フリーDBMSの決定版
PostgreSQL大全

chapter 2

pgbash実用法

会員管理DBを作ってみよう

境田雅明 *SAKAIDA Masaaki* sakaida@psn.co.jp

本章では，シェル上で直接SQLを実行できる「pgbash」というソフトについて，導入から実用までを，会員管理のDBを例にとりながら解説します．

はじめに

PostgreSQLをすでに使っている人は「**pgbash**？それ何？？」と思うかもしれません．**pgbash**は**'99**年**5**月ごろ，**pgsql-jp**メーリングリストで議論され**'99**年**7**月にリリースされた新しいソフトです．**pgbash**を一言で言うならば「**psql**のエッセンスを取り出し，部分的に拡張してシェルに組み込んだ新しいシェル」です．すなわち，シェルのすべての機能を使いながら**SQL**を直接実行できるという便利なシェルなのです．

今回は簡単な会員管理のサンプルを用いて，**pgbash**について解説します．

pgbashとは

pgbashは，**bash-2.03**シェルに**PostgreSQL**用の直接**SQL**や埋め込み**SQL**のインターフェースを組み込んだシェルです．ログインシェルやサブシェル（シェルの中からシェルを起動）として，また，シェルスクリプト実行用のシェルとして利用することができます．ここで「直接**SQL**」とは，**SQL**を入力しその結果を即座に出力する機能を意味します．「埋め込み**SQL**」とは，**fetch into**文のように検索結果をシェル変数に代入し，シェルスクリプト言語で処理する機能を意味します．

pgbashでは，**SQL**文（終端子は**;**〔セミコロン〕）を直接入力することができます．**SQL**文は，**1**つのシェルコマンドとして取り扱われますので，**SQL**文とパイプライン，リダイレクションおよびバックグランドジョブ命令を組み合わせて実行することができます．また，**exec_sql**コマンドを使用すると**SQL**文に各種オプションをつけて実行することができます．

“論より証拠”です．図**1**を見てください． 今までにシェルと**SQL**の経験があるかたならば，図**1**の内容はほとんど理解できるはずです．このように，**pgbash**は，シェルと**SQL**さえわかれば誰でも使用できるとても便利な道具です（ひょっとして，将来「シェルから**SQL**を使えないなんてシェルじゃない」なーんていう時代が来るかもしれません**(^o^)**）．

入手／導入

pgbashの入手

pgbashは表1のURLから入手できます．

また，**pgbash**は，**bash-2.03**のソースプログラムが必要です．これらは表**2**の**URL**から入手できます．

このほか，pgbashの情報は右のWebページを参照してください．

http://www.psn.co.jp/PostgreSQL/pgbash/

●表1 pgbash入手先

http://www.psn.co.jp/PostgreSQL/pgbash/pgbash-2.1.tar.gz
ftp://ftp.psn.ne.jp/pub/PostgreSQL/pgbash/pgbash-2.1.tar.gz

●表2 bash-2.03入手先

ftp://ftp.gnu.org/gnu/bash/bash-2.03.tar.gz
ftp://prep.ai.mit.edu/pub/gnu/bash/bash-2.03.tar.gz
ftp://sunsite.unc.edu/pub/gnu/bash/bash-2.03.tar.gz

●図1 pgbash実行例

```
prompt> /usr/local/bin/pgbash   ←pgbashを起動
pgbash> connect to db@xxx.com as db2 user pgbash;  ←データベース（DB）に接続
# PostgreSQL 7.0.x on i386-unknown-freebsd2.2.7, ..
# CONNECT TO db@xxx.com:5432 AS db2 USER pgbash

pgbash> select * from test limit 100; | more  ←SQL＋パイプライン

name    |tel            |email             |address
--------+---------------+------------------+-------
境田雅明  |06-6123-1234   |sakaida@psn.co.jp |大阪
鈴木一郎  |078-876-9876   |itirou@xxx.co.jp  |兵庫県

以下省略

pgbash> select * from test; > /tmp/sel.dat &
      SQL＋リダイレクション＋バックグランドジョブ
pgbash> ls /tmp
-rw-r--r-- 1 pgbash postgres 616 Apr 1 12:16 sel.dat

pgbash> NAMEDATA='田中太郎'
pgbash> TELDATA='06-6123-1234'
pgbash> insert into test values(
> '$NAMEDATA',  ←シェル変数を利用
> '$TELDATA','tanaka@tanaka.com',
> '大阪府');  ←単語の切れ目で改行可能
INSERT 19232 1

pgbash> connect to db3; > /dev/null.  ←loginユーザ名でDB接続
pgbash> set connection db2;  ←カレントDBをdb2にセット
pgbash> m    ←DBの接続一覧表示

# Connected Databases List (C: current database is '*')
+---+------------+---------+--------------------------+
| C |connect_name|user_name|target_name(dbname@host:..)|
+---+------------+---------+--------------------------+
| * | db2        | pgbash  |db@xxx.com:5432           |
|   | db3        | pgbash  |db3:5432                  |
+---+------------+---------+--------------------------+
(2 row)

pgbash> select * from test;   ←db2に対して検索
pgbash> exec_sql -d db3 "select * from test3"   ←一時的にdb3に接続して検索
pgbash> begin;
pgbash> declare cur cursor for select name,tel from test;
pgbash> fetch in cur into :AA,:BB;   ←（埋め込みSQL）シェル変数に代入
pgbash> echo "名前=$AA, 電話番号=$BB"
名前=境田雅明, 電話番号=06-6123-1234
pgbash> end;
pgbash> history | tail  ←操作履歴表示
 505  declare cur cursor for select name,tel from test;
 506  fetch in cur into :AA,:BB;
 507  echo "名前=$AA, 電話番号=$BB"
 508  end;
 509  history | tail

pgbash> fc declare   ←bashの履歴編集機能を利用して，履歴のdeclare文を修正再実行
pgbash> !fetch  ←履歴のfetchを再実行
pgbash> disconnect all;  ←すべてのD/B接続を切断
pgbash> exit  ←pgbashの終了
prompt>
```

フリーDBMSの決定版

PostgreSQL大全

PostgreSQLのインストール

pgbashをインストールするには**PostgreSQL**があらかじめインストールされている必要があります．**PostgreSQL**のインストールにつきましては**1**章「最新版**PostgreSQL 7.0**登場！」，**6**章「**Postgre SQL**導入パッケージ完全ガイド」もしくは他の文献を参照してください．

なお，**pgbash**は，**PostgreSQL**が**/usr/local/pgsql**ディレクトリにインストールされていることを前提にしています．このディレクトリ以外のディレクトリ（たとえば**,xxx/pgsql**）の場合は，

```
$ ln -s xxx/pgsql /usr/local/pgsql
```

としてシンボリックリンクを張ってください．

pgbashのインストール

pgbash-2.1.tar.gzを，適当なディレクトリで展開します．

```
$ gzip -dc pgbash-2.1.tar.gz | tar xf  -
```

pgbash-2.1ディレクトリが作成されますので，**bash-2.03.tar.gz**を**pgbash-2.1**ディレクトリの中で展開します．これで，**pgbash-2.1**ディレクトリの中に**bash-2.03**ディレクトリが作成されます．

```
$ cd pgbash-2.1
$ gzip -dc bash-2.03.tar.gz | tar xf  -
```

次に**bash-2.03**にパッチをあてます．**pgbash-2.1/patch**ディレクトリの中で**make**を実行してください．パッチの結果は，**make.log**ファイルに記録されます．

```
$ cd patch
$ make
```

その後，**pgbash-2.1/src**ディレクトリに移動して，**configrue**→**make**→**make install**を実行します（**make**は**gnu make**を使用します．**FreeBSD**では**gnu make**をインストールした後に**gmake**を使用します）．

make installを実行すると“**pgbash**”を**/usr/local/bin**ディレクトリへ，“**pgbashrc**”を**/etc**ディレクトリへコピーします．また，**make install**は，スーパーユーザでなければ実行できませんので注意してください．スーパーユーザになれない場合は，手作業で“**pgbash**”と“**pgbashrc**”を適当なディレクトリにコピーしてください．本章での説明では，**pgbash**は**/usr/local/bin**へ，**pgbashrc**は**/etc**へコピーされたものとして説明します．

ユーザの環境設定

ユーザの環境設定は，**bash**シェルの場合と同じですが，**pgbash**では**~/.pgbashrc**ファイルが追加されます．

pgbashをログインシェルとして使用するときは，以下の点について考慮します．

①パスワードファイルのログインシェル名を/usr/local/bin/pgbashに変更
②FTPソフトなどを使用する場合は/etc/shellsに/usr/local/bin/pgbashを追加
③Linuxなど/etc/ld.so.confの設定が必要なOSは，/etc/ld.so.confに/usr/local/pgsql/libを追加して/sbin/ldconfを実行

なお，①～③はスーパーユーザの権限で設定します．

また，**pgbash**をサブシェルとして使用する場合は，**~/.bashrc**，**~/.pgbashrc**の設定が必要です．**~/.bashrc**には，**bash**シェルを使用するうえで必要な設定を，**~/.pgbashrc**は，**pgbash**専用の設定を行います．**~/.pgbashrc**は**/etc/pgbashrc**をホームディレクトリにコピーすることによって作成します．この**~/.pgbashrc**を修正することで，ユーザ独自の操作環境を構築することができます．

なおログインシェルは，**~/.bash_profile**（なければ**~/.bash_login**または**~/.profile**）を使用しますので，通常，**~/.bash_profile**には，**~/.bashrc**と**~/.pgbashrc**の読み込み命令（“．**~/.bashrc**”と“．**~/.pgbashrc**”）を記述しておきます．

●図2　Welcomeメッセージ

```
Welcome to the pgbash-2.1 (bash-2.03)

  Type '[time] SQL; [pipeline][redirection][&]' to execute SQL.
  Type 'exec_sql [option] ["SQL"]' to execute SQL with options.
  Type '?' to HELP. (This help is set at ~/.pgbashrc)

pgbash>
```

●図3　コマンド一覧の表示

```
prompt> ?
   ?       : this help
   h  [SQL]: help <SQL> syntax or all SQL_commands
   o       : help OPTIONs of 'exec_sql' function
   v       : print PGBASH VERSION
   s       : print STATUS after SQL execution
   l       : list all DATABASEs
   m       : list all CONNECTIONs
   d  [TBL]: list tables,indices,columns in <TaBLe>, or all tables
   dt      : list only TABLEs
   di      : list only INDEXes
   ds      : list only SEQUENCEs
   dg      : list GRANT/REVOKE permissions
```

使用方法

それでは，**pgbash**の基本的な使用方法を会員管理のサンプルを作成しながら説明します．使い方などの詳細につきましては，**pgbash**附属のドキュメントや**pgbash**の**Web**ページなどを参照してください．

pgbashの起動と終了

通常使用しているログインシェルでログインした後に，**/usr/local/bin/pgbash** と入力すると，**pgbash**をサブシェルとして使用できます．**pgbash**が起動すると~**/.bashrc**と~**/.pgbashrc**が読み込まれ図**2**の**Welcome**メッセージが表示されます．また，ログインシェル用の環境設定後に，**telnet**でログインすると**pgbash**をログインシェルとして使用できます．

pgbashをシェルスクリプト実行用シェルとして使用する場合は，シェルスクリプトの先頭に“**#!/usr/local/bin/pgbash**”を指定します．

対話型環境では，“**exit**”を入力すると**pgbash**を終了し，ログインシェルの場合はログアウト，サブシェルの場合は元のログインシェルに戻ります．また，シェルスクリプトはそれが終了した時点で**pgbash**が終了したことになります．**pgbash**が終了すると，すべてのデータベース接続は自動的に切断されます．

ヘルプ機能

pgbashプロンプト状態で“**?**”を入力すると~**/.pgbashrc**に定義された図**3**のコマンド一覧が表示されます．

ここで，“**h**”を入力すると**SQL**コマンド一覧が“**h SQL**”を入力すると**SQL**の文法が表示されます．

データベースの接続と切断

データベースに接続するには，**CONNECT**文を使用します．切断は**DISCONNECT**文です．**CONNECT**から**DISCONNECT**（もしくは**pgbash**の終了）まで，データベースは接続中になります．

CONNECT文を発行せずに**SELECT**などの**SQL**を実行した場合は，**CONNECT TO DEFAULT;**（ログインユーザ名を，**PostgreSQL**ユーザ名／データベース名として接続）が自動的に発行されます．

また，複数のデータベースに接続することもできます．この場合，**set connection**文もしくは**exec_sql -d**オプションでデータベース接続名を選択して**SQL**を実行します．

漢字コード

マルチバイト指定（**PostrgeSQL-6.5.x**では**configure -with-mb=EUC_JP**，**PostrgeSQL-7.0.x**では**condifure -enable-multibyte=EUC_JP**）でインストールされた**PostrgeSQL**は，**EUC**漢字コードのテーブル名，属性名，データなどをデータベースに登録することができます．

漢字のデータを取り扱う場合でクライアント端末の漢字コードが**EUC**以外（たとえばシフト**JIS**）の場合は，このことを**PostgreSQL**のバックエンドに知らせる必要があります．

クライアントの漢字コードをバックエンドに知らせるには以下の**2**つの方法があります．

①~/.bashrcもしくは~/.pgbashrcにexport PGCLIENTENCODING='SJIS'を指定
②データベースに接続後にset client_encoding='SJIS';を実行

①が指定されると，シフト**JIS**がすべてのデータベース接続の初期値となります．②が実行されるとデータベースの接続ごとに漢字コードを切り替えることができます．

テーブルの定義

それでは，会員管理のテーブル（**master**）を定義してみます．データ項目は，氏名（**name**），電話番号（**tel**），メールアドレス（**email**），住所（**address**）の**4**項目とし，電話番号とメールアドレスに索引をつけるものとします．なお，**PostgreSQL**ユーザ名は**pgbash**として，すでに作成されているものとします．また，データベース名も**pgbash**として**createdb pgbash**によって作成しておきます．それでは，図**4**のように対話型環境でテーブルを定義してみます．

図**4**を見るとわかるように，［**SQL**文;］と入力するだけで**SQL**を実行することができます．**SQL**文は単語の切れ目で自由に改行することができます．

●図4　テーブルの定義

```
prompt> /usr/local/bin/pgbash
pgbash> connect to pgbash@xxx.com user pgbash;
Password: zzzzzzz
pgbash> drop table master;
pgbash> create table master (
>   name   char(16),    tel    char(16),
>   email  varchar(32), address varchar(8)
>   );
pgbash> create index tel_ind on master(tel);
pgbash> create index email_ind on master(email);
pgbash> dt
# Database = pgbash
+--------+----------+-------+
| Owner  | Relation | Type  |
+--------+----------+-------+
| pgbash | master   | table |
+--------+----------+-------+
(1 row)
```

●図5　シェルスクリプトでのテーブルの定義

```
#!/usr/local/bin/pgbash
connect to pgbash@xxx.com user pgbash zzzzzzz;
drop table master; > /dev/null
create tbale master (
   name   char(16),      tel  char(16),
   email  varchar(32),   address varchar(8)
);
if((SQLCODE==0)); then
   create index tel_ind on master(tel);
   create index email_ind on master(email);
fi
```

対話型環境では，**SQL**文の途中で改行すると“>”プロンプトが表示されますので，続けて**SQL**文を入力します．セミコロンが現れた時点で**SQL**文の終了とみなします．図**4**の白ケイの行の**dt**はテーブル名一覧を表示するコマンドです．このコマンドは**~/.pgbashrc**で処理方法が定義されています．

次にシェルスクリプトでテーブルを定義します．図**5**を見てください．

シェルスクリプトは，先頭に**#!/usr/local/bin/pgbash**をつけて，ファイルのパーミッションを実行許可に設定します．シェルスクリプトにおいても，**SQL**文は単語の切れ目で自由に改行することができます．セミコロンが現れた時点で**SQL**の終了とみなします．図**5**の下の**SQLCODE**は，**SQL**の実行状態を表すシェル変数です．

SQL実行状態を表すシェル変数

pgbashは，**SQL**実行後の処理結果を**SQL**シェル変数にセットします．この**SQL**シェル変数は**SQL**実行ごとに毎回更新されます．主な**SQL**シェル変数を表**3**に示します．

●表3 主なSQL変数

SQLシェル変数	型	内容
$SQLOID	整数型	最新のinsertのOIDの値
$SQLCODE	整数型	SQLエラーコード
		0 正常終了
		100 EOF（End Of File）
		負値 SQLエラー
$SQLERRMC	文字型	SQLエラーメッセージ
$SQLERRML	整数型	SQLエラーメッセージの長さ
$SQLERRD2	整数型	検索結果の行数
$SQLERRD3	整数型	検索結果の列数
$SQLNTUPLE	整数型	SQLERRD2と同じ
$SQLNFIELD	整数型	SQLERRD3と同じ
${SQLFIELDNAME[I]}		列名並び（iは0からSQLNFIELD－1まで）

データの登録

それでは次に，対話型環境でinsert文，copy文を使用してデータを登録する例を図6～8に示します．

続いて，Webページを使用した場合を説明します．リスト1は，HTMLファイル（insert.html）の内容です．

リスト1のWebページを表示してデータを入力し，submitボタンをクリックするとリスト2に示すinsert.cgiが起動します．insert.cgiはデータを書き込み，その結果をホームページに表示します．エラーの場合は，SQLCODEとエラーメッセージを表示します．

リスト2において，2行目のexec 2>&1は，標準エラー出力を標準出力に出力するための指定です．これで，bashが出力するエラーメッセージもホームページに表示することができます．3，4行目のecho "Content-type: text/html"と echo ""は必須です．5行目のexec_sql -iはこのシェルスクリプトをCGIモードに設定します．

●図6 insert文の例

```
pgbash> insert into master values(
> '境田雅明','06-6666-6666',
>      'sakaida@psn.co.jp','大阪');
pgbash> insert into master values(
> '鈴木一郎','078-888-8888',
>       'itirou@orx.co. jp','兵庫');
```

●図7 直接入力のcopy文の例

```
pgbash> copy master from STDIN using delimiters ','; << EOF
境田雅明,078-999-1234,sakaida@psn.co.jp,大阪
鈴木一郎,078-999-9999,itirou@orx.co.jp,兵庫
EOF
```

●図8 ファイルから入力のcopy文の例[注1]

```
pgbash> copy master from input.dat;
```

CGIモードの設定

exec_sql -iを実行すると，GET／POSTメソッドによるWebページからの入力データを自動的にシェル変数にセットします．たとえば，リスト1の

●リスト1 insert.htmlの内容

```
<HTML>
<FORM METHOD=POST ACTION="insert.cgi">
<PRE>
パスワード<INPUT TYPE=password NAME="passwd" SIZE=8>
名前     <INPUT TYPE=text NAME="name"     SIZE=15>
電話番号 <INPUT TYPE=text NAME="tel"      SIZE=15>
Email    <INPUT TYPE=text NAME="email"    SIZE=31>
住所     <INPUT TYPE=text NAME="address" SIZE=7>
</PRE>
<INPUT TYPE=submit VALUE="submit">
<INPUT TYPE=reset  VALUE="reset">
</HTML>
```

注1）PostgreSQLの標準機能では，postgresスーパーユーザだけがcopy文にファイル名を指定できます．この場合，copy文によってバックエンドがファイルの入出力を行うことになります．それに対して，pgbash-2.1は，クライアント側で処理する独自のcopy機能を実装し，一般ユーザであってもファイル名を指定してファイルの入出力を行うことができます．

●リスト2　insert.cgi

```
 1: #!/usr/local/bin/pgbash
 2:  exec 2>&1
 3:  echo "Content-type: text/html"
 4:  echo ""
 5:  exec_sql -i
 6: #
 7: echo "<HTML>"
 8: echo "<font size=4>"
 9: echo "<PRE>"
10: #
11: connect to pgbash@xxx.com as db1 user pgbash $passwd;
12: if((SQLCODE<0)); then
13:     exit
14: fi
15: #
16: insert into master values('$name','$tel','$email','$address');
17:
18: if((SQLCODE==0)); then
19:     echo ""
20:     echo "名前     =$name"
21:     echo "電話番号=$tel"
22:     echo "Email    =$email"
23:     echo "住所     =$address"
24:     echo ""
25:     echo "登録しました"
26: else
27:     echo "書きこみエラー"
28:     echo "SQLCODE=$SQLCODE: $SQLERRMC"
29: fi
30: #
31: echo "</PRE>"
32: echo "</font>"
33: echo "</HTML>"
```

Webページ上の名前name，telが，リスト2ではそのままシェル変数$name，$telとして参照されています．

また，exec_sql -iはselectの検索結果の出力を自動的にHTML出力に切り替え，クッキーが設定されている場合は，$HTTP_COOKIE の値を分解して次のシェル変数に代入します．

```
$HTTP_NCOOKIE          : クッキーの個数

${HTTP_COOKIEKEY[i]} : クッキーのキー名
${HTTP_COOKIEVAL[i]} : クッキーの値
（ただし，iは0からHTTP_NCOOKIE－1までです）
```

データの検索

それでは次に，シェルスクリプト（select.sh）を使用してデータを検索する例を稿末のリスト3に示します．

select.shを実行すると，図9のメニュー画面（番号の入力）が表示されます．

ここで，メニュー番号1を選択して名前の一部分（たとえば“田”）を入力すると図10の結果が表示されます．

また，メニュー番号2を選択して電話番号（たとえば“078-999-1234”）を入力すると図11のように該当する名前，メールアドレス，住所が表示されます．

●図9　メニュー画面

```
############ Menu ############
 名前部分一致検索 ..........1
 電話番号検索 ..............2
 メールアドレス末尾一致検索 ..3
 終了 ......................9
番号：
```

●図10　名前の検索結果（この例では“田”）

```
name     |tel           |email              |address
---------+--------------+-------------------+-------
境田雅明  |078-999-1234  |sakaida@psn.co.jp|兵庫県
境田晴信  |078-999-1234  |harunob@psn.co.jp|兵庫県
田中太郎  |06-6321-5678  |tanaka@tanaka.com|大阪府
山田花子  |06-6543-9876  |hana@yosi.co.jp  |大阪府
(3 rows)
```

●図11　電話番号の検索結果（この例では“078-999-1234”）

```
Name     |tel           |email              |address
---------+--------------+-------------------+-------
境田雅明  |078-999-1234  |sakaida@psn.co.jp|兵庫県
(1 rows)
```

●図12　アドレスの末尾検索結果（この例では“co.jp”）

```
  名　前 |   電 話 番 号      |  メールアドレス
---------+-------------------+---------------------
長島茂也  |Tel.03-9876-1234   |nagasima@yomi.co.jp
長島一哉  |Tel.03-9876-1234   |kazu@yomi.co.jp
栗原居達  |Tel.06-3456-0987   |kurita@paso.co.jp
？？？？  |Tel.06-3456-0987   |kurio@paso.co.jp
野村息子  |Tel.-----------    |musuko@hansi.co.jp
---------+-------------------+---------------------
（合計= 5 行）
```

同様に，メニュー番号**3**を選択してメールアドレスの末尾（たとえば“**co.jp**”）を入力すると図**12**の結果が表示されます．このメールアドレス末尾一致検索では，検索結果を**1**行ずつ**fetch into**文で読み込み，名前が未登録なら“**????**”を，電話番号が未登録なら“**----------**”を表示するように加工して表示しています．さらに名前などの属性名を**echo**文で日本語表示してみました．

続いて，同じような検索を**Web**ページ上で実行してみましょう．リスト**4**，**5**のような**HTML**ファイル（**select.html**），**CGI**ファイル（**select.cgi**）をそれぞれ作成します．

リスト**4**の**Web**ページを表示して検索用データを入力し，“**submit**”ボタンをクリックすると**select.cgi**が起動します．**select.cgi**は，検索結果を**Web**上に表示するものです．

●リスト4　select.html

```
<HTML>
<FORM METHOD=POST ACTION="select.cgi">

パスワード <INPUT TYPE=password NAME="passwd" SIZE=8>

次のいずれかの処理を選択して該当する項目にデータを
入力してください。

<INPUT TYPE=radio NAME="menuno" VALUE="1" CHECKED>
名前部分一致検索
<INPUT TYPE=text NAME=namedata SIZE=15>
（名前の一部分を指定可能）<BR>
<BR>
<INPUT TYPE=radio NAME="menuno" VALUE="2" >
電話番号検索
<INPUT TYPE=text NAME=teldata SIZE=15> <BR>
<BR>
<INPUT TYPE=submit VALUE="submit">
<INPUT TYPE=reset  VALUE="reset">
</HTML>
```

それでは，実際に操作してみましょう．**select.html**を表示すると図**13**のようになります．

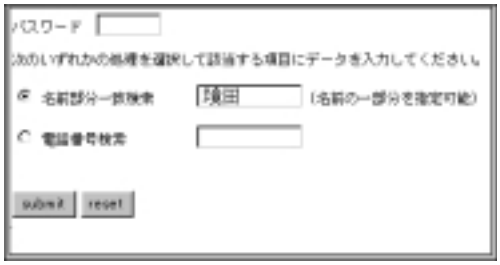
「名前の部分一致」を選択して名前の一部分（たとえば“境田”）を入力しますと図**14**のような結果が表示されます．このように，**select**文を記述する

●リスト5　select.cgi

```
#!/usr/local/bin/pgbash
 exec 2>&1
 echo "Content-type: text/html"
 echo ""
 exec_sql -i
#
echo "<HTML>"
echo "<font size=4>"
echo "<PRE>"
#
connect to pgbash@xxx.com as db1 user pgbash $passwd;
if((SQLCODE<0)); then
    exit
fi
#----- メニュー番号ごとの処理
case $menuno in
   1)
      select * from master where name like '%$namedata%' limit 100;
      if((SQLCODE==0 && SQLNTUPLE==0)); then
         echo "該当データ無し"
      fi ;;
   2)
      select * from master where tel = '$teldata' limit 100;
      if((SQLCODE==0 && SQLNTUPLE==0)); then
         echo "該当データ無し"
      fi ;;

    9) condition=0  ;;
    *) echo "番号エラー" ;;
   esac
   echo ""
#
echo "</PRE>"
echo "</font>"
echo "</HTML>"
```

●図13　ブラウザ上での検索画面



●図14　検索結果

name	tel	email	address
境田 稚明	078-927-1234	sakaida@pon.co.jp	兵庫県
境田 晴信	078-927-1234	harunob@pon.co.jp	兵庫県

(2 rows)

だけでWeb上でテーブル形式の検索結果が閲覧できます．

なお，リスト3で使用されているSQL_NOT_FOUNDは，EOF（End Of File）を表すSQLエラーコードです．

SQLエラーコード

pgbashは，SQLエラーコードを独自のシェル変数にセットしています．シェル変数は

```
$ exec_sql -h errno
```
（対話型環境ではh errno）

で見ることができます．

SQL実行後にexec_sql -s（対話型環境では，s）とすると，実行結果のSQLエラーコードなどを表示することができます（図15）．

制約

●bashコマンド

bashの declare，set，selectコマンドは，SQL文の先頭語と同じになります．pgbashでは，SQL文を優先的に取り扱うため，bashのコマンド名を表4のように変更しています．

●表4　bashのコマンドとの対応

bash	pgbash
declare	declares
set	sets
select	selects

●図15　実行結果のエラーコード例

```
# SQL status (shell variable)
  SQLCODE   = -403   (SQL error code)
  SQLNTUPLES= 0      (number of tuples)
  SQLNFIELDS= 0      (number of fields)
  SQLERRML  = 38     (length of SQLERRMC)
  SQLERRMC  = ERROR:  testxxx: Table does not exist.
```

●シェル変数の利用

SQLをパイプラインやバックグランドジョブで実行した場合，SQL実行後のシェル変数（たとえば，$SQLCODEなど）は参照できません．これは，パイプラインとバックグランド処理が別プロセスで起動されるため，もとのプロセスにシェル変数の値を返せないからです．

●SQL; の記述位置

SQL文は，文の先頭かもしくはタイムスペックの後しか記述できません．ifやwhileの後には記述できません．

おわりに

pgbashは，bashシェルの字句解析部に比較的単純なパッチをあて，SQL文を“exec_sql 'SQL'”に切り替える方式でSQLを解釈しています．このため，SQL文の記述位置などの制約があります．将来的には，SQL文の解釈をもっとしっかりとした考え方で組み込む必要があると思っています．また，pgbashは内部でPostgre SQLのlibpqインターフェースを使用していますが，ODBCインターフェースの組み込みも検討する必要があります．

しかし，現状においてもpgbashは，対話型環境において優れた操作性を発揮します．また，比較的簡単なデータベースアクセス処理プログラムを作成する上でとても便利な道具になります．ぜひ一度試してみてください．

最後になりましたが，pgbashは久保健洋（kubo-t@cx.airnet.ne.jp）さんのorabash／pqbash

（Oracle／PostgreSQLbash組み込みコマンド）のアイデアをもとにして開発されました．また，pgbashの検討や動作確認におきまして，pgsql-jpメーリングリストの皆さまに協力していただきました．ご協力いただきました皆さまに改めて感謝いたします．

●リスト3　シェルスクリプトを利用したデータ検索例

```
#!/usr/local/bin/pgbash
connect to pgbash@xxx.com user pgbash;
if((SQLCODE<0)); then
   exit
fi
#
declares -i condition=1;
while(( condition==1 ))
do
   echo "############ Menu ############"
   echo " 名前部分一致検索 .........1"
   echo " 電話番号検索 ..............2"
   echo " メールアドレス末尾一致検索 ..3"
   echo " 終了 ......................9"
   echo -n "番号："; read menuno

   case $menuno in
   1) echo -n "名前の一部分："; read namedata
      select * from master where name like '%$namedata%' limit 100;
      if((SQLCODE==0 && SQLNTUPLE==0)); then
         echo "該当データ無し"
      fi ;;
   2) echo -n "電話番号："; read teldata
      select * from master where tel = '$teldata' limit 100;
      if((SQLCODE==0 && SQLNTUPLE==0)); then
         echo "該当データ無し"
      fi;;
   3) echo -n "メールアドレスの末尾部分一致："; read emaildata
      begin; > /dev/null
      declare cur cursor for select name, tel, email from master
             where email like '%$emaildata' order by tel
                limit 100; > /dev/null
      if((SQLCODE==0)); then
         declares -i counter=0
          echo "  名　前　|　電 話 番 号　　|  メールアドレス"
          echo "----------+------------------+----------------------"

         declares -i x=0; while(( x < 100 ))
         do
            fetch in cur into :name_data :name_null,
                                  :tel_data :tel_null, :email_data;
             if((SQLCODE<0)); then
                echo "SQLCODE=$SQLCODE ($SQLERRMC)"
               let x=101
             elif((SQLCODE==SQL_NOT_FOUND)); then
               let x=101
             else
               if(( tel_null==-1 )); then
                  tel_data='----------   '
               fi
               if(( name_null==-1 )); then
                  name_data='????   '
               fi
 echo "${name_data:0:10}|Tel.${tel_data:0:14}|$email_data"
              let counter=counter+1
              let x=x+1
             fi
         done
          echo "----------+------------------+----------------------"
         echo "(合計= $counter 行)"
      fi
      end; > /dev/null
      ;;
   9) condition=0  ;;
   *) echo "番号エラー" ;;
   esac
   echo ""
done
```