実践テクニックをご紹介

徒然PostgreSQL散策

第3回 ユーザ定義関数の魅力

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

Mikheev氏コアメンバを去る

冷夏の後は一転して残暑が続くこの頃ですが（この原稿を書いているのは9月中旬です），みなさま体調などは崩されていませんでしょうか．

今回はちょっと残念なお知らせがあります．PostgreSQL開発チーム創立以来のコアメンバの1人であるVadim Mikheev氏が8月10日をもってコアメンバを抜けました．氏の名前は日本の皆様にはあまりなじみがないかもしれませんが，実はPostgreSQLの大きな機能であるトランザクションログ，行ロック，副問い合わせなどをすべて1人で実装した「超」が付くほどのスーパプログラマです．つまり，現在のPostgreSQLが商用DBに匹敵する機能を持つに至ったのはかなりの部分が氏の貢献によるものです．

氏がコアメンバを抜けた理由は，「自分は小さな改良には興味がない．かといって，大きな改良をするほどの時間が取れない」ということだそうです．

私はほとんどのコアメンバに実際に会ったことがありますが，唯一氏だけにはお会いできなかったのが心残りです．ともあれ，また違った方面での今後の活躍を期待しましょう．

ユーザ定義関数とは

PostgreSQLをお使いの方なら，一度は「ユーザ定義関数」というものに触れたことがあると思います．ユーザ定義関数とは，その名のとおり，データベースの一般ユーザが作成できる関数のことです．ユーザ定義関数はデータベースサーバ側（PostgreSQLの用語で言えば「バックエンド」側）で実行されますが，関数を追加したからといってデータベースサーバを再起動する必要もありませんし，普通にデータベースを運用している最中にユーザ定義関数を作成してもまったく問題ありません．

簡単に言うと，ユーザ定義関数はバックエンド側で実行されるSQLなどの処理を関数の中にパックしたものです．「SQLなど」と言いましたが，PostgreSQLではユーザ定義関数をいくつかの言語で書くことができ，言語によって「など」の部分がだいぶ違ってきます．つまり，言語によってSQL以外のプラスアルファの部分を記述する能力がだいぶ違ってきますが，これについては後で述べます．

ユーザ定義関数のメリット

考えてみると，たとえPostgreSQLのユーザ定義関数を使わなくても，何かデータベースでやりたいことを1つにまとめて関数にするのはアプリケーション側（PostgreSQLの用語で言えば「フロントエンド」側）でも十分可能です．では，ユーザ定義関数を使うメリットはどこにあるのでしょうか？

●複数の言語から使用できる関数を作ることができる

フロントエンド側で書いた関数は，当然のことながらその関数を記述した言語経由でなければ利用できません．たとえば，PHPで作った関数をPerlから使うなどということはできません．その点，ユーザ定義関数はSQLの中で使用するため，データベースにアクセス

できる言語ならばどのようなものからもアクセスできます．したがって，頻繁に使うデータデーベース処理をユーザ定義関数にしておくことにより，開発効率を高めることができます．

●データベースサーバへの通信を節約できる

アプリケーションがSQLを実行する際には，必ずフロントエンドからバックエンドへの通信が発生します．たとえば100行のSQL文を実行しなければならない処理では，100回の通信が発生するわけです．このような処理が1日に1回程度ならまだしも，何度も繰り返されるようだと通信のコストがばかになりません．この100行のSQL文をユーザ定義関数にまとめることにより，大幅に通信コストを削減できます．

●効率的な開発の分担ができる

データベースのエキスパートがユーザ定義関数を書き，アプリケーション開発者がそれを使うことにすれば，プロジェクトのメンバ全員がデータベースに精通する必要がなくなります．

ユーザ定義関数のデメリット

一方ユーザ定義関数を使う上で，注意することもあります．

●移植性が低下する

ユーザ定義関数の構文や機能はPostgreSQL独自のものですから，ほかのデータベースに移行しようとする際には少なからず移植の手間が発生します．

●最適化しにくい

PostgreSQLは問い合わせの中身を実行前に調べ，どうやったらもっともすばやく実行できるかを決定します．これを「最適化」と呼び，最適化を実行するサブシステムを「オプティマイザ」と呼びます．通常のSQL文ではオプティマイザが細かくSQL文をチェックして最適化処理ができます．ところが，オプティマイザから見るとユーザ定義関数の中身はブラックボックスですから，最適化は難しく，大雑把な判断しかできません[注1]．

◇ ◇ ◇

というわけで，ユーザ関数を使うかどうかは，メリット，デメリットを比較した上で慎重に考慮する必要があります……で終わってはあまりにも常識的で，つまりません:-) 実際，ユーザ定義関数は非常にいじりがいがあり，思いもよらぬことができるおもしろさがあります．今回はそうした楽しみの一端をご紹介し，みなさんにユーザ定義関数にはまっていただこうと思います．

プログラムレスユーザ定義関数

PostgreSQLにはC言語で記述するユーザ定義関数があります．といっても別にPostgreSQLにCコンパイラが内蔵されているわけではなく，別途コンパイル済のオブジェクトファイルを用意しておいて，ユーザ定義関数が呼ばれたときにPostgreSQLがそのオブジェクトをロードして呼び出すだけです．ということは，オブジェクトさえあればC言語でプログラムを書かなくてもC関数として呼び出すことができるはずです．たとえば，次のような条件にあてはまる関数を持つオブジェクトファイルが利用できます．

- 共有ライブラリになっていること
- 引数がないこと
- 返り値がスカラーであること

このような条件を満たす関数はUNIX系OS標準のCライブラリにも含まれています．たとえば，getpid()という関数は引数がなく，実行中のプロセスIDを正数で返します．getpid()を呼び出すユーザ定義関数を作ってみましょう．

```
$ psql test
test=> CREATE OR REPLACE FUNCTION getpid()
RETURNS INTEGER AS
'/lib/libc.so.6' LANGUAGE 'c';
CREATE FUNCTION
```

注1）今開発中のPostgreSQL 7.4では，SQL関数の中身まで解析して最適化する機能が追加されています．

「/lib/libc.so.6」の部分はシステムによって異なるかもしれません．

この関数を呼び出す例を示します．

```
test=> SELECT getpid();
 getpid
--------
   2146
(1 row)
```

ところで，PostgreSQLはセッションごとにプロセスを起動するので，同じセッションの中では何回getpid( )を呼び出しても結果は同じです．このような関数では，関数定義時に「IMMUTABLE」（不変）という属性を付けることにより，オプティマイザが効率の良い問い合わせプランを作成してくれます．すなわち，

```
CREATE OR REPLACE FUNCTION getpid()
RETURNS INTEGER AS
'/lib/libc.so.6' IMMUTABLE LANGUAGE 'c';
```

とします．こうすることで，たとえば図1のような問い合わせでインデックスが使われるようになります．

「i = 2146」は，getpid( )がこのセッションの中では常に2146を返すことをオプティマイザが認識して数値に置き換えた結果です．関数を呼び出す必要がなくなり，さらに性能が向上します．

テキスト検索ユーザ定義関数 exttable( )

SELECTはデータベースを検索するものですが，ユーザ定義関数をうまく使うことによって，普通のテキストファイルをあたかもテーブルであるかのようにSELECTで検索することができます．さすがにこのレベルになると，プログラムを書かないわけにはいきませんし，SQLやPL/pgSQLといったユーザ定義関数は言語の記述能力が足りないので，C言語を使う必要性が出てきます．

今回ご紹介するのは，exttable( )というC言語で書かれたユーザ定義関数です．インストール方法は以下のようになります．

```
$ cd /usr/local/src/postgresql-7.3.4/contrib
$ tar xfz /tmp/exttable-0.2.tar.gz
$ cd exttable-0.2
$ make
$ make install
$ psql -e -f exttable.sql test
```

なお，インストールにはPostgreSQL 7.3.xのソースが必要になるので，あらかじめ入手，configureまでしておいてください．PostgreSQLのソースは/usr/local/src/postgresql-7.3.4に展開してあるものとします．

exttable( )自体のソースコードはftp.sra.co.jp/pub/cmd/postgres/exttable/から入手できます．exttable-0.2.tar.gzが現時点の最新版です．これを/tmp/に置いたものとします．

exttable( )の使い方

たとえば図2のような「/tmp/sample.txt」というテキストファイルがあったとします．誌面ではわかりにくいと思いますが，これはCSVのようなフォーマットになっていて，1行が1レコード，1行の中の各項目

▼図1　インデックスを利用するgetpid()関数

```
EXPLAIN SELECT * FROM t1 WHERE i = getpid();
                          QUERY PLAN
---------------------------------------------------------------
 Index Scan using t1_pkey on t1  (cost=0.00..4.82 rows=1 width=4)
   Index Cond: (i = 2146)
(2 rows)
```

▼図2　/tmp/sample.txtの内容

```
PostgreSQL構築・運用ガイド 日経BP 2003/6/25 ISBN 4-8222-2343-4
PHP x PostgreSQLで作る最強Webシステム 技術評論社 2003/1/9 ISBN 4-7741-1647-5
PostgreSQL完全攻略ガイド第3版 技術評論社 2001/6/28 ISBN 4-7741-1226-7
PHP4徹底攻略 改訂版 ソフトバンクパブリッシング 2000/9/30 ISBN 4-7973-2097-4
```

はタブで区切られています．

このファイルをexttable( )で「検索」してみましょう（図3）．exttable( )の第1引数はテキストファイルのファイル名で，第2引数が項目の区切り文字です．今回は「\t」すなわちタブ記号を指定しています．AS以降は各項目の列名と型を指定しています．これは，exttable()が「SET OF RECORD」という実行時に動的に型を決めるための仮想的な型を返すからです．もちろんexttable( )が固定の型を返すようにすることもできますが，テキストファイルの中にどのようなデータが入っているかあらかじめ予測することはできないので，このようにしています．

少々見慣れない形にはなっていますが，これも立派なSELECT文ですから，SELECTで使える構文はすべて利用できます．たとえば，発行日の昇順に並べたければ，図4のようにORDER BYを利用できます．

```
SELECT * FROM 書籍 ORDER BY 発行日;
```

VIEWを使ってさらに使いやすく

いつもこのような長いSQLを入力するのはたいへんなので，VIEWを使ってもう少し使いやすくしてみましょう（リスト1）．これによって「書籍」というVIEWが定義されました．先ほどのSELECT文は次のように簡潔に書くことができます．

区切り文字の利用

exttable( )の第2引数に「:」を与えると，区切り文字が「:」になっているファイルを検索できます．たとえば/etc/passwdがそのような「:」区切りのファイルです（図5）．

コマンドの実行結果をSELECT

コマンドの実行結果をファイルに保存することにより，exttable( )で検索の対象にすることができます．

たとえば，psコマンドの結果を検索するためには，リスト2のようなシェルスクリプトを作成し，exttable( )が理解できる形に整形してからファイルに保存します．「$pid $tty $state $runtime $command」の間の空白は1個のタブ記号であることに注意してください．

図6に実行例を示します．

FIFOの利用

コマンドの出力結果をファイルに出力する方法では，一時ファイルが書き込まれるタイミングとexttable( )がそのファイルをアクセスするタイミングの関係を把握することが難しくなります．この問題は名前付きパイプを使うと簡単に解決できます．

▼図3　exttable( )による検索結果

```
SELECT * FROM exttable('/tmp/sample.txt', '\t') AS 書籍(書名 text, 出版者 text , 発行日 date, ISBN text);
                書名                |          出版者          |   発行日   |        isbn
------------------------------------+--------------------------+------------+--------------------
 PostgreSQL構築・運用ガイド         | 日経BP                   | 2003-06-25 | ISBN 4-8222-2343-4
 PHP x PostgreSQLで作る最強Webシステム | 技術評論社             | 2003-01-09 | ISBN 4-7741-1647-5
 PostgreSQL完全攻略ガイド第3版      | 技術評論社               | 2001-06-28 | ISBN 4-7741-1226-7
 PHP4徹底攻略 改訂版                | ソフトバンクパブリッシング | 2000-09-30 | ISBN 4-7973-2097-4
(4 rows)
```

▼図4　ORDER BYの利用

```
SELECT * FROM exttable('/tmp/sample.txt', '\t') AS
書籍(書名 text, 出版者 text , 発行日 date, ISBN text) ORDER BY 発行日;
                書名                |          出版者          |   発行日   |        isbn
------------------------------------+--------------------------+------------+--------------------
 PHP4徹底攻略 改訂版                | ソフトバンクパブリッシング | 2000-09-30 | ISBN 4-7973-2097-4
 PostgreSQL完全攻略ガイド第3版      | 技術評論社               | 2001-06-28 | ISBN 4-7741-1226-7
 PHP x PostgreSQLで作る最強Webシステム | 技術評論社             | 2003-01-09 | ISBN 4-7741-1647-5
 PostgreSQL構築・運用ガイド         | 日経BP                   | 2003-06-25 | ISBN 4-8222-2343-4
(4 rows)
```

▼リスト1　VIEWの利用

```
CREATE VIEW 書籍 AS SELECT * FROM exttable('/tmp/sample.txt', '\t') AS
書籍(書名 text, 出版者 text, 発行日 date, ISBN text);
```

▼図5　区切り文字による結果の切り分け

```
SELECT * FROM exttable('/etc/passwd', ':') AS passwd(ユーザ名 text, パスワード text , uid integer, gid
integer, gecos text, ホームディレクトリ text, ログインシェル text);
 ユーザ名   | パスワード   | uid  | gid  |        gecos        | ホームディレクトリ     | ログインシェル
------------+--------------+------+------+---------------------+------------------------+----------------
 root       | x            |    0 |    0 | root                | /root                  | /bin/bash
 bin        | x            |    1 |    1 | bin                 | /bin                   | /sbin/nologin
 daemon     | x            |    2 |    2 | daemon              | /sbin                  | /sbin/nologin
 adm        | x            |    3 |    4 | adm                 | /var/adm               | /sbin/nologin
 lp         | x            |    4 |    7 | lp                  | /var/spool/lpd         | /sbin/nologin
 sync       | x            |    5 |    0 | sync                | /sbin                  | /bin/sync
 shutdown   | x            |    6 |    0 | shutdown            | /sbin                  | /sbin/shutdown
 halt       | x            |    7 |    0 | halt                | /sbin                  | /sbin/halt
 mail       | x            |    8 |   12 | mail                | /var/spool/mail        | /sbin/nologin
 news       | x            |    9 |   13 | news                | /var/spool/news        |
 uucp       | x            |   10 |   14 | uucp                | /var/spool/uucp        | /sbin/nologin
 operator   | x            |   11 |    0 | operator            | /root                  | /sbin/nologin
 games      | x            |   12 |  100 | games               | /usr/games             | /sbin/nologin
 gopher     | x            |   13 |   30 | gopher              | /var/gopher            | /sbin/nologin
 ftp        | x            |   14 |   50 | FTP User            | /var/ftp               | /sbin/nologin
 nobody     | x            |   99 |   99 | Nobody              | /                      | /sbin/nologin
 apache     | x            |   48 |   48 | Apache              | /home/httpd            | /bin/false
 named      | x            |   25 |   25 | Named               | /var/named             | /bin/false
 xfs        | x            |   43 |   43 | X Font Server       | /etc/X11/fs            | /bin/false
 rpcuser    | x            |   29 |   29 | RPC Service User    | /var/lib/nfs           | /sbin/nologin
 nscd       | x            |   28 |   28 | NSCD Daemon         | /                      | /bin/false
 sshd       | x            |   74 |   74 |                     | /var/empty/sshd        | /bin/false
 rpc        | x            |   32 |   32 | Portmapper RPC user | /                      | /bin/false
 postfix    | x            |  100 |  101 | postfix             | /var/spool/postfix     |
 pop        | x            |  110 |  110 | Pop Account         | /var/spool/mail        | /bin/false
 squid      | x            |   23 |   23 |                     | /var/spool/squid       | /dev/null
(26 rows)
```

▼リスト2　psコマンドの出力を整形するスクリプト

```
ps x|sed 1d|while read pid tty state runtime command
do
  echo "$pid $tty $state $runtime $command"
done
```

▼図6　psコマンドの結果をSELECT

```
$ sh ps.sh > /tmp/ps.out
$ psql test
test=# SELECT * FROM exttable('/tmp/ps.out', '\t') AS ps(pid integer, tty text, state text,
runtime time, command text);
 pid  |  tty  | state | runtime  |                          command
------+-------+-------+----------+-------------------------------------------------------------
[略]
 1368 | tty1  | S     | 00:00:00 | /bin/sh /usr/X11R6/bin/startx
 1379 | tty1  | S     | 00:00:00 | xinit /etc/X11/xinit/xinitrc --
 1389 | tty1  | S     | 00:00:00 | /usr/X11R6/bin/wmaker
 1439 | tty1  | S     | 00:00:00 | kinput2 -canna -cannaserver localhost
 1530 | tty1  | S     | 00:00:00 | ebview
 1531 | tty1  | S     | 00:00:00 | wmclock -shape
 1539 | tty1  | S     | 00:00:00 | wsoundserver
 1541 | tty1  | S     | 00:00:00 | xmms
 1542 | tty1  | S     | 00:00:00 | xmms
 1543 | tty1  | S     | 00:00:00 | xmms
 1544 | tty1  | S     | 00:00:00 | xmms
 1547 | ?     | S     | 01:12:00 | esd -terminate -nobeeps -as 2 -spawnfd 11
 1549 | tty1  | S     | 00:28:00 | emacs
 1550 | ?     | S     | 00:00:00 | /usr/lib/emacs/20.7/i386-vine-linux/emacsserver
 1583 | pts/0 | S     | 00:00:00 | /bin/bash -i
 1698 | pts/1 | S     | 00:00:00 | /bin/bash -i
 1832 | tty1  | S     | 00:00:00 | xmms
 1833 | tty1  | S     | 00:00:00 | xmms
 1840 | pts/1 | S     | 00:00:00 | sh ps.sh
 1841 | pts/1 | R     | 00:00:00 | ps x
 1842 | pts/1 | S     | 00:00:00 | sed 1d
 1843 | pts/1 | S     | 00:00:00 | sh ps.sh
(36 rows)
```

まず，名前付きパイプを作ります．

```
$ mkfifo /tmp/fifo
```

次にps.shの実行結果を名前付きパイプに書き込みます．

```
$ sh ps.sh > /tmp/fifo
```

すると，プロンプトが返ってきません．これは，誰かが/tmp/fifoを読み出すまでブロックするからです．別の端末を立ち上げ，そこから図7のようにexttable()を実行します．図7は，exttable()の第1引数を名前付きパイプにしただけです．出力結果も先ほどとまったく同様になるはずです．また，先ほどの端末も無事にプロンプトが返ってくるはずです．

では，SELECTを先に実行した場合にはどうなるのでしょうか？ この場合は，名前付きパイプにデータが書き込まれるまでSELECTのほうが待たされます．

ユーザ定義関数とセキュリティ

すでに読者のみなさまもお気づきと思いますが，パスワードファイルの表示をしたり，psコマンドの実行結果を表示したりと，exttable()を使うとかなり危険なことができます．exttable()に限らず，ユーザ定義C関数は強力である反面，危険なセキュリティホールになることがあります．1つの対策としては，このような危険な関数の実行権限をスーパユーザのみに与え，一般ユーザには限定された使い方のみを提供する方法があります．

以下，PostgreSQLのスーパユーザで実行します．

① まずexttable()の実行権限を一般ユーザから剥奪します（**リスト3**）

② 次にexttable()を呼び出すだけの「ラッパー関数」を作ります（**リスト4**）

②でのポイントは「EXTERNAL SECURITY DEFINER」です．関数を作成するときにこのキーワードが指定されると，どのユーザがその関数を実行しても，その関数は関数の作成者（この場合PostgreSQLのスーパユーザ）の権限で実行されます．UNIXで言えば，setuidのようなものですね．試してみましょう．ここで，ユーザfooは一般ユーザです．図8のように，exttable()の呼び出しはエラーになりましたが，exttable()を呼び出しているbooks()は普通に呼び出すことができます．

③ このままではちょっと使いにくいので，VIEWを作るとよいでしょう（**リスト5**）

さいごに

今回はユーザ定義関数を使ってどのようなことができるか，またセキュリティ上の考慮点などを説明しました．次回はexttable()を例にとってユーザ定義C関

▼図7　別端末からexttableを実行

```
$ psql test
test=# SELECT * FROM exttable('/tmp/fifo', '\t') AS ps(pid integer, tty text, state text,
runtime time, command text);
```

▼リスト3　一般ユーザから実行権限を剥奪

```
REVOKE EXECUTE ON FUNCTION exttable(text, char) FROM PUBLIC;
```

▼リスト4　ラッパー関数の作成

```
CREATE TYPE book AS (書名 text, 出版者 text, 発行日 date, ISBN text);

CREATE OR REPLACE FUNCTION books() RETURNS SETOF book
EXTERNAL SECURITY DEFINER AS '
SELECT * FROM exttable(''/tmp/sample.txt'', ''\t'') AS
書籍(書名 text, 出版者 text, 発行日 date, ISBN text);'
LANGUAGE 'sql';

GRANT EXECUTE ON FUNCTION books() TO PUBLIC;
```

▼図8　実行の様子

```
test=# SET SESSION AUTHORIZATION foo;
SET
test=# SELECT * FROM exttable('/tmp/sample.txt', '\t') AS 交通費(書名 text, 出版者 text , 発行日 date,
ISBN text);
ERROR:  exttable: permission denied
test=# SELECT * FROM books();
                書名                |          出版者          |   発行日   |        isbn
------------------------------------+--------------------------+------------+--------------------
 PostgreSQL構築・運用ガイド         | 日経BP                   | 2003-06-25 | ISBN 4-8222-2343-4
 PHP x PostgreSQLで作る最強Webシステム | 技術評論社             | 2003-01-09 | ISBN 4-7741-1647-5
 PostgreSQL完全攻略ガイド第3版      | 技術評論社               | 2001-06-28 | ISBN 4-7741-1226-7
 PHP4徹底攻略 改訂版                | ソフトバンクパブリッシング | 2000-09-30 | ISBN 4-7973-2097-4
(4 rows)
```

▼リスト5　VIEWの作成

```
CREATE VIEW 書籍 AS SELECT * FROM books();
GRANT SELECT ON 書籍 TO PUBLIC;
```

数の作成方法を解説する予定です．また，ある意味ユーザ定義関数の限界に挑戦したとも言えるdblink()というcontribの力作を紹介します．なにせ，PostgreSQL本体にも実装されていない「分散データベース」もどき[注2]をユーザ定義関数で強引に実現してしまったのですから:-) お楽しみに．

注2）もちろん本物の分散データベースとは違って，データベースにまたがるトランザクションの一貫性を保証できているわけではありませんし，分散問い合わせの最適化を行ってくれるわけでもありません．そこが「もどき」という所以です．

C O L U M N

JPUG合宿in湘南2003開催される

日本PostgreSQLユーザ会としてははじめての合宿が，葉山町は「湘南国際村センター」（http://www.shonan-village.co.jp/）で9月22日から23日にかけて開催されました．葉山は風光明媚で知られる観光地ですが，「湘南国際村」は葉山や三浦半島，それに富士山まで見渡せる絶好のロケーションにあります．この恵まれた環境の中で，PostgreSQLのインターナルからコミュニティのあり方まで徹底的に議論しようという企画に20名ほどの方が参加されました．

初日は「PostgreSQLの楽しいhack法」（石井），「PostgreSQLアーキテクチャ解説」（片岡裕生氏）というPostgreSQLのソースコードレベルの解説がありました．

▲会場の様子

夜は事務局の部屋に全員が集合，2時近くまでアルコールを片手に熱い議論が戦わされました．また「家が近所」というPHPユーザ会の廣川類氏が飛び入りで論戦に参加，「これはもしかしたらPHPの合宿？」と思うほどPHPに関する濃い話題も交されました．

2日目は注目のレプリケーションツール「PGCluster」を開発されている三谷篤氏のプレゼンがありました．氏は深夜に渡る懇親会の後も徹夜でPGClusterのバグ修正と機能追加に励んでおられたそうで，そのエネルギーには感銘しました．

最後に桑村潤氏から「PostgreSQLコミュニティの過去，現在，未来」という合宿の締めくくりにふさわしい発表がありました．

はじめての合宿ということで準備に苦労しましたが，終わってみると十分報われておつりが山ほどきたな，という印象です．私も含め参加者の方から口々に「またやりたい」という声が聞かれました．次回の合宿で読者の方とお会いできるのを楽しみにしています．