13) Contribs

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 parta 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. sslinfo
F.29. tablefunc
F.30. test_parser
F.31. tsearch2
F.32. uuid-ossp
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;
      p
----------------
 :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 oid`	show info for table with OID *oid*
`-f filenode`	show info for table with filenode *filenode*

Switch	Description
`-t` *`tablename_pattern`*	show info for table(s) matching *`tablename_pattern`*
`-s`	show tablespace OIDs
`-S`	include system objects (those in `information_schema`, `pg_toast` and `pg_catalog` schemas)
`-i`	include indexes and sequences in the listing
`-x`	display more information about each object shown: tablespace name, schema name, and OID
`-q`	omit headers (useful for scripting)
`-d` *`database`*	database to connect to
`-H` *`host`*	database server's host
`-p` *`port`*	database server's port
`-U` *`username`*	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 mostras 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 mostras 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

Aenção: 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 sescala 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
`-i`	Required to invoke initialization mode.
`-s` *`scale_factor`*	Multiply the number of rows generated by the scale factor. For example, `-s 100` will imply 10,000,000 rows in the `accounts` table. Default is 1.
`-F` *`fillfactor`*	Create the `accounts`, `tellers` and `branches` tables with the given fillfactor. Default is 100.

Table F-15. pgbench benchmarking options (fase dos testes)

Option	Description
`-c` *`clients`*	Number of clients simulated, that is, number of concurrent database sessions. Default is 1.
`-t` *`transactions`*	Number of transactions each client runs. Default is 10.
`-N`	Do not update `tellers` and `branches`. This will avoid update contention on these tables, but it makes the test case even less like TPC-B.
`-S`	Perform select-only transactions instead of TPC-B-like test.
`-f` *`filename`*	Read transaction script from *`filename`*. See below for details. `-N`, `-S`, and `-f` are mutually exclusive.
`-n`	No vacuuming is performed before running the test. This option is *necessary* if you are running a custom test scenario that does not include the standard

Option	Description
	tables `accounts`, `branches`, `history`, and `tellers`.
`-v`	Vacuum all four standard tables before running the test. With neither `-n` nor `-v`, pgbench will vacuum `tellers` and `branches` tables, and will remove all entries in `history`.
`-D` *varname=value*	Define a variable for use by a custom script (see below). Multiple `-D` options are allowed.
`-C`	Establish a new connection for each transaction, rather than doing it just once per client thread. This is useful to measure the connection overhead.
`-l`	Write the time taken by each transaction to a logfile. See below for details.
`-s` *scale_factor*	Report the specified scale factor in pgbench'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 `branches` table. However, when testing custom benchmarks (`-f` option), the scale factor will be reported as 1 unless this option is used.
`-d`	Print debugging output.

Table F-16. pgbench common options (útil em ambas as fases)

Option	Description
`-h` *hostname*	database server's host
`-p` *port*	database server's port
`-U` *login*	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 verfsão 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 PostgresSQL".
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?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 Korry 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;