PHP & PostgreSQL
最新バージョン詳細情報

PostgreSQL 7.2 の新機能

日本PostgreSQLユーザー会 理事長 石井達夫
ISHII Tatsuo　ishii@postgresql.jp

PostgreSQL 7.2登場！

2002年2月にリリースされたPostgreSQL 7.2は可用性を高め，大規模データへの対応を進めるなど，ミッションクリティカルな用途にも積極的に対応することを狙った最新バージョンです．

10ヶ月ぶりのメジャーバージョンアップ

PostgreSQL 7.1がリリースされたのは2001年4月ですから，7.2は10ヶ月ぶりのメジャーバージョンアップということになります編集注．期待されていたレプリケーションこそ実装されませんでしたが，VACUUMがロックされなくなるなど，大規模システムでPostgreSQLを運用する際に問題になっていた点が多数改善されています．7.2での改善点は多岐にわたりますが，誌面の都合もありますので，本稿では主に可用性や性能といった重要な点での改良点を見ていきます．

7.1と7.2の違いの詳細については，SRAのページ

http://osb.sra.co.jp/

にも記述があります．

ロックしないVACUUM

PostgreSQLは削除があってもすぐにデータを削除するのではなく，単にそこに「削除された」という印だけを付けます．更新も同様で，すぐにそのデータを書き換えるのではなく，古いデータに削除された印を付けた後新しいデータを追加します．このような方式を追記型記憶管理と呼びます．これによってMVCC（Multi Version Concurrency Control，Oracleでいうところの「読み取り一貫性」）を実現し，かつクラッシュ後のデータベース再起動処理を簡潔にしています．

しかし，追記型記憶管理においては，放っておくと削除された領域がずっと残り続け，ディスク領域を圧迫するという問題点があります．それだけではなく，物理的にテーブルファイルの大きさが肥大しますので，性能にも大きな影響を与えるようになります．

この問題を解決するのがVACUUM注1です．VACUUMはPostgreSQL独自のSQLコマンドで，VACUUMを実行すると削除領域が取り除かれ，そのテーブルが圧縮されます．

これまでのVACUUMの欠点

ただし，VACUUMにも欠点があります．

①VACUUMの実行中は処理対象のテーブルにロックがかかり，同じテーブルをアクセスするとVACUUMが終わるまで待たされる

②VACUUM処理は削除領域の大きさに比例して処理時間がかかるので，更新や削除が多数行われた大規

編集注）今年4月3日，PostgreSQL 7.2のバグ修正版であるPostgreSQL 7.2.1がリリースされました．
注1）「バキューム」と発音します．

模なテーブルにVACUUMをかけると，ときには終了するまで何時間もかかることがある

このため，巨大なテーブルを扱い，かつ更新頻度が高いようなシステムでは24時間連続してPostgreSQLを運用することが困難でした．

新型VACUUM

この問題を一気に解決するのが7.2の新型VACUUMです．7.2では，VACUUM処理の最中であっても検索はもちろん，更新も自由にできます．どこに秘密があるのでしょうか？ 実は，7.2ではVACUUMは削除領域を圧縮するのではなく，単にその領域が再利用可能であることを共有メモリ上のテーブルにマークするだけなのです．更新や挿入で新しいデータが追加される際は，可能ならばその削除領域を再利用します．これによって更新トランザクションが高頻度で処理されるような場合でも，削除領域が次の更新ですぐに再利用されるので，全体として無駄なく記憶領域が利用されることが期待できます[注2]．

7.1までと違って7.2のVACUUMは検索や更新処理を阻害しないので，今までよりもずっと頻繁にVACUUMをかけることができます．つまり，従来せいぜい一晩に1回程度のVACUUMであったのが，1時間に1回，場合によっては10分に1回でもよいかもしれません．より頻繁にVACUUMをかけることにより，PostgreSQLの高い性能を維持することができるわけです．

では，自分のシステムにおいて，どの程度の頻度でVACUUMをかけるのが適当なのでしょうか？ これは，テーブルの中にどの程度削除領域がたまったかが1つの判断になります．7.2では，筆者が作成したcontrib/pgstattuple関数を使って，削除領域の状態を知ることができます．

pgstattupleの使い方

pgstattupleはcontribモジュールであり，標準ではインストールされていません．contribモジュールは，以下のようにしてインストールします[注3]．

```
$ cd postgresql-7.2/contrib
$ make install
$ psql -f pgstattuple/ pgstattuple.sql test[注4]
```

pgstattupleは関数であり，引数としてテーブル名を取り，返り値として削除領域の割合を返します．

図1は，同じcontribモジュールのpgbenchを使い，1000トランザクションを実行した後にpgstattupleを呼び出したときの状態です．ご覧のように，87%以上が削除領域になっています．

ここでVACUUMをかけると図2のようになります．削除領域がなくなり，代わりに再利用可能領域（free/reusable space）が91%以上になったことがわかります．

なお，7.1までのVACUUMと同様に削除領域や再利用可能領域を物理的に削除したい場合は，

```
VACUUM FULL
```

としてください．

●図1　1000トランザクション実行直後

```
test=# select pgstattuple('branches');
NOTICE:  physical length: 0.05MB live tuples:
10 (0.00MB, 0.98%) dead tuples: 900 (0.04MB,
87.89%) free/reusable space: 0.00MB (3.43%)
overhead: 7.70%
 pgstattuple
-------------
   87.890625
(1 row)
```

●図2　VACUUMをかけた後

```
test=# select pgstattuple('branches');
NOTICE:  physical length: 0.05MB live tuples:
10 (0.00MB, 0.98%) dead tuples: 0 (0.00MB,
0.00%) free/reusable space: 0.04MB (91.32%)
overhead: 7.70%
 pgstattuple
-------------
           0
```

注2）インデックス領域については，VACUUMでも不要領域の圧縮が行われませんので，ときどきREINDEXコマンドを使ってゴミ掃除することが必要です．
注3）PostgreSQL 7.2のソースが必要です．
注4）“test”は自分が使用するデータベース名に読み替えてください．

オプティマイザ，統計情報関連の改良

同じSQL文を実行するにしても，結果を取得する方法はいろいろあります．たとえば，テーブルサイズが小さいときはテーブルをくまなく調べるのがもっとも効率が良いのですが，検索結果が少なくテーブルが大きいときは，インデックスを利用するほうが速くなります．こうした考慮を人間が行うのは非常に煩わしいのですが，PostgreSQLには賢い問い合わせオプティマイザが付属しており，そのような計算を自動的に行ってくれます．

しかし，オプティマイザがいくら賢くても，オプティマイザが判断材料にするデータが不十分ではどうにもなりません．7.2では，より緻密にテーブルの統計情報を取得するようになっています．

見積結果の比較

例をお見せしましょう．テーブルt1は，“CREATE TABLE t1(t TEXT);”と定義され，2002年1月1日から2002年10月27日までの300日のすべての日付が文字列として“2001-01-01”のように順に格納されています．ここで，2002年1月1日から2002年5月30日まで全部で何日あるかPostgreSQLに見積もらせてみましょう（正解は150日です）．これが正確にできないと，正確な問い合わせプランが作成できません．

PostgreSQL 7.1では，図3のようになります．rows=100が見積もり行数です．正解は150ですから，かなりずれてしまっています．このケースではわざと日付のデータを文字列として格納していますが，このような場合最小値の“2002-01-01”から最大値の“2002-10-27”まで一様に文字列が分布していると見なし，その仮定のもとで計算します．これが見積もりのずれとなって現われているのです[注5]．

同じことを7.2でやってみましょう．図4をご覧になるとわかるように，非常に正確に見積もりが行われています．これは，最大値と最小値ではなく，10個所のデータをサンプリングしてそれに基づいて計算しているからです．もしも10個所のサンプリングでは不足な場合は，

```
ALTER TABLE t1 ALTER COLUMN t SET STATISTICS
20;
```

でサンプリングを20個所に増やすこともできます．

また，7.2では統計情報を取得するANALYZEがVACUUMから独立しています．ANALYZEコマンドはテーブルの検索や更新を妨げないので，データの更新を大量に行ったら積極的にANALYZEコマンドを発行して常に統計情報を最新にするように心掛けると良いでしょう．

SMP対応の向上

従来PostgreSQLでは，共有メモリ上のデータを管理する際に，スピンロック（spin lock）とセマフォ（semaphore）の2つのしくみが使われてきました．ス

●図3　PostgreSQL 7.1による実行例

```
test=# EXPLAIN SELECT COUNT(*) FROM t1 WHERE t <  '2002-05-31';
NOTICE:  QUERY PLAN:

Aggregate  (cost=6.00..6.00 rows=1 width=0)
  ->  Seq Scan on t3  (cost=0.00..5.75 rows=100 width=0)
```

●図4　PostgreSQL 7.2による実行例

```
test=# EXPLAIN SELECT COUNT(*) FROM t1 WHERE t <  '2002-05-31';
NOTICE:  QUERY PLAN:

Aggregate  (cost=6.13..6.13 rows=1 width=0)
  ->  Seq Scan on t4  (cost=0.00..5.75 rows=150 width=0)
```

注5）日付型のデータをそれ以外の型のデータ型に無理に格納すると，このように不都合が起きることがあります．7.1でもDATE型を使用すればこのようなことは起きません．テーブル設計の際に適切なデータ型を使用するように心掛けましょう．

ピンロックは非常に短時間のロックを管理し，セマフォはトランザクションのロックなどの長時間のロックを管理します．7.2ではその中間に“Light Weight Lock”と呼ばれる第3番目のロックを追加し，きめ細かくロックを管理するようになりました．

SMP[注6]はCPUを複数個使い，性能を向上させようとするものです．従来，PostgreSQLはロックのオーバヘッドのためにCPUの数を増やしても性能が向上しない傾向がありましたが，7.2ではロックを細かく管理したことでその点が改善されています．

図5をご覧ください．このグラフはPostgreSQLコアメンバのひとりTom Lane氏が4CPUのLinuxで測定した結果です．縦軸がTPS[注7]，横軸は同時実行ユーザ数です．下のほうに重なるように表示されているのが7.1.3です．一方上のほうにある2本の線は7.2での測定結果です．ベンチマークの種類によって数値が異なっていますが，7.2では7.1.3の2倍から3倍の性能が得られていることがわかります．

部分インデックス

部分インデックス（partial index）は，通常のインデックスとは違って，列のすべてのデータをインデックスとして持つのではなく，一部の値だけをインデックス化するものです．実際の業務では，検索の対象となるデータというのは意外に限られるものです．

たとえば，商品の売り上げデータを考えてみます．おそらく経営者の興味があるのは，そのうち売上額が一定以上のものです．販売額が10万以上のデータだけにインデックスを付けるには，以下のようにします．

```
CREATE INDEX 売り上げインデックス ON 売り上げ(販売額)
WHERE 販売額 > 10000;
```

もちろんこうしたからといって，販売額が10万に満たないものを検索できなくなるわけではありません．ただインデックスが使えないだけです．

部分インデックスを使うことにより，必要な部分だけにインデックスを作成することができるので，ディスク領域が節約できる他，インデックスが相対的に縮小されることにより，性能の向上も図れます．

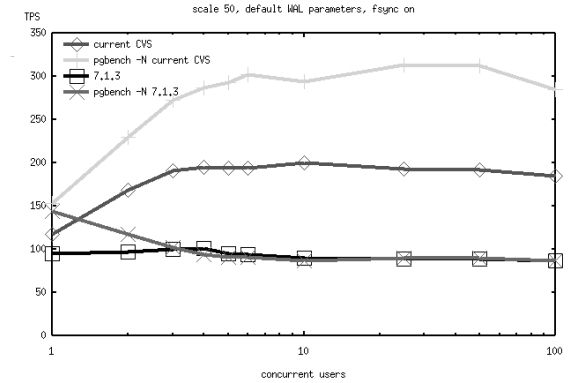
7.1との非互換性

7.1.xから7.2に移行するに当っての注意点はいろいろありますが，比較的はまりやすいところをご紹介しておきます．

CURRENTがなくなった

「現在」をあらわす特殊なキーワードCURRENTが廃止され，かわりにCURRENT_TIMESTAMP，CURRENT_DATE，CURRENT_TIMEのどれかを使うようになりました．

図5／4way SMPマシンにおける7.1と7.2の性能比較



注6）Symmetric Multiple Processor．
注7）1秒間に実行したトランザクション数．

CHAR，VARCHARのサイズ指定

従来CAHR(n)，VARCHAR(n)のnの意味はバイト数でしたが，7.2では文字数になります．これによって文字コードの違いによるバイト数の違いを意識する必要がなくなりましたが，nの意味がバイト数であることに依存しているようなアプリケーションでは注意が必要です．

CHAR，VARCHARのサイズ制限

CHAR(n)，VARCHAR(n)のnを超えてデータを入力すると，7.1までは暗黙のうちにデータを切り捨てていましたが，7.2からはエラーになります．

マルチバイト文字のチェック

明らかに不正なデータを含むマルチバイト文字を入力しようとすると，エラーになるようになりました．7.1で気が付かないうちにSJISのデータをEUC-JPのデータベースに入力していて，7.2に移行しようとするとエラーに遭遇することがあります．このような場合は，pg_dumpの出力テキストを編集するか，7.1のデータベースを修正してからpg_dumpする必要があります．

OIDの存在しないテーブル

7.2では，すべてのテーブルがOIDを持つとは限りません，システムカタログを直接見るようなアプリケーションで，OIDに依存しているような場合は修正が必要になるでしょう[注8]．

まとめ

PostgreSQL7.2は，ロックしないVACUUMを実装し，またSMPシステムへの対応をするなど，可用性とスケーラビリティを大幅に向上させています．今までは小中規模のWebシステムで広く利用されていたPostgreSQLですが，7.2をきっかけに大規模Webシステムでも採用されていくものと思われます．今後の展開が楽しみです．

注8）7.2ではユーザテーブルでもOIDなしのテーブルが作成できます．