-encoding.php

http://www.php.net/manual/pt_BR/function.pg-client-encoding.php

Instalar Xampplite e postgresql_phpgenerator para fazer demonstração com banco dba_projeto.

12.2) Integração com Java (jdbc)

Configuração do Java em Windows

No prompt:

doskey.com /insert

No painel de controle

Adicionar ao path:

c:\j sdk\bin;c:\j sdk\jre\bin;%PATH%

Criar a variável classpath

.;c:\j sdk\lib\tools.jar;c:\j sdk\lib\dt.jar;c:\j sdk\lib\htmlconverter.jar;c:\j sdk\jre\lib;c:\j sdk\jre\lib\rt.jar;c:\j sdk\jre\lib\ext\postgresql-8.3-603.jdbc3

No Linux (Ubuntu)

Para saber qual versão está usando, qual seu path e alterar, se for o caso:

sudo update-alternatives --config java

Então copie o driver jdbc para o diretório, por exemplo (postgresql-8.3-603.jdbc3.jar):

/usr/lib/jvm/java-6-sun/jre/lib/ext

O driver para integrar aplicações em Java ao PostgreSQL é o JDBC.

Para seleccionar a versão correta do JDBC devemos ter em mãos a versão do PostgreSQL e a versão do Java a usar e visitar o site <http://jdbc.postgresql.org/download.html#jdbcselection>

A versão for Windows já traz o respectivo JDBC. Na dúvida:

- JDK 1.1 - JDBC 1. Note que com a versão 8.0 o suporte ao JDBC 1 foi removido
- JDK 1.2, 1.3 - JDBC 2.
- JDK 1.3 + J2EE - JDBC 2 EE. Contém adicional suporte para classes javax.sql.
- JDK 1.4, 1.5 - JDBC 3. Contém suporte para SSL e javax.sql, mas não requer J2EE
- JDK 1.6 - JDBC4. Suporte para métodos JDBC4 é limitado.

Para a Versão Atual (8.3) do PostgreSQL

[JDBC3 Postgresql Driver, Version 8.3-603](#) (preferir este, mais maduro)

[JDBC4 Postgresql Driver, Version 8.3-603](#)

Pequenos exemplos em Java acessando banco PostgreSQL através do JDBC:

Para rodar os exemplos abaixo precisamos ter o J2SE (JDK) instalado.

Comando para compilar:

```
javac nome.java
```

Executando:

```
java nome.class
```

Este exemplo apenas testa a conexão com o servidor e o banco de dados.

Apenas copie o driver JDBC para o diretório \jre\lib\ext do Java, crie um arquivo chamado jdbc_teste.java com este conteúdo:

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;

public class jdbc_teste2 {
    public static void main(String[] argv) {
        System.out.println("Checando se o Driver está registrado com DriverManager.");

        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException cnfe) {
            System.out.println("Driver não encontrado!");
            System.out.println("Deixe mostrar o relatório do que encontrei e sair.");
            cnfe.printStackTrace();
            System.exit(1);
        }

        System.out.println("Driver Registrado corretamente, tentarei uma connection.");

        Connection c = null;

        try {
            // The second and third arguments are the username and password,
```

```
// respectively. They should be whatever is necessary to connect
// to the database.
c = DriverManager.getConnection("jdbc:postgresql://localhost:5433/dba_projeto", "postgres",
"postgres");
} catch (SQLException se) {
    System.out.println("Erro ao conectar: imprimindo o resultado e saindo.");
    se.printStackTrace();
    System.exit(1);
}

if (c != null)
    System.out.println("Conexão bem sucedida ao banco!");
else
    System.out.println("Nunca deve ver isso.");
}
}
```

Agora outro arquivo que mostra os registros de uma tabela

Crie um arquivo jdbc_teste2.java com o conteúdo abaixo:

```
import java.sql.*;

public class jdbc_teste {
    public static void main(String args[]) {
        String url = "jdbc:postgresql://localhost:5433/dba_projeto";
        Connection con;
        String query = "select * from clientes";
        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("Cliente " + 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());
}
}
```

Mais um pequeno exemplo, consultando uma tabela:

Crie um arquivo com nome jdbc_teste3.java, com o conteúdo das duas páginas seguintes:

```
import java.sql.*;

public class jdbc_teste3 {
    public static void main(String args[])
    {
```



```
String url = "jdbc:postgresql://localhost:5433/dba_projeto";
System.out.println("-----");
System.out.println("Esta é a URL: " + url);
Connection db;
ResultSet rs;
//Statement sq_stmt;
try
{
    Class.forName( "org.postgresql.Driver" );
}
catch ( java.lang.ClassNotFoundException e )
{
    System.err.print( "ClassNotFoundException: " );
    System.err.println( e.getMessage () );
}

System.out.println("Driver do PostgreSQL selecionado. ");

try
{
    db = DriverManager.getConnection( url, "postgres", "postgres" );
}
catch ( SQLException ex )
{
    System.err.println( "SQLException: " + ex.getMessage() );
}

System.out.println("Conexão aberta. ");
try
{
    db = DriverManager.getConnection( url, "postgres", "postgres" );
    Statement sq_stmt = db.createStatement();
    String sql_str = "SELECT * FROM clientes";
```

```
rs = sq_stmt.executeQuery(sql_str);
while (rs.next())
{
    System.out.println("-----");
    String cpf = rs.getString("cpf");
    String nome = rs.getString("nome");
    String email = rs.getString("email");
    System.out.println("CPF: " + cpf);
    System.out.println("Nome: " + nome);
    System.out.println("Email: " + email + " ");
}
System.out.println("-----");
System.out.println("-----");
}
catch ( SQLException ex )
{
    System.err.println( "SQLException: " + ex.getMessage() );
}

System.out.println("Consulta efetuada. ");

System.out.println("Conexão fechada. ");
System.out.println("-----");
}
}
```

Gerador de Relatórios

Agora vamos configurar um cliente Java para conectar ao PostgreSQL através do JDBC. Este cliente é o **gerador de relatórios** através do JDBC.

Instalar

- Instalar o iReport (<http://ireport.sf.net>)
- Abrir o iReport

Configurar Fonte de Dados

- Menu Data – Conexões/Fonte de Dados
- Novo
- Conexão de Banco de Dados JDBC – Próximo
- Nome (nome da conexão: conTeste)
- Driver (selecione org.postgresql.Driver)
- Caminho do JDBC (ajuste o necessário)
jdbc:postgresql://localhost:5432/MYDATABASE (original)
jdbc:postgresql://localhost:5432/dba_projeto
- Endereço do servidor (localhost)
- Banco de dados (dba_projeto)
- Usuário (postgres)
- Senha (postgres)

Antes de conectar ou de testar devemos copiar o driver JDBC para a pasta:

C:\Program Files\JasperSoft\iReport-2.0.4\lib

- Clique em Salvar

Criar Relatório

Clicar no botão New report (varinha mágica) para criar um novo relatório com o assistente

- Em Conexões selecionar a conexão criada (conTeste)
- Clique na caixa Consulta SQL e digite:
select codigo, valor from pedidos;
- E clique em Próximo
- Clique na seta dupla para a direita >> e Próximo
- Próximo em Agrupar...

- Próximo em layout
- Encerrar

Compilar

- Clique no menu Criar – Compilar e Salve

Executar

- Criar – Executar relatório (usar conexão ativa)

Caso não seja exibido clique em Criar e deixe selecionados: Visualizar JRViewer e Utilizar o virtualizador de Relatórios.

Um outro gerador de relatórios muito bom é o BIRT do grupo Eclipse:

<http://www.eclipse.org/birt>

12.3) Integração com MS Access

Para aplicações usadas por um único usuário, o MS Access atende geralmente. Ele tem problemas para atender diversos usuários simultâneos.

A conexão do MS Access e de muitos outros aplicativos for Windows com o PostgreSQL se dá através do psqLODBC, que pode ser encontrado em:

<http://www.postgresql.org/ftp/odbc/versions/msi/>

Baixar a última versão, descompactar e instalar usando o arquivo psqldb.msi ou atualizar uma versão existente com o arquivo upgrade.bat.

Após a instalação teremos um novo driver no ODBC do Windows.

Vamos criar uma Conexão ODBC

Adicionar um banco do PostgreSQL no ODBC

Iniciar – Painel de Controle – Ferramentas Administrativas – ODBC

Adicionar – PostgreSQL ANSI ou UNICODE (depende da codificação do banco)

Database – dba_pojoeto

Server – localhost

Username – postgres

Password – postgres

Então clique em Test

Observe que em Options existem diversas opções extra.

Clique em Save para guardar e OK.

Criar o Banco

Agora abra o Access e crie um banco de dados em branco chamado dba_projeto.

Clicar com o botão direito na janela bancos de dados (área livre)

Importar

Arquivos do tipo (Selecionar o último - Bancos de dados ODBC())

Clique na Aba acima - Fontes de dados de máquina

Selecione a PostgreSQL30W ou outro nome dado, a que criamos e clique em OK

Selecione uma das tabelas para importar e clique em OK

caso apenas vinculemos, as tabelas continuam no PostgreSQL e podemos usá-la e alterá-las, mas seus registros continuarão na tabela que está no PostgreSQL.

Já importando estamos trazendo uma cópia para o Access, uma cópia estática.

12.4) Integração com Visual C++

E também .NET e outros for Windows na parte III do Livro "PostgreSQL 8 for Windows".

Também o Capítulo 13 do PostgreSQL Prático:

http://pt.wikibooks.org/wiki/PostgreSQL_Prático/Conectividade

Conectando com Visual BASIC

```
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
```

```
cnn.Execute StrSql2
```

Outro exemplo:

Criar um DSN ODBC "pgresearch" via ADO e use:

```
Dim gcnResearch As ADODB.Connection
Dim rsUIId As ADODB.Recordset

' open the database
Set gcnResearch = New ADODB.Connection
With gcnResearch
    .ConnectionString = "dsn=Dpgresearch"
    .Properties("User ID") = txtUsername
    .Properties("Password") = txtPassword
    .Open
End With
```

Conexão com Visual Studio

<http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=1687>

Using C#

```
using CoreLab.PostgreSql;
...
PgSqlConnection oPgSqlConnection = new PgSqlConnection();
oPgSqlConnection.ConnectionString =
    "User ID=myUsername;" +
    "Password=myPassword;" +
    "Host=localhost;" +
    "Port=5432;" +
    "Database=myDatabaseName;" +
    "Pooling=true;" +
    "Min Pool Size=0;" +
    "Max Pool Size=100;" +
    "Connection Lifetime=0";
oPgSqlConnection.Open();
```

Using VB.NET

```
Imports CoreLab.PostgreSql
...
Dim oPgSqlConnection As PgSqlConnection = New PgSqlConnection()
oPgSqlConnection.ConnectionString =
    "User ID=myUsername;" & _
    "Password=myPassword;" & _
    "Host=localhost;" & _
    "Port=5432;" & _
    "Database=myDatabaseName;" & _
    "Pooling=true;" & _
    "Min Pool Size=0;" & _
    "Max Pool Size=100;" & _
    "Connection Lifetime=0"
oPgSqlConnection.Open()
```

For more information, see: [PostgreSQLDirect](#) .NET Data Provider. Download [here](#). Support forms [here](#).

Conectando ao .NET com PostgreSQL

Vamos agora instalar o **.NET Data Provider para PostgreSQL** chamada **Npgsql**. O download do provedor pode ser feito no endereço:

<http://pgfoundry.org/frs/download.php/1408/Npgsql2-MS-Net-bin.zip>

Descompacte o arquivo zipado e abra a pasta principal , dentro dela você verá duas pastas , abra a pasta **bin** e copie os arquivos **Npgsql.dll** e **Mono.Security.dll** para o diretório do projeto de sua aplicação VB .NET.

Após copiar estes arquivos clique com o botão direito do mouse sobre o nome do projeto e selecione a opção **Add Reference**, navegue até o local onde a **Npgsql.dll** foi copiada selecione-a e clique em **OK**.

Agora já temos tudo pronto para usar o **PostgreSQL** no **VB 2005 Express**, e é isso que vou fazer no próximo artigo: [VB 2005 - Acessando o PostGreSQL II](#)

Veja o original para as capturas e mais detalhes:

http://www.macoratti.net/07/11/vbn5_pg1.htm

http://www.macoratti.net/07/11/vbn5_pg2.htm

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>

A outra é o driver do próprio OO:

<http://dba.openoffice.org/drivers/postgresql/index.html>

- 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 OOBBase

- 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 OOBBase

Pronto. Agora todas as tabelas do banco bdteste estão disponíveis no banco criado no OOBBase.

Também podemos agora criar consulta com assistentes, criar formulários e relatórios com facilidade.

Fonte: http://pt.wikibooks.org/wiki/PostgreSQL_Pr%C3%A1tico/Ferramentas/OpenOffice_Base

13) Contribs

13.1) Relação de módulos de contribuição (contrib)

13.2) Alguns dos módulos em detalhes

13.1) Relação de módulos de contribuição (contrib)

Instalar Nova Contrib

No Windows, para instalar uma nova contrib, execute novamente o programa de instalação.

No Linux devemos instalar o devido pacote ou compilar os fontes do PostgreSQL.

Os contribs são módulos que são distribuídos paralelamente ao PostgreSQL e ficam geralmente numa pasta chamada contrib, dentro da pasta principal do PostgreSQL. São utilitários para diversos tipos de finalidades, todas relacionadas ao PostgreSQL.

Diversos contribs que chegam a ser muito importantes acabam, com o tempo, fazendo parte da distribuição do PostgreSQL.

Na versão 8.3.1 são em um total de 34 os contribs:

<http://www.postgresql.org/docs/8.3/interactive/contrib.html>

- F.1. [adminpack](#)
- F.2. [btree_gist](#)
- F.3. [chkpass](#)
- F.4. [cube](#)
- F.5. [dblink](#)
- F.6. [dict_int](#)
- F.7. [dict_xsyn](#)
- F.8. [earthdistance](#)
- F.9. [fuzzystrmatch](#)
- F.10. [hstore](#)
- F.11. [intagg](#)
- F.12. [intarray](#)
- F.13. [isn](#)
- F.14. [lo](#)
- F.15. [ltree](#)
- F.16. [oid2name](#)
- F.17. [pageinspect](#)
- F.18. [pgbench](#)
- F.19. [pg_buffercache](#)
- F.20. [pgcrypto](#)
- F.21. [pg_freespacemap](#)
- F.22. [pgrowlocks](#)
- F.23. [pg_standby](#)
- F.24. [pgstattuple](#)

F.25. [pg_trgm](#)
F.26. [seg](#)
F.27. [spi](#)
F.28. [sslnfo](#)
F.29. [tablefunc](#)
F.30. [test_parser](#)
F.31. [tsearch2](#)
F.32. [uuid-oss](#)
F.33. [vacuumlo](#)
F.34. [xml2](#)

No Windows e ao instalar pelos repositórios de distribuições Linux eles já vêm compilados, mas ao instalar o PostgreSQL através dos fontes, os contribs também precisam ser compilados.

Compilar:

Podemos compilar todos ao mesmo tempo, acessando o diretório com o fonte do contrib e executando:

```
make
```

Instalar:

```
make install
```

Para compilar somente um dos contribs, acessar seu diretório e executar:

```
make  
make install
```

Após compilar podemos rodar o binário, quando existir ou podemos importar os .sql executando:

```
psql -U postgres -d dbname -f /pathdopostgresql/contrib/module.sql
```

13.2) Alguns dos módulos em detalhes

1) adminpack

Traz funções para o pgAdmin e outras ferramentas de administração e gerenciamento com funcionalidades adicionais, como gerenciamento remoto e log de arquivos do servidor.

Funções implementadas

Somente o superusuário pode executar essas funções. Aqui uma lista das funções:

```
int8 pg_catalog.pg_file_write(fname text, data text, append bool)  
bool pg_catalog.pg_file_rename(oldname text, newname text, archivename text)
```

```
bool pg_catalog.pg_file_rename(oldname text, newname text)
bool pg_catalog.pg_file_unlink(fname text)
setof record pg_catalog.pg_logdir_ls()

/* Renaming of existing backend functions for pgAdmin compatibility */
int8 pg_catalog.pg_file_read(fname text, data text, append bool)
bigint pg_catalog.pg_file_length(text)
int4 pg_catalog.pg_logfile_rotate()
```

2) chkpass

Este módulo implementa um tipo de dados chamado **chkpass** que destina-se a armazenar senhas criptografadas.

Para usar este contrib, assim como outros, entrar no banco e executar:

```
\i 'C:/Program Files/PostgreSQL/8.2/share/contrib/chkpass.sql'
```

Agora o tipo chkpass já está disponível. Vejamos:

```
create temp table teste(s chkpass);
insert into teste values ('ola');
postgres=# select * from teste;
      s
-----
```

```
:uQFSOxjtmww3M
(1 registro)
```

Este tipo usa a função de criptografia crypt() e não pode ser indexável.

3) cube

Este módulo implementa um tipo de dados cube para representar cubos multidimensionais.

<http://www.postgresql.org/docs/8.3/interactive/cube.html>

4) dblink

É um módulo que suporta conexões para outros bancos de dados, inclusive de outros clusters

```
dblink_connect -- opens a persistent connection to a remote database
dblink_connect_u -- opens a persistent connection to a remote database, insecurely
dblink_disconnect -- closes a persistent connection to a remote database
dblink -- executes a query in a remote database
dblink_exec -- executes a command in a remote database
dblink_open -- opens a cursor in a remote database
dblink_fetch -- returns rows from an open cursor in a remote database
dblink_close -- closes a cursor in a remote database
dblink_get_connections -- returns the names of all open named dblink connections
```

[dblink_error_message](#) -- gets last error message on the named connection
[dblink_send_query](#) -- sends an async query to a remote database
[dblink_is_busy](#) -- checks if connection is busy with an async query
[dblink_get_result](#) -- gets an async query result
[dblink_cancel_query](#) -- cancels any active query on the named connection
[dblink_current_query](#) -- returns the current query string
[dblink_get_pkey](#) -- returns the positions and field names of a relation's primary key fields
[dblink_build_sql_insert](#) -- builds an INSERT statement using a local tuple, replacing the primary key field values with alternative supplied values
[dblink_build_sql_delete](#) -- builds a DELETE statement using supplied values for primary key field values
[dblink_build_sql_update](#) -- builds an UPDATE statement using a local tuple, replacing the primary key field values with alternative supplied values

Importando para o banco postgres:

Exemplo de uso:

Cluster1 (porta = 5432)

```
\i 'C:/Program Files/PostgreSQL/8.2/share/contrib/dblink.sql'
```

Cluster2, banco2 (porta=5433)

```
create table t(c int);
insert into t values(1);
insert into t values(2);
insert into t values(3);
insert into t values(4);
insert into t values(5);
```

Cluster1, banco1 (Consultando o outro banco):

Podemos criar um nome para uma conexão com:

```
select * from dblink_connect('conexao', 'hostaddr=127.0.0.1 dbname=db user=postgres port=5433');
```

E então referir-se apenas pelo nome assim:

```
select * from dblink('conexao','select c from t') as t1(c1 int);
```

```
select * from dblink('conexao','select c from t') as t1(c1 int);
```

```
select * from dblink('conexao','select * from tabela') as tabela1(c int, b int, a char(45));
```

Inserindo registros no outro banco do outro cluster:

```
select dblink_exec('conexao','insert into t values(5)');
```

```
select dblink_exec('conexao','insert into tabela values(8,8,"Manoel")');
```

Observe que se usa duas aspas simples para campos tipo string.

```
select dblink_exec('conexao', 'SELECT * FROM tabela WHERE c < 3');  
  
select dblink_current_query();
```

Exercício:

- Criar um novo cluster na porta 5433 com o usuário postgres
- Startar o servidor do novo cluster e acessar o psql
- Importar o dblink neste novo cluster (é suficiente apenas neste)
- Neste novo cluster, usando o banco postgres, executar uma consulta que traga nome e cpf de todos os registros da tabela clientes do banco dba_projeto do cluster default.

<http://www.postgresql.org/docs/8.3/interactive/dblink.html>

Um bom documento: "**Interconectando Servidores PostgreSQL com DBlink**"

5) lo

Suporte ao gerenciamento de grandes objetos, também chamados de LOs ou BLOBs. Ele inclui um tipo de dados **lo** e uma trigger **lo_manage**.

O JDBC assume que BLOBs (Binary Large Objects) são armazenados em tabelas.

O módulo lo conserta isso, pois anexa uma trigger para a tabela que contém uma coluna com referência ao LO.

Exemplo simples:

```
CREATE TABLE image (title TEXT, raster lo);  
  
CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON image  
FOR EACH ROW EXECUTE PROCEDURE lo_manage(raster);
```

<http://www.postgresql.org/docs/8.3/interactive/lo.html>

6) oid2name

Utilitário que ajuda o DBA a examinar a estrutura de arquivos do PostgreSQL

oid2name switches

Switch	Description
-o <i>oid</i>	show info for table with OID <i>oid</i>
-f <i>filenode</i>	show info for table with filenode <i>filenode</i>

Switch	Description
<code>-t tablename_pattern</code>	show info for table(s) matching <i>tablename_pattern</i>
<code>-s</code>	show tablespace OIDs
<code>-S</code>	include system objects (those in <i>information_schema</i> , <i>pg_toast</i> and <i>pg_catalog</i> schemas)
<code>-i</code>	include indexes and sequences in the listing
<code>-x</code>	display more information about each object shown: tablespace name, schema name, and OID
<code>-q</code>	omit headers (useful for scripting)
<code>-d database</code>	database to connect to
<code>-H host</code>	database server's host
<code>-p port</code>	database server's port
<code>-U username</code>	username to connect as

Mostrando todos os bancos e seus OIDs e Tablespaces:

C:\Program Files\PostgreSQL\8.2\bin\oid2name -U postgres

Tablespaces:

C:\Program Files\PostgreSQL\8.2\bin\oid2name -s -U postgres

Mais detalhes em: <http://www.postgresql.org/docs/8.3/interactive/oid2name.html>

Get size of Postgres DB from filesystem

Get the size accurately from postgres local filesystem, i guess there is some sql stuff that can do that but that does the job as well for me :

```
#!/bin/bash
/usr/lib/postgresql/8.1/bin/oid2name -U postgres | while read -a e; do
name=${e[1]}
oid=${e[0]}
[[ $oid == "All" || $oid == "Oid" || -z $oid || -z $name ]] && continue
typeset -a size
size=( du -s /var/lib/postgresql/8.1/main/base/$oid )
size=${size[0]}
printf [...]
```

7) pgbench

O objetivo deste é executar testes de benchmark no PostgreSQL. Exe executa a mesma sequência de

SQL em múltiplas sessões concorrentes e então calcula a taxa média das transações (transações por segundo). Por default, pgbench testa um cenário que é ligeiramente baseado no TCP-B envolvendo cinco comandos SELECT, INSERT e UPDATE por transação. Contudo é fácil de testar outros casos escrevendo seu próprio script de transação.

O pgbench não é a melhor ferramenta para medir o desempenho do SQL em aplicações Web mas é bom para medir o desempenho do servidor do PostgreSQL.

TCP-B - <http://www.tpc.org/tpcb/default.asp> ou http://www.tpc.org/tpcb/spec/tpcb_current.pdf

A saída típica do pgbench é como esta:

```
transaction type: TPC-B (sort of)
scaling factor: 10
number of clients: 10
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
tps = 85.184871 (including connections establishing)
tps = 85.296346 (excluding connections establishing)
```

As quatro primeiras linhas mostram os parâmetros mais importantes da configuração. As próximas linhas mostram o número de transações completadas e destinadas (o último é apenas o produto do número de clientes pelo número de transações) estes serão iguais a menos que a execução falhe antes da conclusão. As duas últimas linhas mostram a taxa TPS aparecendo com e sem a contagem do tempo para iniciar a sessão do banco.

Transações como TCP-B requerem algumas específicas tabelas para sua configuração.

Pgbench deve ser chamado com a opção -i (inicializar) para criar e popular essas tabelas.

Inicializando:

```
pgbench -i [ outras-opções ] nomebanco
```

nomebanco é o banco que já deve ter sido criado e onde estamos pretendendo realizar os testes. Para a conexão também existem os parâmetros -h, -p e -U como no exemplo abaixo:

```
pgbench -i -h 127.0.0.1 -p 5432 -U postgres bdteste
```

Ação: pgbench -i cria quatro tabelas: accounts, branches, history e tellers. Cuidado, se existirem tabelas com estes nomes elas serão excluídas.

Para o default fator de escala 1, as tabelas criadas aparecem a seguir com seus respectivos registros:

```
table          # of rows
-----
```

```
branches      1
tellers       10
accounts     100000
history       0
```

Podemos (e em muitos casos, devemos) incrementar o número de registros usando a opção `-s` (fator de escala). A opção `-F` (fator de preenchimento) talvez também deva ser usada na fase de inicialização.

Após a inicialização podemos executar o benchmark com um comando que não inclua a opção `-i`.

As opções mais importantes para o teste são:

- `-c` (número de clientes)
- `-t` (número de transações)
- `-f` (especifica um script personalizado para as transações)

Abaixo aparece a lista completa.

Table F-14. pgbench initialization options (fase de inicialização)

Option	Description
<code>-i</code>	Required to invoke initialization mode.
<code>-s</code> <code>scale_factor</code>	Multiply the number of rows generated by the scale factor. For example, <code>-s 100</code> will imply 10,000,000 rows in the <code>accounts</code> table. Default is 1.
<code>-F fillfactor</code>	Create the <code>accounts</code> , <code>tellers</code> and <code>branches</code> tables with the given <code>fillfactor</code> . Default is 100.

Table F-15. pgbench benchmarking options (fase dos testes)

Option	Description
<code>-c clients</code>	Number of clients simulated, that is, number of concurrent database sessions. Default is 1.
<code>-t</code> <code>transactions</code>	Number of transactions each client runs. Default is 10.
<code>-N</code>	Do not update <code>tellers</code> and <code>branches</code> . This will avoid update contention on these tables, but it makes the test case even less like TPC-B.
<code>-S</code>	Perform select-only transactions instead of TPC-B-like test.
<code>-f filename</code>	Read transaction script from <code>filename</code> . See below for details. <code>-N</code> , <code>-S</code> , and <code>-f</code> are mutually exclusive.
<code>-n</code>	No vacuuming is performed before running the test. This option is <i>necessary</i> if you are running a custom test scenario that does not include the standard tables <code>accounts</code> , <code>branches</code> , <code>history</code> , and <code>tellers</code> .

Option	Description
<code>-v</code>	Vacuum all four standard tables before running the test. With neither <code>-n</code> nor <code>-v</code> , <code>pgbench</code> will vacuum <code>tellers</code> and <code>branches</code> tables, and will remove all entries in <code>history</code> .
<code>-D</code> <code>varname=value</code>	Define a variable for use by a custom script (see below). Multiple <code>-D</code> options are allowed.
<code>-C</code>	Establish a new connection for each transaction, rather than doing it just once per client thread. This is useful to measure the connection overhead.
<code>-l</code>	Write the time taken by each transaction to a logfile. See below for details.
<code>-s</code> <code>scale_factor</code>	Report the specified scale factor in <code>pgbench</code> 's output. With the built-in tests, this is not necessary; the correct scale factor will be detected by counting the number of rows in the <code>branches</code> table. However, when testing custom benchmarks (<code>-f</code> option), the scale factor will be reported as 1 unless this option is used.
<code>-d</code>	Print debugging output.

Table F-16. `pgbench` common options (útil em ambas as fases)

Option	Description
<code>-h hostname</code>	database server's host
<code>-p port</code>	database server's port
<code>-U login</code>	username to connect as

Qual a transação executada atualmente no `pgbench`?

A transação default atualmente executa 7 comandos por transação:

1. `BEGIN;`
2. `UPDATE accounts SET abalance = abalance + :delta WHERE aid = :aid;`
3. `SELECT abalance FROM accounts WHERE aid = :aid;`
4. `UPDATE tellers SET tbalance = tbalance + :delta WHERE tid = :tid;`
5. `UPDATE branches SET bbalance = bbalance + :delta WHERE bid = :bid;`
6. `INSERT INTO history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);`
7. `END;`

Caso especifiquemos `-N`, os passos 4 e 5 não serão incluídos.

Caso seja especificado `-S` somente o `SELECT` é executado.

Boas Práticas

Algumas recomendações com a finalidade de receber resultados mais úteis do uso do pgbench.

- Nunca execute testes que rodem por somente alguns segundos. Incremente a opção -t para que execute por pelo menos alguns minutos. Em alguns casos precisaremos de horas para receber alguns números reprodutíveis.
- Para o default tipo TCP-B cenário de teste a opção de inicialização -s deve ser pelo menos igual ao número de clientes que se pretende testar (-c); caso contrário as medições não serão coerentes.
- Caso o autovacuum esteja habilitado os resultados poderão ser imprevisíveis.
- Uma libitação do pgbench é que ele próprio pode se tornar o gargalo caso estejamos testando um grande número de sessões cliente. Isso pode ser aliviado rodando o pgbench em uma máquina diferente da que roda o servidor do banco.

"Well, firstly: pgbench is not a good benchmarking tool. It is mostly used to generate load. Secondly, the numbers are suspicious: do you have fsync turned off? Do you have write caching enabled? If so, you'd want to make sure that cache is battery backed. Thirdly, the effects of caching will be seen on subsequent runs."

Exercícios:

Desabilitar o autovacuum no postgresql.conf e restartar o servidor.

Lembrando que na versão 8.2.7 no Windows, o pgbench é um executável do diretório bin.

- Criar o banco para o teste
C:\Program Files\PostgreSQL\8.2\bin>createdb -U postgres pgb_teste
C:\Program Files\PostgreSQL\8.2\bin>pgbench -i -U postgres pgb_teste -P postgres

Agora vamos incrementar o fator de escala para 100, para termos um total de 10.000.000 de registros na tabela accounts.

```
C:\Program Files\PostgreSQL\8.2\bin>pgbench -i -s 100 -U postgres pgb_teste -P postgres
```

Vamos testar com 1.000.000 de registros:

```
C:\Program Files\PostgreSQL\8.2\bin>pgbench -i -s 10 -U postgres pgb_teste -P postgres
```

Agora vamos ao teste de fato:

```
C:\Program Files\PostgreSQL\8.2\bin>pgbench -U postgres -s 10 -c 10 -t 3000  
pgb_teste -P postgres
```

starting vacuum...end.

transaction type: TPC-B (sort of)

scaling factor: 10

number of clients: 10

number of transactions per client: 3000

number of transactions actually processed: 30000/30000

tps = 110.873759 (including connections establishing)

tps = 111.188119 (excluding connections establishing)

Mais detalhes, em especial sobre scripts personalizados e geração de arquivos de log por transação:
<http://www.postgresql.org/docs/8.3/interactive/pgbench.html>

Outro teste:

```
C:\Program Files\PostgreSQL\8.2\bin>pgbench -U postgres -c 20 -t 4000 pgb_teste -P postgres
```

starting vacuum...end.

transaction type: TPC-B (sort of)

scaling factor: 10

number of clients: 20

number of transactions per client: 4000

number of transactions actually processed: 80000/80000

tps = 101.996193 (including connections establishing)

tps = 102.152872 (excluding connections establishing)

Anotar os valores para comparação com futuros testes.

Em Micro com Intel Core 2 Duo, 2GB de RAM, SATA 160GB e Linux Ubuntu 7.10:

```
marcofrota@cmiin04:~$ su - postgres
```

```
postgres@cmiin04:~$ createdb pgb
```

```
postgres@cmiin04:~$ /usr/lib/postgresql/8.2/bin/pgbench -i pgb
```

```
postgres@cmiin04:~$ /usr/lib/postgresql/8.2/bin/pgbench -s 10 -c 10 -t 3000 pgb
```

transaction type: TPC-B (sort of)

scaling factor: 1

number of clients: 10

number of transactions per client: 3000

number of transactions actually processed: 30000/30000

tps = 1328.642752 (including connections establishing)

tps = 1331.249780 (excluding connections establishing)

8) pgcrypto

Traz diversas funções de criptografia para uso no PostgreSQL.

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/pgcrypto.html>

9) pgstattuple

Traz estatísticas a nível de tuplas.

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/pgstattuple.html>

10) sslinfo

Mostra informações sobre o SSL dos clientes que se conectam ao servidor.

Detalhes em: <http://www.postgresql.org/docs/8.3/interactive/sslinfo.html>

11) tsearch2

Já foi integrada ao PostgreSQL 8.3.

Mas para manter compatibilidade com versões anteriores fica também como contrib para buscas textuais.

Mais detalhes em:

<http://www.postgresql.org/docs/8.3/interactive/tsearch2.html>

<http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/>

Primeira Edição da DBFree Magazine com o artigo "Indexação textual no PostgreSQL".

[http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?](http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=tsearch2.pdf)

[action=AttachFile&do=get&target=tsearch2.pdf](http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=tsearch2.pdf) ou

[http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?](http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=tsearch2.odt)

[action=AttachFile&do=get&target=tsearch2.odt](http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=tsearch2.odt)

Capítulo 25 do livro:

The comprehensive guide to building, programming, and administering PostgreSQL databases, Second Edition. Autoria de Korrry Douglas e Susan Douglas, Editora SAMS

Contribs

Alguns programadores desenvolvem ferramentas, módulos e exemplos que são úteis a quem trabalha com PostgreSQL.

Como sua utilidade é restrita e também a equipe pretende manter o core do PostgreSQL o menos possível, as contribs não são incorporados ao PostgreSQL. Com o tempo, quando alguma destas contribuições tornam-se muito importantes ela acaba por ser incorporada ao core, como foi o caso da T-Search agora na versão 8.3.

Importar uma contrib no Windows:

```
C:\Program Files\PostgreSQL\8.2\bin>psql -U postgres -d dnocs < ..\share\contrib\cube.sql
```

No Linux ou instalamos o pacote dos contribs ou compilamos o diretório contrib nos fontes, como também podemos compilar somente o diretório do cube.

Algumas Contribs**cube**

Traz um tipo de dados cubo, que pode ser tridimensional ou com cinco ou seis dimensões.

```
\c dnocs
```

```
-- Definindo um cubo:
```

```
SELECT '4'::cube AS cube;
```

```
-- Definindo um ponto no espaço:
```

```
SELECT '4, 5, 6'::cube AS cube;
```

```
-- Definindo uma caixa n-dimensional representada por um par de pontos opostos:
```

```
SELECT '(0,0,0,0),(1,-2,3,-4)'::cube AS cube;
```

```
-- O cubo inicia em (0,0,0,0) e termina em (1,-2,3,-4).
```

```
-- Ao definir um cubo devemos ficar atento para que os pontos tenham a mesma dimensionalidade:
```

```
-- Calcular a interseção de dois cubos:
```

```
SELECT cube_inter('(1,2,3,4),(0,0,0,0)', '(0,0,0,0),(-1,-2,-3,-4)');
```

```
-- Mostrará o ponto em comun entre os cubos.
```

```
-- União entre cubos
```

```
SELECT cube_union('(1,2,3,4),(0,0,0,0)', '(0,0,0,0),(-1,-2,-3,-4)');
```

```
SELECT cube_union(cube_union('(1,2,3,4),(0,0,0,0)', '(0,0,0,0),(-1,-2,-3,-4)'), '(0,0,0,0),  
(9,-10,11,-12)');
```

```
-- Localizando certo ponto em um cubo
```

```
SELECT cube_contains('(1,-2,3,-4),(0,0,0,0)', '(0,-1,1,-2)');
```

```
-- Podemos usar os operadores < e > em cubos:
```

```
SELECT '(0,0,0,0),(1,2,3,4)'::cube < '(0,0,0,0),(2,2,3,4)'::cube;
```

```
-- O operador << procurar um cubo à esquerda de outro cubo
```

```
SELECT '(-2,-3),(-1,-2)'::cube << '(0,0),(2,2)'::cube;
```

```
-- O operador >> procurar um cubo à direita de outro cubo
```

Cubos e Índices

```
CREATE TABLE mycubes(a cube DEFAULT '0,0'::cube);
```

Definindo um índice do tipo GiST:

```
CREATE INDEX cube_idx ON mycubes USING gist (a);
```

```
create table mytexts(a varchar primary key);
```

```
insert into mytexts values ('Joao');
```

```
insert into mytexts values ('Pedro');
```

```
insert into mytexts values ('Ribamar');
```

```
insert into mytexts values ('Manoel');
```

Para garantir que o SGBD não executará uma varredura sequencial (sequential scan), então executaremos:

```
SET enable_seqscan TO off;
```

```
SELECT * FROM mytexts WHERE a='Pedro';
```

Caso enable_seqscan tivesse como on o PostgreSQL executaria uma varredura sequencial, pois a tabela é muito pequena em termos de registros.

A situação muda se ao invés de usarmos = usarmos ~.

```
SELECT * FROM mytexts WHERE a ~ 'Pedro';
```

```
EXPLAIN SELECT * FROM mytexts WHERE a ~ 'Pedro';
```

Veja que agora o PostgreSQL voltou a usar o sequential scan, mesmo desabilitado.

Trabalhando com ISBN e ISSN

A contrib destes está no arquivo isn.sql.

International Standard Book Number (ISBN) e International Standard Serial Number (ISSN).

```
CREATE TABLE myisbn(name text, number isbn);
```

```
INSERT INTO myisbn VALUES('Apache Administration', '3-8266-0554-3');
```

```
SELECT * FROM myisbn;
```

Testando um ISBN inválido:

```
INSERT INTO myisbn VALUES('no book', '324324324324234');
```

```
SELECT * FROM myisbn WHERE number > '3-8266-0506-3'::isbn;
```

14) Guia de Replicação no PostgreSQL com Slony-I (Usando o PGAdmin, PostgreSQL-8.2.6)

14.1) Conceitos

14.2) No Windows XP

14.3) No Linux Ubuntu 7.10

14.1) Conceitos de replicação

Replicação de banco de dados é a **cópia dos dados de uma base de dados original para outra base**. Isto permite que possa **disponibilizar os dados em um ou mais sites**. Isto melhora a disponibilidade dos dados. A cópia destes dados podem ser parcial ou total.

A replicação poderá ser usada para Sistemas distribuídos, Sistema de alta disponibilidade, ou talvez em processo de ETL em Data Warehouse (*).

Nos conceitos atuais de replicação a primeira classificação das ferramentas é relativo a sua arquitetura é **síncrona ou assíncrona**.

Replicação **síncrona** é uma **cópia fiel em qualquer momento dos dados e transações**. No mecanismo **assíncrono**, o **servidor Master assume toda carga de escrita, atualização, deleção e leitura, com a diferença que o Slave apenas disponibiliza leitura da base** conforme a figura 2.

Nos servidores Master podem ocorrer transações de leitura, escrita, atualização e deleção, já no servidor Slave, poderá ocorrer apenas operações de leitura, conforme figura 1.

Entre servidores Multimaster você poderá ter síncrono ou assíncrono, já em servidores Master e Slave, assíncrono. Isto é determinado pela ferramenta utilizada, e o que determina qual arquitetura será utilizada é o objetivo de implementação. Segue abaixo um comparativo de ferramentas para replicação no PostgreSQL:

	Slony-I	Mamooth	Postgres-R	PG-Replicator
Tipo:	Assíncrono	Assíncrono	Síncrono	Assíncrono
Uso:	Master - multislave	Master - multislave	MultiMaster	Multimaster
Propaga DDL	Não	Não	Não	?
Sist. Operacional	Todos	Todos	Todos	Linux
BLOB	Não	?	?	Sim

Nestas três arquiteturas propostas acima, cada uma tem uma aplicabilidade específica preferencial.

- Síncrono Multimaster: Balanceamento de carga ou alta disponibilidade;

- Assíncrono Multimaster: Alta disponibilidade;
- Assíncrono Master-Slave: Relatórios diversos níveis.

Fig1 – Servidores Master estão ambos sincronizados transmitindo escrita, atualizações e deleções em tempo real.

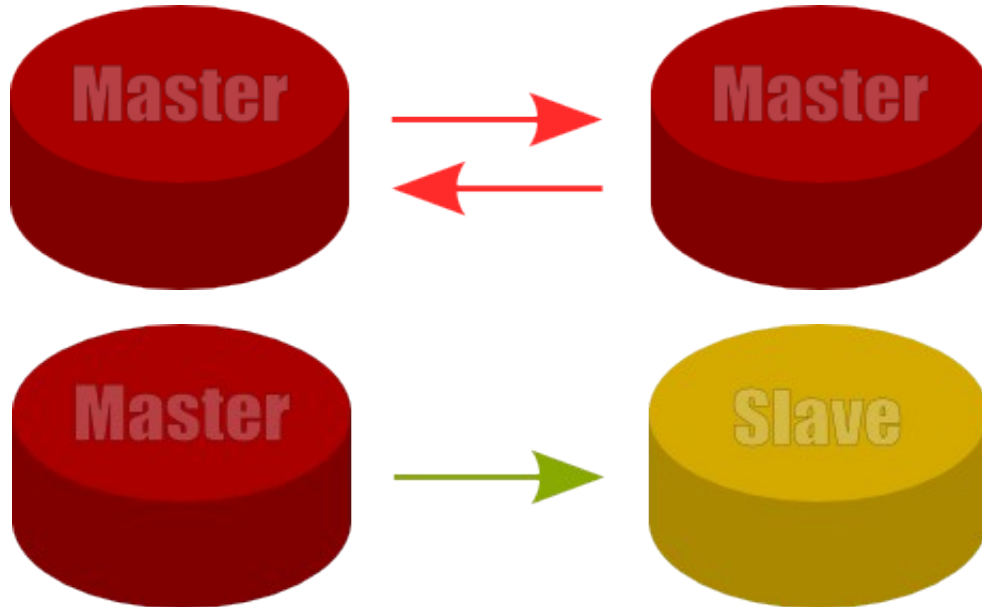
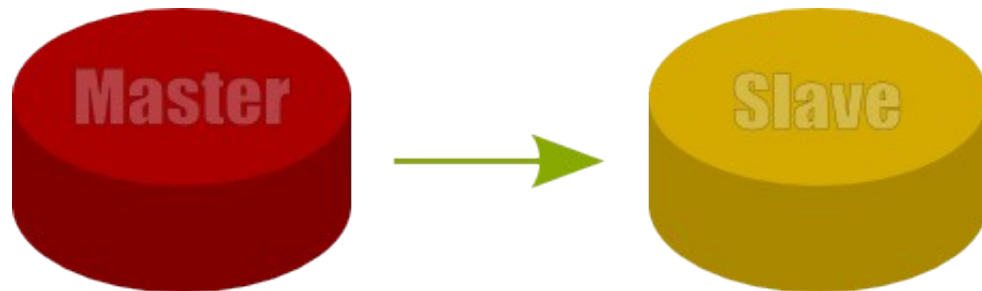


Fig2 - Servidores Master e Slave estão ambos sincronizados transmitindo escrita, atualizações e deleções em tempo real do Master para o Slave.



Para quem ainda não está familiarizado com as particularidades do mecanismo síncrono ou assíncrono talvez se pergunte o que aconteceria se um “Servidor A” replicasse assincronamente ao “Servidor B” e vice versa, isto não seria parecido com o mecanismo síncrono? A resposta disso é imediata, o servidor Slave é atualizado apenas e exclusivamente pelo mecanismo de sincronização. Isto provocaria um “lock”, travamento, de entrada de dados em ambos os servidores, ficando ambos apenas disponível para leitura, o que é logicamente um erro. A maior aplicabilidade dos mecanismos síncronos são para proporcionar alta disponibilidade, enquanto o mecanismo assíncrono pode ser usado para melhoria de performance e disponibilidade de relatórios complexos, e por que não dizer parte constituinte de um *Data Warehouse*, no processo de Extração, Transformação e Carga(ETL). **A alta disponibilidade é desejável em sistemas críticos que não podem parar.** Em caso de crash de um servidor, outro servidor assume de forma transparente para o usuário final do sistema. Para nosso primeiro exemplo de replicação imaginem uma empresa matriz no Distrito Federal e com filiais nos estados do Rio de Janeiro e São Paulo. A estrutura de rede necessariamente uma máquina deve “enxergar” a outra, seja por IP fixo, VPN ou outra. A matriz possui a gerência de material e recursos humanos da empresa toda e as filiais estão on-line podendo acompanhar atualizações na gestão de material e dos recursos humanos. Se houver uma queda temporária da rede a(s) filial(ais) estarão com o último estado consistente da base de dados, podendo manter seus serviços com menor risco, como por exemplo de vender algo que já foi vendido pela matriz ou outra filial, ou de não vender por não saber que o estoque já foi renovado e está disponível.

(*) Um *data warehouse* (ou **armazém de dados**, ou **depósito de dados** no Brasil) é um [sistema de computação](#) utilizado para armazenar informações relativas às atividades de uma organização em [bancos de dados](#), de forma consolidada. O desenho da base de dados favorece os relatórios, a

análise de grandes volumes de dados e a obtenção de informações estratégicas que podem facilitar a tomada de decisão.

O *data warehouse* possibilita a análise de grandes volumes de dados, coletados dos sistemas transacionais ([OLTP](#)). São as chamadas séries históricas que possibilitam uma melhor análise de eventos passados, oferecendo suporte às tomadas de decisões presentes e a previsão de eventos futuros. Por definição, os dados em um *data warehouse* não são voláteis, ou seja, eles não mudam, salvo quando é necessário fazer correções de dados previamente carregados. Os dados estão disponíveis somente para leitura e não podem ser alterados.

A ferramenta mais popular para exploração de um *data warehouse* é a *Online Analytical Processing* [OLAP](#) ou Processo Analítico em Tempo Real, mas muitas outras podem ser usadas.

Fonte: http://pt.wikipedia.org/wiki/Data_warehouse

Dentre os mecanismos de replicação iremos no capítulo seguinte mostrar o Slony, atualmente é o mais conhecido pela internet e aparentemente é o projeto que ganha mais adeptos.

Algumas características do Slony são:

- Replicação Master Multislave;
- Assíncrono;
- Multiplataforma;
- A definição da tabela a ser replicada na base “Master” deve ser idêntica a “Slave”;
- Replicação de tabelas e seqüências. Os demais objetos devem ser importados na base Slave de acordo com a necessidade do usuário;
- Em caso de falha o servidor slave não para, e quando retomada a comunicação, o banco slave retorna automaticamente a forma do master em “pouco tempo”;
- Slony deve ser instalado em cada servidor que desejar ser master ou slave;
- Baixo tráfego pela rede;
- As tabelas e seqüências slave ficam permanentemente com lock para escrita, atualização ou deleção. Qualquer tratamento deve ser feito pelo banco através de functions ou triggers;
- Inadequado para clusters multimaster;

Conceitos no Slony

O Slony é um replicador Master multi-Slave, assíncrono, e é multi-plataforma, ou seja, podemos instalá-lo no Windows, Linux, UNIX, Free BSD, e outros.

Fonte: <http://www.insphired.com/content/view/57/1/>

Alerta: muito importante verificar se a versão do PostgreSQL é compatível com a do Slony. Veja neste site detalhes: <http://developer.pgadmin.org/~hiroshi/Slony-I/>

O PGAdmin tem capacidade de criar e administrar replicação de clusters Slony-I já há algum tempo, porém ele foi projetado para permitir que o usuário trabalhe diretamente com os níveis mais baixos de conceitos do Slony, como listens (escutas) e paths (caminhos). Isto é muito flexível mas

não é muito amigável. Nós estamos esperando incluir alguns assistentes no futuro para tornar mais fácil de configurar e modificar clusters, mas enquanto isso, aqui está um bom atalho para criar um cluster de replicação com 3 nós.

14.2) No Windows XP

Neste exemplo um servidor master é configurado com dois slaves diretos. Este exemplo foi escrito e testado originalmente usando Slony-I v1.2.11 e PostgreSQL-8.2.5, rodando numa única máquina com o Windows XP. Repetido em Windows XP, PostgreSQL-8.2.6 e Slony-I 1.2.12.

Obs.: os três servidores, do master, do slave1 e do slave2, neste exemplo, estão numa mesma máquina (localhost). Numa situação real, geralmente cada um tem um IP diferente.

O utilitário (contrib) pgbench é utilizado para gerar o schema de teste e a carga de trabalho.

Replicando um banco de dados para dois slaves

1 – Crie três bancos: master, slave1 e slave2 e instale plpgsql em todos.

2 – Criar um schema do pgbench no banco master:

```
C:\Program Files\PostgreSQL\8.2\bin>pgbench -i -U postgres master -P postgres -p 5432
```

3 – Adicione na tabela history uma chave primária com nome history_pkey nos campos tid, bid e aid.

4 – Gerar um dump, somente do schema do banco master e importar nos bancos slave1 e slave2.

```
pg_dump -s -p 5432 -U postgres master > schema.sql
psql -p 5432 -U postgres slave1 < schema.sql
psql -p 5432 -U postgres slave2 < schema.sql
```

5 – Criar um script de configuração para cada engine do Slony. Os arquivos deverão conter somente duas linhas como as seguintes:

```
c:\slony\master.conf
```

```
cluster_name='pgbench'
conn_info='host=127.0.0.1 port=5432 user=postgres dbname=master
password=postgres'
```

Crie um arquivo para cada banco, ajuste o dbname e faça outros ajustes se precisar.

6 – Instalar o serviço do Slony

```
slon -regservice Slony-I
```

7 – Registrar cada um dos engines

Uso do comando slon:

```
slon [options] clustername conninfo
```

```
slon -addengine Slony-I C:\slony\master.conf
```

```
slon -addengine Slony-I C:\slony\slave1.conf
```

```
slon -addengine Slony-I C:\slony\slave2.conf
```

8 – No PGAdmin

Antes indicar em Arquivo – Opções

Em caminho do slony indique - [C:\Arquivos](#) de Programas\PostgreSQL\8.2\share

Sob o Nó Replicação do banco master criar um novo cluster Slony-I usando as seguintes opções:

```
Join existing cluster: Unchecked
```

```
Cluster name: pgbench
```

```
Local node: 1 Master node
```

```
Admin node: 99 Admin node
```

9 – Em cada um dos bancos slave1 e slave2 criar um cluster de Replicação com as opções:

slave1:

```
Join existing cluster: Checked
```

```
Server: <Select the server containing the master database>
```

```
Database: master
```

```
Cluster name: pgbench
```

```
Local node: 10 Slave node 1
```

```
Admin node: 99 - Admin node
```

e slave2:

```
Join existing cluster: Checked
```

```
Server: <Select the server containing the master database>
```

```
Database: master
```

```
Cluster name: pgbench
```

```
Local node: 20 Slave node 2
```

```
Admin node: 99 - Admin node
```

10 – Criar no master, paths para ambos os slaves e em cada slave voltar para o master. Criar os

paths sob cada nó no master usando a string de conexão do script de configuração. Observe que futuras reconstruções do cluster devem exigir que novos paths sejam definidos. Todos os paths devem ser preenchidos, tanto os do master quanto os dos slaves. Usar a string de conexão dos arquivos de configuração adequadamente:

```
host=127.0.0.1 port=5432 user=postgres dbname=master password=postgres
host=127.0.0.1 port=5432 user=postgres dbname=slave1 password=postgres
host=127.0.0.1 port=5432 user=postgres dbname=slave2 password=postgres
```

Veja na próxima página os detalhes

Exemplo de preenchimento dos Paths

Tomemos como exemplo os nós do banco master:

Ao expandir Replication – pgbench - Nodes.

Então vemos: Master node, Slave node 1, Slave node 2 e Admin node. Ao expandir cada um destes, encontraremos Paths(0), que deverá ser preenchido com todas as possibilidades, como por exemplo, o do Master node: clique sobre Paths(0) com o botão direito e New Path, então aparece um diálogo para entrarmos com informações do "10 – Slave node 1", apenas com Connect info vazio, onde devemos entrar com as informações de conexão do Slave1:

```
host=127.0.0.1 port=5432 user=postgres dbname=slave1 password=postgres
```

E Ok.

Então clicamos novamente com o botão direito do mouse sobre Paths(1) e New Path. Agora ele nos solicita as informações para o Slave2 e entramos em Connect info com:

```
host=127.0.0.1 port=5432 user=postgres dbname=slave2 password=postgres e Ok.
```

Clique novamente com o botão direito sobre Paths(2) e New Path. Observe que agora não aparece nenhum servidor. Mesmo que procuremos na combo, nada aí. Ou seja, ele nos solicita somente os que precisa, o que ajuda e muito a quem está iniciando.

De forma semelhante preencha todos os paths do master e dos slaves, de forma que ao final fique como o que aparece na próxima página.

Banco Master

- Replication

- pgbench

- Nodes (4)

- Master node

- Paths(2)

- Slave node 1

- Slave node 2

- listens(0)

```

Slave node 1
  Paths(2)
    Master node
    Slave node 2
  listens(0)
Slave node 2
  Paths(2)
    Master node
    Slave node 1
  listens(0)
Admin node
  Paths(3)
    Master node
    Slave node 1
    Slave node 2
  listens(9)
Replication Sets(1)

```

Banco Slave1

```

Replication
  pgbench
    Nodes (3)
      Master node
        Paths(1)
          Slave node 1
        listens(1)
      Slave node 1
        Paths(1)
          Master node
        listens(0)
      Admin node
        Paths(2)
          Master node
          Slave node 1
        listens(2)
    Replication Sets(0)

```

Banco Slave2

```

Replication
  pgbench
    Nodes (4)
      Master node
        Paths(2)
          Slave node 1
          Slave node 2
        listens(0)
      Slave node 1
        Paths(2)

```

Master node
Slave node 2
listens(0)
Slave node 2
Paths(2)
Master node
Slave node 1
listens(4)
Admin node
Paths(3)
Master node
Slave node 1
Slave node 2
listens(6)
Replication Sets(0)

11 – Criar um Replication Set no master usando as seguintes configurações:

ID: 1

Comment: pgbench set

12 – Adicionar as tabelas para o Replication set (master) com as configurações:

(master – Replication – pebench – Replication Sets – pgbench set - Tables)

Table: public.accounts

ID: 1

Index: accounts_pkey

Table: public.branches

ID: 2

Index: branches_pkey

Table: public.history

ID: 3

Index: history_pkey

Table: public.tellers

ID: 4

Index: tellers_pkey

13 – No master node criar uma nova subscrição para cada slave usando as seguintes opções:

(pgbench set – botão direito – new object – new subscription)

Origin: 1

Provider: 1 – Master node

Receiver: 10 – Slave node 1

Origin: 1

Provider: 1 – Master node

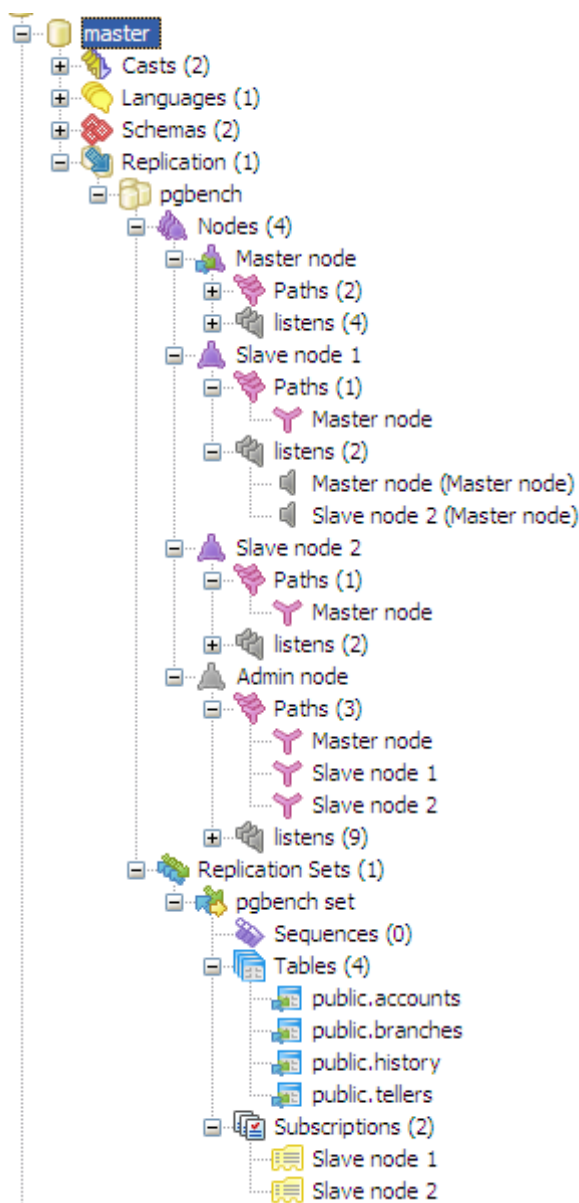
Receiver: 20 – Slave node 2

14 - Iniciar o serviço slon:

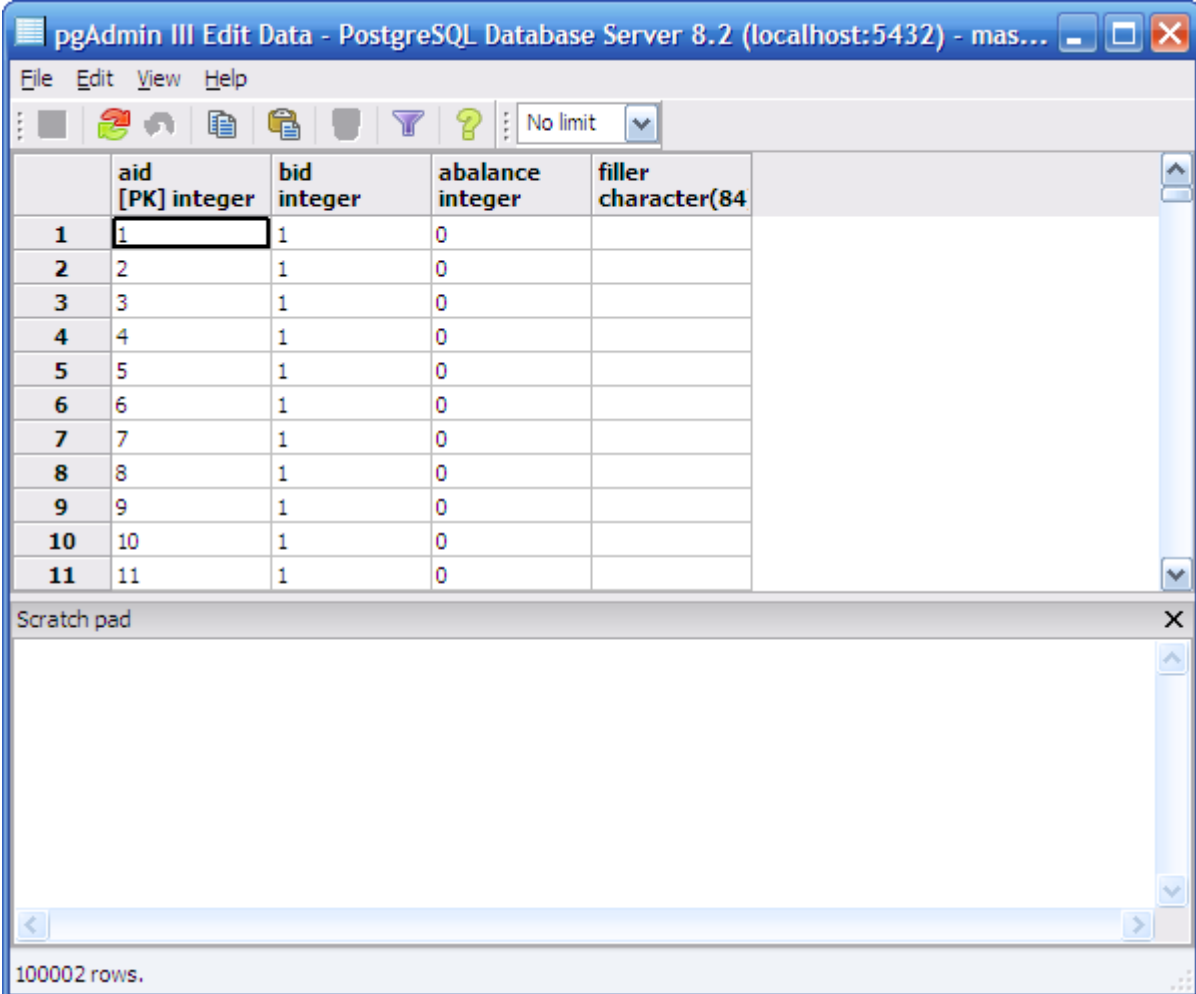
```
net start Slony-I
```

A replicação inicial deve começar e pode ser monitorada na aba Estatística do PGAdmin para cada nó. Basta seleccionar o cluster pgbench, expandir os 4 nós e seleccionar Estatística.

Veja o resultado no banco master após a conclusão:



Veja agora a tabela account no master:



pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - mas...

File Edit View Help

No limit

	aid [PK] integer	bid integer	abalance integer	filler character(84)
1	1	1	0	
2	2	1	0	
3	3	1	0	
4	4	1	0	
5	5	1	0	
6	6	1	0	
7	7	1	0	
8	8	1	0	
9	9	1	0	
10	10	1	0	
11	11	1	0	

Scratch pad

100002 rows.

Agora no Slave 1:

The screenshot shows the pgAdmin III 'Edit Data' window for a PostgreSQL database server. The window title is 'pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - slave1 ...'. The interface includes a menu bar (File, Edit, View, Help), a toolbar with icons for various database operations, and a table of data. The table has five columns: 'aid [PK] integer', 'bid integer', 'abalance integer', and 'filler character(84)'. The data is organized into 11 rows, with the first row highlighted. Below the table is a 'Scratch pad' area. At the bottom of the window, it indicates '100002 rows.'.

	aid [PK] integer	bid integer	abalance integer	filler character(84)
1	1	1	0	
2	2	1	0	
3	3	1	0	
4	4	1	0	
5	5	1	0	
6	6	1	0	
7	7	1	0	
8	8	1	0	
9	9	1	0	
10	10	1	0	
11	11	1	0	

Scratch pad

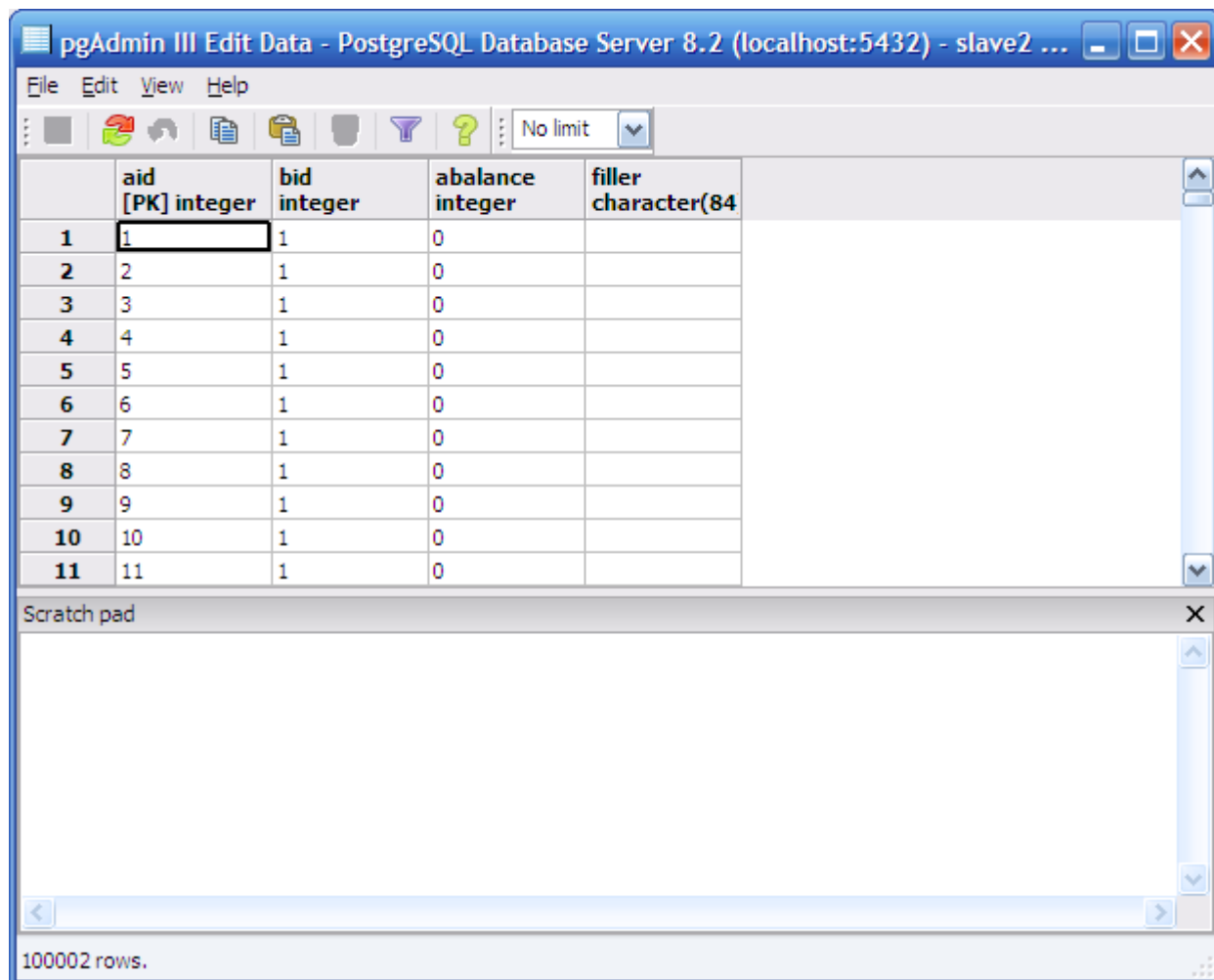
100002 rows.

Após inserir um registro na tabela account do banco master e dar um refresh, este mesmo registro foi visto nos slaves.

Lembrando que somente as tabelas do master permitem escrita. As dos slaves são somente leitura.

Orinalmente eram 100.000 registros. Inseri mais dois e os mesmos foram também vistos nos slaves.

Veja o slave2



pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - slave2 ...

File Edit View Help

No limit

	aid [PK] integer	bid integer	abalance integer	filler character(84)
1	1	1	0	
2	2	1	0	
3	3	1	0	
4	4	1	0	
5	5	1	0	
6	6	1	0	
7	7	1	0	
8	8	1	0	
9	9	1	0	
10	10	1	0	
11	11	1	0	

Scratch pad

100002 rows.

Original em inglês: <http://people.planetpostgresql.org/dpage/index.php?/archives/51-Setting-up-Slony-I-with-pgAdmin.html>

Tradução de Ribamar FS.

14.3) No Linux Ubuntu 7.10

Replicando um banco de dados para dois slaves

Instalar o PostgreSQL-8.2.7, PostgreSQL-8.2-slony1, PostgreSQL-contrib-8.2 (que traz o pgbench), slony1-bin e slony1-doc

1 – Crie três bancos: master, slave1 e slave2 e instale plpgsql em todos.

2 – Criar um schema do pgbench no banco master:

```
su - postgres
/usr/lib/postgresql/8.2/bin/pgbench -i -U postgres master -P postgres -p 5432
```

3 – Adicione na tabela history uma chave primária com nome history_pkey nos campos tid, bid e aid.

4 – Gerar um dump, somente do schema do banco master e importar nos bancos slave1 e slave2.

```
su - postgres

/usr/lib/postgresql/8.2/bin/pg_dump -s -U postgres master > schema.sql -p 5432
/usr/lib/postgresql/8.2/bin/psql -U postgres slave1 < schema.sql -p 5432
/usr/lib/postgresql/8.2/bin/psql -U postgres slave2 < schema.sql -p 5432
```

5 – Criar um script de configuração para cada engine do Slony (daemon em *nix). Os arquivos deverão conter somente duas linhas como as seguintes:

/var/lib/postgresql/slony/master.conf e slave1.conf e slave2.conf, com o conteúdo:

```
cluster_name='pgbench'
conn_info='host=127.0.0.1 port=5432 user=postgres dbname=master password=postgres'
```

Alerta: Cuidado para não inserir ; ao final das linhas. Isso aconteceu comigo e deu trabalho para descobrir. Quando editei o master.log que fica no /var/lib/postgresql/slony/ eu percebi pela mensagem de erro.

Crie um arquivo para cada banco, ajuste o dbname e faça outros ajustes se precisar.

6 – Registrar cada um dos engines

Uso do comando slon - slon [options] clustername conninfo

```
su - postgres
slon -f /var/lib/postgresql/slony/master.conf > master.log 2>& 1 &
slon -f /var/lib/postgresql/slony/slave1.conf > slave1.log 2>& 1 &
```

```
slon -f /var/lib/postgresql/slony/slave2.conf > slave2.log 2>& 1 &
```

7 – No PGAdmin

Antes indicar em Arquivo – Opções - Caminho do slony - /usr/share/slony1

Sob o Nó Replicação do banco master criar um novo cluster Slony-I usando as seguintes opções:

```
Join existing cluster: Unchecked
Cluster name: pgbench
Local node: 1 Master node
Admin node: 99 Admin node
```

8 – Em cada um dos bancos slave1 e slave2 criar um cluster de Replicação com as opções:

```
slave1:
Join existing cluster: Checked
Server: <Select the server containing the master database>
Database: master
Cluster name: pgbench
Local node: 10 Slave node 1
Admin node: 99 - Admin node
```

```
e slave2:
Join existing cluster: Checked
Server: <Select the server containing the master database>
Database: master
Cluster name: pgbench
Local node: 20 Slave node 2
Admin node: 99 - Admin node
```

9 – Criar no master, paths para ambos os slaves e em cada slave voltar para o master. Criar os paths sob cada nó no master usando a string de conexão do script de configuração. Observe que futuras reconstruções do cluster devem exigir que novos paths sejam definidos. Todos os paths devem ser preenchidos, tanto os do master quanto os dos slaves. Usar a string de conexão dos arquivos de configuração adequadamente:

```
host=127.0.0.1 port=5432 user=postgres dbname=master password=postgres
host=127.0.0.1 port=5432 user=postgres dbname=slave1 password=postgres
host=127.0.0.1 port=5432 user=postgres dbname=slave2 password=postgres
```

Observação: No banco master, no Nó master criar dois pats, um para cada slave.
Em cada Nó Slave, criar um path para o master.

Repetir o procedimento para os bancos slaves.

10 – Criar um Replication Set no master usando as seguintes configurações:

ID: 1
Comment: pgbench set

11 - Adicionar as tabelas para o Replication set (master) com as configurações:

(master – Replication – pebench – Replication Sets – pgbench set - Tables)

Table: public.accounts
ID: 1
Index: accounts_pkey

Table: public.branches
ID: 2
Index: branches_pkey

Table: public.history
ID: 3
Index: history_pkey

Table: public.tellers
ID: 4
Index: tellers_pkey

12 - No master node criar uma nova subscrição para cada slave usando as seguintes opções:

Origin: 1
Provider: 1 - Master node
Receiver: 10 - Slave node 1

Origin: 1
Provider: 1 - Master node
Receiver: 20 - Slave node 2

13 - Concluído

A replicação inicial deve começar e pode ser monitorada na aba Estatística do PGAdmin para cada nó. Basta selecionar o cluster pgbench, expandir os 4 nós e selecionar Estatística.

Fontes consultadas:

- Artigo do pontapé inicial - <http://people.planetpostgresql.org/dpage/index.php?/archives/51-Setting-up-Slony-I-with-pgAdmin.html>
- <http://www.insphired.com/content/view/57/1/>
- Lista internacional do PostgreSQL - <http://archives.postgresql.org/>
- Página oficial do Slony - <http://www.slony.info/>
- Documentação online do Slony - <http://linuxfinances.info/info/slonyintro.html>
- Documentação para download - <http://main.slony.info/downloads/1.2/source/slony1-1.2.13-docs.tar.bz2>
- Em pdf - <http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o?action=AttachFile&do=get&target=slony.pdf>
- Bons artigos sobre Replicação com slony no Linux:
<http://www.linuxjournal.com/article/7834>
<http://www.vivaolinux.com.br/artigos/impressora.php?codigo=4536>
- Wikipédia – <http://pt.wikipedia.org>
- <http://www.google.com.br>
- Capítulo 24 do livro
The comprehensive guide to building, programming, and administering PostgreSQL databases,
De Korry Douglas e Susan Douglas, SAMS editora.

15) Ferramentas

Acesso Remoto (linux e windows)
Administração (ads, webmin, pgadmin, phppgadmin, EMS, SQL Maestro)
Instaladores (Xampplite)
Geradores de Aplicativos em PHP (SQL Maestro)
Modelagem e Engenharia Reversa (DDT, DBVisualizer)
Monitoramento (Pureagent, explain)
Benchmark
Replicação
pgAgent
tsearch2
Exemplos de Bancos de Dados
Instalação do PostgreSQL-8.3.1 dos fontes no Ubuntu 7.10
Outros

15.1) PostgreSQL Autodoc
15.2) Gerando relatórios com o BIRT
15.3) Instalando o Xampplite e o Gerador de Aplicativos em PHP
15.4) DbVisualizer
15.5) pTop
15.6) Instalação do PostgreSQL-8.3.1 dos fontes no Linux Ubuntu 7.10

15.1) PostgreSQL Autodoc

O autodoc é um utilitário que roda para tabelas do PostgreSQL e retorna documentos [HTML](#), [Dot](#), [Dia](#) e [DocBook XML](#) com o DDL e diagramas das tabelas. Existe integração com o DIA (<http://www.gnome.org/projects/dia/>) e com o GraphViz (<http://www.research.att.com/sw/tools/graphviz/>).

Autodoc site oficial - <http://www.rbt.ca/autodoc/>

Instalar

Para quem tem Linux Ubuntu basta atualizar seus repositórios e no terminal executar:
sudo apt-get install postgresql-autodoc

Aproveitar e instalar também o DIA para visualizar os diagramas:
sudo apt-get install dia

Instalar também o GraphViz:
sudo apt-get install graphviz

Para Executar

Acesse um terminal e faça login como usuário do PostgreSQL
su - postgres


```
postgres@cmiin07 postgresql_autodoc -help
```

Exemplo:

Com este exemplo estou gerando diagramas e DDLs de um esquema (comercial) do banco dba_projeto2.

```
postgres@cmiin07 postgresql_autodoc -u postgres -d dba_projeto2 -s comercial -p 5433 -password=postgres
```

Ele gerará um arquivo em HTML contendo a estrutura dos objetos do esquema, gerará um arquivo do diagrama para o DIA, um XML e vários outros.

Agora um exemplo abrangendo todo o banco, que contém dois esquemas:

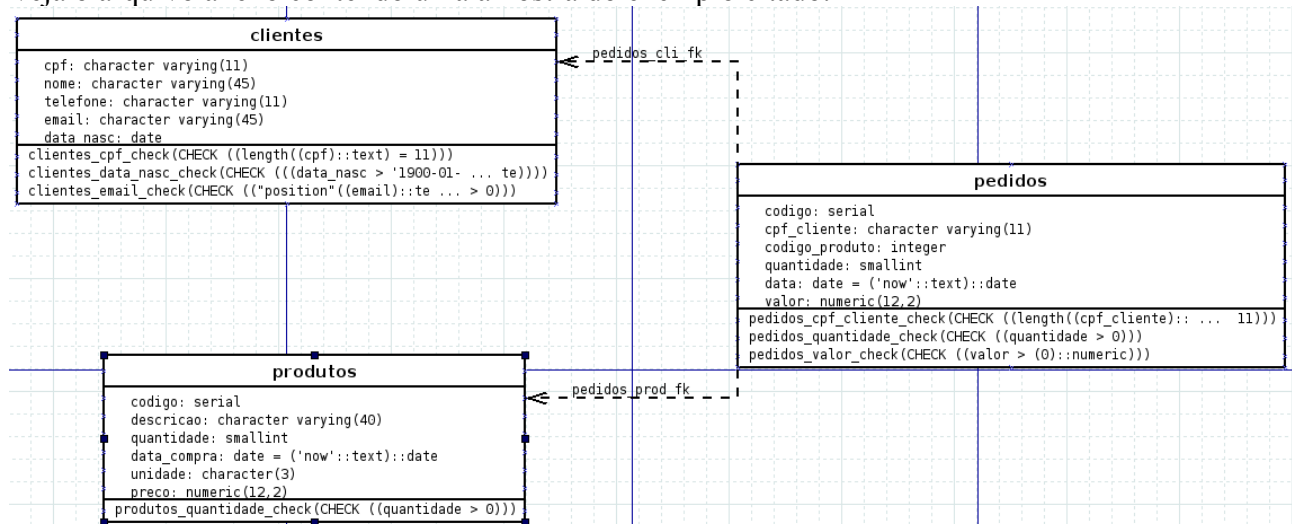
```
postgres@cmiin07 postgresql_autodoc -u postgres -d dba_projeto2 -p 5433 -password=postgres
```

Agora transformando o .dot em png:

```
postgres@cmiin07 dot -Tpng -o dba_projeto2.png dba_projeto2.dot
```

Com este comando gerará uma imagem oriunda do .dot.

Veja o arquivo anexo contendo uma amostra do exemplo citado.



Pelo visto existem muito mais recursos nesta ferramenta.

15.2) Gerando relatórios com BIRT

Através do Eclipse/BIRT podemos gerar relatórios com qualidade profissional a partir dos bancos de dados.

Site oficial – <http://www.eclipse.org/birt>

Após o download apenas descompactar.

Gerando Relatórios através de bancos do PostgreSQL.

Geraremos um relatório tomando as colunas descricao e quantidade da tabela produtos do banco dba_projeto.

Projeto

- Abrir o Eclipse/BIRT
- File – New – Project
- Expandir Business Intelligence and Reporting Tools e selecionar Report Project e Next
- Digite um nome para o projeto: pjt_produtos e clique em Finish

Relatório

- Clique com o botão direito sobre o pjt_produtos – New – Report
- Selecione pjt_produtos e digite um nome para o relatório: rpt_produtos.rptdesign e Next
- Deixe Blank Report e clique em Finish.

Indicar um Banco para o Relatório

Antes devemos ter o driver JDBC para a versão do PostgreSQL do servidor.

<http://jdbc.postgresql.org/download.html> ou aqui diretamente:

<http://jdbc.postgresql.org/download/postgresql-8.3-603.jdbc3.jar>

- Clicar na aba Data
 - Clicar em Data Sources com o botão direito – New Data Source
 - Clique em JDBC Data Source em Next
 - Clique no botão Manage Drivers
 - Clicar no botão Add e indicar o driver jdbc e OK.
 - Em Driver Class selecione (org.postgresql.Driver)
 - Em Database URL digite: jdbc:postgresql://localhost:5432/dba_projeto
 - Em Username digite postgres e em Password digite postgres
 - Clique no botão Test Connection. Se tudo correu bem a mensagem será de sucesso.
- Caso contrário corrija e tente novamente.

Finalmente clique em Finish.

Aproveite para salvar as alterações clicando no disquete acima.

Indicar uma Consulta para o Relatório

- Ainda na aba Data clique em Data Sets com o botão direito e New Data Set
- Aceite o default e clique em Next
- Expanda public, depois expanda produtos
- Duplo clique em descricao – digite uma vírgula e duplo clique em quantidade
- Clique logo após from e duplo clique em produtos
- De forma que ao final a consulta apareça assim:

```
select public.produtos.descricao,public.produtos.quantidade
```

from public.produtos

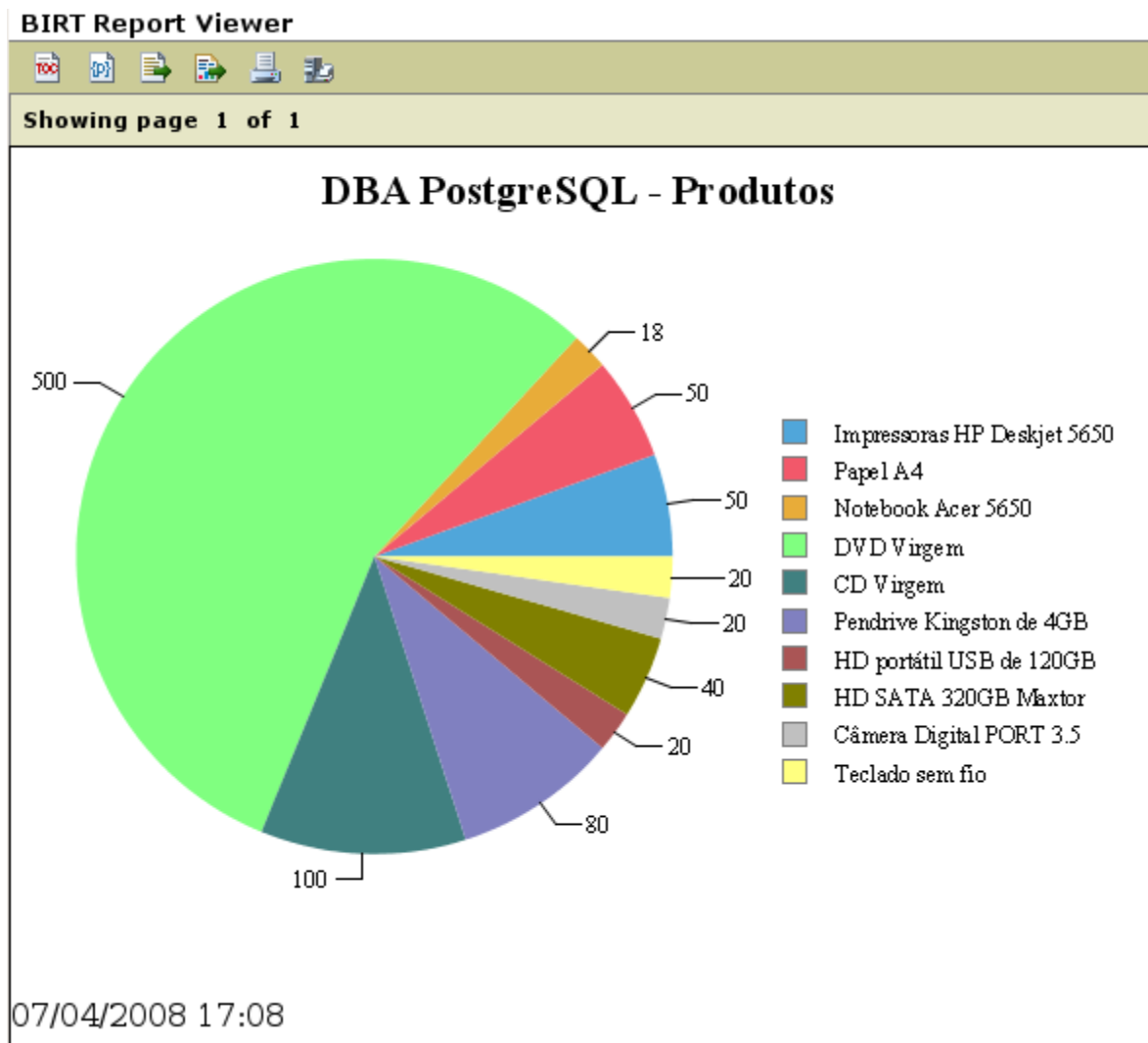
Então clique em Finish.

- Clique em Preview Results para visualizar os registros e OK

Desenhar o Relatório em forma de Gráfico (Torta)

- Clique na aba Palet
- Clique no controle Chart e arraste e solte na área livre do relatório
- Selecione Pie e em Dimension selecione 2D With Depth e clique em Next
- Em Select Data clique em Use Data Set
- A direita clique em <None> e selecione Data Set
- Clique em quantidade, arraste e solte na caixa abaixo de Series 1
- Clique em descricao, arraste e solte na caixa à direita de Category Definition e Next
- Clique em Title e apague o conteúdo da caixa Chart Title e digite nela DBA Projeto – Produtos
- Clique em Finish e clique no disquete para salvar
- Então clique em Preview para uma visualização mais adequada.
- Clique em File – View Report (selecione um dos formatos)

Veja como ficou nosso relatório



Visualizando Relatórios pela Web

Lembrando que este gerador de relatórios também tem fácil integração com linguagens web, podendo ser visualizados na Web através de um link. Para isso veja detalhes aqui:

http://pt.wikibooks.org/wiki/Aplicativos_em_PHP/Recursos_Extras/Geradores_de_Relat%C3%B3rios

15.3) Instalando o Xampplite e o Gerador de Aplicativos em PHP

O Xampplite é um pacote de instaladores para o Apache, PHP, MySQL e outros, que existe para Windows, Linux e outros sistemas operacionais. Após fazer o download e descompactar já está tudo pré-configurado e pronto para usar.

Download – <http://xampp.sf.net>

Descompactar no raiz do C:\

Após descompactar executar o arquivo xampp-control.exe e startar o Apache para que o PHP possa

funcionar. Caso queira pode startar também o MySQL.

Caso o suporte ao PostgreSQL não esteja habilitado basta editar o arquivo `apache\bin\php.ini` e remover o ponto e vírgula à direita da linha com “`extension=php_pgsql.dll`”.

Após startar o apache podemos executar arquivos do PHP, que obrigatoriamente deverão estar em:

`c:\xampplite\htdocs\teste.php`

E serão chamados pelo navegador com:

<http://localhost/teste.php>

Agora vamos instalar o PostgreSQL PHP Generator:

<http://www.sqlmaestro.com/products/postgresql/phpgenerator/>

Basta fazer o download e instalar.

Após instalar execute e crie um aplicativo em PHP usando as tabelas clientes e produtos do banco `dba_projeto`. Observe que será criado um aplicativo com três arquivos.

Usando o Gerador

- Na tela Propriedades de Conexões – apenas informe os dados do banco e clique em Test connect para saber se tá tudo OK e clique em Next.
Selecione as tabelas clientes e produtos (este gerador não trabalha com relacionamentos) e Next
- Na tela das Querys clique em Next
- Na tela de seleção de tabelas e querys selecione abaixo o diretório:
`C:\xampplite\htdocs` e clique em Next
- Na tela Fields for script apenas clique em Next
- Na tela Style apenas Next
- Na tela Navigator Next
- Na tela Security Next
- Na tela Generation log clique em Generate.

Agora já temos uma aplicação em PHP acessando o banco `dba_projeto`.

Inicie o Apache com o arquivo `C:\xampplite\xampp_control.exe` e chame o navegador no endereço:

http://localhost/dba_projeto

Veja que apresenta os dois arquivos criados. Clique em um deles e observe que haverá um link para o outro.

Observe que já vem com paginação, por exemplo a tabela de clientes exhibe seus registros em duas páginas com ordenação pelos campos e busca.

15.4) DbVisualizer

Esta é uma ferramenta que gera um Diagrama de Entidades Relacionamentos (DER), ou seja, para engenharia reversa.

Site oficial - <http://www.dbvis.com/products/dbvis/>

Screenshots - <http://www.dbvis.com/products/dbvis/features/features.jsp?page=screens>

Características - <http://www.dbvis.com/products/dbvis/features/features.jsp?page=matrix>

Específicas para o PostgreSQL -

<http://www.dbvis.com/products/dbvis/doc/main/doc/ug/databaseSpecific/postgresql.html>

Criando uma Conexão com o PostgreSQL via JDBC (ele já traz o driver JDBC)

- Abrir o DbVisualizer
- Clicar em Connections com o botão direito e Create Database Connection
- Clique em Use Wizard
- Entre com um nome para a conexão e Next
- Selecione o driver do PostgreSQL na lista e Next
- Entre com os dados da conexão e clique em Test Connection. Se tudo Ok clique em Finish.
- Clique no nome do banco à esquerda e acima
- Role a lista e selecione as tabelas do banco, segurando a tecla Ctrl
- Então clique acima em References para visualizar o gráfico.

15.5) pTop

Monitorando o PostgreSQL com ptop

Escrito por Guedes - <http://makeall.wordpress.com/2008/04/02/monitorando-o-postgresql-com-ptop/>

Em um ambiente corporativo com aplicações de missão crítica, qualquer instante de instabilidade pode causar um grande prejuízo para seu cliente, e conseqüentemente para sua empresa. Neste universo, onde todas as camadas de infra-estrutura que dão suporte ao funcionamento de suas aplicações precisam estar altamente-disponíveis, faz-se necessário tomar medidas preventivas a fim de identificar, pro-ativamente, qualquer incidente que pode acarretar em um grande problema.

Neste cenário, o papel do DBA é muito importante, pois ele pode identificar, já na camada de Banco de Dados, possíveis candidatos a problemas futuros. No entanto, ele precisa de ferramentas de apoio para tal tarefa, a fim de tornar seu trabalho produtivo e pró-ativo e não reativo, como em muitas empresas.

Sendo assim, gostaria de falar hoje sobre o nosso amigo *ptop*, que é uma espécie de ‘top’ para o PostgreSQL. Inspirado no top dos sistemas UNIX-like, ele permite:

- Visualizar a instrução SQL sendo executada por um processo;
- Visualizar o plano de execução de um SELECT rodando no momento;
- Visualizar os *locks* de um determinado processo;

- Visualizar as estatísticas das tabelas de usuário;
- Visualizar as estatísticas dos índices de usuário;

Obtendo e Instalando

Site oficial - http://pgfoundry.org/frs/?group_id=1000300

A última versão estável do ptop é a 3.6.1 mas já existem versões beta disponíveis que podem ser baixadas e testadas. No nosso caso utilizaremos a última versão estável, fazendo o download e salvando em um diretório temporário, como por exemplo: **/tmp**.

Vamos supor que você deseja monitorar via *ptop* um ambiente de desenvolvimento com as seguintes características:

Nome do bando de dados: dba_projeto

Porta: 5432

Usuário: postgres

Senha: postgres

(Para esse exemplo foi criado um usuário monitor no PostgreSQL, sem qualquer privilégio a mais, apenas de conexão ao banco)

Salve o arquivo baixado no servidor que deseja monitorar e siga os passos:

Obs.: No Ubuntu para instalar o pg_config use: `sudo apt-get install libpq-dev`

```
# Descompacte o arquivo ...
tar zxvf ptop-3.6.1.tar.gz

# Acesse o diretorio criado
cd ptop-3.6.1/

# Leia os arquivos README e INSTALL
# (sim eles não têm esses nomes à toa... ":D )
less README
less INSTALL

# Configure e compile o código
./configure && make

# Após compilar um arquivo executável sera criado
ls -la ptop
-rwxr-xr-x 1 gudes gudes 140282 2008-04-01 20:48 ptop

Usage: ptop [-ISTWbcinqu] [-x x] [-s x] [-o field] [-z username]
          [-p PORT] [-U USER] [-d DBNAME] [-h HOSTNAME] [number]
```

Visão geral do ptop

Para executar o *ptop* use:

./ptop -U usuario -d banco -h localhost -p porta -W

Será solicitada a senha do usuário.

Com o *ptop* sendo executado, é possível ver uma série de informações úteis sobre o servidor e os processos do PostgreSQL. A semelhança do mesmo com o utilitário *top* do UNIX é visível e não é mera coincidência...

Principais comandos do *ptop* e suas telas

Obtendo ajuda no *ptop*: pressione ‘?’ ou ‘h’

Obtendo o plano de execução de um processo: pressione ‘E’ (maiúsculo) e digite o número de um determinado processo (PID) para visualizar o plano de execução do mesmo.

Obtendo as estatísticas das tabelas em uso: pressione ‘R’ (maiúsculo)

Obtendo as estatísticas dos índices em uso: pressione ‘X’ (maiúsculo)

Visualizando os locks de um determinado processo: pressione ‘L’ (maiúsculo) e digite o número do processo (PID) que deseja visualizar

DESAFIO 1: tente alterar a ordem de classificação dos processos (por cpu, tempo de execução ou utilização de recursos, por exemplo).

DESAFIO 2: tente alterar o número de processos a serem mostrados.

15.6) Instalação do PostgreSQL-8.3.1 dos fontes no Linux Ubuntu 7.10

Através dos Repositórios

Instalar

```
sudo apt-get install postgresql-8.3
```

Com isso teremos a versão 8.3 instalada em pouco tempo. O Ubuntu irá buscar o postgresql em seus servidores e o instalará e configurará para nós:

Detalhes

Diretório de dados - /var/lib/postgresql/8.3/main/base

Diretório do pg_hba.conf e do postgresql.conf - /etc/postgresql/8.3/main

autovacuum.conf - /etc/postgresql-common

pg_dump, pg_dumpall, psql e outros binários - /usr/bin

Acessando pelo prompt

Trocar a senha do super-usuário

```
sudo passwd postgres
```


Acessando a console psql

```
sudo -u postgres psql
```

Instalando através dos Fontes

Este método de instalação dá um pouco mais de trabalho mas em contrapartida é o que oferece um maior controle e uma maior possibilidade de customização.

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 (opcionais, para autenticação)

Veja como encontrar esses requisitos no Ubuntu:

```
sudo apt-get install build-essential libreadline5-dev zlib1g-dev gettext
```

Ou faça o download do site: <http://packages.ubuntu.com/> e instale com:

```
dpkg -i nome.deb
```

Ou então duplo clique no gerenciador de arquivos Nautilus.

Obs.: estes pacotes podem mudar de nome devido ao aparecimento de novas versões.
E use make ao invés de gmake.

Download - <http://www.postgresql.org/ftp/source/>

Descompacte em /usr/local/src e instale no diretório default, que é /usr/local/pgsql.

```
sudo tar xzpvf postgresql-8.3.1.tar.gz -C /usr/local/src  
cd /usr/local/src/postgresql-8.3.1
```

Caso já tenha o postgresql instalado no Ubuntu, pule as etapas de criação do grupo e do usuário na instalação abaixo, como também terá que alterar a porta no script postgresql.conf, logo após a criação do cluster com o comando initdb e, no caso, os comandos deverão passar também a porta, por exemplo:

```
bin/createdb -p 5433 bdteste  
bin/psql -p 5433 bdteste.
```

Configurar, Compilar e Instalar

```
sudo ./configure
sudo make
sudo make install
sudo groupadd postgres
sudo useradd -g postgres -d /usr/local/pgsql postgres
sudo mkdir /usr/local/pgsql/data
sudo chown postgres:postgres /usr/local/pgsql/data
sudo passwd postgres
su - postgres
bin/initdb -D /usr/local/pgsql/data
bin/pg_ctl -D /usr/local/pgsql/data start
bin/createdb teste
bin/psql teste
```

Isso irá criar um cluster com encoding UTF-8 (default do Ubuntu)

Caso prefira iso-8859-1, antes de executar o initdb, exporta a variável LANG:

```
export LANG=pt_BR.iso-8859-1
bin/initdb --encoding latin1 -D /usr/local/pgsql/data
```

Copiar o script de inicialização "linux" para o /etc/init.d

```
sudo cp /usr/local/src/postgresql-8.3.0/contrib/start-script/linux /etc/init.d/postgresql-8.3.0
```

Dar permissão de execução ao script:

```
sudo chmod u+x /etc/init.d/postgresql-8.3.0
```

Adicionar ao Path

```
su - postgres
gedit /etc/bash.bashrc (e adicione a linha abaixo):
PATH=/usr/local/pgsql/bin:$PATH
```

16) Módulo 2 - Aula Extra - BLOBs

Um grande objeto é um método de armazenamento (não um tipo de dados) que habilita o PostgreSQL a armazenar colunas grandes separadamente da tupla destinada a registrar os dados. Um objeto grande aparece como um OID dentro da tupla.

Uma coluna queimada (TOASTed column) é uma coluna onde o servidor automaticamente armazena valores grandes fora da tupla para vários tipos. O fato da coluna ser queimada é invisível à camada do SQL.

Objetos grandes não são necessários tanto quanto foram antes do recurso (TOAST) ser introduzido na versão 7.1. No entanto, objetos grandes podem ser acessados com interfaces cliente e servidor em C através de operações com arquivos do tipo Unix como open, close, lseek, read, e write. Objetos queimados (TOASTED) são sempre tratados como objetos inteiros.

O problema de objetos grandes é que eles não são armazenados separadamente de forma automática e devem ser manipulados separadamente dos outros valores no registro. Isto significa que você deve usar comandos especiais para inserir, atualizar e remover grandes objetos. O pg_dump também possui argumentos especiais para extrair e restaurar objetos grandes.

Caso você esteja armazenando imagens em um banco de dados onde esteja lendo-as e mostrando-as, então, provavelmente, o tipo bytea irá funcionar muito bem. A imagem, caso seja muito grande, deveria ser queimada (TOASTED). Se você possui uma aplicação de gerenciamento de imagens que manipula ou analisa o conteúdo das imagens, então o armazenamento de objetos grandes seria a melhor alternativa. Com o armazenamento de objetos grandes você poderia procurar por um pedaço específico, ler e gravar as alterações usando a interface em C. Se você utiliza uma simples coluna do tipo bytea (queimado), você seria obrigado a ler a imagem inteira e gravá-la também inteira, mesmo que a alteração fosse em um só bit.

As interfaces em C também são úteis como funções do lado do servidor para análise de dados. Sua função em C do lado do servidor que reconhece uma imagem pode varrer e procurar por padrões ou aplicar transformações na imagem (morphing) sem necessitar transferir qualquer parte da imagem para o cliente. A interface cliente pgsql, também suporta a funcionalidade de objetos grandes.

Para criar uma tabela que contenha um objeto grande, defina a coluna com o tipo OID. O OID será o identificador do objeto grande neste caso.

O próximo problema é como pegar um objeto grande do seu sistema de arquivo para ser armazenado dentro da tabela. O seu objeto grande se encontra do lado cliente ou servidor? Um arquivo do lado cliente é um arquivo que vive na máquina onde a parte cliente está rodando. Esta distinção é válida somente se o cliente estiver numa máquina diferente de onde estiver instalado o banco de dados. Um arquivo do lado servidor é um arquivo que reside na mesma máquina que o servidor de banco de dados.

Há uma diferença entre carregar arquivos do lado servidor e carregar arquivos do lado cliente. Para arquivos do lado servidor, você pode invocar a função lo_import() na sua instrução INSERT. Todo o trabalho é executado pelo servidor de banco de dados.

Para arquivos do lado cliente, o cliente tem que ter o trabalho de abrir o arquivo e enviá-lo para o servidor. Isto é feito no programa cliente. Isto também pode ser feito usando psql (que é um programa cliente especial) utilizando dois passos. Primeiro, importe o objeto grande e então insira seu OID no devido registro da tabela utilizando o valor da variável :LASTOID do psql.

Quando você selecionar uma linha que contenha um objeto grande, o que você verá na coluna do objeto grande será um número OID. Para que acesse o conteúdo do objeto grande, você deve também extraí-lo para um arquivo ou abri-lo e manipulá-lo através da sua interface cliente ou de uma função do servidor.

Para extrair a imagem da tabela mypictures para a máquina servidora, use `lo_export()`. Especifique a coluna do objeto grande e o arquivo destino de saída. O arquivo de destino de saída deve estar liberado para ser gravado pelo banco de dados do servidor e pertencerá ao dono do processo do banco de dados.

Para extrair a imagem de dentro de um arquivo na máquina cliente usando o `psql`, é preciso saber alguns truques. Sem o recurso de um shell script, você deve prestar atenção com um olho mágico no valor da coluna da figura e usá-lo na declaração `\lo_export`. (Se alguém souber como fazer isso somente com SQL e `psql` por favor, me avise!)

O código a seguir mostra a criação de uma tabela que contém um objeto grande. Então, carrega uma imagem a partir da máquina servidora e exporta uma cópia dela na máquina servidora. Então, carrega uma imagem a partir da máquina cliente e exporta uma cópia dela na máquina cliente.

```
--
-- Minha tabela de figuras.
--
CREATE TABLE mypictures (
    title TEXT NOT NULL primary key,
    picture OID);

-- A figura de Rosas Vermelhas está na Máquina Servidora
-- Carrega e exporta uma cópia na Máquina Servidora
--
INSERT INTO mypictures (title, picture)
VALUES ('Red Roses', lo_import('/tmp/redroses.jpg'));
SELECT lo_export(picture, '/tmp/redroses_copy.jpg') FROM mypictures
WHERE title = 'Red Roses';

--
-- A figura de Rosas Brancas está na Máquina Cliente
-- Carrega e exporta uma cópia na Máquina Cliente
--
\lo_import '/tmp/whiteroses.jpg'
INSERT INTO mypictures (title, picture) VALUES ('White Roses', :LASTOID);

SELECT * from mypictures;

--      title      | picture
-- +-----+-----+
-- Red Roses      | 3715516
-- White Roses    | 3715518
-- (2 rows)

\lo_export 3715518 '/tmp/whiteroses_copy.jpg'
```

Tudo isso funciona, no entanto, o que acontece quando você apaga uma linha ou atualiza um objeto grande? Apagando uma linha (registro) de uma tabela do modo usual apaga a linha, no entanto, o objeto grande é deixado pendente. Você pode dizer que o objeto largo ainda existe usando

\lo_list

no psql. Este comando lista todos os OIDs de objetos grandes conhecidos no sistema.

Por outro lado, às vezes o objeto grande não é deixado pendente porque outros registros referenciam a mesma imagem. Como os objetos grandes são, bem, grandes, pensou-se que seria melhor somente tratar os identificadores dos objetos, o OID. Aquele OID pode ser referenciado por outros registros se o desenvolvedor não quiser múltiplas cópias do mesmo objeto grande e aquele OID também pode estar envolvido em uma transação no mesmo ou em diferentes registros. Isto é um problema que o desenvolvedor deve resolver.

Neste exemplo, nós assumimos que quando um registro é removido, então, queremos também remover o objeto grande. Isto implica que nunca haverá outro registro se referenciando ao mesmo objeto grande.

Para fazer isso use uma regra (rule) para chamar a função lo_unlink() do servidor. Usando as interfaces de objetos grandes é possível gravar em objetos grandes e modificá-los. Quando você estiver usando somente SQL, então você precisa descartar (drop) e substituir o objeto grande. Isto também poderia ser feito por uma regra. Assim que você apontar um novo valor para o objeto grande, ele somente não aponta mais para o antigo. Mas faz isso somente se receber um novo valor.

```
CREATE RULE droppicture AS ON DELETE TO mypictures
DO SELECT lo_unlink( OLD.picture );

CREATE RULE reppicture AS ON UPDATE TO mypictures
DO SELECT lo_unlink( OLD.picture ) where OLD.picture <> NEW.picture;
```

Para verificar se aquela regra está funcionando, no psql selecione um registro da tabela e observe o registro o qual você quer remover. Use \lo_list para mostrar os objetos grandes. Descarte (remova) o registro de maneira usual. Selecionando da tabela, voce sabe que o registro da tabela foi removido e, usando \lo_list você pode ver que o objeto grande correspondente também foi removido.

```
=# select * from mypictures;
   title   | picture
-----+-----
White Roses | 3715592
Red Roses   | 3715593
(2 rows)

=# update mypictures set picture=lo_import('/tmp/redroses_copy.jpg')
   where title='Red Roses';
   lo_unlink
-----
1
(1 row)

=# \lo_list
      Large objects
   ID      | Description
-----+-----
3715592 |
3715598 |
(2 rows)

=# select * from mypictures;
   title   | picture
```

```
-----+-----  
White Roses | 3715592  
Red Roses   | 3715598  
(2 rows)
```

ATENÇÃO: Note que se você não apontar mais um objeto grande que está sendo referenciado por um registro existente E você possui estas regras no lugar, você não pode descartar (remover) o registro. Você deve descartar a regra, remover o registro e recriar a REGRA. A variável LO_TRANSACTION que determina a maneira pela qual se trata operações com objetos grandes não tem mais efeito para esta finalidade e **não existe mais na versão 7.4**.

Colaboradores: elein em varlena.com

Fonte: http://www.postgresql.org.br/Blobs_%28ou_como_armazenar_arquivos_dentro_do_banco%29

17) Encoding

Alteração de encoding em sistemas Debian

Autor: Andre Luiz Facina <andre.luiz.facina@gmail.com>

Data: 03/06/2008

Alteração de encoding em sistemas Debian

Essa dica é muito útil para sistemas Debian/Ubuntu mais novos que, por padrão, instalam o sistema em UTF-8.

Adicionar a entrada pt_BR ao arquivo /var/lib/locales/supported.d/local:

```
# echo "pt_BR ISO-8859-1" >> /var/lib/locales/supported.d/local
```

Aqui ficou assim no Ubuntu 7.10:

```
pt_BR ISO-8859-1
```

```
#pt_BR.UTF-8 UTF-8
```

```
#en_US.UTF-8 UTF-8
```

Adicionar o alias pt_BR em /etc/locale.alias:

```
# echo "pt_BR pt_BR.ISO-8859-1" >> /etc/locale.alias
```

Aqui ficou:

```
...
```

```
# it with the rest of us. Send it using the `glibcbug' script to
```

```
# bugs@gnu.org.
```

```
pt_BR      pt_BR.ISO-8859-1
```

```
bokmal     nb_NO.ISO-8859-1
```

```
bokmål     nb_NO.ISO-8859-1
```

```
catalan    ca_ES.ISO-8859-1
```

```
...
```

Modificar o locale padrão nos arquivos /etc/environment e /etc/default/locale:

```
LANG="pt_BR"
```

```
LANGUAGE="pt_BR:pt:en"
```

Meu /etc/environment

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"  
#LANG="pt_BR.UTF-8"  
#LANGUAGE="pt_BR:pt:pt_PT"  
  
LANG="pt_BR.ISO-8859-1"  
LANGUAGE="pt_BR:pt"
```

Meu /etc/default/locale

```
LANG="pt_BR.ISO-8859-1"  
#LANG="pt_BR.UTF-8"  
LANGUAGE="pt_BR:pt"  
#LANGUAGE="pt_BR:pt:pt_PT"
```

Executar os comandos abaixo como root:

```
# locale-gen  
# dpkg-reconfigure locales
```

Pronto :)

Agora reinicie.

Original de André Luiz Facina

<http://www.vivaolinux.com.br/dicas/verDica.php?codigo=10510>

18) Usando o PostGIS

http://www.webgis.com.br/postgis/docs/capitulo4_Usando_PostGIS.htm

Criando um banco de dados espacial com PostgreSQL + PostGIS

O Postgis é um excelente módulo espacial para o PostgreSQL. Serve tanto a aplicativos para publicação web, como o Mapserver, quanto para uso em intranets, com Grass, QGis etc. Seu diferencial é a possibilidade de executar análise espacial consultas que somente um Banco de Dados SQL pode!

OBS: esse artigo foi escrito originalmente para constar no portal [VivaoLinux](#). Foi postado lá, mas devido à imensa fila de espera e à vontade de que esse artigo fosse para o ar, decidi colocá-lo aqui também. Além, claro, de um detalhe importante: aqui é copyleft! :)

Criando um banco de dados espacial com PostgreSQL + PostGIS

Primeiro How-To adicionado! [Criando um banco de dados espacial com PostgreSQL + PostGIS](#)

Introdução

Mapas digitais. Eis uma daquelas áreas intrigantes, que cada vez mais contam com programas e alternativas em software livre.

É uma área tradicionalmente dominada por grandes corporações de software, que vendem programas bons, mas geralmente acima da casa das dezenas de milhares de dólares... Como então nós, simples mortais, ou apenas moradores de países subdesenvolvidos, podemos fazer nossos mapas no computador, seja para publicar na internet, seja para imprimir, seja para o que for?

Antigamente, a solução era fazer na mão, braçalmente nos editores de imagem ou então tentar entrar, de alguma forma, no restrito e anti-ético mercado do mapeamento digital - por ser restrito, mil vezes mais que o mercado de software tradicional.

Atualmente, há diversas soluções... e a cada dia aparece uma nova. Quer publicar na internet? Mapserver (<http://mapserver.gis.umn.edu/>). Quer fazer geoprocessamento pesado? Grass (<http://grass.itc.it/>) é uma ótima solução. Ou então, usar o Spring - mas sempre lembrando que nesse caso, como diz nosso compa Uchoa, não é Software Livre pois não cumpre as 4 leis básicas do software livre.

Nesse artigo, no entanto, quero explorar outra área, que faz parte também desse universo paralelo do mapeamento digital e do geoprocessamento: Banco de Dados Espacial, quando surge o PostGIS, módulo do PostgreSQL.

O PostgreSQL e o PostGIS tem concorrentes peso pesado... como o Oracle e sua extensão espacial, o "Spatial"... Mas o PostgreSQL é um senhor banco de dados, que isso fique bem claro. E na opinião de muitos, o mais parrudo e potente BD livre da atualidade.

Desde já aviso: é um artigo grande, pois é praticamente impossível fazer alguma coisa sem compreender alguns conceitos básicos. Inicialmente pensei até em subdividi-lo, mas achei que não haveria problema em mantê-lo grande, como uma referência unificada sobre o tema.

Durante todo ele, são instalados os seguintes programas:

- PostgreSQL 8.x
- Postgis 1.x
- libGeos / libGeos-dev
- proj4
- qGis 0.74

Simbora?

Por que um banco de dados espacial?

• Raster x Vector

Uma primeira distinção a ser feita, quando trabalhamos com mapas digitais, é a existência de dados vetoriais e dados raster.

Raster

O dado Raster é uma image, um mapa de pixels - um bitmap - com informações de cor em todos os pontos.

Uma imagem de 4x4 pixels, por exemplo:

```
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
```

Imagens de satélite e fotos aéreas são exemplos de dados Raster.

Vector / Vetor

Para informações mais "simples", utilizam-se vetores, que são arquivos infinitamente mais leves e fáceis de trabalhar, para determinados tipos de mapeamentos.

São possíveis formas de armazenamento de dados. Por exemplo, uma farmácia poderia ser definida com um ponto. Mas uma reserva florestal precisaria ser definida como um polígono... e uma estrada, como uma linha.

Shape x Banco de dados Espacial

Dentre os diversos formatos para armazenar mapas, o Shape, da ESRI, é o mais popular, servindo como um intercâmbio entre os vários programas. No entanto, é um formato proprietário. O mapa "Brasil", por exemplo, teria três arquivos:

- brasil.shp (contém a informação vetorial)
- brasil.shx (contém os índices)
- brasil.dbx (banco de dados associado ao vetor)

No entanto, como podemos ver acima, o banco de dados está separado do arquivo que contém o mapa. Isso significa que, para operações mais complexas, é necessário possuir um aplicativo que faça essa integração, permitindo a realização de joins entre tabelas e análise espacial.

Num banco de dados espacial, isso fica muito mais simples, pois a informação do vetor é apenas mais uma coluna, do tipo "geometry". Seria algo como:

BRASIL

```
| cod_estado | nome_estado | uf | pop_1991 | geometry |  
| 01 | Amazonas | AM | 4099021 | polygon  
(010600000000100000000103000000001000000E2000000093E00...) |
```

Nesse número gigantesco estão as coordenadas (latitude e longitude) do polígono do estado, no caso, Amazonas - em um hexadecimal legível para o Postgis.

Padrões

Segundo o [Open Geospatial Consortium - OGC](#), são definidos 7 tipos para armazenamento, em WKT (well-known text):

- * Ponto - POINT(0 0)
- * Linha - LINESTRING(0 0,1 1,1 2)
- * Polígono - POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- * Multiponto - MULTIPOINT(0 0,1 2)
- * Multilinha MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- * Multipolígono - MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- * Coleção de geometrias - GEOMETRYCOLLECTION(POINT(2 3),LINESTRING((2 3,3 4)))

Projeções cartográficas

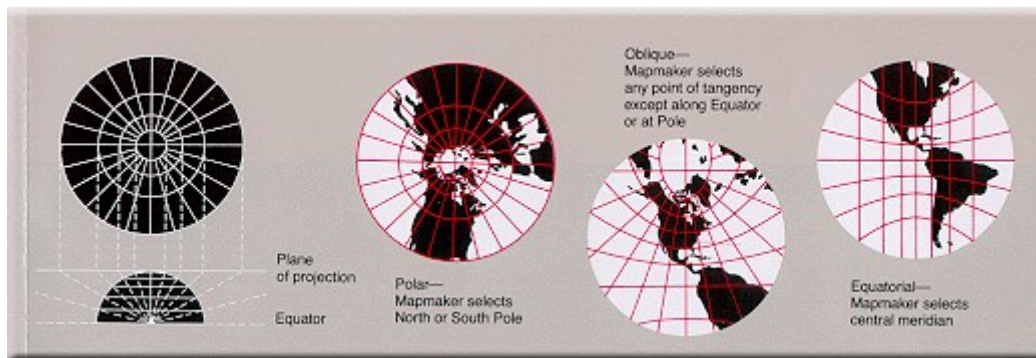
- PROJEÇÕES CARTOGRÁFICAS

Todo mapa possui uma projeção. Lembra das aulas de geografia? UTM, Mercator etc?

Então! Todo mapa é projetado pois, na verdade, a Terra é redonda, 3D, e o mapa tradicional - por enquanto - é 2D, bidimensional. A projeção assim é uma forma de "esticar" a realidade para fazê-la caber numa folha de papel.

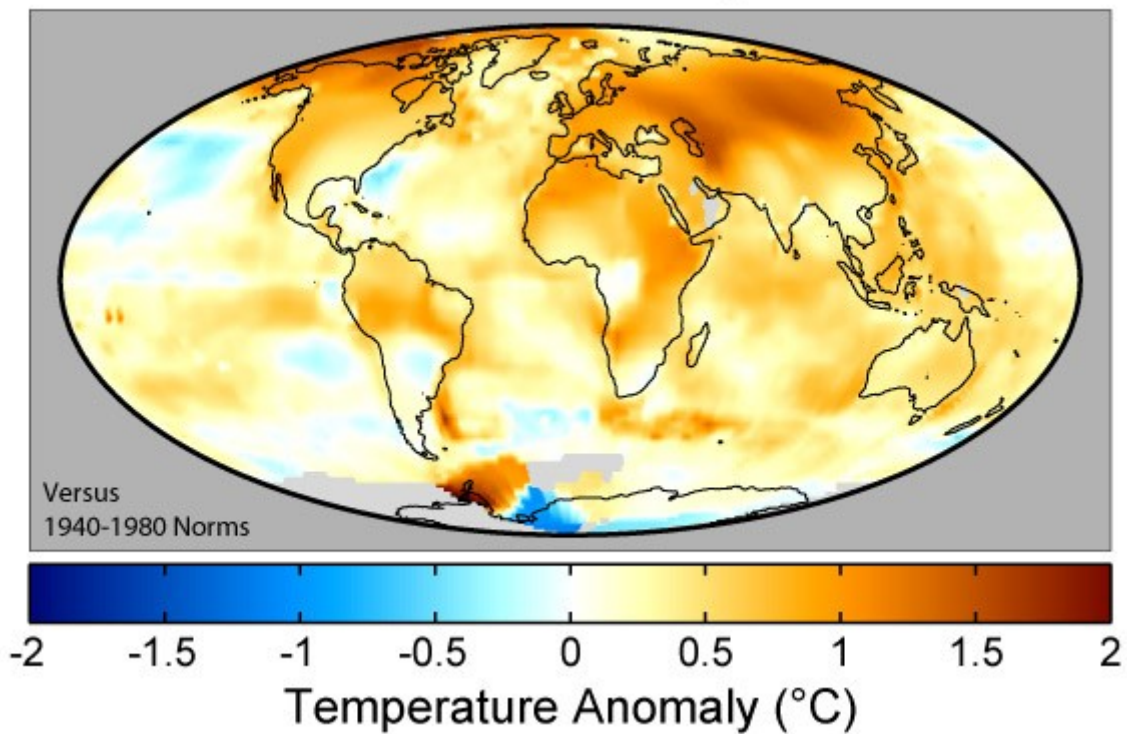
projeção de Mercator - fonte: (<http://wikipedia.org>)

Sendo assim, cada projeção "estica" - ou melhor, PROJETA - a Terra de forma diferente. E cada uma dessas projeções vai destacar uma coisa e perder em outra - distorções de todas as formas. Lembra daquele mapa em que a Groenlândia era maior que o Brasil?



(Projeção Gnomonica - privilegia as distâncias)

1995-2004 Mean Temperatures



(Projeção Mollweide - privilegia as áreas)

fonte: [Wikipedia](http://en.wikipedia.org/wiki/Map_projection)

Existem projeções que privilegiam a representação correta das áreas. Outras, das distâncias. Outras, das formas... mas cada tem as suas distorções. Já tentou esticar uma bola daquelas "dente de leite", quando furam? Então, é a mesma coisa! Informação adicional pode ser encontrada em lugares como a Wikipedia (http://en.wikipedia.org/wiki/Map_projection)

Para isso, existe uma biblioteca excelente chamada Proj - PROJ.4 - Cartographic Projections Library (<http://www.remotesensing.org/proj/>)

Existe no apt, quando só é preciso:

```
# apt-get install proj
```

A biblioteca pode tanto ser usada em linha de comando como por outros programas. Por aqui, só vamos citá-la e lembrar que ela é importante para um banco de dados espacial - poderemos, desta forma, reprojeter em um mesmo mapa camadas de dados com diferentes projeções.

Análise espacial

- Um dos objetivos mais requisitados quando se deseja fazer um Banco de Dados Espacial é a possibilidade de realizar análises espaciais. O que é isso? Vou dar apenas uma leve introdução ao assunto, já que isso é algo complexo e que é tema de toda a área do Geoprocessamento.

A análise espacial, nesse caso específico, está ligada à possibilidade de fazer pesquisas ligadas à uma análise vetorial. Isto é: é possível determinar as intersecções entre elementos espaciais de camadas diferentes, criar buffers (áreas) a partir de pontos e uma série de outras funcionalidades.

Por exemplo: gostaria de localizar todas as farmácias contidas no bairro da Bela Vista. A resposta não será nem a totalidade da camada "farmácias" e nem a camada "Bairro da Bela Vista", mas o cruzamento entre elas. Ou então, a partir de um ponto, descobrir quais são as farmácias num raio de 500m. E assim por diante!

Há uma infinidade de funções desse tipo, e para elas existe a biblioteca GEOS (<http://geos.refrations.net/>), compatível com o Open Gis Consortiun - OGC.

Você pode pegar diretamente na página ou, no Debian / Ubuntu:

```
$ apt-get install libgeos libgeos-dev
```

Instalo também a libgeos-dev pois vamos compilar o postgis com acesso às funcionalidades da GEOS. Desta forma, é necessária a biblioteca de desenvolvimento - além dos binários. Quem quiser compilar na mão, sem problemas também!

Deixo aqui uma pequena lista de funções disponíveis no PostGis usando essa biblioteca, só pra dar água na boca:

- Crosses
- Touches
- Overlaps
- Relate
- Boundary
- Buffer

Instalando o PostgreSQL e o PostGIS (finalmente!)

- Depois de um pouco de teoria, vamos ao ponto: instalar um banco de dados espacial que faça análise espacial

PostgreSQL

Baixe a última versão do PostgreSQL (<http://www.postgresql.org/download/>)
Em seguida, desempacote:

```
# tar jxvf postgresql-8.x.tar.bz2
```

É fundamental, para utilizar o Postgis, compilar com as seguintes flags:

```
# LDFLAGS=-lstdc++ ./configure CC=/usr/bin/gcc-3.4 [+ as suas opções para o postgresql]
```

Durante a compilação, talvez ele peça a instalação de alguma outra biblioteca, como a readline. Você pode instalar ou então utilizar --without-readline.

Continue com:

```
# make
# make install
# adduser postgres
# mkdir /usr/local/pgsql/data
# chown postgres /usr/local/pgsql/data
# su - postgres
postgres@localhost$ /usr/local/pgsql/bin/initdb -D postgres@localhost$
/usr/local/pgsql/data
postgres@localhost$ /usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1
&
```

Para automatizar a inicialização - isto é, colocar como deamon, copie para

```
# cp /usr/src/postgresql-8.xx/contrib/start-scripts/linux /etc/init.d/postgresql
# chmod +x /etc/init.d/postgresql
# updaterc.d postgresql defaults
```

(rc.d no caso de algumas outras distros)

O PostgreSQL está pronto para receber o PostGIS!

POSTGIS:

Baixe a última versão em <http://postgis.refractory.net/>

Instale:

```
# tar zxvf postgis-1.0.x.tar.gz
```

Mova o diretório para dentro da árvore fonte do PostgreSQL:

```
# mv postgis-1.0.x/ /usr/src/postgresql-8.xx/contrib/
```

Vá até o diretório:

```
#
cd /usr/src/postgresql-8.xx/contrib/postgis-1.0.x/
```

Altere as configurações, se quiser utilizar o Proj e o Geos:

```
#
vi Makefile.config
USE_PROJ ?= 1
PROJ_DIR ?= /usr/lib
##
USE_GEOS ?= 1
GEOS_DIR ?= /usr/lib
```

Salve o arquivo e continue:

```
# make  
# make install
```

O Postgis está pronto. Falta agora criar um banco de dados espacial. É o próximo passo!

Habilitando tabela espacial e carregando seu banco

- Com o Postgis preparado, vamos criar um banco de dados habilitado para dados e consultas espaciais:

```
$ su postgres
```

Crie um usuário para os mapas e um banco de dados:

```
postgres@localhost:/$ createuser usuariomapas  
postgres@localhost:/$ createdb bdmapas -U usuariomapas
```

A partir daí, adicionaremos as funções do Postgis ao banco:

```
postgres@localhost:/$ createlang plpgsql bdmapas  
postgres@localhost:/$ psql -f lwpostgis.sql -d bdmapas
```

O arquivo lwpostgis, nessa instalação, fica no diretório share/contrib/, dentro da árvore do PostgreSQL instalado (no nosso caso, em /usr/local/pgsql/).

Se o Commit deu certo, está pronto: você já tem um banco de dados espacial com o PostGis.

Depois dessas alterações, colete as estatísticas - o que acelera suas consultas:

```
postgres@localhost:/$ vacuumdb -z bdmapas
```

- Carregando seu Banco de Dados**

Existem diversos formatos para armazenar mapas. São arquivos que trabalham com vetores

```
postgres@localhost:/$ /usr/local/pgsql/bin/psql -f postgres@localhost:/$  
/usr/local/pgsql/share/contrib/lwpostgis.sql -d mapas  
postgres@localhost:/$ /usr/local/pgsql/bin/vacuumdb -z mapas
```

usando o loader de shapes

```
postgres@localhost:/$ shp2pgsql [
```

Carregando diretamente (do diretório dos shapes):

```
postgres@localhost:/$ /usr/local/pgsql/bin/shp2pgsql -c QUADRAS_region quadras mapas |  
/usr/local/pgsql/bin/psql -d mapas
```

Ou via SQL dump:

```
postgres@localhost:/$ shp2pgsql -D roads1 roads_table my_db > roads.sql  
postgres@localhost:/$ psql -d my_db -f roads.sql
```


- **Índices**

Os índices servem para acelerar as buscas em uma tabela de banco de dados. No caso de um banco de dados espacial, os índices aceleram muito - às vezes, reduzem o tempo das buscas em até 70%. No entanto, não vale a pena indexar tabelas quando são poucos registros. Isso se chega na tentativa e erro, mas uma regra que costuma valer é que a partir da centena de registros, passa a valer a pena indexar.

O Postgis usa o índice GIST (Generalized Search Tree), que serve para dados irregulares (arranjos de números inteiros, dados espectrais, etc).

Entre em algum cliente SQL (pode ser o phpPgAdmin ou via linha de comando)

phpPgAdmin:

<http://localhost/phppgadmin>

Ou via linha de comando:

```
$ /usr/local/pgsql/bin/psql -d mapas
```

```
CREATE INDEX eixos_idx ON eixos USING gist (the_geom)
```

aonde:

CREATE INDEX -> cria o índice

eixos_idx -> nome para o seu índice

ON eixos -> define a tabela que se quer indexar

USING gist (the_geom) -> escolhe o índice a ser usado (gist) e o campo a que vai ser aplicado (the_geom)

Recomenda-se depois executar:

```
VACUUM ANALYZE nometabela nomecoluna;
```

```
SELECT UPDATE_GEOMETRY_STATS(nometabela, nomecoluna);
```

Onde baixar mapas?

Existem mapas disponíveis para download na página do IBGE, na seção de downloads:

<http://www.ibge.gov.br/>

Testando com qgis

Para finalizarmos, vamos dar uma olhada nos dados, utilizando o cliente de mapas QuantumGis (<http://www.qgis.org>). O qGIS, como também é chamado, tem versões para linux e window\$. Suporta conexões Shape e PostGis, que podem ser mescladas em um mesmo mapa. É possível também abrir conexões com diferentes bancos PostGIS simultaneamente.

Pode ser instalado via apt:

```
# apt-get install qgis
```

É um programa de fácil uso, com interface visual feita em QT. Para criar uma nova conexão, basta:

-> camadas

-> adicionar uma camada PostGIS

Se abre um "assistente" para adição de conexões.

Com o qGIS, pode se editar as cores de um mapa, legendas, mapa de referência, além de fazer consultas simples. Para fazer consultas avançadas, é preciso entrar no assistente dele.

O qGIS tem uma ótima ferramenta que é a exportação de arquivos mapfile, arquivo base para o Mapserver (<http://mapserver.gis.umn.edu/>). O Mapserver é um outro excelente programa, que merece mais outros artigos. Nele é possível criar serviços web (WMS - Web Map Server), publicação de mapas na web entre outras coisas.

Por último, fico devendo uma demonstração de análise espacial no qGis. No entanto, é possível fazer tais consultas via SQL diretamente, resultando um dado espacial ou um atributo.

Por exemplo: quais são as "rodovias pavimentadas" que "cruzam" a cidade "tal"? É possível obter:

- a) uma lista com nomes das rodovias
- b) uma lista de geometrias das citadas rodovias.

O SQL (a) seria algo como:

```
"SELECT nome_rodovia from (select * from rodovias_pavimentadas where  
crosses(rodovias_pavimentada.the_geom, cidade.the_geom) and cidade.nome='São Paulo')  
as foo using unique gid using SRID=-1"
```

É isso! Acredito que os passos iniciais estão aí... existe uma boa oferta de documentação pela internet, reforçado pela grande presença da comunidade lusófona que produz tutoriais e how-to em português. E fica a dica para um próximo artigo: a configuração do Mapserver!

Fernão Lopes Ginez de Lara: <http://www.vivaolinux.com.br/artigos/impressora.php?codigo=4390>

19) Introdução às Redes de Computadores

As informações aqui são bem resumidas, com o intuito de simplificar, portanto caso precise de mais detalhes deve procurar alguma publicação mais detalhada, como o Guia Foca Linux ou um bom livro.

Redes – é a ligação de dois ou mais computadores para compartilharem seus recursos: arquivos, periféricos (impressoras, scanners, etc), etc.

Principais Tipos de Redes:

- LAN (Local Area Network) e
- WAN (Wide Area Network)

As redes **LAN**, são redes locais, com dois ou mais computadores interligados localmente.

As redes **WAN** são formadas por duas ou mais redes LAN separadas ao redor do planeta, podendo estar em salas separadas ou em países distantes (como é o caso da Internet ou de redes de grandes multinacionais ao redor do planeta).

Vários são os elementos que fazem parte de uma rede. Vejamos alguns destes conceitos.

HUBs -Concentrador (também chamado **HUB**) em linguagem de [informática](#) é o aparelho que interliga diversas [máquinas](#) (computadores) que pode ligar externamente [redes TAN](#), [LAN](#), [MAN](#) e [WAN](#).

O Hub é indicado para redes com poucos terminais de rede, pois o mesmo não comporta um grande volume de informações passando por ele ao mesmo tempo devido sua metodologia de trabalho por [broadcast](#), que envia a mesma informação dentro de uma rede para todas as máquinas interligadas. Devido a isto, sua aplicação para uma rede maior é desaconselhada, pois geraria lentidão na troca de informações.



Ribamar FS – <http://pg.ribafs.net> – ribafs@ribafs.net

Switch - Um **switch**, que pode ser traduzido como **comutador**, é um dispositivo utilizado em [redes de computadores](#) para reencaminhar quadros (ou [tramas](#) em Portugal, e *frames* em inglês) entre os diversos nós. Possuem diversas portas, assim como os [concentradores](#) (*hubs*) e a principal diferença entre o comutador e o concentrador é que o comutador segmenta a rede internamente, sendo que a cada porta corresponde um segmento diferente, o que significa que não haverá colisões entre pacotes de segmentos diferentes — ao contrário dos [concentradores](#), cujas portas partilham o mesmo [domínio de colisão](#).



MAC - O **endereço MAC** (do [inglês](#) *Medium Access Control*) é o endereço físico da estação, ou melhor, da [interface de rede](#). É um endereço de 48 [bits](#), representado em [hexadecimal](#). O [protocolo](#) é responsável pelo controle de acesso de cada estação à rede [Ethernet](#). Este endereço é o utilizado na camada 2 ([Enlace](#)) do [Modelo OSI](#).

Exemplo:

00:00:5E:00:01:03

NAT - Em [redes de computadores](#), **NAT**, *Network Address Translation*, também conhecido como *masquerading* é uma técnica que consiste em reescrever os [endereços IP](#) de origem de um pacote que passam por um [router](#) ou [firewall](#) de maneira que um [computador](#) de uma [rede interna](#) tenha acesso ao exterior ([rede pública](#)).

Broadcast - **Broadcast** ou **Radiodifusão** é o processo pelo qual se transmite ou difunde determinada informação, tendo como principal característica que a mesma informação está sendo enviada para muitos receptores ao mesmo tempo. Este termo é utilizado em [telecomunicações](#) e em [informática](#).

Em [Redes de computadores](#), um endereço de *broadcast* é um endereço IP (e o seu endereço é sempre o último possível na rede) que permite que a informação seja enviada para todas as máquinas de uma [LAN](#), [MAN](#), [WAN](#) e [TANS](#), [redes de computadores](#) e [sub-redes](#). A [RFC](#) (Request for comments), [RFC 919](#) é a RFC padrão que trata deste assunto.

Roteador (também chamado **router** ou **encaminhador**) é um equipamento usado para fazer a comutação de [protocolos](#), a comunicação entre diferentes [redes de computadores](#) provendo a comunicação entre computadores distantes entre si.

Roteadores são dispositivos que operam na camada 3 do [modelo OSI](#) de referência. A principal característica desses equipamentos é selecionar a rota mais apropriada para repassar os [pacotes](#) recebidos. Ou seja, encaminhar os pacotes para o melhor caminho disponível para um determinado destino.



Tunelamento - Em [informática](#), a definição de Tunnelling é a capacidade de criar túneis entre duas máquinas por onde certas informações passam.

Em se tratando de um ramo do protocolo [TCP/IP](#), o [SSH](#) e o [Telnet](#), pode-se criar uma conexão entre dois computadores, intermediada por um servidor remoto, fornecendo a capacidade de redirecionar pacotes de dados.

Rede ponto-a-ponto

É um tipo de configuração física de enlaces (links) de comunicação de dados, onde existem apenas dois pontos de dispositivos de comunicação em cada uma das extremidades dos enlaces. Geralmente é utilizado cabeamento Coaxial para realizar essas conexões.

Cliente-servidor é um modelo computacional que separa [clientes](#) e [servidores](#), sendo interligados entre si geralmente utilizando-se uma [rede de computadores](#). Cada instância de um cliente pode enviar requisições de dado para algum dos servidores conectados e esperar pela resposta. Por sua vez, algum dos servidores disponíveis pode aceitar tais requisições, processá-las e retornar o resultado para o cliente. Apesar do conceito ser aplicado em diversos usos e aplicações, a arquitetura é praticamente a mesma.

Uma comunicação é dita **half duplex** (também chamada semi-duplex) quando temos um dispositivo Transmissor e outro Receptor, sendo que ambos podem transmitir e receber dados, porém não simultaneamente, a transmissão tem sentido bidirecional. Durante uma transmissão half-duplex, em determinado instante um dispositivo A será transmissor e o outro B será receptor, em outro instante os papéis podem se inverter. Por exemplo, o dispositivo A poderia transmitir dados que B receberia; em seguida, o sentido da transmissão seria invertido e B transmitiria para A a informação se os dados foram corretamente recebidos ou se foram detectados erros de transmissão. A operação de troca de sentido de transmissão entre os dispositivos é chamada de turn-around e o tempo necessário para os dispositivos chavearem entre as funções de transmissor e receptor é chamado de turn-around time.

Exemplos

- Walk Talkie
- Transmissão de fibra ótica.

Uma comunicação é dita **full duplex** (também chamada apenas duplex) quando temos um dispositivo Transmissor e outro Receptor, sendo que os dois podem transmitir dados simultaneamente em ambos os sentidos (a transmissão é bidirecional). Poderíamos entender uma linha full-duplex como funcionalmente equivalente a duas linhas [simplex](#), uma em cada direção. Como as transmissões podem ser simultâneas em ambos os sentidos e não existe perda de tempo com turn-around (operação de troca de sentido de transmissão entre os dispositivos), uma linha full-duplex pode transmitir mais informações por unidade de tempo que uma linha half-duplex, considerando-se a mesma taxa de transmissão de dados.

Exemplo

Aparelho telefônico; Vídeo Conferência; PCI-Express; Protocolo TCP ([Transmission Control Protocol](#)).

O [cabeamento](#) por **par trançado** (Twisted pair) é um tipo de fiação na qual dois condutores são entrançados um ao redor do outro para cancelar [interferências](#) eletromagnéticas de fontes externas e interferências mútuas ([linha cruzada](#) ou, em inglês, crosstalk) entre cabos vizinhos. A taxa de giro (normalmente definida em termos de giros por metro) é parte da especificação de certo tipo de cabo. Quanto maior o número de giros, mais o ruído é cancelado. Foi um sistema originalmente produzido para transmissão [telefônica](#) analógica que utilizou o sistema de transmissão por par de fios aproveita-se esta tecnologia que já é tradicional por causa do seu tempo de uso e do grande número de linhas instaladas.

Wireless - A tecnologia *wireless* (sem fios) permite a conexão entre diferentes pontos sem a necessidade do uso de cabos - seja ele telefônico, coaxial ou óptico - por meio de equipamentos que usam [radiofrequência](#) (comunicação via ondas de rádio) ou comunicação via [infravermelho](#), como em dispositivos compatíveis com [IrDA](#).

Wireless é uma tecnologia capaz de unir terminais eletrônicos, geralmente computadores, entre si devido às ondas de rádio ou infravermelho, sem necessidade de utilizar cabos de conexão entre eles. O uso da tecnologia wireless vai desde transceptores de rádio como [walkie-talkies](#) até [satélites artificiais](#) no espaço.

Classes de Redes

Classe	Máscara de Rede	Endereço da Rede	
A	255.0.0.0	0.0.0.0	– 127.255.255.255
B	255.255.0.0	128.0.0.0	– 191.255.255.255
C	255.255.255.0	192.0.0.0	– 223.255.255.255
Multicast	240.0.0.0	224.0.0.0	– 239.255.255.255

O tipo de endereço que você deve utilizar depende exatamente do que estiver fazendo.

Referência rápida de máscara de redes

A tabela abaixo faz referência as máscaras de rede mais comuns e a quantidade de máquinas máximas que ela atinge. Note que a especificação da máscara tem influência direta na classe de rede usada:

Máscara (Forma octal)	Máscara (Forma 32 bits)	Número Máximo de Máquinas
Classe A:		
/8	/255.0.0.0	16,777,215
Classe B:		
/16	/255.255.0.0	65,535
/17	/255.255.128.0	32,767
/18	/255.255.192.0	16,383
/19	/255.255.224.0	8,191
/20	/255.255.240.0	4,095
/21	/255.255.248.0	2,047
/22	/255.255.252.0	1,023
/23	/255.255.254.0	511
Classe C		
/24	/255.255.255.0	255
/25	/255.255.255.128	127
/26	/255.255.255.192	63
/27	/255.255.255.224	31
/28	/255.255.255.240	15
/29	/255.255.255.248	7
/30	/255.255.255.252	3
/32	/255.255.255.255	1

Qualquer outra máscara fora desta tabela (principalmente para a classe A), deverá ser redimensionada com uma calculadora de IP para chegar a um número aproximado de redes/máquinas aproximados que deseja.

Referência: Guia Foca Linux - <http://focalinux.cipsga.org.br/guia/avancado/ch-rede.html#s-rede-ip-classes>

Endereços reservados para uso em uma rede Privada

Se você estiver construindo uma rede privada que nunca será conectada a Internet, então você pode escolher qualquer endereço que quiser. No entanto, para sua segurança e padronização, existem alguns endereços IP's que foram reservados especificamente para este propósito. Eles estão especificados no RFC1597 e são os seguintes:

ENDEREÇOS RESERVADOS PARA REDES PRIVADAS			
Classe de Rede	Máscara de Rede	Endereço da Rede	
A	255.0.0.0	10.0.0.0	- 10.255.255.255
B	255.255.0.0	172.16.0.0	- 172.31.255.255
C	255.255.255.0	192.168.0.0	- 192.168.255.255

IP - O endereço IP, de forma genérica, pode ser considerado como um conjunto de números que representa o local de um determinado *equipamento* (normalmente [computadores](#)) em uma [rede privada](#) ou [pública](#).

Máscara de subrede – também conhecida como **subnet mask** ou **netmask**. Uma [subrede](#) é uma divisão de uma rede de computadores.

Classe	Bits iniciais	Início	Fim	Máscara de Subrede padrão	Notação CIDR
A	0	1.0.0.1	126.255.255.254	255.0.0.0	/8
B	10	128.0.0.1	191.255.255.254	255.255.0.0	/16
C	110	192.0.0.1	223.255.255.254	255.255.255.0	/24

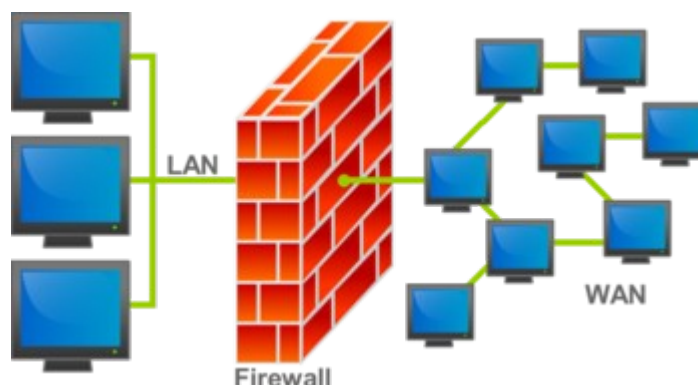
Uma rede “classful” é uma rede que possui uma máscara de subrede 255.0.0.0, 255.255.0.0 ou 255.255.255.0.

Gateway – Endereço IP do servidor.

DNS – Domain Name System (Sistema de Nomes de Domínios) é um sistema de gerenciamento de nomes [hierárquico](#) e distribuído operando segundo duas definições:

- Examinar e atualizar seu [banco de dados](#).
- Resolver nomes de servidores em endereços de rede (IPs).

Firewall é o nome dado ao dispositivo de uma [rede de computadores](#) que tem por objetivo aplicar uma política de segurança a um determinado ponto de controle da rede. Sua função consiste em regular o tráfego de dados entre redes distintas e impedir a transmissão e/ou recepção de acessos nocivos ou não autorizados de uma rede para outra. Este conceito inclui os equipamentos de [filtros de pacotes](#) e de [proxy](#) de aplicações, comumente associados a redes [TCP/IP](#).



Squid - o Squid é um popular [servidor Proxy](#) em [software livre](#). Um dos melhores softwares para a função do mercado. Seu uso é variado, ele pode esconder petições

repetidas, esconder [www](#), [DNS](#), e outros recursos de [rede](#) compartilhados para um grupo de pessoas. É projetado principalmente para rodar em sistemas [UNIX](#).

IPTables - O **netfilter** é um módulo que fornece ao [sistema operacional Linux](#) as funções de [firewall](#), [NAT](#) e [log](#) de utilização de [rede de computadores](#).

iptables é o nome da ferramenta do [espaço do usuário](#) que permite a criação de regras de *firewall* e NATs. Apesar de, tecnicamente, o iptables ser apenas uma ferramenta que controla o módulo netfilter, o nome "iptables" é frequentemente utilizado como referência ao conjunto completo de funcionalidades do netfilter. O iptables é parte de todas as [distribuições](#) modernas do Linux.

Serviço	Porta						
PostgreSQL	5432						
MySQL	3306						
SSH	22						
FTP	21 e 20						
POP	110						
IMAP	143						
SMTP	25						
TELNET	23						
HTTP	80						
HTTPS	443						

Velocidade de Redes Ethernet

10 MBPs	TCP/IP
100 MBPs	RFC
1 GBPs	INTERNIC (http://www.internit.com)
10 GBPs	UDP

Upgrade de Versões no PostgreSQL

Existem dois tipos de Upgrade: entre versões menores e entre versões maiores.

Entre Versões Menores

Exemplo: temos a 8.3.1 e desejamos fazer upgrade para a 8.3.2 ou 8.3.3.

Neste caso apenas baixamos a versão mais recente e atualizamos sem dificuldades.

No Windows existe um arquivo .bat para a atualização.

Se através dos repositórios em Linux, basta instalar a versão mais recente.

- Parar a versão atual
- Instalar os binários (atualizar) da nova versão
- Iniciar a nova versão

Entre Versões Maiores

Exemplo: temos a 8.2.1 e desejamos fazer upgrade para a 8.3.2 ou 8.3.3.

Neste caso precisamos efetuar um backup de todos os bancos na versão atual, instalar a versão nova e restaurar o backup na versão nova.

- Na versão atual:
`pg_dumpall > backup_full.sql`
- Parar o servidor da versão atual
- Mover a pasta do PostgreSQL atual para outro lugar
- Instalar o PostgreSQL versão mais recente e criar o cluster
- Restaurar as alterações do postgresql.conf e o pg_hba.conf
- Iniciar o novo PostgreSQL
- Restaurar os dados do anterior:
`psql -U postgres -d postgres -f backup_full.sql`

Essas diferenças de upgrade ocorrem devido a que entre versões maiores ocorrerem alterações internas no formato do sistema de tabelas e dos arquivos de dados.

Nas versões menores a equipe apenas adiciona correções de bugs

Initdb

Por padrão, ao executar o initdb, se a opção "-U foo" é informada, o usuário foo (do SGBD) será o superusuário do PostgreSQL; caso não tenha opção -U, ele definirá o \$USER (do SO) como superusuário do PostgreSQL, ou seja, o nome do superusuário do PostgreSQL será o mesmo do nome do usuário do SO que executou o initdb. E mais, ao inicializar um agrupamento (aka cluster) (com initdb), você é obrigado a criar um superusuário como também é o caso ao instalar um SO (que por padrão é o usuário root). No PostgreSQL, o nome do superusuário é definido no momento do initdb; ele não é fixo. Os usuários comuns do SGBD podem ser criados após a inicialização do agrupamento.

Euler