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 seqüencialmente 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 registro 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. Imaginne 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 restora 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 exempo 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 efetudas 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 comecamos 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/walls
000000010000000000000000
000000010000000000000001
000000010000000000000002
 000000010000000000000003
000000010000000000000004
```

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 baclup-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 simplesmete 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 terminanos 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
000000010000000000000000
000000010000000000000001
000000010000000000000002
000000010000000000000003
000000010000000000000004
000000010000000000000005
000000010000000000000005.002EA2B8.backup
000000010000000000000006
000000010000000000000007
000000010000000000000008
000000010000000000000009
00000001000000000000000A
00000001000000000000000B
00000001000000000000000C
00000001000000000000000D
00000001000000000000000E
```

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 destroido) 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 -xzvf 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
2:38:00pm 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 iniciamo 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 teste3. 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 parace com isso:
$ cat $PGDATA/recovery.conf
restore_command= 'cp /tmp/walls/%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 Postgreql quando parar.Se voce quer recuperar todas as mudancas simplesmente omita 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 "000000010000000000000006" from archive
LOG: restored log file "000000010000000000000007" from archive
LOG: restored log file "000000010000000000000008" from archive
LOG: restored log file "000000010000000000000009" from archive
...
LOG: archive recovery complete
LOG: database system is ready

Quando o processo de recuperacao se completa, Postgresql renomeiao 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_commando 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 Korry 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 seqüência 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 a 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 seqüencialmente,

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 inclui-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 000000010000123400055CD, o arquivo de histórico de cópia de segurança vai ter um nome parecido com 000000010000123400055CD.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 despender 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 seqüencialmente 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