PostgreSQL

Chapter 2

Installation
PostgreSQLをセットアップしよう

Chapter 2
Installation～PostgreSQLをセットアップしよう

2-1 インストールの前に

本章では，PostgreSQLのインストール方法を説明します．対象とするバージョンは基本的に6.3.2ですが，必要に応じて6.4についても言及します．

最近はPostgreSQLのコンパイル済みパッケージが手に入る場合もありますが，ここでは，PostgreSQLの最も基本的なインストール方法であるソースからコンパイルする方法を解説します．この方法は少々面倒ですが，以下のような利点もあります．

- パッケージが提供されていないプラットフォームでもPostgreSQLが利用できるようになる
- パッケージに添付されていないインターフェースも自由に追加できる
- C言語ユーザ定義関数などの高度な機能を使いたい場合には，ソースにアクセスできることが必須

ポイントさえ押さえればPostgreSQLのインストールは決して難しいものではないので，ぜひ挑戦してみてください．

なお，本書付属のCD-ROMに収録されているPlamo LinuxにはPostgreSQL 6.3.2のコンパイル済みパッケージが収録されています．

2.1.1 ドキュメント

インストールの際に参照するドキュメントを表2.1.1に示します．インストールの前に一読しておいてください．FAQ[注1]とFAQ_Linuxについては，メーリングリストのメンバーの桑村氏が翻訳されたFAQとFAQ_Linux日本語版をCD-ROMのja_doc/の下に収録しました．こちらもぜひご覧ください．

●注1
ご存知の向きも多いかと思いますが，FAQとは“Frequently Asked Questions”をあらわします．日本語でいうと「よくある質問」といった意味です．

●表2.1.1 インストール時に参照するファイル

ファイル名	説明	入手方法
INSTALL	ソースに付属するインストール用ドキュメント	ソースディレクトリ/INSTALL
FAQ	ソースに付属するFAQ	ソースディレクトリ/doc/FAQ
FAQ_FreeBSD	FreeBSD固有のFAQ	ソースディレクトリ/doc/FAQ_FreeBSD
FAQ_Linux	Linux固有のFAQ	ソースディレクトリ/doc/FAQ_Linux

2-2 インストールのための準備

2.2.1 ソースの入手方法

付属のCD-ROMのpgsql/以下にPostgreSQLのソースが収められていますが，インターネットでは以下のURLから入手できます．

```
ftp://ftp.jaist.ac.jp/pub/dbms/PostgreSQL/postgresql-6.3.2.tar.gz
ftp://ftp.sra.co.jp/pub/cmd/postgres/6.3.2/postgresql-6.3.2.tar.gz
ftp://ftp.postgreSQL.org/pub/postgresql-6.3.2.tar.gz
```

6.4の場合は以下になります．

```
ftp://ftp.jaist.ac.jp/pub/dbms/PostgreSQL/postgresql-v6.4.tar.gz
ftp://ftp.sra.co.jp/pub/cmd/postgres/6.4/postgresql-v6.4.tar.gz
ftp://ftp.postgreSQL.org/pub/postgresql-v6.4.tar.gz
```

2.2.2 必要なツール類

PostgreSQLをインストールするためには，以下のツールが必要です．

GNU make（gmake）

GNU makeは3.75以上のバージョンが必要です．

```
gmake -v
```

というコマンドでバージョンを確認してください．実行例（図2.2.1）では，バージョ

図 2.2.1　GNU make のバージョンを調べる

```
$ gmake -v
GNU Make version 3.76.1, by Richard Stallman and Roland McGrath.
Copyright (C) 1988, 89, 90, 91, 92, 93, 94, 95, 96, 97
        Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

Report bugs to <bug-gnu-utils@prep.ai.mit.edu>.
```

ンが3.76.1と表示されています．

GNU makeは，Linuxでは最初からインストールされていることが多いです．FreeBSDではパッケージ[注1]を使ってインストールするのが簡単でしょう．

●注1
2.2.6および2.2.7-RELEASEでは，gmake-3.76.1.tgzを使います．

gcc

コンパイルにはGNU Cコンパイラ（gcc）を使用することをお勧めします．バージョンはできるだけ新しい方がよいのですが，2.7.2以上ならおそらく大丈夫です．egcsももちろん利用できます．gccのバージョンは

```
gcc -v
```

で確認できます．実行例（図2.2.2）では2.7.2.1と出ています．

gccに関しては，Linux，FreeBSDではほとんどの場合インストール済みだと思います．

図 2.2.2　GNU C コンパイラのバージョンを調べる

```
$ gcc -v
Reading specs from /usr/lib/gcc-lib/powerpc-unknown-linux/2.7.2.1-
ppclinux/specs
gcc version 2.7.2.1-ppclinux
```

bison

bisonはGNUの提供する「コンパイラコンパイラ」の一種で，PostgreSQLではSQLパーサ，ecpgパーサなどで使われています．bisonもLinux，FreeBSDではほとんどの場合インストール済みだと思います．

flex

flexは，GNUの提供する字句解析ツールです．バージョン2.5.3はバグのため使えないので注意してください．2.5.2か2.5.4以上ならOKです．flexもLinuxやFreeBSDでは標準でインストールされている場合が多いと思います．

flexのバージョンは

```
$ flex --version
flex version 2.5.4
```

のようにして確認できます．実行例では2.5.4と出ています．

patch

コンパイルには直接関係ありませんが，各種パッチを当てるために必要です．LinuxやFreeBSDでは最初からインストールされていると思います．

2-3 コンパイルとインストール

2.3.1 必要なディスク容量

PostgreSQLは標準では/usr/local/pgsql/にインストールします．本書でも標準通り/usr/local/pgsql/にインストールするものとします．インストール後のディレクトリ構造は表2.3.1のようになります．

このうちbin，lib，include，manはインストール後は変化しない領域で，合計して3～10Mバイト程度です．dataはデータベースが作成される領域なので，PostgreSQLを利用するにつれて増えていきます．たとえば，付属のregressionテスト（後述）を実行すると約20Mバイトを消費しますので，できれば/usr/local/pgsql以下は50Mバイト程度は確保してください．

このほか，ソースを展開しコンパイルする領域が必要です．これはどこでもいいのですが，本書では/usr/local/src/以下で作業するものとします．ソースはコンパイル後は30～60Mバイト程度になります．仮に/usr/local以下が1つのパーティションになっているとすると，インストールしてregression testするだけで83～120Mバイトほどディスクが必要となります．

なお，Linuxの場合，コンパイラオプションにデフォルトで-gオプション[注1]が有効になっているためディスクを消費してしまいますが，これをoffにすればディスクが節約できます．

●注1
デバッグ用のシンボルを生成するオプションです．

表2.3.1 /usr/local/pgsql/以下に作成されるディレクトリ

ディレクトリ	内容
bin	すべての実行プログラム
lib	ライブラリ，テンプレートファイルなど
include	ヘッダファイル
man	オンラインマニュアル
data	データベース領域

2.3.2　インストールのステップ

PostgreSQLのインストールは大きく分けて，

①専用アカウント作成などの準備
②ソースを展開し，パッチを当てる
③コンパイルし，できたバイナリを/usr/local/pgsqlなどにインストールする
④環境設定を行う
⑤初期設定を行う

の段階を踏みます．以下，順を追って説明します．

2.3.3　専用アカウント作成などの準備

postgresアカウントの作成

PostgreSQLを運用するためには，専用のUNIXアカウントを作成することをお勧めします．このアカウントは，UIDが0以外ならどんなアカウントでもかまいません．たとえば「自分以外は絶対にPostgreSQLを使わない」ということであれば，自分がふだんログインするのに使っているアカウントを流用することもできます．

しかし，複数のユーザがPostgreSQLを利用する通常の環境では，PostgreSQLを運用するための専用のアカウントを用意すべきです．ここでは，この専用アカウントを“postgres”とします．

念のためにUNIXユーザの追加の方法を説明します．

■ Linuxの場合

rootになってadduserコマンドを実行します．

```
$ su
Password: [rootのパスワードを入力]
# /usr/sbin/adduser postgres
```

となります．

■FreeBSDの場合

rootになってadduserコマンドを実行します.

```
$ su
Password: [rootのパスワードを入力]
# /usr/sbin/adduser postgres
```

となります.

カーネルの再構築

PostgreSQLでは，System Vのshared memory（共有メモリ）機能が必要です. FreeBSDではデフォルトではこの機能は使えません. 使用中のカーネルのコンフィギュレショーションファイルをチェックし，以下のキーワードがない場合には追加してカーネルを再構築してください.

```
options SYSVSHM
options SYSVSEM
options SYSVMSG
```

ソースディレクトリとインストールディレクトリの作成

ソースディレクトリの/usr/local/src/とインストールディレクトリの/usr/local/pgsqlを作成します.

```
# mkdir /usr/local/pgsql
# chown postgres:postgres /usr/local/pgsql
# mkdir /usr/local/src
# chown postgres:postgres /usr/local/src
```

CD-ROMのマウント

本書付属のCD-ROMをマウントします. プラットフォームによってCD-ROMのマウントの仕方が違う場合がありますが，筆者の環境では以下のコマンドで/mnt/cdromにマウントされます.

```
# mount /mnt/cdrom
```

CD-ROMがマウントされるディレクトリ/mnt/cdromは，使用するプラットフォームに合わせて適当に読み替えてください．

2.3.4　ソースを展開しパッチを当てる

ソースの展開

これ以降のステップはpostgresユーザになって実行します．いったんログアウトしてからpostgresでログインし直すか，suコマンドを使います．

```
# su - postgres
```

PostgreSQLのソースを展開します[注2]．

```
$ cd /usr/local/src
$ tar xfz /mnt/cdrom/pgsql/postgresql-6.3.2.tar.gz
```

●注2
6.4ではpostgresql-6.3.2.tar.gzをpostgresql-v6.4.tar.gzにして実行します．

これで，postgresql-6.3.2（6.4ではpostgresql-v6.4）というディレクトリができるはずです．この中にPostgreSQLのすべてのソースとドキュメントが収められています．トップディレクトリには表2.3.2のファイル／ディレクトリがあります．

実際にコンパイル／インストールを行うのはsrcの下ですが，後述のregression testではcontrib/の下も使用します．

●表2.3.2　postgresql-6.3.2に作成されるディレクトリおよびファイル

COPYRIGHT	著作権とライセンス条件について
HISTORY	改定履歴
INSTALL	インストール方法
README	概要
contrib/	ユーザ提供の関数など
doc/	ドキュメント
migration/	旧バージョンからの移行方法*
register.txt	WWWとサポート用のメーリングリストの説明
src/	ソース一式

＊）旧バージョンからの移行方法はINSTALLに記述されるようになったため，migrationは6.4にはありません．

各種パッチの適用

PostgreSQL 6.3.2のリリース後に見つかったバグなどを修正するためのパッチを適用します.

```
$ cd postgresql-6.3.2
$ sh /mnt/cdrom/pgsql/patches-6.3.2/PATCH
```

6.4の場合は以下を実行します.

```
$ cd postgresql-v6.4
$ sh /mnt/cdrom/pgsql/patches-6.4/PATCH
```

PATCHはシェルスクリプトで，パッチをまとめて適用するものです．各パッチの説明は/mnt/cdrom/pgsql/patches-6.3.2/README_jp.txt[注3]にあります．6.3.2の場合，パッチの量が多いため，うまくパッチが当たらないプラットフォームがあるかもしれません．その場合にはパッチ済みのソースが

```
/mnt/cdrom/pgsql/pacthes-6.3.2/postgresql-6.3.2-patched.tar.gz
```

という名前で置いてありますので，これをお使いください.

なお，CD-ROMに収録した6.3.2および6.4用のパッチのオリジナルはそれぞれ以下のURLです.

```
ftp://ftp.sra.co.jp/pub/cmd/postgres/6.3.2/patches/
ftp://ftp.sra.co.jp/pub/cmd/postgres/6.4/patches/
```

本書の刊行以降に追加されたパッチはここから取得できます.

●注3
6.4の場合は/mnt/cdrom/pgsql/patches-6.4/README_jp.txtにあります.

2.3.5 いよいよコンパイル

configureの実行

PostgreSQLは，移植性を高めるために，configureというツールを実行してプラットフォームに適したMakefileを生成する方法を採っています．したがって，コンパイルを開始する前にまずconfigureを実行する必要があります.

```
$ cd src
```

▶ x86/linuxの場合

```
$ ./configure --with-template=linux-elf --with-mb=EUC_JP
```

▶ Linux/PPCまたはMkLinuxの場合

```
$ ./configure --with-template=linux-elf-ppc --with-mb=EUC_JP
```

▶ FreeBSDの場合

```
$ ./configure --with-template=freebsd --with-mb=EUC_JP
```

他のプラットフォームでも，--with-template=以降を適当に設定することによりconfigureが実行できます．指定可能なキーワードは，src/template以下のファイル名です．

6.4では，--with-template=…の部分は，Linux，FreeBSDに関しては必要ありません[注4]．ただしSolarisの場合は，6.3.2でも6.4でも--with-template=…が必要です．

- SPARC版Solaris 2.4/2.5/2.6の場合 --with-template=solaris_sparc_gcc
- x86版Solaris 2.4/2.5/2.6の場合 --with-template=solaris_i386_gcc

--with-mb=で指定しているのは文字コードです．指定できる文字コードは表2.3.3のものです．データベースの中で複数の言語を混在できるかどうかは，文字コードによります．たとえば，EUC_JPでは英語と日本語しか使えませんが，MULE_

●注4
本来ですと，このようにプラットフォームを指定しなくてもconfigureが自動的にプラットフォームを検知してくれるのですが，6.3.2ではバグのためにこの機能がうまく働きません．6.4ではこのバグは修正されています．

MULE INTERNALコードの入出力方法

MULE INTERNALコードを入出力するためには，Muleのシェルバッファを使うのが簡単です．

① M-x shellでシェルバッファを作ります．
② C-x C-pで入出力文字コードを設定します．mule internalコードは，*internal*で設定できます．

日本語，韓国語，中国語，フランス語などが混在したテストデータが

/usr/local/src/posgresql-6.3.2/src/test/regress/sql/mule_internal.sql

にありますのでお試しください．なお，6.4ならコンパイルし直さなくてもMULE_INTERNAL用のデータベースが作れますので，手軽にテストできます．

Linuxでディスク容量を節約する方法

Makefile.custom による設定では，一部不可能なカスタマイズがあります．前述のデバッグオプションの件がそれに当たります．

Linuxでは，Makefile.globalの中に

```
CFLAGS= -I$(SRCDIR)/include -I$(SRCDIR)/
backend   -O2 -g
```

という行があり，デバッグ用のシンボルを生成する“-g”が設定済みとなっています．このためディスクをたくさん消費してしまいます．この-gを外せばかなりディスクが節約できますが，そのためには以下の方法があります．

①Makefile.globalを直接編集し，-gを外す
②template/linux-elfを編集し，“CFLAGS:-O2 -g”となっているのを“CFLAGS:-O2”とする

①の方法ではconfigureを再実行するたびにMakefile.globalを編集し直す必要がありますが，オリジナルのソースに手を加えることはありません．逆に，②ではconfigureのたびにMakefile.globalを編集する必要がなくなりますが，オリジナルのソースに手を加えることに抵抗のある方もいらっしゃるかもしれません．好みに応じてどちらからを選べばよいでしょう．

INTERNALやUNICODEなら世界各国の言語を混在することができます．文字コードについては，/usr/local/src/postgresql-6.3.2/doc/README.mb.jpをご覧ください．6.4では，データベース生成時に文字コードを指定できるので，ここで指定する文字コードはデフォルトの文字コード程度の意味になります．

configureが無事終了すると，GNUmakefile，Makefile.globalとMakefile.portなどのmakefileが生成されます．

- **GNUmakefile　主makefile[注5]**
- **Makefile.global　makeが参照する各種オプション設定値**
- **Makefile.port　makefiles/Makefile.*プラットフォーム*へのシンボリックリンク．プラットフォーム固有の設定値がある**

●注5
Makefileというファイルもありますが，これはダミーです．

●表2.3.3　指定できる文字コード

EUC_JP	日本語EUC
EUC_CN	GBをベースにした中文EUC
EUC_KR	韓国語EUC
EUC_TW	台湾のEUC
UNICODE	UNICODE．ただしエンコーディングはUTF-8で，サポートするのはUCS-2の範囲，すなわち0xffffまで
MULE_INTERNAL	マルチリンガルエディタMuleの内部コード．ただし，Type Nの不定長文字はサポートしていない
LATIN1	ドイツ語，フランス語などのヨーロッパ系言語であるISO8859 Latin 1．6.4では Latin2～5もサポートされる

このほか，include/config.hにはCコンパイラが参照する各種設定値があります．

以上のファイルはconfigureが自動生成するものですから，原則として変更を行いません．ユーザ設定はMakefile.customというファイルを作成することにより行います．Makefile.customはMakefile.globalが参照します．

コンパイル

GNU Makeでコンパイルします．

```
$ gmake all
```

マシンの性能によりますが，コンパイルには5分から1時間ほど時間がかかります．ちなみに，筆者の環境（PowerBook 2400c/180，LinuxPPC）では約15分かかりました．

コンパイルが正常に終われば，

```
All of PostgreSQL is successfully made.  Ready to install.
```

と表示されます．まれに，コンパイルエラーがあるにもかかわらずmakeが正常終了することがあるので，いちおうmakeの出力を確認してください．

コンパイルがうまくいかない場合

メーリングリストで報告された事例と対処方法を挙げておきますので，参考にしてください．

Q FreeBSD 2.1.xでうまくPostgreSQLがコンパイルできません．

A 残念ながらFreeBSD 2.1.xではコンパイルは難しいようです．FreeBSD 2.2.xにアップグレードすることをお勧めします．

Q Solarisでコンパイルはうまくいったように見えるのですが，その後，

```
QUERY: LOAD '/usr/local/pgsql/src/test/regress/regress.so';
WARN:Load of file /usr/local/pgsql/src/test/regress/regress.so
failed: ld.so.1: /usr/local/pgsql/bin/postgres:
fatal: relocation
error: symbol not found: palloc: referenced in
```

```
/usr/local/pgsql/src/test/regress/regress.so
```

のようなエラーがregression testで出ます．

A GNUのasとldを使っているのが原因です．

- /usr/ccs/binをcommand search path上で先頭にする
- 環境変数GCC_EXEC_PREFIXに/usr/ccs/bin/をセットする

として，GNUのasとldを使わないようにしてください[注6]．

●注6
PostgreSQLの再コンパイルが必要です．

Q Solaris 2.4でコンパイルすると，getrusageが未定義エラーになります．

A Solaris 2.4にはgetrusage()がないためです．ソースに付属のgetrusage()の代用品を以下の方法で使うようにしてください．

① src/backend/port/Makefileの27行目

```
OBJS+=
```

を

```
OBJS+= getrusage.o
```

に変更．

②同じディレクトリのgetrusage.cを変更．8行目の`#if 0`を削除．対応する`#endif`（55行目）も削除．

③src/backendで`gmake install`を実行．

インストール

コンパイルが無事に終了したらインストールです．

```
$ gmake install
```

を実行することにより，/usr/local/pgsql以下にプログラムやライブラリがインストールされます．インストールが正常に終われば図2.3.1のメッセージが表示されます．インストールがうまくいかない場合は，/usr/local/pgsql以下への書き込み権限があるかどうか，ディスク容量は十分かどうかなどを確認してください．なお，6.4ではデ

●図2.3.1　インストール後に表示されるメッセージとその日本語訳

```
(1998-04-07)
PostgreSQL has a Web site at http://www.postgresql.org/ which carries details
on the latest release, upcoming features, and other information to make your
work or play with PostgreSQL more productive.

You may wish to subscribe to the PostgreSQL user-support mailing list.
To do this, send e-mail to pgsql-questions-request@postgresql.org with the
text "subscribe" in the message body. You should receive back an e-mail
welcoming you to the list. Other mailing lists and support information are
detailed on the Web site.

Thank you for choosing PostgreSQL, a leading-edge freeware product.
```

```
(1998年4月7日)

　PostgreSQLのWebページは http://www.postgresql.org/にあります．このページには，最新バージョンの情報，将来予定されている機能追加など，PostgreSQLを仕事や趣味により有効に活用するのに役立つ情報があります．
　PostgreSQLのユーザサポート用のメーリングリストを購読することもできます．このためには，pgsql-questions-request@postgresql.org宛に，本文に“subscribe”と書いてメールを送ってください．メーリングリストの購読を歓迎するメールが返送されてくるはずです．他のメーリングリストやサポート情報については，Webページ上に詳細な情報があります．
　最先端のフリーウェア製品であるPostgreSQLを選んでくださってありがとうございました．
```

フォルトでオンラインマニュアルがインストールされなくなったため，以下を実行してください．

```
$ gmake install-man
```

ただし，次に説明する要領でドキュメントをインストールすれば，オンラインマニュアルもいっしょにインストールされるのでこのステップは必要ありません．ただこのオンラインマニュアルは一部内容が古いので，できればdocの下を参照した方がいいでしょう（図2.3.2）．

ドキュメントのインストール

PostgreSQLのドキュメントは基本的にSGML形式で管理されています（doc/src以下に格納）．しかしSGMLを直接閲覧できるブラウザは普及していないため，HTMLに変換されたファイルも付属していますので，これをインストールしましょう．

```
$ cd /usr/local/src/postgresql-6.3.2/doc …6.3.2の場合
$ cd/usr/local/src/postgresql-v6.4/doc …6.4の場合
$ make install
```

●図 2.3.2　6.4 をインストール後に表示されるメッセージとその日本語訳

```
(1998-09-01)
PostgreSQL has a Web site at http://www.postgresql.org/ which carries details
on the latest release, upcoming features, and other information to make your
work or play with PostgreSQL more productive.

Please check the following URL for a listing of the current user-support
mailing lists:

        http://www.postgresql.org/supp-mlists.shtml

All of the mailing lists are currently archived and viewable at:

        http://www.postgresql.org/mhonarc

And, so that we have an idea of who is using what, please connect to the
following registration URL:

        http://www.postgresql.org/register.shtml


Thank you for choosing PostgreSQL, the most advanced open source database engine.
```

```
(1998年9月1日)

PostgreSQLのWebページは http://www.postgresql.org/にあります．このページには，最新バージョンの情報，将来予定されている機能追加など，PostgreSQLを仕事や趣味により有効に活用するのに役立つ情報があります．
  以下のURLで，現在有効なユーザサポート用メーリングリストの一覧を調べてみてください．

http://www.postgresql.org/supp-mlists.shtml

メーリングリストのアーカイブは以下の場所にあり，参照することができます．

http://www.postgresql.org/mhonarc

  また，私たちは，どんな人たちがどのバージョンを使っているのか知りたいと思います．どうか以下のユーザ登録用URLをご利用ください．

http://www.postgresql.org/register.shtml

  最も進歩したオープンソースデータベースエンジンであるPostgreSQLを選んでくださってありがとうございます．
```

これで/usr/local/pgsql/doc以下にドキュメントがインストールされます．内容は

- admin/：管理者向けドキュメント
- programmer/：プログラマ向けドキュメント
- tutorial/：チュートリアル
- user/：エンドユーザ向けドキュメント

のようになっています．このほかにpostgres/がありますが，これは上記の内容をひとつにまとめたようなものです．

2-4 環境設定

インストールが完了したら，PostgreSQLを利用するために必要な環境設定を行います．本節で説明する環境設定は，ユーザ“postgres”だけでなく，PostgreSQLを利用するすべてのユーザにも必要です．

2.4.1 コマンドサーチパス

PostgeSQLのコマンドはすべて/usr/local/pgsql/binの下にあるので，これを実行できるようにコマンドサーチパスに追加します．

▶ bashの場合

.bashrcに以下を追加します．

```
PATH="$PATH":/usr/local/pgsql/bin
```

▶ csh/tcshの場合

.cshrcに以下を追加します．

```
setenv PATH="$PATH":/usr/local/pgsql/bin
```

2.4.2 環境変数

以下の環境変数を設定します．

- PGLIB…/usr/local/pgsql/libのある場所
- PGDATA…データベース領域のある場所

- LD_LIBRARY_PATH…PostgreSQLのクライアント用sharedライブラリの置いてある場所
- MANPATH…PostgreSQLのオンラインマニュアルの置いてある場所

▶ bashの場合

.bashrcに以下を追加します.

```
export POSTGRES_HOME=/usr/local/pgsql
export PGLIB=$POSTGRES_HOME/lib
export PGDATA=$POSTGRES_HOME/data
export MANPATH="$MANPTH":$POSTGRES_HOME/man
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH":"$PGLIB"
```

以上が完了したら,

```
source ~/.bashrc
```

を実行してください.

▶ csh/tcshの場合

.cshrcに以下を追加します.

```
setenv POSTGRES_HOME /usr/local/pgsql
setenv PGLIB $POSTGRES_HOME/lib
setenv PGDATA $POSTGRES_HOME/data
setenv MANPATH "$MANPTH":$POSTGRES_HOME/man
setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":"$PGLIB"
```

以上が完了したら

```
source ~/.cshrc
```

を実行してください.

2-5 初期設定

2.5.1 データベースの初期化

データベースインスタンス

PostgreSQLにはデータベースを管理するためのディレクトリが必要です．これが環境変数PGDATAで指定したディレクトリです．このディレクトリを作成することを「データベースの初期化」と呼びます．初期化が終わったディレクトリを「データベースインスタンス」と呼び，後述のpostmasterが管理します．1つのデータベースインスタンスには1つのpostmasterプロセスが対応します．ポート番号を変えれば複数のpostmasterプロセスを起動することができますので，1つのホストに複数のデータベースインスタンスが共存できます．

データベースインスタンスの中には複数の「データベース」があり，さらにデータベースの中には複数のテーブルなどのオブジェクトが含まれます．

1つのデータベースサーバへの接続では，1つのデータベースしか使えません．データベースへの接続を切り替えれば複数のデータベースを扱うことができますが，複数のデータベースにまたがるような検索はできません．

1つのデータベースインスタンスには以下のものが含まれます．

- データベースインスタンスの管理情報
- データベースディレクトリ“base”

createdbコマンドあるいはSQL文の“create database”によってデータベースが作成されると，そのたびにそのデータベースと同じ名前のディレクトリがbaseの下に作成されます．ただしひとつだけ例外があり，“template1”というデータベースはinitdbを実行したときに作成されます．template1は特別なデータベースで，データベースを作成する際の雛型として使われます．たとえば，ユーザ定義関数などはtemplate1デ

ータベースに登録しておくことによって，それ以降作成されたデータベースにも定義がコピーされます．

初期化の実行

ではさっそくデータベースの初期化を行いましょう．環境変数PGDATAにすでにデータベース場所が指定されていますので，ここは単に，

```
$ initdb
```

でOKです．これ以外の場所をデータベースインスタンスにしたい場合は，

```
$ initdb --pgdata=/pgsql/db
```

のように，--pgdata=を使ってデータベースインスタンスを作成するディレクトリを明示的に指定します．initdbで指定するディレクトリは，書き込みができることと，指定したディレクトリが存在していないことが必要です．もしもなんらかの理由で同じディレクトリをもう一度使いたい場合は，そのディレクトリを削除してからinitdbを実行してください．

なお6.4では，initdbのときにデフォルトの文字コードを指定できます．たとえば韓国語EUCを使う場合は，

```
$ initdb --pgencoding=EUC_KR
```

とします．initdbで文字コードの指定を省略した場合は，configureで指定した文字コードが採用されます．

ここで注意してもらいたいのは，initdbを起動したユーザ（今回の例ではpostgres）が，そのデータベース領域の所有者となることです．このユーザはPostgreSQLの「スーパユーザ」と呼ばれ，UNIXのスーパユーザ（root）がシステムの管理を行うように，データベースの管理を行います．そのため，**スーパユーザにはデータベースに関するすべてのセキュリティチェックが適用されません．**誤った操作を行うと取り返しのつかない結果を招きかねないことになるので，ご注意ください．

2.5.2 postmasterの起動

postmasterは一度起動されたらずっと動き続け，クライアントからの接続要求を待ちます．接続要求があると，データベースエンジンを起動し，データベースを利用できる状態にします．したがって，postmasterが動いていないとPostgreSQLを利用することはできません．

postmasterを起動する際に環境変数PGDATAまたは引数でデータベースインスタンスを指定しますが，このデータベースインスタンスを作成したユーザだけがpostmasterを起動できることに注意してください．たとえば，ユーザpostgresで作成したデータベースインスタンスを，ユーザfooがpostmasterの起動時に指定することはできません．

データベースインスタンスは-Dオプションで指定しますが，今回は環境変数PGDATAを指定していますので，

```
$ postmaster -S
```

だけでOKです．-Sは，端末からpostmasterを切り離し，デーモンとして動かすオプションです．-Sオプションを使わない場合は，

```
$ postmaster &
```

と&を付けて明示的にバックグラウンドでの起動を指定してください．以下，postmasterで指定可能なオプションを説明します．

■ -D パス名

データベースインスタンスへのパスを指定します．環境変数PGDATAに優先します．

■ -p ポート番号

postmasterが使用するポート番号を指定します．省略時は5432が使われます．ポート番号を変えることにより，複数のpostmasterを同時に立ち上げることができます．

■ -b パス名

バックエンド（データベースエンジン）であるpostgresへのパス名を指定します．今回の例では，/usr/local/pgsq/bin/postgresです．-bを省略した場合はコマンドサーチパスから検索されますが，効率が悪くなるので-bを使ったほうがよいでしょう．

■ -i

セキュリティの関係上，postmasterはデフォルトではUNIXドメインのソケットからしか接続を受け付けません．つまり，他のホストから接続できないわけです．-iを指定すると，他のホストからの接続を許可します．

自ホストであっても，クライアントによっては-iが必要なケースがあります．たとえばPHP/FIでは，データベースホスト名を省略した場合，“localhost”へ接続しようとします．localhostは自ホストを意味しますが，この場合はUNIXドメインのソケットを使用しませんので，結局-iが指定されていないpostmasterに接続できません．

セキュリティの関係からですが，単に-iを指定しただけでは，実は自ホストからしか接続できません．外からの接続を許可するためには，/usr/local/pgsql/data/pg_hba.confというファイルを設定する必要があります．pg_hba.confは普通のテキストファイルで，viなどのテキストエディタで編集できます．pg_hba.confでは，#で始まる行はコメントで無視されますが，この中の

```
#host          all         0.0.0.0       0.0.0.0            trust
```

の頭の#を消せば，すべてのホストのすべてのユーザが無条件で接続できるようになります．ある特定のホストからの接続を許可するには，そのIPアドレスを指定します．たとえば，

```
host          all         192.168.10.1   255.255.255.255   trust
```

と書けば，IPアドレス192.168.10.1を持つホストだけが接続を許可されるようになります．

255.255.255.255はアドレスマスクで，これを変えることにより特定のネットワークアドレスを持つホストからの接続をコントロールできます．

```
host          all         192.168.10.0   255.255.255.0   trust
```

この行は192.168.10.xのホスト（xは任意）の接続を許可します．つまり，まず接続を要求して来たホストのIPアドレスと255.255.255.0のANDを取り，その結果と192.168.10.0と255.255.255.0のANDを取った結果が一致すれば接続が許可されるわけです．

たとえば，IPアドレス192.168.10.1からの接続要求を考えてみましょう．192.168.10.1と255.255.255.0のANDは192.168.10.0になります．次に192.168.10.0と255.255.255.0のANDを取ると192.168.10.0になるので接続は許可されます．

192.168.11.1ではどうでしょう．192.168.11.1と255.255.255.0のANDは192.168.11.0

で192.168.10.0と一致しないので，接続は許可されません．

このほか，IPアドレスだけでなく，データベースを指定して接続の許可／不許可が設定できますし，パスワードを設定してユーザ認証を行うこともできます．詳細は「2.8 セキュリティ機能」で解説します．

■ -B バッファ数

データベースが共有メモリ上で管理するバッファの数を指定します．1個のバッファの大きさは8Kバイトで，デフォルトでは64個のバッファが設定されてます．バッファが大きいほど性能がよくなりますが，システム設定によって利用できる共有メモリの大きさに制限があるので注意してください．

■ -d デバッグレベル

デバッグレベルを1から3までの数字で指定することにより，デバッグ情報が出力されます．数字が大きいほどデバッグ情報が詳細になります．出力先は後述のバックエンドオプションで指定できますが，デフォルトではpostmasterを起動した端末が出力先です．この場合，-Sを指定すると出力されません．

■ -o バックエンドオプション

バックエンド（postgres）の認識するオプションを記述します．詳細はオンラインマニュアル(postgres)(1)を参照してください．ここでは利用頻度の高いものを示します．

■ -S ソートメモリ

ソート処理に使うバッファ[注1]の大きさをKバイト単位で指定します．デフォルトは512です．このサイズが大きいほど，ORDER BYなどのソート処理や，JOINで行われる内部のソート処理が高速化します．

-Sオプションを指定するときは，postmaster自身のオプションと混同されないように"などを使います．たとえば以下のようにします．

```
postmaster -S -o "-S 1024"
```

●注1
共有メモリではありません．

2.5.3 regression test

以上でひとまずインストールが完了しました．さっそく使い始めたいところですが，

まずは気を落ち着けて，PostgreSQLシステムが正常に動いているかどうか確認テストを行いましょう．PostgreSQLにはこのテストを自動的に行う“regression test”というツールが付属しています．

```
$ cd /usr/local/src/postgresql-6.3.2注2/test/regression
$ gmake all runtest
```

●注2
6.4の場合はposstgresql-6.3.2をpostgresql-v6.4としてください．

筆者の環境での実行例を図2.5.1に示します．regression testでは実際に問い合わせを発行して得られた結果と，あらかじめ用意しておいた「正しい」結果を比較し，一致すればokと表示します．テストに使われるSQL文はsqlの下にあります．PostgreSQLで使えるSQL文を理解する上でとても参考になりますので，一度見ておくとよいでしょう．

regression testの結果が全部okならば問題ありませんが，この実行例を見ると，と

●図2.5.1　regression testの実行例

```
gcc -I../../include -I../../backend     -O2 -g -g -Wall -Wmissing-
prototypes -I../../interfaces/libpq -I../../include -fpic   -c regress.c
-o regress.o
gcc -shared -o regress.so regress.o
cd input; make all; cd ..

[途中省略]

============== destroying old regression database... =================
ERROR:  destroydb: database regression does not exist.
destroydb: database destroy failed on regression.
============== creating new regression database...   =================
============== running regression queries...         =================
boolean .. ok
char .. ok
char2 .. ok
char4 .. ok
char8 .. ok
char16 .. ok
varchar .. ok
text .. ok
strings .. ok
int2 .. ok
int4 .. ok
oid .. ok
oidint2 .. ok
oidint4 .. ok
oidname .. ok
float4 .. ok
float8 .. failed
numerology .. ok
point .. ok
lseg .. ok
box .. ok
path .. ok
```

```
polygon .. ok
circle .. ok
geometry .. failed
timespan .. ok
datetime .. ok
reltime .. ok
abstime .. ok
tinterval .. ok
horology .. ok
comments .. ok
create_function_1 .. ok
create_type .. ok
create_table .. ok
create_function_2 .. ok
constraints .. ok
triggers .. ok
copy .. ok
create_misc .. ok
create_aggregate .. ok
create_operator .. ok
create_view .. ok
create_index .. ok
sanity_check .. ok
errors .. ok
select .. ok
select_into .. ok
select_distinct .. ok
select_distinct_on .. ok
subselect .. ok
aggregates .. ok
transactions .. ok
random .. failed
portals .. ok
misc .. ok
arrays .. ok
btree_index .. ok
hash_index .. ok
select_views .. ok
alter_table .. ok
portals_p2 .. ok
euc_jp .. ok
ACTUAL RESULTS OF REGRESSION TEST ARE NOW IN FILE regress.out
rm regress.o
```

ところどころ“failed”が見受けられます．これは，テストの結果を判断するアルゴリズムが，文字列を比較するという単純なものであるために起こる現象です．典型的なケースは以下です．

■エラーメッセージの違い

テストケースには，故意にエラーになるような問い合わせが含まれています．このときに出力されるエラーメッセージはプラットフォームによって異なることがあります．

■浮動小数点の表現の違い（float8など）．

これもプラットフォームによって異なることがあります．

■乱数を使ったテスト（random）

乱数はその性質上，実行結果がプラットフォームによって異なることがあります．

テスト項目がfailした場合，期待される結果との差分がregression.diffsというファイルに記録されますので，上記にあてはまるかどうか，ある程度判断の材料になります．

2.5.4　postmasterの自動起動

今のままでは，システムを再起動するたびに手動でpostmasterを再起動する必要があります．システムの起動時に自動的にpostmasterが起動されるようにするためには，システムに変更が必要です．

変更方法についてはINSTALLやcontrib/Linux/にも書いてありますが，ここでは非常に簡単な方法を示します[注3]．

●注3
この作業はrootになって行ってください．

▶ Linuxの場合

/etc/rc.d/rc.localにリスト2.5.1を追加します．

▶ FreeBSDの場合

/usr/local/etc/rc.dにpgsql.shというファイルを作成し，リスト2.5.2の内容を書き込んだあと，以下のように実行権を与えます．

●リスト2.5.1　/etc/rc.d/rc.localへの追加

```
POSTGRESDIR=/usr/local/pgsql
if [ -x $POSTGRESDIR/bin/postmaster -a -d $POSTGRESDIR/data ];then
        rm -f /tmp/.s.PGSQL.5432
        su - postgres -c "postmaster -S -i"
        echo -n 'postmaster '
fi
```

●リスト2.5.2　pgsql.sh

```
#! /bin/sh
POSTGRESDIR=/usr/local/pgsql
if [ -x $POSTGRESDIR/bin/postmaster -a -d $POSTGRESDIR/data ];then
        rm -f /tmp/.s.PGSQL.5432
        su - postgres -c "postmaster -S -i"
        echo -n 'postmaster '
fi
```

```
# chmod 755 pgsql.sh
```

ここで注意事項を．

/tmp/s.PGSQL.5432はpostmasterが使用するUNIXドメインのソケットで，“5432”はポート番号を表します．このスクリプトでは，システムのクラッシュなどでソケットが残ってしまった場合に備えて削除を行っています．使用するポート番号が標準の5432と異なる場合はこの数字を変えてください．

2-6 ユーザ登録＆ユーザ DB 作成

ここまでのステップでPostgreSQLのインストールは完了しています．しかし，現在の状態ではpostgresユーザしかPostgreSQLを利用できません．一般ユーザがPostgreSQLを使えるようにするためには，ユーザ登録とユーザDBの作成を行う必要があります．

2.6.1 ユーザ登録

UNIXに「ユーザ」という概念があるように，PostgreSQLにも「データベースユーザ」（あるいは単に「ユーザ」）があります．PostgreSQLを使う場合は，あらかじめユーザ登録を行っておく必要があります．なお，postgresというユーザ[注1]はinitdbのときに自動的に登録済みとなっています．

●注1
正確にはinitdbを発行したユーザのことです．

```
createuser ユーザ名
```

PostgreSQLのユーザを登録するためには，createuserコマンドを使います．createuserを実行できるのは，PostgreSQLのスーパユーザだけです．ここではユーザpostgresで実行します．たとえば“ishii”というユーザを登録する場合は図2.6.1のようになります．

①の1000はユーザを管理する「ユーザID」で，1～2147483648の値を指定します．デフォルト値はUNIXのユーザIDと同じ値です．通常はデフォルト値でよいでしょう．登録しようとするユーザがUNIXユーザとして登録されていない場合は，

●図 2.6.1 新規にユーザを登録する（下線部を入力）

```
% createuser ishii

Enter user's postgres ID or RETURN to use unix user ID: 1000 -> 1000 ──→ ①
Is user "ishii" allowed to create databases (y/n) y ──→ ②
Is user "ishii" allowed to add users? (y/n) n ──→ ③
createuser: ishii was successfully added
```

```
Enter user's postgres ID -> 12345
```

のように適当な値を設定します．

②では，このユーザに新しいデータベースを作る権限を与えるかどうかを設定します．通常は“y”と答えてかまいません．

③では，このユーザに新しいユーザを追加する権限を与えるかどうかを設定します．ここで“y”と答えると，このユーザは任意のユーザを追加できるようになるばかりでなく，PostgreSQLのスーパユーザと同じ権限を持つようになるので注意してください．通常は“n”と答えておきましょう．

ユーザ名は31バイト以内のアルファベットa～z，A～Z，数字の0～9，記号の_（アンダースコア）が使えます．-（ハイフン）などの記号は使えません．日本語もいちおう使えるようですが，問題点がないかどうか完全には確認できていません．これ以外のオプションについてはオンラインマニュアル[注2]をご覧ください．

なお，一度登録したユーザを削除するためには，destroyuserコマンドを使います．

●注2
createdb(1)のこと．

```
% destroyuser foo
```

destroyuserコマンドはPostgreSQLのスーパユーザだけが実行できます．

また，PostgreSQLでは，ユーザにパスワードの設定ができますが，その方法については2.8節で説明します．

2.6.2　データベースの作成

PostgreSQLでは複数のデータベースを持つことができます．たとえば，以下のような使い方が考えられます．

- 部署ごとにデータベースを使い分ける
- 学部，学科ごとにデータベースを使い分ける
- ユーザごとにプライベートなデータベースを持つ

ただしデータベースが異なると，その間でJOINなどによるテーブルの相互参照はできませんので注意が必要です．

データベースを作成するには，createdbコマンドを使います．

```
$ createdb データベース名
```

PostgreSQLでは，データベース名を省略した場合，ユーザ名と同じデータベース名がデ

フォルト値になるので，ユーザ用のプライベートなデータベースなどはユーザ名と同じにしておくのがよいでしょう．

本書のインストール例では，データベースは

```
/usr/local/pgsql/data/base/
```

の下に作られます．/usr/local/pgsql/data/base/のあるパーティションのディスクが不足している場合には，別のディレクトリにデータベースを作ることもできます．そのためには，まずinitlocationというコマンドでそのディレクトリを初期化します．initlocationの引数はデータベースディレクトリです．使用例を示します．

```
$ initlocation /psqgl/data
```

次にcreatedbを実行します．

```
$ createdb -D /pgsql/data test
```

6.4では絶対パスのロケーションが指定できなくなりました．以下のように環境変数を使ってください．

```
$ initlocation /pgsql/data
$ export PGDATA2=/pgsql/data
$ postmaster -S -i
$ createdb -D PGDATA2 test
```

その他のcreatedbのオプションについては，オンラインマニュアルをご覧ください．

なお，バージョン6.4ではデータベースを作る際に文字コードを指定できます．これは，initdbで指定した文字コードと異なる文字コードのデータベースを作りたい場合に使用します．文字コードの指定は-Eオプションを使用します．

```
$ createdb -E 文字コード
```

文字コードは，configureやinitdbと同様のものが使えます．

データベースを消去するには，destroydbコマンドを使います．

```
$ destroydb test
```

destroydbコマンドはPotgreSQLのスーパユーザだけが実行できます．

2-7 基本的な使い方

ひと通り基本的な設定ができたので，さっそくPostgreSQLを使ってみることにしましょう．ここでは“test”というデータベースを使います．前節で解説したcreatedbで作っておいてください．

2.7.1 psql

PostgreSQLでは，psql[注1]というSQLインタプリタを利用するのがもっとも基本的な使い方です．

psqlは起動されるとSQL文の入力待ちとなります．SQL文が入力されるとそれをバックエンド[注2]に転送し，実行結果がバックエンドから戻って来るとそれを表示します．

psqlは会話的に使うことを想定されていますが，多くのUNIXコマンドがそうであるように，シェルスクリプトなどからバッチ的に実行することもできます．実際，createdbやcreateuserはpsqlを呼び出すシェルスクリプトとして実装されています．

psqlは非常にシンプルなコマンドですが，応用範囲は広く，きちんとマスターすれば多くの仕事がpsqlで片付きます．たとえば，CGI用のスクリプトをpsqlで作ることも十分可能です．

psqlの基本的な使い方は，

```
$ psql データベース名
```

です．データベース名を省略すると，ユーザ名と同じデータベース名を指定したものとみなします．

データベースはcreatedbによってあらかじめ作成済みでなければなりません．どんなデータベースがあるかわからないときは，

```
$ psql -l
```

●注1
sqlplusと同じようなもの，といえばOracleを使ったことのある方ならピンと来るでしょう．

●注2
PostgreSQLのデータベースサーバです．

でデータベースの一覧表を表示できます．

　psqlは，EUCコードを使えば日本語を受け付けます．クライアント側の都合でSJIS（シフトJIS）を使いたいときは，環境変数PGCLIENTENCODINGにSJISを設定してください．ただし，この場合でもあくまでデータベース自体はEUCコードでデータが管理されます．環境変数PGCLIENTENCODINGは，単にバックエンドに対してコード変換の必要性を教えるだけです．

▶ bashのとき

```
$ export PGCLIENTENCODING=SJIS
```

▶ csh/tcshのとき

```
$ setenv PGCLIENTENCODING SJIS
```

　日本語を使用する場合は，

```
$ psql -n データベース名
```

のように，-nオプションを指定する必要があります．psqlでは，bashやtcshと同じようなコマンドヒストリや行編集の機能をGNUのreadlineライブラリを使って実現しているのですが，このライブラリは日本語を通さないため，-nオプションを使ってreadlineライブラリを無効化する必要があるからです．readlineライブラリを使わないでPostgreSQLを作成した場合は-nオプションは不要です．また，

```
$ psql test < test.sql
```

のように，ファイルからコマンドを入力する場合も-nは不要です．

　psqlが正常に起動されると図2.7.1のように表示されます．test=>はpsqlのプロ

●図2.7.1　pgsqlを実行したところ

```
$ psql test
Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT for copyright terms of POSTGRESQL

   type \? for help on slash commands
   type \q to quit
   type \g or terminate with semicolon to execute query
 You are currently connected to the database: test

test=>
```

ンプトです．この状態で任意のSQL文か，「バックスラッシュコマンド」というpsqlの内部コマンドを入力することができます．

2.7.2　テーブルの作成

SQLでは，テーブルの作成は“create table”というSQL文を使います．たとえば，「品名」と「値段」をそれぞれ“hinmei”と“nedan”カラムとして持つ“shinamono”というテーブルを作ってみましょう．

```
create table shinamono (hinmei text, nedan int);
```

といったSQL文をpsqlのプロンプトから入力してみてください．なお，textは可変長文字列のデータ型，intは整数を扱うデータ型です[注3]

●注3
PostgreSQLで使えるデータ型については3.3節をご覧ください．

```
test=> create table shinamono (hinmei text, nedan int);
CREATE
test=>
```

SQLでは“create”“table”などの構文上のキーワードとテーブル名，カラム名は大文字と小文字が区別されません．したがってこのSQL文は

```
CREATE TABLE SHINAMONO (HINMEI TEXT, NEDAN INT);
```

と入力してもまったく同じです．

また，SQL文は複数行に分けて書くことができます．この場合，psqlは;が入力されるまでtest->というプロンプトを出して入力を促進します．

```
test=> create table shinamono (
test->     hinmei text,
test->     nedan int
test-> );
```

SQLでは，構文要素の間のタブやスペースは無視されますので，この例のように適当にインデントを付けて見やすくすることができます．

SQL文中にはコメントを付けることもできます．

```
test=> create table shinamono (
```

●図 2.7.2　\dコマンドで確認

```
test=> \d shinamono

Table    = shinamono
+--------------------------------+--------------------------------+-------+
|              Field             |              Type              | Length|
+--------------------------------+--------------------------------+-------+
| hinmei                         | text                           |   var |
| nedan                          | int4                           |     4 |
+--------------------------------+--------------------------------+-------+
test=>
```

●図 2.7.3　テーブル名やカラム名には日本語も使える

```
test=> create table 品物 (
test->     品名 text,
test->     値段 int
test-> )
CREATE;
test=> \d 品物

Table    = 品物
+--------------------------------+--------------------------------+-------+
|              Field             |              Type              | Length|
+--------------------------------+--------------------------------+-------+
| 品名                           | text                           |   var |
| 値段                           | int4                           |     4 |
+--------------------------------+--------------------------------+-------+
test=>
```

```
test->     hinmei text,       -- 品名
test->     nedan int    -- 値段
test-> );
```

このように，コメントに日本語も使えます．

作ったテーブルの構造（スキーマ定義）はpsqlの\dコマンドで確認できます（図2.7.2）．日本語のテーブル名やカラム名も使えます（図2.7.3）．ただし，日本語のテーブル名やカラム名は実際にプログラムを書く際に不便だったりするのであまりお勧めできません．あくまでデモ程度に留めておくのがよいでしょう．

2.7.3　データ登録

テーブルができたところでデータを登録してみましょう．データを登録するには，insertを使います．

```
test=> insert into shinamono values ('みかん', 100);
INSERT 18419 1
test=> insert into shinamono values ('りんご', 150);
INSERT 18420 1
test=> insert into shinamono values ('メロン', 2500);
INSERT 18421 1
test=>
```

SQLでは，文字列は' 'で囲む必要があることに注意してください．

2.7.4　データの表示

データ登録できたのでテーブルの中身を表示してみます．データの表示にはselectを使います．

```
test=> select * from shinamono;
hinmei|nedan
------+-----
みかん|  100
りんご|  150
メロン| 2500
(3 rows)
```

selectの次の＊は「ターゲットリスト」と呼ばれ，どの項目を表示するかを指定します．＊を指定すると，テーブルの全カラムを表示しますが，特定のカラムだけを表示することもできます．

```
test=> select hinmei from shinamono;
hinmei
```

```
------
みかん
りんご
メロン
(3 rows)
```

価格の合計を表示することもできます.

```
test=> select sum(nedan) from shinamono;
 sum
----
2750
(1 row)
```

合計ができるのなら，平均もできそうですね.

```
test=> select avg(nedan) from shinamono;
avg
---
916
(1 row)
```

合計や平均のほかに，最小値（min)，最大値（max)，個数（count）などが使えます.

もしもすべての品物の値段が10％上がったらそれぞれいくらになるでしょう？

```
test=> select nedan, cast(nedan as float) * 1.1 as takai from
shinamono;
nedan|takai
-----+-----
  100|  110
  150|  165
 2500| 2750
(3 rows)
```

このように，ターゲットリストにはカラムだけでなく，カラムに対する演算も書けます．ここでcastは，明示的な型変換を行います．nedanカラムがint型なのに対し，1.1は実数ですから，castを使って型を合わせているわけです．PostgreSQLでは，cast

と同じ働きをする::演算子があります．これを使えば，

```
select nedan::float * 1.1 from shinamono;
```

となります．こちらの方が簡潔で使いやすいのですが，SQL標準ではありません．
なお，6.4では型変換が自動的に行われるので，

```
select nedan * 1.1 from shinamono;
```

だけで大丈夫です．また“as takai”は，値上がり後の値段の表題を“takai”とする指示です．

2.7.5 Update

次に，値段を変更してみましょう．たとえば，みかんの値段を10円安くしたいとします．データの変更には“update”を使います．

```
test=> update shinamono set nedan = nedan -10 where hinmei = 'みかん';
UPDATE 1
test=> select * from shinamono;
hinmei|nedan
------+-----
りんご|  150
メロン| 2500
みかん|   90
(3 rows)
```

ここで，where以降を省略してしまうと，すべての品物の値段が10円安くなってしまうので注意しましょう．

2.7.6 Delete

最後に削除ですが，deleteを使います．みかんの行を削除してみましょう．

```
test=> delete from shinamono where hinmei = 'みかん';
```

PostgreSQLの表現

「テーブル」「カラム」「行」に対して，さまざまな呼び方がありますが，これらはほぼ同じ意味です．

・テーブル
表，リレーション，クラス（PostgreSQL特有の表現）

・カラム
列，アトリビュート，属性

・行
タプル，レコード，インスタンス（PostgreSQL特有の表現）

```
DELETE 1
test=> select * from shinamono;
hinmei|nedan
------+-----
りんご|  150
メロン| 2500
(2 rows)
```

where以降を省略すると，すべての行が削除されてしまうのでくれぐれも注意してください．

2.7.7　Drop table

deleteでは，テーブルの中身は消去できますが，テーブルは残ります．テーブル自体を削除するにはdrop tableを使います．

```
test=> drop table shinamono;
DROP
```

2.7.8 Copy

PostgreSQLでは，テキストファイルからデータベースへデータを取り込むことと，逆にデータベースのデータをテキストファイルに出力することができます．このためのコマンドがpsqlの\copyコマンドです．

テキストファイルからのデータ入力

まず，テキストファイルにデータを用意します．今，このファイルが/tmp/in.datという名前で置いてあるとします．in.datの中身はこんな感じです．

```
みかん　100
りんご　150
メロン　2500
```

ご覧のように，1行に1レコード分，各カラムのデータをそのまま書いておきます．カラムの間はタブ1個で区切ります．このファイルの文字コードは基本的にサーバ側と同じ文字コードでなければなりません[注4]．このファイルからデータベースにデータを取り込むには，psqlから以下のコマンドを実行します．

●注4
ただし，PGCLIENTENCODINGがセットしてあれば，その文字コードにしておきます．

```
test=> \copy shinamono from /tmp/in.dat
```

うまくいけば，

```
Successfully copied.
```

と表示されます．

なお，同じテキストファイルに対して何度も\copyを実行すると，そのたびに同じデータが重複して追加されます．

テキストファイルへのデータ出力

逆にテキストファイルにデータベースからデータを吐き出すのも\copyです．

```
test=> \copy shinamono to /tmp/in.dat
```

やはりこれもうまくいけば，

```
Successfully copied.
```

と表示されます．出力されたデータの文字コードはサーバ側と同じ文字コードです．ただし，PGCLIENTENCODINGがセットしてあれば，その文字コードになります．

2.7.9　psqlのヘルプ機能

```
create table/insert/update/delete/drop table
```

などのSQLの簡単な使い方を，psqlのヘルプ機能で知ることができます．\hと入力することにより，図2.7.4のようにSQLの一覧が表示されます（;はいりません）．

特定のSQLコマンドのヘルプを見るには，

```
\h コマンド名
```

●図2.7.4　pgsqlのヘルプ画面

```
test => \h
type \h <cmd> where <cmd> is one of the following:
    abort                   abort transaction       alter table
    alter user              begin                   begin transaction
    begin work              cluster                 close
    commit                  commit work             copy
    create                  create aggregate        create database
    create function         create index            create operator
    create rule             create sequence         create table
    create trigger          create type             create user
    create view             declare                 delete
    drop                    drop aggregate          drop database
    drop function           drop index              drop operator
    drop rule               drop sequence           drop table
    drop trigger            drop type               drop user
    drop view               end                     end transaction
    explain                 fetch                   grant
    insert                  listen                  load
    lock                    move                    notify
    reset                   revoke                  rollback
    select                  set                     show
    update                  vacuum

type \h * for a complete description of all commands
test=>
```

●図2.7.5　selectコマンドの使用例

```
test=> \h select
Command: select
Description: retrieve tuples
Syntax:
select [distinct on <attr>] <expr1> [as <attr1>], ... <exprN> [as
<attrN>]
        [into [table] <class_name>]
        [from <from_list>]
        [where <qual>]
        [group by <group_list>]
        [having <having_clause>]
        [order by <attr1> [ASC | DESC] [using <op1>], ... <attrN> ]
        [union [all] select ...];
```

●表2.7.1　バックスラッシュコマンドの一覧

\?	バックスラッシュコマンドの表示
\df	関数の一覧表を表示
\do	オペレータの一覧表を表示
\dS	システムテーブルの一覧表を表示
\dt	ユーザテーブル一覧表を表示
\dT	型の一覧表を表示
\l	データベースの一覧表を表示
\z	ACL（アクセス権限）付きでテーブルの一覧表を表示

と入力します（図2.7.5）．selectにはずいぶんといろいろな使い方があるものですね．

2.7.10　バックスラッシュコマンド

すでに述べた\h，\d，\copy以外でよく使うバックスラッシュコマンドを紹介します（表2.7.1）．これ以外のバックスラッシュコマンドについては，オンラインマニュアル（psql(1)）をご覧ください．

2.7.11　psqlの起動オプション

psqlの起動時に指定できるオプションを上手に使えば，シェルスクリプトの中でpsqlがより効果的に活用できるようになります．ここではよく使われるものを紹介します．

詳しくは，オンラインマニュアルをご覧ください．

■ -c query

SQL文をquery（クエリー）で与えます．通常，-qオプションと併用し，シェルスクリプトの中からpsqlを呼び出します．-qオプションはpsqlが起動されたときのwelcomeメッセージを出さないようにします．さらに-tオプションも併用すると，selectの結果に表示される(10rows)などのメッセージやカラム名も出力しなくなります．なお，SQLコマンドの中にシェルの特殊文字が含まれる場合は，"などを使ってクエリーをquoteしておく必要があります．

■ -e

バックエンドに送ったSQL文を表示します．ファイルからSQL文を入力しているときに，何が実行されているかを知りたいときに使います．

■ -f filename

filenameから入力を受け付けます．

■ -h hostname

ホストhostnameのバックエンドに接続します．

■ -H

出力がHTMLのテーブルになります．CGIなどに使うと便利です．一緒に-Tオプションを使うと，<table>タグへのオプションが指定できます．たとえば，デフォルトではテーブル出力の際に区切り線が表示されませんが，

```
-T border
```

とすれば，区切り線が表示されます．もっとも，セルのバックグラウンドカラーを指定するなどの細かいことはできませんから，凝ったことをしたい場合はpsqlの出力をsedなどで加工したほうがよいでしょう．

■ -l

データベースの一覧を表示し，終了します．対話的な使い方はできません．

■ -o filename

出力結果をfilenameに格納します.

■ -p port

接続の際に使用するポート番号を指定します．デフォルトは5432です.

■ -u

このオプションを指定すると，psqlは接続前にパスワードを聞いてきます．これは，後述のセキュリティ機能を設定したときに使用します.

2-8 セキュリティ機能

本節では，PostgreSQLのセキュリティ機能について説明します．個人でデータベースを使う場合は別として，実際の業務においてはセキュリティ管理は重要な問題です．不用意に貴重なデータが削除されたり，部外者に取り引きデータを見られたりすることを防がなければなりません．

PostgreSQLでは，以下のように必要に応じてセキュリティを細かく設定できます．

- 前述のホスト名によるセキュリティは，host based authentication（HBA）と呼ばれ，バックエンド接続時の基本的なセキュリティを提供します．
- 個々のユーザごとにパスワードによる認証が可能です．パスワードの認証機構にはcrypt authenticationとflat file authenticationの2通りの方法があります．
- テーブルごとにアクセス権を設定できます．これはSQLの標準で決まっている方法で，SELECT/INSERT/UPDATE/DELETEの各機能をユーザ／グループに対して許可／不許可を設定できます．

2.8.1 host based authentication（HBA）

前述のように，HBAの設定はpg_hba.confで行います．pg_hba.confのフォーマットは

```
host DBNAME IP_ADDRESS ADDRESS_MASK USERAUTH [AUTH_ARGUMENT]
```

のようになります．以下，各キーワードについて解説します．

- **host**
  固定キーワードです．
- **DBNAME**

●表2.8.1　ユーザ認証の方法

ident	ident（RFC1413）による認証を行います
trust	認証を行いません
reject	条件がマッチした場合，接続を拒否します
password [passwd_file]	flat file パスワードによる認証を行います
crypt	PostgreSQLのシステムテーブルpg_shadowを使って認証を行います passwordとcryptについては次章で説明します
kbr4/kbr5	Kerberos V4/V5による認証を行います．kbr4/kbr5による認証については， 桑村氏のページ（http://www.rccm.co.jp/%7Ejuk/）に詳しい解説があります．

データベース名を指定します．“ all ”を指定すると，すべてのデータベースが対象となります．

- **IP_ADDRESS，ADDRESS_MASK**

  2.5節を参照してください．

- **USERAUTH**

  PostgreSQLユーザの認証を行う方法を指定します（表2.8.1）

pg_hba.confは上から順に見て行き，条件にマッチした行が見つかるとそれが適用されます．

```
host        all        192.168.0.1   255.255.255.255   reject
host        all        0.0.0.0       0.0.0.0           trust
```

したがってこの例では，192.168.0.1以外のホストからの接続が許可されます．

2.8.2 flat file password authentication

flat file password authenticationでは，基本的にデータベースディレクトリ（/usr/local/pgsql/data/）にパスワードファイルを用意し，それを使ってユーザ認証を行います．

```
host  unv     133.65.96.250   255.255.255.255 password passwd
```

たとえば，上の行はunvというデータベースに対して133.65.96.250のホストからのみ接続を許可し，ユーザは/usr/local/pgsql/data/passwdを使って認証を行います．

認証用のパスワードファイルはpg_passwdコマンドで作成します．pg_passwdは

postgresユーザで実行します.

```
$ cd /usr/local/pgsql/data
$ pg_passwd passwd
File "passwd" does not exist.  Create? (y/n): y ←最初だけ聞かれます
Username: foo                    ←認証の対象となるユーザ名
New password:                    ←パスワードを入力（エコーされません）

Re-enter new password:           ←確認のためパスワードをもう一度入力
```

すでに登録済みのユーザのパスワードの変更もpg_passwdで行います．ただし，postgresアカウント以外ではpg_passwdを実行できないので，ユーザが自分でパスワードを変更することはできません.

認証の対象となるユーザで接続するには，-uオプションを使います.

```
Username: foo          ←認証の対象となるユーザ名
Password:              ←パスワードを入力（エコーされません）
```

認証に失敗した場合は，次のメッセージが表示されます.

```
Connection to database 'test' failed.
User authentication failed
```

flat file password authenticationでは，ユーザが入力したパスワードは，rloginやtelnetと同様，そのままの形でネットワーク上を流れます．したがって，悪意あるユーザがネットワークを流れるパケットを盗聴すれば，パスワードが人に知られてしまいます．この点，次に紹介するcrypt authenticationではパスワードが暗号化されてからネットワークに送出されるので，より安全と言えます.

なお,

```
host  unv     133.65.96.250   255.255.255.255 password
```

のようにpg_hba.confのエントリからパスワードファイル名を省略すると，次に述べるcrypt authenticationのパスワードデータベースを見にいくようになります．ただ，このような使い方にメリットがあるのかどうかは疑問です.

2.8.3 crypt authentication

●注1
viewとは，実テーブル（この場合pg_shadow）への検索結果から作られる仮想的なテーブルです．viewを作成するにはcreate viewというSQL文を使います．viewは普通のテーブルと同じように検索できますが，更新はできません．

PostgreSQLのシステムテーブルpg_userとpg_shadowを使って認証を行う方法です．実際にユーザ名とパスワードが格納されているのはpg_shadowの方で，このテーブルはPostgreSQLのスーパユーザ以外はアクセスできません．pg_userはpg_shadowから作られたview[注1]で，一般ユーザもアクセスできますが，パスワードカラムは“****”となっていて，もちろんパスワードを読み取ることはできません．

crypt authenticationでは，パスワードが暗号化されてからネットワークに流れるので，盗聴の心配がありません．crypt authenticationを使うためには，pg_hba.confで“crypt”というキーワードを使います．

FreeBSDにおけるcrypt authenticationの問題点

FreeBSDでは暗号に関する輸出規制の関係で，暗号化の部分に北米向けとは異なる互換パッケージが使われています．このため，FreeBSDにおけるcrypt authenticationの動作は他のプラットフォームとは異なる動作をします．この結果，FreeBSD上のバックエンドにFreeBSD以外のプラットフォームから接続できない，逆にFreeBSDのクライアントからFreeBSD以外のバックエンドに接続できない，という問題が生じます（もちろんFreeBSDどうしでは問題ありません）．

メーリングリストでは，この問題に対して2つの解決方法が提案されました．

①DES互換パッケージをインストールする

北米向けの暗号化パッケージDESと互換のdesというパッケージをインストールする方法です．以下のURLが参考になります．

```
http://www.y-min.or.jp/~nob/FreeBSD/des.html
http://www.jp.FreeBSD.ORG/%7Eryuchi/QandA/HTML/290.html
http://www.jp.FreeBSD.ORG/%7Eryuchi/QandA/HTML/291.html
http://www.jp.FreeBSD.ORG/www.freebsd.org/ja/handbook/handbook55.html
http://www.jp.FreeBSD.ORG/www.freebsd.org/ja/handbook/handbook56.html#58
```

②SSLeay（http://www.ssleay.org/）というパッケージをインストールし，付属のlibcrypto.aをリンクする

具体的には，Makefile.globalの

```
LDFLAGS=   -lcrypt -lnsl -ldl -lm -lbsd -lreadline -ltermcap
```

を

```
LDFLAGS=   -L/usr/local/ssl/lib/lib -lcrypto -lnsl -ldl -lm -lbsd -lreadline -ltermcap
```

に変更します．なお，プラットフォームによってLDFLAGSの内容が異なりますが，libcryptの代わりに/usr/local/ssl/lib/lib/libcrypto.aをリンクするのがポイントです．

これでうまくいくはずなのですが…筆者の環境（FreeBSD2.2.6-RELEASE+SSLeay 0.8.1）では成功していません．

```
host  unv     133.65.96.250   255.255.255.255 crypt
```

また，pg_shadowテーブルにユーザをパスワード付きで登録することも必要です．そのためには，SQL文の“create user”を使います．

```
create user foo with password "foo";
```

これでユーザfooが，パスワードfooで登録されます．

パスワード付きのユーザにpsqlで接続する場合は，-uオプションを使います．

```
$ psql -u test
Username: foo          ←認証の対象となるユーザ名
Password:              ←パスワードを入力（エコーされません）
```

すでに登録済みのユーザのパスワードを変更するには“alter user”文を使います．

```
alter user foo with password "bar";
```

これでパスワードが“bar”に変更されます．create userもalter userもPostgreSQLのスーパユーザしか実行できないことに注意してください．

2.8.4 テーブルごとのアクセス権設定

SQL文のgrant/revokeにより，select/insert/update/delete/ruleの実行権をユーザ／グループに対して許可／不許可の設定ができます．SQLの標準によれば（PostgreSQLもそうですが），テーブルを作成しただけでは，そのテーブルを作成したユーザ以外にはいっさいアクセスが許されていませんので，必要に応じてgrant文を使って，他のユーザがアクセスできるようにします．

grant

grantは，select/insert/update/deleteの実行権を与えます．基本的な構文は以下のようになります．

```
grant 権限の種類 on テーブル名 to 対象;
```

「権限の種類」は，all，select，insert，update，delete，ruleのどれかです．権限は，で区切って複数列挙することができます．allを指定すると，すべての権限が対象

となります.

「テーブル名」は許可の対象となるテーブルです．やはり,で区切って複数のテーブルを列挙できます.

「対象」は許可の対象となるユーザなどです．publicとすると，すべてのユーザが対象となります．このほか,「グループ」を定義し，そのグループに対して許可する方法もありますが，ここでは触れません.

grant文の使用例を挙げると

```
grant select on shinamono to public;
```

これですべてのユーザがshinamonoテーブルをselectできるようになります.

revoke

grantと逆に，権限を取り上げるにはrevokeを使います.

```
revoke 権限の種類 on テーブル名 from 対象;
```

権限の種類，テーブル名，対象の指定の仕方はgrantとまったく同じです.

アクセス権限の確認

現在そのテーブルに与えられているアクセス権限はpsqlの\zコマンドで確認できます.

```
test=> \z
Database    = test
 +-----------------+-------------------------------------------+
 |  Relation       |            Grant/Revoke Permissions       |
 +-----------------+-------------------------------------------+
 | shinamono       |                                           |
 +-----------------+-------------------------------------------+
```

初期状態ではこのように権限リストは空です．ここでユーザfooに対してselect権限を与えてみます.

```
test=> grant select on shinamono to foo;
```

```
CHANGE

test=> \z
Database    = test
 +-----------------+-----------------------------------------------------+
 |  Relation       |            Grant/Revoke Permissions                 |
 +-----------------+-----------------------------------------------------+
 | shinamono       | {"=","foo=r"}                                       |
 +-----------------+-----------------------------------------------------+
```

この権限リストの見方ですが，= の左側がユーザです．public の場合は何も表示されません．= の右側は権限リストです．

```
r      select
a      insert
w      update/delete
R      rule
```

この例では，一般ユーザ（public）には何の権限もなく，ユーザfoo にはselect 権限が与えられていることがわかります．