Conheça os novos recursos do PostgreSQL 8

O passo do elefantinho

Foi lançada há pouco tempo a nova versão do PostgreSQL, importante banco de dados de código aberto. No final das contas, foram reunidos tantos recursos novos e necessários que o pulo no número da nova versão parece justificável: PostgreSQL 8.

Por Joachim Wieland

A concorrência está animada e aumenta entre os bancos de dados livres: há mais de um ano o antigo *SAP DB* está disponível sob uma licença livre, distribuído pela MySQL AB sob o nome de *MaxDB*. Em agosto do ano passado, a IBM e a Computer Associates lançaram quase ao mesmo tempo seus produtos *Cloudscape* e *Ingres* sob uma licença de código aberto. Os bancos de dados de código aberto há muito estabelecidos precisam preparar sua tática contra a antiga concorrência comercial. O *PostgreSQL* vem mostrar sua estratégia nessa disputa com sua recém-lançada versão 8. Duas das inúmeras novidades saltam especialmente aos olhos dos administradores de banco de dados, pois são recursos que somente estavam disponíveis em produtos para a elite: conjuntos de objetos de banco de dados (*tablespaces*) e restauração a partir dos logs de transação (*Point-In-Time Recovery*).

Divide et impera

Os bancos de dados não deixam os discos rígidos em paz. Por isso, um truque recomendado é distribuir os acessos de leitura e escrita de operações concomitantes em muitos discos. Além disso, discos complementares ajudam a equilibrar a falta de espaço. Até agora, não era uma tarefa desejável gerenciá-los com o PostgreSQL, pois havia somente um diretório central para os dados. Não era possível utilizar a capacidade de armazenamento disponível em qualquer lugar. Nesta nova versão, o administrador pode lançar mão de tablespaces. Eles atuam como um contêiner de objetos do banco de dados que se integram em qualquer quantidade em qualquer local de armazenamento. Com isso, o administrador pode tanto distribuir a carga de I/O como também aproveitar melhor o espaço disponível.

Além de bancos de dados completos, também pode-se incluir, de acordo com a necessidade, somente esquemas, tabelas ou índices individuais. A divisão da tabela e de seu índice correlato por diferentes discos rígidos atua especialmente como uma melhoria de desempenho. Assim, os cabeçotes de leitura e gravação do disco pularão menos e podem se movimentar de um setor para outro suavemente. Quando se inserem, em discos diferentes, tabelas associadas entre si com vínculos, obtém-se um efeito semelhante.

Isso funciona bem quando uma grande quantidade de linhas é devolvida. Neste caso, o planejador reconhece que uma verificação de índice é mais lenta que a leitura seqüencial (`EXPLAIN` ou `EXPLAIN ANALYZE` exibem o cálculo de custo do planejador). Caso os locais de armazenamento das tabelas logicamente vinculadas estejam no momento fisicamente separados, as operações paralelas de leitura não se impedem mutuamente e podem fornecer uma taxa de transferência máxima.

Quem já conhece o recurso das tablespaces de outros bancos de dados (como o *Oracle*) irá se surpreender, pois o PostgreSQL não conhece nenhum parâmetro que determine seu tamanho. O banco de dados atribui às tablespaces somente um caminho. Assim elas alocam somente a memória que realmente será utilizada.

O administrador cria um novo tablespace com o seguinte comando SQL:

```
CREATE TABLESPACE nome_do_tablespace ➥
[OWNER nome_do_usuário]LOCATION caminho
```

Agora ele pode colocar um banco de dados, um esquema, uma tabela ou um índice no contêiner recém criado. Veja o exemplo a seguir:

```
CREATE TABLE teste (a INT, b INT) ➥
TABLESPACE hd_master;
```

Garantia de depósito

Como qualquer banco, os bancos de dados também protegem seus "depósitos" da maneira mais eficiente possível contra perdas. A maioria deles utiliza uma estratégia com vários passos: de modo geral, os backups salvam a base de dados completa em mídias seguras. A partir dali, é possível recuperá-las por completo após uma falha do sistema. Os assim chamados arquivos de log de transação (veja o quadro **Arquivos de log de transação**) registram como complemento todas as alterações nos arquivos de dados. Por meio deles pode-se cancelar operações interrompidas e não finalizadas ou com erro (*rollback*).

Sem precauções adicionais, os arquivos de log de transação cresceriam de maneira ininterrupta, acabando por exceder a capacidade do sistema de arquivos. Via de regra, o banco de dados evita esse excesso com um processo de rotação: novos registros sobrescrevem os mais antigos. Esse processo limita o espaço utilizado; entretanto, restringe também o histórico de alterações documentados nos logs. Caso ele não retorne até o último backup, há uma paradoxo no conceito de segurança: todos os dados que foram alterados nos logs entre a cópia de segurança completa mais atual e a mais antiga não poderão ser reconstruídos após uma perda total de dados. O PostgreSQL 8.0 encara esse risco por meio de um processo chamado "restauração a partir dos logs de transação" (*Point-In-Time Recovery*). Ele protege todos os logs de transação antes que sejam sobrescritos e assegura, dessa forma, que todo o estado do banco de dados possa ser reconstruído a partir do final do último backup. Os erros de usuário são registrados "de carona" (quem nunca esqueceu o `WHERE` num comando `DELETE`?).

Apesar de ser um recurso novo, a restauração não é complicada. O administrador executa primeiro o último backup completo do diretório `data`, bem como de todos os tablespaces utilizados. Então, copia no diretório `data` o arquivo `recovery.conf`, no qual ele indica onde os arquivos de log de transação se encontram e até que ponto a restauração deve chegar. A seguir, um exemplo desse processo:

```
restore_command = 'cp /var/lib/pgsql/➥
archive/%f %p'
recovery_target_time = '2004-11-24 ➥
15:15:00.0000'
```

Como período para o processo de refazer (*rollforward*), pode-se definir tanto um ID de transação como também – conforme o exemplo – um período de tempo determinado. Caso o PostgreSQL inicie e encontre o arquivo descrito, ele entra no modo de restauração, reconstrói o banco de dados e renomeia o arquivo de controle de `recovery.conf` para `recovery.done`. A **listagem 1** exibe um trecho da mensagem de uma restauração desse tipo.

Intercâmbio de dígitos

Uma outra novidade no PostgreSQL 8 ataca um problema que não pode ocorrer num banco de dados na fase de operações. Geralmente é um indício da falta de um plano, de competência de quem fez a modelagem ou de um teste do banco depois de pronto: a alteração posterior dos tipos de coluna. Para fazer isso, normalmente os analistas inserem uma nova coluna na tabela, já com o tipo apropriado. Depois, os dados convertidos da coluna a ser alterada são então copiados para a nova. Por último, apaga-se a antiga coluna e atribui-se o nome original à nova coluna.

A versão 8 do PostgreSQL "dá uma colher de chá" e assume automaticamente esse trabalho manual. Basta usar o comando mostrado a seguir: ➥

Listagem 1: Restauração a partir dos logs de transação (Point-In-Time Recovery)

```
01 LOG:  Iniciar reconstrucao do pacote
02 LOG:  comando_restauracao = »cp /var/lib/pgsql/archive/%f %p«
03 LOG:  momento_restauracao = 2004-11-24 15:15:00+01
04 LOG:  Arquivo_log do pacote reconstruído »000000010000000000000007.00EB5DD4.backup«
05 LOG:  Arquivo_log do pacote reconstruido »000000010000000000000007«
06 LOG:  Ponto de verificacao: 0/7EB5DD4
07 LOG:  Refazer em 0/7EB5DD4; Desfazer em 0/0; Shutdown FALSE
08 LOG:  Proximo ID_transacao: 1827; proximo OID: 131918
09 LOG:  Executa reconstrucao automatica
10 LOG:  Refazer inicia em 0/7EB5E10
11 LOG:  Arquivo_log do pacote reconstruido »000000010000000000000008«
12 LOG:  Arquivo_log do pacote reconstruido »000000010000000000000009«
13 LOG:  Reconstrucao finaliza antes do término da transacao 1921, horario 2004-11-24 15:15:08 CET
14 LOG:  Refazer finalizado em 0/9D980F4
15 LOG:  Novo timeline-ID escolhido: 2
16 LOG:  Reconstrucao do pacote fechada
17 LOG:  sistema_banco_de_dados esta pronto
```

```
ALTER TABLE ALTER COLUMN … TYPE …[USING …]
```

Desse modo, o sistema naturalmente pode assumir uma conversão automática desde que possa distinguir entre o tipo de dados original e o novo.

Para todos os outros casos... não, não precisa puxar o MasterCard: para eles está disponível o parâmetro `USING`. Aqui, é uma expressão quem vai definir as regras da conversão.

Essa expressão pode ser formada de maneira extremamente flexível: o desenvolvedor pode incorporar tanto outra coluna da tabela como também funções de conversão pré-definidas ou automaticamente escritas.

A **figura 1** ilustra um exemplo que converte uma tabela chamada *historia* com uma coluna chamada *ano* sobre a função definida pelo usuário `converter_de_romano()`, que transforma algarismos romanos (*VarChar*) em números decimais (*Int*).

Mantendo a Segurança

Com a versão 8.0, novas possibilidades surgem também para os desenvolvedores de aplicativos de banco de dados. O suporte a pontos de restauração (*save points*) é uma opção bastante interessante. Essa ampliação permite que se insira, em uma nova transação, outras *subtransações*. Quando o banco de dados encontra uma definição `SAVEPOINT nome`, ele marca o *status* alcançado até aquele momento. As operações subseqüentes podem ser canceladas posteriormente por meio do comando `ROLLBACK TO SAVEPOINT nome` dentro da transação até esse ponto congelado. Caso não seja mais necessário um save point, então é possível liberá-lo usando o comando `RELEASE SAVEPOINT nome` (Com isso, são cancelados também todos os save points anteriormente definidos). Veja o exemplo a seguir para compreender melhor o princípio:

Listagem 2: Manipulação de Exceção em PL/PgSQL

```
01 CREATE FUNCTION exemplo_excecao(int) RETURNS int AS $$
02 DECLARE
03   param ALIAS FOR $1;
04 BEGIN
05   BEGIN
06     INSERT INTO … VALUES (…)
07   EXCEPTION
08     WHEN INTEGRITY_CONSTRAINT_VIOLATION THEN
09       -- Constraints como, por exemplo, UNIQUE/NOT NULL/CHECK… ou uma
10       -- chave externa foram violados.
11       …
12     WHEN UNIQUE_VIOLATION OR FOREIGN_KEY_VIOLATION THEN
13       -- é possível também uma codificação mais exata
14       …
15     WHEN DIVISION_BY_ZERO THEN
16       -- esses tipos de erro também podem ser rastreados
17       …
18     WHEN OTHERS THEN
19       -- todos os erros não explicitamente rastreados…
20       RAISE EXCEPTION 'Erros em exemplo_excecao: %", param;
21   END;
22   RETURN 1;
23 END;
24 $$ LANGUAGE plpgsql;
```

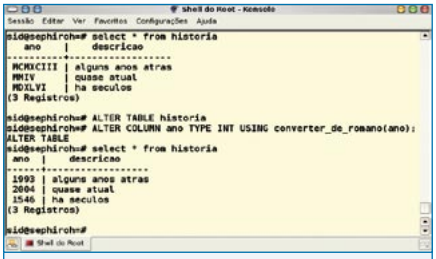

Figura 1: O PostgreSQL 8.0 oferece suporte mais sofisticado para as alterações posteriores do tipo de dados de uma coluna.

```
BEGIN TRANSACTION;
INSERT INTO tabela VALUES (1);
SAVEPOINT sp;
INSERT INTO tabela VALUES (2);
ROLLBACK TO SAVEPOINT sp;
INSERT INTO tabela VALUES (3);
COMMIT;
```

Neste exemplo foram inseridos na tabela os valores de 1 a 3. Entretanto, o comando `ROLLBACK TO SAVEPOINT sp;` cancela a inserção do valor 2.

Esse recurso é especialmente útil no caso de transações mais longas. O que acontecia nas versões anteriores era que uma transação precisava ser cancelada por completo sempre que um comando nela contido produzisse um erro. Em caso de dúvida, o cliente precisava reiniciá-la. Agora ele pode retornar a uma transação no ponto certo de restauração.

Essa possibilidade foi incorporada também no *PL/PgSQL*, uma linguagem procedural que pode ser escrita por meio de comandos SQL, mais as variáveis e a lógica de programação. Não é à toa que o nome se assemelha ao PL/SQL do Oracle. Nesta versão, pelo conceito de save points, as funções do PL/PgSQL fornecem suporte também para exceções graças às quais um erro em uma função pode ser rastreado e trabalhado adequadamente. Na versão anterior, um erro encerrava não somente a própria função em si, mas também a transação corrente. A **listagem 2** contém a estrutura de uma função PL/PgSQL com manipulação de exceção.

Simplesmente complexo

De longe, um ponto forte do PostgreSQL são as funções de servidores livres programáveis em diferentes linguagens. Além do SQL e do PL/PgSQL, o desenvolvedor pode escolher entre PL/Perl, PL/Python e PL/Tcl. As linguagens PL/Java, PL/R, PL/Ruby e PL/sh não estão disponíveis na distribuição padrão, mas podem ser encontradas em [1], [2], [3] e [4]. Há até mesmo uma linguagem PL/PHP, que ainda se encontra em desenvolvimento.

A PL/Perl foi estendida na versão 8.0 nas assim chamadas *Set Returning Functions*, inclusive funções que podem retornar não uma, mas várias linhas de resultado. Com o também novo suporte de SPI (*Server Programming Interface* – Interface de programação do servidor), o acesso a outros objetos do banco de dados é possível por meio das funções PL/Perl. Elas podem retirar por si só comandos do SQL e analisar os resultados. Além disso, o programador agora pode definir gatilhos (*triggers*) nessas linguagens.

Especialmente para essas funções torna-se interessante a possibilidade de receber ou retransmitir tipos complexos (compostos) como parâmetros. A versão 8.0 oferece suporte para esses tipos de dados em alguns locais nos quais, anteriormente, só eram permitidos tipos escalares. Assim, esse suporte permite que o construtor `row()` crie tipos compostos a partir dos mais simples. Dessa forma, os três tipos escalares `integer`, `text` e `real`, por meio de `row(3, 'PostgreSQL', 47.11)`, podem ser resumidos em um valor composto.

Listagem 3: Tipos de dados compostos e PL/Perl

```
01 CREATE TYPE ender AS (nome text, rua text, cidade text, cep int);
02
03 CREATE TABLE pessoas (pid serial, endereco ender);
04 INSERT INTO pessoas (endereco)
05      VALUES (row('Heinz Meier', 'Am Hang 13', '94849 Irgendwo', NULL));
06
07 CREATE OR REPLACE FUNCTION lies_cep(endereco) RETURNS endereco AS $$
08   my $endereco = $_[0];
09   my $cidade = $endereco->{'cidade'};
10   $cidade =~ s/^\s*(\d*)\s+//;  # descarta o prefixo numérico
11
12   return { nome => $endereco->{'nome'},  rua => $endereco->{'rua'},
13            cidade => $cidade, cep => $1 || 0 };
14 $$  LANGUAGE plperl;
15
16 SELECT lies_cep(endereco) FROM pessoas;
```

O exemplo da **listagem 3** ilustra o manuseio de tipos compostos em PL/Perl. Em primeiro lugar, define-se o tipo de dado `endereco`; em seguida aplica-se uma tabela com esse tipo de dados e nela é inserido um valor. Esse valor corresponde, justamente, à definição de tipo, porém mistura nomes de cidades e códigos postais no campo `cidade` e deixa o campo `CEP` vazio. Portanto, no exemplo é aplicada também a função `lies_cep()` que corrige esse erro: ela recebe um valor pelo tipo `adresse`, extrai o CEP do campo de nomes, copia-o no campo predefinido para ele e devolve o conjunto de dados corrigido na forma de valor pelo tipo `adresse`:

Os exemplos na **listagem 2** e na **listagem 3** demonstram também o caractere cifrão. Até então, as funções sempre eram fechadas por apóstrofes, o que trazia a desvantagem de se precisar trocar todas as apóstrofes (') por aspas duplas ('') na função completa. Com isso, uma linha (*string*) vazia se tornava uma seqüência composta de apóstrofes (''''). Com os caracteres cifrão é possível definir funções (e também todas as outras variáveis de texto) dentro de `$$ … $$` e, entre eles, utilizar aspas normais.

Arquivos de log de transação

Os arquivos de log de transação de um banco de dados funcionam como um relatório ou diário de bordo do sistema de dados. No log são registradas todas as alterações planejadas com relação aos dados (tabelas, índices, entre outras). Em curtos espaços de tempo elas são transferidas do log para a base de dados. Caso o computador trave durante um processo de gravação é possível determinar, com base nos arquivos de log de transação, quais alterações já foram executadas e quais ainda estão pendentes. Essa informação permite que a transação seja completada ou cancelada. Em todos os casos, a consistência dos dados permanece protegida.

Além disso, uma outra vantagem é o aumento da velocidade, pois é mais fácil anexar dados seqüenciais aos arquivos de log de transação do que posicionar o cabeçote do disco rígido continuamente sobre locais exatos nos arquivos de dados. Os arquivos de log de transação também podem ser armazenados em dois discos rígidos. Antes do PostgreSQL 8.0, os arquivos de log de transação eram sobrescritos após um certo tempo. Agora, pode-se arquivá-los e não somente completar um número reduzido de informações deles originadas, como ainda executar todas de um período determinado (*Point-In-Time Recovery*).

Nos bastidores

Entre as mudanças da nova versão do PostgreSQL há muitas que beneficiam os usuários normais de banco de dados, embora não tenham contato direto com elas. Por exemplo, o planejador e o otimizador foram melhorados e também utilizam um índice quando os tipos de dados neles contidos não são idênticos aos tipos da consulta, mas podem ser convertidos.

Da mesma forma, a *caching* do banco de dados foi revista. Nesta versão, ela

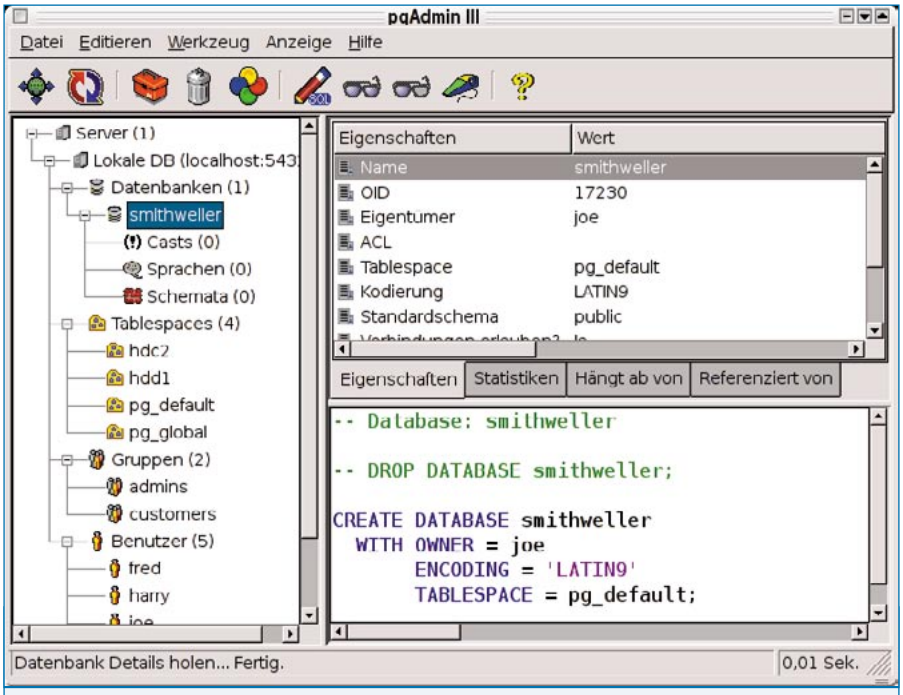

Figura 2: A ferramenta gráfica de administração *pgAdmin III* foi especialmente adaptada para a nova versão do PostgreSQL.

SQL programado

No servidor do banco de dados, os procedimentos armazenados (*stored procedures*) aliviam o cliente de muitos subcálculos e geralmente levam a um ganho no desempenho. O PostgreSQL oferece particularmente muitas possibilidades de programação deste código em muitas linguagens.

Um exemplo retirado da prática: Um servidor de email implementa como antispam o método da lista negra (*black list*). Para tanto, ele utiliza um banco de dados que administra uma lista branca (*white list*) para endereços IP, uma para remetentes e outra para destinatários. Devem valer as seguintes regras: cada email será rejeitado temporariamente por quatro minutos. Cada combinação de endereço de IP de remetente e destinatário expirará após 12 horas, caso entre somente uma mensagem para essa combinação. Caso haja mais emails, a combinação deverá ser apagada 30 dias após a última atividade.

Todas essas regras precisam ser incluídas no servidor de email ou em um programa externo. Entretanto, há a alternativa de facilmente armazenar um procedimento no servidor de banco de dados sem intermediários que considere as listas brancas, administre os instantes em que os emails são enviados e saiba após quanto tempo um conjunto de dados expira. Dessa forma, o servidor de email consulta em uma única função (`DeferGreylisting(ip, sender, recipient)`) se o email deve ser aceito ou ignorado por enquanto. A função verifica todas as condições de acordo com o registro do banco de dados, sem que com isso uma transferência de dados entre o servidor de email e o servidor de banco de dados se acumule.

No PostgreSQL, o PL/PgSQL é a bola da vez para essa tarefa. Neste caso, trata-se de SQL com construtores de programação, inclusive cláusulas IF/ELSE e loops e, a partir da versão 8.0, também com manipulação de exceção. Além disso, as funções nessa linguagem são pré-compiladas, o que resulta em uma execução muito rápida. A desvantagem neste caso é que o programador quase não tem acesso aos módulos já disponíveis.

Caso ele queira aproveitar módulos pré-construídos, é possível migrar para uma outra linguagem no PostgreSQL, como a PL/Perl. No exemplo, ele poderia querer que o banco de dados exibisse em uma seleção, além do endereço de IP, o nome do *host* associado. Isso pode ser feito por meio de uma função PL/Perl que utilize o módulo *Net::DNS*; isso possibilitaria também ao banco de dados a execução de verificações de DNS (*DNS lookups*).

considera, além do momento do último acesso a uma página de armazenamento, também a freqüência dos acessos no passado. Ambos os valores determinam quais páginas devem ser mantidas no cache e quais devem ser descartadas. Esse novo algoritmo é chamado de ARC (*Adaptive Replacement Cache* – Cache de Substituição Adaptativa). Anteriormente era usado somente um algoritmo chamado LRU (*Least Recently Used* – Menos Usados Recentemente).

Nas versões anteriores, as páginas de armazenamento alteradas eram sobrescritas no disco rígido a cada poucos minutos, o que aumentava bastante a atividade do disco naquele momento. Com o novo *Background Writer*, os acessos de gravação se distribuem mais razoavelmente, o que leva a um pico de carga menor e, portanto, a um aumento geral da taxa de transferência de dados do sistema. Além disso, agora o processo de limpeza também "come" menos recursos da máquina, pois não libera mais espaço de armazenamento utilizado. Neste processo, o equilíbrio entre o tempo de execução e o de carga pode ser configurado.

Detalhes tão pequenos...

Mesmo após tudo isso, a lista de novidades do PostgreSQL 8 ainda não se esgotou. Vale mencionar, por exemplo, a rotação de logs incorporada ao programa ou a nova capacidade do *pg_dump* de resolver dependências. Dessa forma, foram criados dumps que o *pg_restore* pode sobrescrever sem que ocorram falhas devido a problemas de dependência. Outra novidade é o *expression index* (índice de expressões), que não só pode indexar os resultados de uma função (o que já era possível antes), mas também trabalhar com quaisquer expressões.

Com freqüência os usuários também perguntam sobre uma ferramenta gráfica de administração do PostgreSQL. Existem

muitas delas, porém poucas são especialmente adaptadas para a versão 8.0, como o *phpPgAdmin* [5] e o *pgAdmin III* [6]. Enquanto o phpPgAdmin tem uma interface web, o pgAdmin é um programa nativo, que graças à utilização de *wxWidgets* funciona tanto em Linux quanto em Windows®. A **figura 2** apresenta uma foto da tela do pgAdmin III.

Desde o lançamento da versão 7.4 do PostgreSQL, há mais de um ano, os desenvolvedores vêm trabalhando duro no quesito padronização: no final de 2003, a nova versão da especificação SQL (chamada *SQL-2003*) foi adotada. Assim, foram introduzidas no PostgreSQL 8.0 diversas alterações que melhoram a conformidade do banco de dados com esse novo padrão. Alguns recursos bastante conhecidos, como por exemplo as seqüências, são agora parte do SQL e podem ser adicionados ao repertório de padrões dos desenvolvedores de bancos de dados. Na especificação SQL-2003, a nova linguagem procedural PL/Java orienta-se também de acordo com a especificação da conexão no servidor Java. A conversão de outros recursos do SQL-2003 está planejada para a próxima versão.

Há um recurso do novo PostgreSQL que causa sensação, principalmente fora do mundo Linux: após incontáveis pedidos, agora esse sistema de banco de dados também tem uma versão para o Windows® (Antes disso, ele rodava somente debaixo do ambiente *Cygwin* [7], que emula uma API Unix sobre o Windows®). A versão nativa para Windows® leva a uma melhora de desempenho considerável. No site *PgFoundry* [8] você encontra um instalador para o PostgreSQL que, após apenas alguns cliques, instala em seu computador, como num "passe de mágica", a versão Windows®. E para facilitar ainda mais as coisas a interface gráfica de administração pgAdmin, da qual já falamos anteriormente, é instalada ao mesmo tempo.

Essa nova versão dá um grande passo em direção ao futuro dos sistemas livres de banco de dados. O PostgreSQL 8.0 mostra-se bem equipado para a concorrência com outros SGDBs de código aberto, como o MySQL. Atualmente, quem quer que esteja procurando um bom banco de dados deve olhar com carinho nesse incrível sistema. ■

Informações

[1] PL/Java: gborg.postgresql.org/project/pljava/projdisplay.php

[2] PL/R: www.joeconway.com/plr

[3] PL/Ruby: moulon.inra.fr/ruby/plruby.html

[4] PL/sh: developer.postgresql.org/~petere/pgplsh

[5] phpPgAdmin: phppgadmin.sourceforge.net

[6] PgAdmin III: www.pgadmin.org/pgadmin3/index.php

[7] Cygwin, implementação da API POSIX para sistemas Windows®: www.cygwin.com

[8] Instalador do PostgreSQL para Windows® pgfoundry.org/projects/pginstaller

[9] História da SQL: www.mcjones.org/System_R/SQL_Reunion_95/sqlr95.html

[10] Comparação entre as diferentes implementações de SQL: http://troels.arvin.dk/db/rdbms/

[11] Site oficial do PostgreSQL: www.postgresql.org

[12] PostgreSQL na Wikipedia: en.wikipedia.org/wiki/PostgreSQL

[13] Guia de otimização do PostgreSQL: www.varlena.com/varlena/GeneralBits/Tidbits/perf.html

Sobre o autor

Joachim Wieland é formado em Tecnologia da Informação pela Escola Superior Técnica da Renânia –Vestfália – Aachen (RWTH-Aachen), na Alemanha, e foi infectado pelo "vírus" PostgreSQL quatro anos atrás.