PHP
Hypertext
Preprocessor

Part-2

PHPと
データベースの連携

第２部

PHP

Hypertext Preprocessor

Chapter-1 データベースのすすめ

ある程度の大きさのシステムを作る場合、データベースが必須になってきます。これはPHPにかぎらず、どんな言語を使ったシステムでも同じです。

データベースというと、普通の人から見ると縁遠いもののように思われがちですが、それは従来のデータベースシステムが「重厚・長大」で、価格的にも一般ユーザには手の届かないものであったことによるものでしょう。

幸い最近では、無償で使えるフリーのデータベースが手に入るようになりました。無償とはいえ、本書で紹介するPostgreSQLのように、商用データベースに迫る機能や性能を持っているものもあるのですから、これを利用しない手はありません。ぜひこの機会にPHPを使ってデータベースにチャレンジしてみてはいかがでしょう。

1.1 データベースを使う四つの理由

ところで、データベースとは何でしょうか。実際にデータベースを使うことのメリットはどこにあるのでしょうか。データベースは難解といわれますが、本当にそうなのでしょうか。本章ではこういった疑問を解明していきます。

ごく簡単にいえば、データベースとは、コンピュータが管理するデータの入れ物です。データを入力して保存しておき、あとから取り出すことのできるものならなんでもデータベースとみなすこともできますが、一般にはこういった目的に特化したソフトウェアを「データベース・マネージメント・システム」と呼びます。少々長いので本書では単に「データベース」あるいは「データベース・システム」と呼ぶことにします。

単にデータを出したり入れたりするだけなら、テキストエディタやワープロ、表計算ソフトでも可能ですが、本格的なデータベースにはそれなりの長所があります。

● 速い

100個くらいのファイルなら必要なものを人間が目で探したり、grepのようなファイル検索ソフトで検索してもたいして時間はかかりません。しかし、これが100万個だったらどうなるでしょう。検索方法にもよりますが、データベースなら100万個から目的のデータを見つけるのに要する時間は通常、秒単位であり、比較にならないほど高速です。

● **整理整頓**

「あれ、あのファイルはどこにあったかな？」とディスクの中を探しまわった経験はどなたもお持ちでしょう。複数のパソコンを使っていると、どのパソコンにファイルを作ったかすら忘れてしまうことも珍しくありません。データベースを使えば、このような心配は無用です。また、データベースは通常ネットワークに対応しているので、ネットワーク上のどのマシンからも同じようにアクセスできます。

● **安心**

データベースは、データの安全性について特別の配慮をしています。たとえば、データの処理中にトラブルがあっても、データが破壊されにくいような構造になっています。また、複数の人が同じデータに同時にアクセスしてもデータがおかしなことにならないような工夫もなされています。

● **高度な問い合わせ機能**

データベースは、データを検索、操作するためのSQL（エス・キュー・エル）という特別な言語を持っています。SQLは非常に強力で、たとえば「関東地区で去年の12月と比べ、今年の12月の売り上げが10%以上伸びている店舗と、そこに勤めている従業員の数を求めよ」などという質問がSQLなら1行で書けます。普通のプログラミング言語なら結構な量のコードを書くことになるでしょう。

このように、データベースは実用的なシステムを作る際にぜひとも欲しくなる機能を備えています。

1.2 PostgreSQL

実はデータベースにもいろいろなタイプがあります。その中で現在最も広く使われているのが、リレーショナル（関係）データベースと呼ばれるものです。本書では、リレーショナルデータベースのひとつであるPostgreSQL（「ポストグレス」あるいは「ポストグレエスキューエル」と読みます）を例にとって、PHPからデータベースを使う方法を紹介します。PostgreSQLには以下のような特徴があります。

● **フリー＆オープンソース**

フリーソフトなので、無償で利用できます。しかも、営利目的にも利用できるライセンスであり、企業向けの本格的なシステム構築も可能です。もちろんソースもすべて公開されており、改造も自由です。

● **本格的なデータベース機能**

トランザクション、主キー、外部キーなどの機能を備えた本格的なデータベースです（トランザクションや主キーについてはあとで述べます）。また、データベースの標準言語である

SQLをサポートしています。基本的な機能においては商品として販売されているデータベースに引けを取りません。

● サポート

フリーソフトというとサポートが心配されますが、PostgreSQLはコミュニティの活動が活発で、メーリングリストを通じて良好なサポートが得られます。また、書籍などの資料も比較的充実しており、情報源に困りません。それでも心配な場合には、商用サポートを利用することもできます。

● プラットフォームを選ばない

PostgreSQLは、Linuxをはじめ、ほとんどのUNIX系のシステムに移植されています。もちろんSolarisなどの商用のUNIXでも動きますし、Windows NTで使うこともできます。むしろ、使えないプラットフォームをさがすほうが難しいほどです。

● 扱いが簡単

データベースというと、導入や設定が難しいものだという印象がありますが、PostgreSQLは簡単な初期化コマンドを実行するだけですぐに使えるようになります。運用の手間もほとんどかかりません。

● ネットワークに対応

PostgreSQLは、ネットワーク越しに利用できるシステムです。極端な話、日本のサイトにあるWWWサーバが、インターネットを経由して米国のサイトに置いてあるPostgreSQLのサーバにアクセスすることも可能です。

● PHPから呼び出しが可能

もちろんこれが一番大事なことですが、PHPから呼び出すことができます。そのほか、C言語やPerl、Javaなどさまざまな言語からPostgreSQLを利用することができます。

● オブジェクト指向機能

PostgreSQLには、以下のような先進的なオブジェクト指向機能が備わっています。

・配列の利用
・テーブルの継承
・ユーザ定義データ型

本書ではこれらについての詳細な解説は行ないません。PostgreSQL付属のマニュアルや部末の参考文献[1][5]を参照してください。

Hypertext Preprocessor

HP

Chapter-2 PostgreSQLを使ってみよう

本章では、実際にPostgreSQLを使いながらデータベースについて実践的に学んでいきます。本章に沿ってPostgreSQLを操作していくことにより、データベースに関する知識が自然に身につくことと思います。

PostgreSQLのインストール手順については第4部を参照してください。

2.1 データベースユーザの登録

PostgreSQLのユーザには**スーパユーザ**と**一般ユーザ**の二種類があります。UNIXにはスーパユーザというものがあり、システムの管理のために特別な権限が与えられています。PostgreSQLのスーパユーザもまったく同じで、データベースの管理を行なう特別なユーザとなっています。スーパユーザは一切のセキュリティチェックが適用されないため、使い方を誤るとデータベースを壊しかねません。かならず「一般ユーザ」を登録し、通常の作業は一般ユーザで行なうようにしてください。

PostgreSQLのユーザを登録するには、PostgreSQLのスーパユーザになってcreateuserというUNIXコマンドを実行します。ユーザ名にはアルファベットで始まる31バイト以内のアルファベット(小文字)、数字、アンダースコア(_)の組み合わせが使えます。空白は使えません。日本語のユーザ名も作れますが、プログラミングの際に煩わしいことになる可能性が高いので、お勧めしません。

ここでは、fooというデータベースユーザを登録してみましょう。PostgreSQLでは、データベースユーザ名とUNIXのユーザ名が一致している必要はありませんが、本書ではUNIXユーザとしてもfooというアカウントがすでに存在しているものとして話を進めます。

```
# su - postgres
$ createuser foo

Shall the new user be allowed to create databases? (y/n) y
```

このユーザが新しいデータベースを作ることができるかどうかをyまたはnで答えます。通常はyでよいでしょう。

```
Shall the new user be allowed to create more new users? (y/n) n
```

このユーザが新しいユーザを追加できるかどうかをyまたはnで答えます。この質問にyと答えるとスーパユーザと同等の権限を持つユーザを作成することになるので、通常はnと答えておきます。

```
CREATE USER
```

createuserコマンドが成功すれば、このメッセージが表示されます。

createuserとは反対に、ユーザを削除するにはdropuserコマンドを使います。dropuserコマンドはスーパユーザだけが実行できます。dropuserの使い方についてはオンラインマニュアルを参照してください。

UNIXと同様、PostgreSQLのユーザにもパスワードが設定できます。パスワードを設定する方法は「3.10.5　データベースを使ったBasic 認証」で説明します。

2.2 データベースの作成

ユーザ登録が終わったら次はデータベースの作成です。fooというアカウント名でUNIXにログインし、以下のコマンドを実行します。

```
$ createdb foo
CREATE DATABASE
```

これでfooという名前のデータベースが作成されました。このようにユーザ名と同じ名称のデータベースを作っておくと、psqlコマンド（次節で説明します）を使ったときにデータベース名を指定する必要がなくなります。以後、本章では特に断らないかぎり、ユーザ名としてfoo、データベースとしてもfooを使うことにします。

データベースを削除するにはdropdbコマンドを使います。ただし、他人が作ったデータベースを削除することはできません（PostgreSQLスーパユーザなら可能です）。

Column　データベース名の使い分け

データベースは、名前が重複しない限りいくらでも作成することができますが、データベースが異なるとその中に格納されているデータはまったく別のものとして扱われ、お互いに関連を持つことができません。したがって、実際の業務で複数のユーザが使用するようなデータは、ひとつのデータベースに収納しておかなければなりません。逆にユーザと同じ名前のデータベースは、そのユーザ専用で利用するのがよいでしょう。

2.3 psql入門

以上で準備ができたので、いよいよデータベースを実際に使ってみましょう。PostgreSQLには対話的にデータベースを操作できるpsqlというコマンドが付属しています。とりあえずpsqlを起動し、データベースにログインしてみます。

```
$ psql
Welcome to psql, the PostgreSQL interactive terminal.

Type:  ¥copyright for distribution terms
       ¥h for help with SQL commands
       ¥? for help on internal slash commands
       ¥g or terminate with semicolon to execute query
       ¥q to quit

foo=>
```

上記のようなメッセージが表示されれば、データベースに正常にログインできています。

ユーザ名と異なるデータベースを使いたい場合はpsqlの引数で指定します。たとえば、データベースtestを使用する場合は

```
$ psql test
```

とします。

使用できるデータベースがわからない場合はpsql -lでデータベースの一覧表を見ることができます。

```
$ psql -l
             List of databases
     Name       |   Owner   |   Encoding
----------------+-----------+---------------
 template0      | postgres  | EUC_JP
 template1      | postgres  | EUC_JP
 foo            | foo       | EUC_JP
(3 rows)
```

Nameはデータベース名、Ownerはデータベースの所有者名、Encodingはそのデータベースの文字コードを表します。EUC_JPは日本語の文字コードのひとつで、「日本語用のEUC」を意味します。

PostgreSQLは日本語だけでなく、各国の文字を使うことができます。日本語EUC以外に

使える文字コードについては、PostgreSQLのソースに付属のドキュメント（doc/README.mb.jp）を参照してください。

このほかにもpsqlには数多くのオプションがあり、**psql -h**でそれらを表示できます。詳しくはオンラインマニュアルをご覧ください。

Column

psqlのオンラインヘルプ

psqlにはSQL文の使い方を表示するオンラインヘルプ機能があります。オンラインヘルプを見るには、¥hのあとにSQL文を指定します。たとえば、CREATETABLEの使用方法は、以下のようにして調べることができます。

```
foo=> ¥h CREATE TABLE
Command:     CREATE TABLE
Description: define a new table
Syntax:
CREATE [ [ LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name (
    { column_name data_type [ DEFAULT default_expr ] [ column_constraint [, ... ]
]
    | table_constraint }  [, ... ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | UNIQUE | PRIMARY KEY |
  CHECK (expression) |
  REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) |
  PRIMARY KEY ( column_name [, ... ] ) |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ]
) ]
    [ MATCH FULL | MATCH PARTIAL ] [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

また、単に¥hとすると、利用可能なSQL文の一覧が表示されます。

2.4 テーブルの作成

データベースは複数の**テーブル**を持つことができます。テーブルは日本語では**表**ですが、その名の通りリレーショナルデータベースではすべてのデータを表形式で扱います。さきほどのデータベース一覧表も実はPostgreSQLが内部的に使用しているデータベース一覧表テーブル（pg_databaseという名前のテーブル）の内容を表示しているのです。*1

テーブルには**行**と**列**があり、それぞれテーブルの縦方向と横方向を表しています。*2

列には名前があります。これを**列名**あるいは**アトリビュート**と呼びます。列は名前だけでなく、**データ型**を持ちます。PostgreSQLでは、SQL標準で規定されている各種データ型とPostgreSQL固有のデータ型を使用することができます。**表2-1**に代表的なデータ型を挙げておきます。

基本的に列名とデータ型はテーブルを作るときに決まり、以後は変化しません。それに対して行のほうは、テーブルを作ったときにはひとつもありませんが、データを追加するたびに増えていきます。

テーブルを作成するには、psqlに対して**CREATE TABLE**というSQL文を入力します。

CREATE TABLE文の基本形は以下のようになります。

```
CREATE TABLE テーブル名 (列名1 データ型, 列名2 データ型...);
```

SQL文では文字列（'abc'のように'で囲ったもの）以外は大文字と小文字を区別しないので、

```
create table テーブル名 (列名1 データ型, 列名2 データ型...);
```

でも同じことですが、本書ではSQLのキーワード（CREATE、TABLEなど）を大文字で、それ以外を小文字で表記することにします。

テーブル名、列名で使える文字については、ユーザ名と同じ規則が適用されます。**表2-2**にある予約語は使えません。*3

それでは実際にテーブルを作ってみましょう。このテーブルは小学生の夏休みの宿題で定番の「お天気日記」です。**図2-1**をotenki.sqlという名前でファイルにセーブし（漢字コードはかならずEUCにしてください）、

```
$ psql -e -f otenki.sql
```

として実行してください。

*1
PostgreSQLではこのようにシステム内部で使用するテーブルを「システムテーブル」あるいは「システムカタログ」と呼び、名前が pg_ で始まることになっています

*2
行は「タプル」、列は「カラム」と呼ばれることもあります。さらにPostgreSQLのマニュアルではテーブルを「クラス」、行を「インスタンス」と呼んでいる箇所があります。

*3
delimited identifierという二重引用符（"）による記法を使えないこともありませんが、のちのちめんどうなことになるのでお勧めしません。

表 2-1 PostgreSQL で使用可能な主なデータ型

データ型	意　味
CHAR(n)	n 文字の文字列（n バイトに満たない場合は空白で埋められる）
VARCHAR(n)	最大 n 文字の可変長文字列
TEXT	可変長文字列
SMALLINT	2 バイト整数
INTEGER	4 バイト整数
BIGINT	8 バイト整数
NUMERIC	多倍長整数+小数（有効桁数1000桁まで）
DECIMAL	多倍長整数+小数（有効桁数1000桁まで）
FLOAT	浮動小数点
DOUBLE PRECISION	倍精度浮動小数点
DATE	日付
TIME	時刻
TIMESTAMP	日付と時刻
INTERVAL	時間間隔
BOOL	真/偽
OID	オブジェクト ID

表 2-2 予約語

ALL	ANALYSE	AND	ANY
AS	ASC	BETWEEN	BINARY
BOTH	CASE	CAST	CHECK
COLLATE	COLUMN	CONSTRAINT	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER
DEFAULT	DEFERRABLE	DESC	DISTINCT DO
ELSE	END	EXCEPT	FALSE
FOR	FOREIGN	FREEZE	FROM
FULL	GROUP	HAVING	ILIKE
IN	INITIALLY	INNER	INTERSECT
INTO	IS	ISNULL	JOIN
LEADING	LEFT	LIKE	LIMIT
NATURAL	NEW	NOT	NOTNULL
NULL	OFF	OFFSET	OLD
ON	ONLY	OR	ORDER
OUTER	OVERLAPS	PRIMARY	PUBLIC
REFERENCES	RIGHT	SELECT	SESSION_USER
SOME	TABLE	THEN	TO
TRAILING	TRUE	UNION	UNIQUE
USER	USING	VERBOSE	WHEN
WHERE			

図 2-1　お天気データベースの定義

```
-- お天気日記テーブルの作成
DROP TABLE otenki;
CREATE TABLE otenki (
  day DATE DEFAULT 'today' PRIMARY KEY,  -- 日付(主キー)
  tenki TEXT DEFAULT '晴れ',       -- 天気(晴れ、曇...)
  ondo INTEGER DEFAULT 25 CHECK(ondo IS NULL OR ONDO > -50 AND ONDO < 50),  -- 温度
  uryou INTEGER DEFAULT 0 CHECK(uryou IS NULL OR URYOU >= 0), -- 雨量
  CONSTRAINT otenki_check
  CHECK(tenki IS NULL OR
  (tenki = '晴れ' AND uryou = 0 OR tenki = '曇' OR tenki = '雨'))
);
```

実行結果

```
$ psql -e -f otenki.sql foo
DROP TABLE otenki;
DROP
CREATE TABLE otenki (
  day DATE DEFAULT 'today' PRIMARY KEY,
  tenki TEXT DEFAULT '晴れ',
  ondo INTEGER DEFAULT 25 CHECK(ondo IS NULL OR ONDO > -50 AND ONDO < 50),
  uryou INTEGER DEFAULT 0 CHECK(uryou IS NULL OR URYOU >= 0),
  CONSTRAINT otenki_check
  CHECK(tenki IS NULL OR
  (tenki = '晴れ' AND uryou = 0 OR tenki = '曇' OR tenki = '雨'))
);
psql:/home/t-ishii/public_html/red-mammoth-php4-second/otenki.sql:11: NOTICE:
CREATE TABLE / PRIMARY KEY will create implicit index 'otenki_pkey' for table
'otenki'
CREATE
```

ご覧のように、otenkiテーブルが作成されました。SQLでは、改行や空白、タブなどは無視されます。また、--以降は**コメント**として扱われます。

dayのあとに**PRIMARY KEY**とあるのは、この列が**主キー**であることの指定です。主キーとは、その列の値を指定すれば、行がユニークに定まるような列のことです。したがって、主キーとなる列では重複した値が許されません（もしも重複した値があると、行がユニークに決まりません）。お天気日記ですから、同じ日の日記が二つあってはいけないので、日付を主キーにするのが適当です。

そのほか、ondoやuryouの列の定義に CHECK(ondo...) のような部分がありますが、これはデータに対する制約条件です。制約については「3.7　データ入力フォーム」で説明します。

作成したテーブルの内容はpsqlの**バックスラッシュ（¥）コマンド**である**¥d**で確認できます。psqlを起動し、**¥d otenki**と入力してみてください。

```
¥d otenki
               Table "otenki"
 Column |  Type   |        Modifiers
--------+---------+--------------------------
 day    | date    | not null default 'today'
 tenki  | text    | default '晴れ'
 ondo   | integer | default 25
 uryou  | integer | default 0
Primary key: otenki_pkey
Check constraints: "otenki_ondo" ((ondo IS NULL) OR ((ondo > -50) AND (ondo < 50)))
                   "otenki_uryou" ((uryou IS NULL) OR (uryou >= 0))
                   "otenki_check" ((tenki IS NULL) OR ((((tenki = '晴れ'::text) AND
(uryou = 0)) OR (tenki = '曇'::text)) OR (tenki = '雨'::text)))
```

Columnは列名、**Type**は型名を表します。

Modifierとあるのは、その列のオプション的な情報です。day列は主キーですが、主キーの場合はNULL（データが存在しない、あるいは定まらないことを示すSQLの予約語）が許されないので、**not null**が自動的に指定されます。

Index: otenki_pkeyはday列に対応する**インデックス**です。インデックスとは、データを高速に検索するためのしくみで、PostgreSQLでは主キーが指定されると自動的にインデックスが作成されます。

Check constraints:は制約を示します。**制約**については「3.7　データ入力フォーム」で説明します。

CREATE TABLEにはこのほかにも数多くのオプションがあります。詳しくはオンラインマニュアルの「create_table（7）」か、Reference Manual「I. SQL Commands」の「CREATE TABLE」を参照してください。

2.5 データの追加

テーブルに行を追加するには、SQL文の**INSERT**を使います。INSERTの基本形は以下のようになります。値の並びはテーブルの列の並びに対応している必要があります。

```
INSERT INTO テーブル名 VALUES(値1, 値2, ...);
```

早速INSERTを使ってデータを登録しましょう。先ほど作成したotenki.sqlに以下を追加し、psqlから同じように実行してください。[*4]

```
INSERT INTO otenki VALUES('2002-8-1','晴れ',30,0);
INSERT INTO otenki VALUES('2002-8-2','晴れ',30,0);
```

*4 付属CD-ROMのarticles/2/otenki.sqlに同じものが収録されています。

```
INSERT INTO otenki VALUES('2002-8-3','曇',27,10);
INSERT INTO otenki VALUES('2002-8-4','曇',27,8);
INSERT INTO otenki VALUES('2002-8-5','曇',26,7);
INSERT INTO otenki VALUES('2002-8-6','晴れ',29,0);
INSERT INTO otenki VALUES('2002-8-7','雨',27,120);
INSERT INTO otenki VALUES('2002-8-8','雨',26,140);
INSERT INTO otenki VALUES('2002-8-9','雨',25,200);
INSERT INTO otenki VALUES('2002-8-10','雨',25,100);
INSERT INTO otenki VALUES('2002-8-11','曇',27,10);
```

```
DROP TABLE otenki;
DROP
CREATE TABLE otenki (
  day DATE DEFAULT 'today' PRIMARY KEY,
  tenki TEXT DEFAULT '晴れ',
  ondo INTEGER DEFAULT 25 CHECK(ondo IS NULL OR ONDO > -50 AND ONDO < 50),
  uryou INTEGER DEFAULT 0 CHECK(uryou IS NULL OR URYOU >= 0),
  CONSTRAINT otenki_check
  CHECK(tenki IS NULL OR
  (tenki = '晴れ' AND uryou = 0 OR tenki = '曇' OR tenki = '雨'))
);
psql:/home/t-ishii/public_html/red-mammoth-php4-second/otenki.sql:11:
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index 'otenki_pkey'
for table 'otenki'
CREATE
INSERT INTO otenki VALUES('2002-8-1','晴れ',30,0);
INSERT 5243050 1
INSERT INTO otenki VALUES('2002-8-2','晴れ',30,0);
INSERT 5243051 1
INSERT INTO otenki VALUES('2002-8-3','曇',27,10);
INSERT 5243052 1
INSERT INTO otenki VALUES('2002-8-4','曇',27,8);
INSERT 5243053 1
INSERT INTO otenki VALUES('2002-8-5','曇',26,7);
INSERT 5243054 1
INSERT INTO otenki VALUES('2002-8-6','晴れ',29,0);
INSERT 5243055 1
INSERT INTO otenki VALUES('2002-8-7','雨',27,120);
INSERT 5243056 1
INSERT INTO otenki VALUES('2002-8-8','雨',26,140);
INSERT 5243057 1
INSERT INTO otenki VALUES('2002-8-9','雨',25,200);
INSERT 5243058 1
INSERT INTO otenki VALUES('2002-8-10','雨',25,100);
INSERT 5243059 1
INSERT INTO otenki VALUES('2002-8-11','曇',27,10);
INSERT 5243060 1
```

もちろんpsqlを起動して直接INSERT文を直接入力してもかまいませんが、日本語が混在しているSQL文を入力する場合は、プラットフォームによっては**-n**オプションを追加してpsqlを起動する必要があるかもしれません。

INSERTについての詳細はオンラインマニュアル「insert (7)」か、Reference Manual「I. SQL Commands」の「INSERT」の項を参照してください。

2.6 データの表示

テーブルの行を取り出すには、**SELECT**文を使います。SELECTを使って行を取り出すことを**問い合わせ**、あるいは**クエリ**(query)と呼びます。

SELECT文は非常に強力で、すべての機能を紹介するだけで1冊の本を書けるほどです。詳しくは部末の参考文献[2][3][4]などを参照してもらうとして、ここでは基本的な点についてのみ説明しておきます。SELECTの基本形は以下の通りです。

SELECT 選択項目**1,**選択項目**2... [FROM** テーブル**1,** テーブル**2... [WHERE** 条件式**]]**

選択項目には列名などを指定します。選択項目にアスタリスク(*)のみを指定すると、そのテーブルの全列が選択項目になります。

条件式を指定することにより必要な行だけを取り出すことができます。WHERE以降を省略すると、すべての行を取り出します。

先ほどのotenkiテーブルの内容を取り出してみましょう。以下をpsqlから実行してください。

```
foo=> SELECT * FROM otenki;
    day      | tenki | ondo | uryou
-------------+-------+------+-------
 2002-08-01 | 晴れ   |   30 |     0
 2002-08-02 | 晴れ   |   30 |     0
 2002-08-03 | 曇     |   27 |    10
 2002-08-04 | 曇     |   27 |     8
 2002-08-05 | 曇     |   26 |     7
 2002-08-06 | 晴れ   |   29 |     0
 2002-08-07 | 雨     |   27 |   120
 2002-08-08 | 雨     |   26 |   140
 2002-08-09 | 雨     |   25 |   200
 2002-08-10 | 雨     |   25 |   100
 2002-08-11 | 曇     |   27 |    10
(11 rows)
```

WHEREを使う

WHEREを使って目的の行を取り出してみましょう。

例 雨の日のデータを取り出す

```
foo=> SELECT * FROM otenki WHERE tenki = '雨';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-07 | 雨    |   27 |   120
 2002-08-08 | 雨    |   26 |   140
 2002-08-09 | 雨    |   25 |   200
 2002-08-10 | 雨    |   25 |   100
(4 rows)
```

例 温度が26度以下の日のデータを取り出す

```
foo=> SELECT * FROM otenki WHERE ondo <= 26;
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-05 | 曇    |   26 |     7
 2002-08-08 | 雨    |   26 |   140
 2002-08-09 | 雨    |   25 |   200
 2002-08-10 | 雨    |   25 |   100
(4 rows)
```

AVGを使う

SQLでは平均を求めることもできます。

```
foo=> SELECT AVG(ondo) AS 平均温度 FROM otenki;
   平均温度
---------------
 27.1818181818
(1 row)
```

ご覧のように、ASを使うと結果として表示される列名を変更することができます。

SUMを使う

SQLでは合計を求めることもできます。

```
foo=> SELECT SUM(uryou) AS 合計雨量 FROM otenki;
 合計雨量
----------
      595
(1 row)
```

このほか、最大(MAX)、最小(MIN)も利用できます。

2.7 データの変更

データを変更するにはUPDATE文を使います。UPDATE文の基本形は以下のとおりです。

UPDATE テーブル名 **SET** 列名1 **=** 値1**,** 列名2 **=** 値2 **[WHERE** 条件式**];**

8月3日の雨量を20にする例を示します。

```
foo=> SELECT * FROM otenki WHERE day = '2002-8-3';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-03 | 曇    |   27 |    10
(1 row)
foo=> UPDATE otenki SET uryou = 20 WHERE day = '2002-8-3';
UPDATE 1
foo=> SELECT * FROM otenki WHERE day = '2002-8-3';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-03 | 曇    |   27 |    20
(1 row)
```

値は相対値で指定することもできます。8月4日の雨量を10増やす例を示します。

```
foo=> SELECT * FROM otenki WHERE day = '2002-8-4';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-04 | 曇    |   27 |     8
(1 row)
```

```
foo=> UPDATE otenki SET uryou = uryou + 10 WHERE day =
'2002-8-4';
UPDATE 1
foo=> SELECT * FROM otenki WHERE day = '2002-8-4';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-04 | 曇    |   27 |    18
(1 row)
```

UPDATEについての詳細は、オンラインマニュアル「update (7)」か、Reference Manual「I. SQL Commands」の「UPDATE」の項を参照してください。

2.8 データの削除

行を削除するには**DELETE**文[*5]を使います。DELETE文の基本形は以下のとおりです。

DELETE FROM テーブル名 **[WHERE** 条件式**];**

8月1日のデータを含む行を削除する例を示します。

```
foo=> SELECT * FROM otenki WHERE day = '2002-8-1';
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2002-08-01 | 晴れ  |   30 |     0
(1 row)

foo=> DELETE FROM otenki WHERE day = '2002-8-1';
DELETE 1

foo=> SELECT * FROM otenki WHERE day = '2002-8-1';
 day | tenki | ondo | uryou
-----+-------+------+-------
(0 rows)
```

WHERE以降を指定しないでDELETEを実行すると、テーブルの行が**すべて**削除されてしまうので、くれぐれも注意してください。

DELETEについての詳細はオンラインマニュアル「delete (7)」か、Reference Manual「I. SQL Commands」の「DELETE」の項を参照してください。

*5 ある行の特定の列だけを削除することはできません。

Hypertext Preprocessor

Chapter-3 PHPとPostgreSQLの連携

第2章では、PostgreSQL単独の使い方を説明しました。本章では、いよいよPHPからPostgreSQLをアクセスします。

3.1 準備

PHPからデータベースを使う場合、通常、データベースユーザがnobodyになるため、PostgreSQLにnobodyのユーザ登録が必要です。postgresユーザでcreateuserコマンドを実行してください。

```
$ createuser nobody
Shall the new user be allowed to create databases? (y/n) n
n
Shall the new user be allowed to create more new users? (y/n) n
n
CREATE USER
```

また、第2章で作成したお天気テーブルは、今のままではfoo以外からアクセスできません。そこでGRANTというSQLコマンドでnobodyユーザにもアクセスを許可することにします。fooユーザでログインし、psqlを起動後、以下を実行してください。

```
GRANT ALL ON otenki TO nobody;
```

本章で紹介するサンプルプログラムは付属CD-ROMの/examples/2/以下に収録してあります。実際に表示するためにはfooユーザでログインして以下の手順を実施します。

```
$ cd ~foo
$ chmod 755 .
$ mkdir public_html
$ cd public_html
$ cp -r /mnt/cdrom/2/examples2/ .
```

これらのサンプルプログラムはフリーソフトであり、利用するのはもちろん、改造するなどして自由に活用していただいてかまいません。再利用するにあたっての条件は、COPYRIGHTというファイルに書いてあります。

以後、本章ではプログラムのファイル名において、~foo/public_html/examples/の部分を省略して表記します。たとえば、~foo/public_html/examples/ex1/ex1.phpは、単にex1/ex1.phpと表記します。

これから九つのサンプルについて解説しますが、サンプル1ならex1、サンプル2ならex2というようにディレクトリ名をつけています。

サンプルプログラムは、ブラウザに以下のようなURLを入力して表示することができます。サンプル8と9を除き、exで始まるファイルが「メインプログラム」なので、ブラウザで表示するときはそれをURLとして指定します。

http://localhost/~foo/examples/プログラムのファイル名

たとえば、ex1/ex1.phpなら、

http://localhost/~foo/examples/ex1/ex1.php

というURLになります。

サンプル8と9は、ex*.phpではなく、index.htmlを指定してください。たとえば、

http://localhost/~foo/examples/ex8/index.html

というURLになります。

読者の便宜を図るために、メニュー画面を用意してあります。ここからサンプルを選んで動かすこともできます。以下のURLで表示してください（**図2-2**）。

http://localhost/~foo/examples/index.html

図2-2　メニュー画面

これで準備ができました。

3.2 PHPを使ってテーブル内容を表示する

3.2.1 サンプルプログラム1

それでは手はじめに第2章のお天気テーブルをPHPを使って表示してみましょう。まずはリスト2-1に掲載したサンプルプログラム1の実行結果を見てください（図2-3）。

図2-3 サンプルプログラム1の実行例

リスト2-1 サンプルプログラム1（ex1/ex1.php）

```
 1  <html>
 2  <head><title>Example 1</title></head>
 3  <body>
 4
 5  <?php
 6  @$con = pg_connect("dbname=foo");        // データベースに接続する
 7  if ($con == false) {
 8    print("データベースに接続できませんでした。理由: $php_errormsg<br>¥n");
 9    exit;
10  }
11  $sql = "SELECT day AS 年月日, tenki AS 天気, ondo AS 温度, uryou AS 雨量 FROM
otenki ORDER BY day";
12
13  @$result = pg_query($sql); // selectを実行
14  if ($result == false) {
15    printf("SQL実行に失敗しました。理由: " . pg_last_error($con));
16    exit;
17  }
18  $rows = pg_num_rows($result);            // 行数を取得
19  $columns = pg_num_fields($result);       // 列数を取得
```

```
20
21  print("<table border>¥n");
22
23  for ($j = 0;$j < $rows;$j++) {
24    if ($j == 0) {
25      print("<tr>");
26      for ($i = 0;$i < $columns;$i++) {
27        $str = pg_field_name($result,$i);        // 列名の取り出し
28        print("<th>$str</th>");
29      }
30      print("</tr>¥n");
31    }
32
33    print("<tr>");
34    for ($i = 0;$i < $columns;$i++) {
35      $str = pg_fetch_result($result,$j,$i);     // データの取り出し
36      print("<td>$str</td>");
37    }
38    print("</tr>¥n");
39  }
40
41  pg_free_result($result);                       // 検索結果の解放
42  pg_close($con);                                // データベースとの接続切断
43  print("</table>¥n");
44  ?>
45  </body>
46  </html>
```

pg_で始まる関数がPostgreSQLにアクセスするための関数です。

●pg_connect（6行目）

PHPからPostgreSQLを使う場合、まずPostgreSQLに接続するという手順が必要になります。これを実行するのがpg_connectです。

引数は接続の詳細を指定するパラメータで、

キーワード=値

のように指定します。

キーワードには、host（接続先のホスト名）、dbname（データベース名）、password（接続パスワード）などがあります。hoseには接続先のホスト名、つまりpostmasterが動いているホスト名を指定します。サンプルのように省略すると、自ホストに対してUNIXドメインソケットを使った接続になります。自ホストへの接続にはこの方法をお勧めします。

dbnameはデータベース名です。省略するとユーザ名と同じデータベース名を指定したことになります。

なお、@pg_connect()のように@をつけているのは、PHPが表示するエラーメッセージを抑止するためです。このような場合、エラーが発生するとグローバル変数$php_errormsgにエラー内容の文字列が入っているので、サンプルのようにそれを表示すればエラーの詳細が通知できますが、そのまま表示すると英文でわかりにくいだけでなく、不必要なシステムの詳細を漏らすことになるので、セキュリティ上も好ましくありません。実際の業務プログラムではもっと適切なエラーメッセージ、たとえば「システムエラーが発生しました。システム管理者に連絡してください」のようなメッセージを表示したほうがよいでしょう。

●pg_query（13行目）

pg_queryは第1引数にpg_connectが返却したデータベースへの接続ハンドルをとり、第2引数で与えられたSQL文を実行します。

ところで、ここで実行しているSELECT文（11行目）ですが

```
SELECT day AS 年月日, ....
```

のようにASを使ってCREATE TABLEで定義した列名dayではなく、日本語の「年月日」などとして見やすくしています。ORDER BY dayは日付順にソートする指定です。

●pg_num_rows（18行目）

pg_num_rowsはpg_query()の結果を引数とし、検索された行数を返します。

●pg_num_fields（19行目）

pg_num_fieldsはpg_query()の結果を引数とし、列数を返します。

●pg_field_name（27行目）

pg_field_nameは、pg_queryの結果と列番号（0から数えます）を引数とし、対応する列の名前を返します。ここでは、表の見出し部分を表示するのに使っています。

●pg_fetch_result（35行目）

pg_fetch_resultはpg_queryの結果から指定行、列のデータを返します。ここでは、表のデータ部分を表示するのに使っています。

●pg_free_result（41行目）

pg_free_resultはpg_queryの作成したメモリ中の検索結果を解放します。本サンプルプログラムのようにpg_queryを実行したあとですぐにpg_closeを呼ぶような場合、pg_closeによって後処理が行なわれるのでpg_free_resultをかならずしも呼ばなくてもよいのですが、続けてpg_queryを使用する場合はかならずpg_free_resultを呼び出して不要になったリソースを解放するようにしてください。

●pg_close（42行目）

pg_closeはpg_connectが確立したデータベースとの接続を切断します。実際にはPHPがスクリプトの最後で自動的にpg_closeを呼び出してくれるので、pg_closeを呼ばなくても実害はありません。

3.3 クラスライブラリで汎用化

テーブル内容を表示するような基本的な処理は、どのアプリケーションでもかならず必要になります。そのたびに同じようなスクリプトを書くのは非効率的です。そこで汎用ライブラリ化し、必要に応じてアプリケーションから呼び出せるようにしましょう。次のような関数にしてみます。

```
PgSelect($dbname, $sql) // $dbanme: データベース名 $sql: SQL文
```

これで、この関数を呼び出すだけで、アプリケーションからテーブル内容を表示できるようになります。しかし、あるとき表の枠を「なし」で表示したくなりました。そこで引数を追加して

```
// $dbanme: データベース名 $sql: SQL文 $boder: 枠の有無
PgSelect($dbname, $sql, $border)
```

としてみました。これで枠の有無を制御できるようになりました。ところがこんどはセルの背景色を変えたくなりました。また引数を追加するのでしょうか？ これではきりがありません。このような問題を解決するのがオブジェクト指向の考え方です。

今回は検索と結果の表示を行なうクラスPgSelect（ex2/lib/pgselect.inc）を作成します。

3.3.1 PgSelectクラス

PgSelectクラス（**リスト2-2**）はデータベースに対して検索を行ない、結果をHTMLのテーブルのかたちで表示します。その際、見栄えや表示形式をカスタマイズしやすいように表示上のポイントとなる箇所を別々のメンバ関数にしています（**表2-3**）。

表2-3　PgSelectクラスのカスタマイズ可能メンバ関数

機能	関数名	引数
テーブル開始タグ印字	printTableHeader	なし
列名印字	printHeader	列番号、列名
データ印字	printData	列番号、データ

PgSelectクラスのコンストラクタ（11～13行目）は、別に定義してあるDbConnectクラスを呼び出し、データベースへの接続を確立します。このようにクラスの内部でデータベースへの接続を管理するので、PgSelectクラスの利用者はデータベースを接続管理を意識する必要がなくなります。DbConnectクラスについてはあとで説明します。

実際に検索を実行するのはdoSelect（31行目）で、引数としてSQL文を取ります。もっとも、doSelectが直接pg_queryを呼び出してSQLを実行しているわけではなく、DbConnectクラスが提供する関数doQueryを使って間接的にSQLを実行しています。

リスト2-2 PgSelectクラス（ex2/lib/pgselect.inc）

```
<?php
/*
 * 検索結果をテーブル形式で表示する
 */
require_once("dbconnect.inc");

class PgSelect {
  var $db; // データベース接続ハンドル

  // コンストラクタ
  function PgSelect() {
    $this->db = new DbConnect();
  }

  // テーブル開始タグの印字
  function printTableHeader() {
    print("<table border>¥n");
  }

  // 列名の印字
  function printHeader($i, $str) {
    print("<th>$str</th>");
  }

  // データの印字
  function printData($i, $str) {
    printf("<td>%s</td>¥n",htmlspecialchars($str));
  }

  // 検索の実行
  function doSelect($sql) {
    $result = $this->db->doQuery($sql);      // selectを実行
    $rows = pg_num_rows($result);            // 行数を取得
    $columns = pg_num_fields($result);       // 列数を取得

    $this->printTableHeader();

    for ($j = 0;$j < $rows;$j++) {
      if ($j == 0) {
        print("<tr>");
        for ($i = 0;$i < $columns;$i++) {
          $str = pg_field_name($result,$i); // 列名の取り出し
          $this->printHeader($i, $str);
        }
        print("</tr>¥n");
      }
```

```
47          print("<tr>");
48          for ($i = 0;$i < $columns;$i++) {
49            $str = pg_fetch_result($result,$j,$i); // データの取り出し
50            $this->printData($i, $str);
51          }
52          print("</tr>¥n");
53        }
54        print("</table>¥n");
55        return true;
56      }
57    }
58    ?>
```

3.3.2 DbConnectクラス

DbConnectクラス（ex2/lib/dbconnect.inc）（**リスト2-3**）はデータベースへの接続を管理します。

コンストラクタ（15行目）は、getConnection（22行目）、getConnectionはdoConnect（27行目）を呼び出してデータベースへの接続を行ないます。この際、実際にデータベースへ接続するpg_connectの引数は別の設定ファイルdef.incに書かれており、dbConnectクラスはそれを読み込んでいるだけです（5行目）。

def.incの内容は以下のようになっており、データベースへの接続に必要な引数をまとめてあります。

```
<?php
define("DBNAME", "foo");// データベース名
define("HOST", "");     // ホスト名
define("PORT", "");     // ポート番号
define("USER", "");     // ユーザ名
define("PASSWORD", ""); // パスワード
?>
```

このようにすることで、接続したいデータベースに合わせて設定変更する個所がここだけになり、管理が容易になります。

直接pg_queryを呼び出さず、データベースへの問い合わせを行なうdoQuery（63行目）を設けているのは、主にエラー処理やデバッグのためです。開発中は、SQL文の誤りに起因するエラーが多発しますが、このようにSQL文を扱う場所を一カ所にすることによってSQL文の内容を印字するなどのカスタマイズが容易になります。また、場合によってはSQL文の実行時間を計測し、性能的にボトルネックになっている部分を探す手助けにすることも考えられます。

リスト2-3 DbConnectクラス（ex2/lib/dbconnect.inc）

```
 1  <?php
 2  /*
 3   * PostgreSQLデータベースへの接続を行なうクラス
 4   */
 5  require_once("def.inc");
 6
 7  class DbConnect {
 8    var $dbname = DBNAME;      // データベース名
 9    var $host = HOST;          // ホスト名
10    var $port = PORT;          // ポート番号
11    var $user = USER;          // ユーザ名
12    var $password = PASSWORD; // パスワード
13    var $con = false;          // コネクションハンドル
14
15    function DbConnect() {     // コンストラクタ
16      $this->getConnection();
17    }
18
19    // コネクションハンドルを返す
20    function getConnection() {
21      if ($this->con == false) {
22        return($this->doConnect());
23      }
24      return($this->con);
25    }
26
27    function doConnect() {
28      // データベースに接続する
29      $constr = "";
30      if ($this->dbname != "") {
31        $constr .= "dbname={$this->dbname} ";
32      }
33      if ($this->host != "") {
34        $constr .= "host={$this->host} ";
35      }
36      if ($this->port != "") {
37        $constr .= "port={$this->port} ";
38      }
39      if ($this->user != "") {
40        $constr .= "user={$this->user} ";
41      }
42      if ($this->password != "") {
43        $constr .= "password={$this->password} ";
44      }
45
46      @$this->con = pg_connect($constr);
47      if ($this->con == false) {
```

```
48          print("データベースに接続できませんでした。理由: $php_errormsg<br>¥n");
49          exit;
50        }
51        return($this->con);
52      }
53
54      // データベースとの接続切断
55      function doClose() {
56        if ($this->con != false) {
57          pg_close($this->con);
58          $this->con = false;
59        }
60      }
61
62      // データベースへの問い合わせの実行
63      function doQuery($sql) {
64        if ($this->con == false) {
65          $this->doConnect();
66        }
67        @$result = pg_query($this->$con, $sql); // selectを実行
68        if ($result == false) {
69          printf("SQL($sql)の実行に失敗しました。理由: " . pg_last_error($this->con));
70          return false;
71        }
72        return $result;
73      }
74    }
75    ?>
```

3.3.3 サンプルプログラム2

次にこれらのクラスを利用するサンプルプログラムを示します（**リスト2-4**）。

主な処理をクラスライブラリ化したために、サンプルプログラム1（**リスト2-1**）に比べると非常にシンプルになっています。処理らしい処理は、PgSelectクラスのインスタンスの生成（7行目）、検索と結果表示（8行目）くらいのものです。

ところで、5行目でpgselect.incを読み込んでいますが、実際にはpgselect.incはlibディレクトリの下にあります。いったいどうやってpgselect.incのあり場所を見つけているのでしょうか？　PHPにはrequireやincludeが読み込むファイルのあり場所を指定するinclude_pathという変数があり、この変数を設定することによってrequire/includeするファイルを探すリストを指定することができます。

リスト2-4　サンプルプログラム2（ex2/ex2.php）

```
 1  <html>
 2  <head><title>Example 2</title></head>
 3  <body>
 4  <?php
 5  require_once("pgselect.inc");
 6
 7  $sel = new PgSelect;
 8  $sel->doSelect("SELECT day AS 年月日, tenki AS 天気, ondo AS 温度, uryou AS
雨量 FROM otenki ORDER BY day");
 9  ?>
10  </body>
11  </html>
```

この方法には複数あって、php.iniファイルで指定する方法がひとつ、そして今回採用している.htaccessで指定する方法の二通りがあります。php.iniを使う方法は、すべてのPHPアプリケーションで同じ設定を使わなければならないのが問題になることがあります。その点、.htaccessで設定する方法では、スクリプトが置いてあるディレクトリ単位で制御できるので、柔軟性があります。

今回使用している.htaccesを示します。

```
# includeパスの設定
php_value include_path ".:./lib"
# "magic quote"をオフにする
php_flag magic_quotes_gpc off
php_flag magic_quotes_runtime off
```

2行目がincludeパスの設定で、ここでは、.(カレントディレクトリ)とlib以下が検索の対象となっています。

.htaccesを有効にするためには、Apacheの設定変更が必要です。以下にそのための設定を示します。httpd.confファイルに以下の設定を追加します。

```
<Directory /home/*/public_html>
     AllowOverride Options
</Directory>
```

もしfooのホームディレクトリが/home以下にない場合は、それに合わせて変更してください。また、Optionsは、allとしてもかまいませんが、Optionsとした場合に比べると、セキュリティリスクも高くなります。

一方、lib/.htaccessは以下のような設定になっています。

```
Order deny,allow
Deny from all
```

これはすべてのアクセスを拒否します。こうしておくことにより、libの下に置いてあるクラスライブラリなどのファイルのURLを直接指定してアクセスされる心配がなくなります。

3.4 表示のカスタマイズ

サンプルプログラム2による表示結果は**図2-2**とまったく同じでおもしろくありません。サンプルプログラム2に変更を加えて以下のようなカスタマイズを行なってみましょう。

①温度は棒グラフで表示
②雨量は右詰めで表示

①は、棒グラフの「種」となる四角のイメージファイルを用意しておき、

```
<td><img src="red.png" alt="グラフの種" width=100 height=10></td>
```

のようにして棒グラフの長さを指定することによって実現できます。
②は、

```
<td><align="right">25</td>
```

のようにして右詰めを指示できます。
つまり、これらを実現するためには、HTMLテーブルのデータ部の印字を変更するだけで十分であることがわかります。そこでPgSelectクラスのデータ印字関数printDataを変更します。**リスト2-5**に変更後のプログラムを示します。
8行目以降PgSelectクラスを継承した新しいクラスmyPgSelectを作成し、printDataの定義を変更しています。それにともなって34行目で呼び出すクラス名がmyPgSelectに変わっています。実行結果を**図2-4**に示します。
このように、クラスと継承を利用すれば、汎用性を失うことなく少ない変更でカスタマイズが可能になります。

リスト2-5 サンプルプログラム3（ex3/ex3.php）

```
 1  <html>
 2  <head><title>Example 3</title></head>
 3  <body>
 4
 5  <?php
 6  require_once("pgselect.inc");
 7
 8  class myPgSelect extends PgSelect {
 9    // PgSelect クラスの printData()を override
10    function printData($i, $str) {
```

```
11      switch ($i) {
12      case 2:   // 温度は棒グラフで表示
13        $width = (int)$str * 4;
14            print("<td><img  src=¥"red.png¥"  alt=¥"グラフの種¥"  width=$width
height=10></td>");
15        break;
16      case 3:   // 雨量は右詰めで表示
17        printf("<td align=¥"right¥">$str</td>");
18        break;
19      default:  // その他は左詰めで表示
20        print("<td>$str</td>");
21        break;
22      }
23    }
24  }
25
26  $sel = new myPgSelect;
27  $sel->doSelect("SELECT day AS 年月日, tenki AS 天気, ondo AS 温度, uryou AS
雨量 FROM otenki ORDER BY day");
28  ?>
29  </body>
30  </html>
```

図2-4　サンプルプログラム3の実行結果

3.5 セッション管理

表示するデータが増えてくると、画面に全データが入り切らず、ブラウザのスクロール機能を使わないとすべてのデータを見ることができなくなります。そこで、商用サイトの検索エンジンなどでは、表示データを適当な件数に分けて表示するようにしています。PgSelectクラスにもこのような機能を組み込むことを考えましょう。

実は、これは見た目よりもむずしいことなのです。というのは、あるページから別のページに移動したときに、WWWサーバは前のページのことはすっかり忘れてしまうからです。前のページが何ページだったか覚えていないことには次のページを表示できません。こう書くと「私のブラウザには『前』『次』のボタンがついていて、以前にどこのページを表示したか覚えているじゃないか」といわれる人もいるかもしれません。確かにブラウザは前のページのことを覚えています。しかしWWWサーバはそうではありません。サーバ側で動くPHPにとっても同じことです。

この問題を解決するためには、次のページに移動するときに前のページの情報を何とかして一緒に持っていくことができればよいわけです。このように、ページからページにまたがる情報を管理することを**セッション管理**と呼びます。

幸いPHP4にはセッション管理の機能があるので、これを利用することにします。PHPのセッション管理機能はなかなか優れもので、オブジェクトをそのままセッション管理の対象にすることができます。もちろんPgSelectクラスから作ったオブジェクトも管理できます。ということは、ページの状態を表すメンバ変数とそれにともなう若干のロジックを追加すれば目的を達成できそうです。

ではまず必要な変数について検討してみましょう。

● ユーザ入力のSELECT文

あるページで検索した結果は、次のページまで「持ち越す」ことはできません。これは、ページを表示し終わるたびにデータベースとの接続を切断しなければならないからです。接続を切断すれば、データベースに対して「前のページで表示したデータの次のデータをください」などということはできないのです。そこで次のページを表示するときは、もう一度同じSELECT文で検索し、前のページの続きのデータから表示します。つまりSELECT 文を覚えておく必要があるわけです。

● 表示オフセット

前のページで検索結果中1～n件まで表示した場合、次のページではn＋1件から表示しなければなりません。このnのことを**表示オフセット**と呼ぶことにします。表示オフセットもセッション管理の対象になります。

3.5.1 PgSelectクラスの拡張

以上をふまえてPgSelectクラスを拡張します（**リスト2-6**）。

① 12-14行目に$mode_var、$next、$prevという変数が追加されています。これらは、ページを変更するリクエストを制御する変数を管理するのに使います。たとえば、次のページを表示するリンクを

```
http://localhost/~foo/examples/ex4.php?mode=next
```

と設定したとします。すると、次に表示するページでは、$_GET["mode"]変数に"next"が設定されているので、次のページを表示するリクエストを処理できるようになります。

問題は、処理の切り分けに使う変数名modeで、これをスクリプトの中に明示的にコーディングしてしまうと、PgSelectクラスを使う側（ex4.php）でそのことを意識せざるを得なくなります。今回は処理の切り分けはページの「前」「次」だけの制御だけですが、処理の切り分けが必要になるのはこれだけとはかぎらず、ほかのクラスでも処理の切り分け用の変数を使うことは十分考えられます（実際次の節でそのような例が出てきます）。つまり、PgSelectクラスが固定で処理切り分け用のmodeという変数を持つのは好ましくないのです。同様の次ページ、前ページを意味する文字列も固定では持たないようにしています。

② $maxl（15行目）は、1ページに一度に表示する行数です。

以上の変数は、継承先のクラスで書き換えることができます。

③内部変数として、ユーザ入力SELECT文と表示オフセットを保持するメンバ変数を追加します（18～19行目）。

④「前のページ」「次のページ」要求を識別する関数をいくつか追加しています。

- **IsPageChangeRequested()**
  「前のページ」または「次のページ」の要求があれがtrueを返します
- **modeVar()**
  処理切り分け用変数の内容を返します
- **IsPagePrevRequested()**
  「前のページ」の要求があれがtrueを返します
- **IsPageNextRequested()**
  「次のページ」の要求があれがtrueを返します

⑤前のページ、次のページへのリンクを表示するメンバ関数を追加します（65～77行目）。この関数の中では、「前」や「次」を押したときに変数処理切り分け変数に表示方向がセットされるように、URLに引数をつけています。リンク先は自分自身で、PHPでは**$_SERVER['PHP_SELF']**が自分自身のURLを表す変数として使えます。

リスト2-6　ページ表示機能を追加したPgSelectクラス（ex4/lib/pgselect.inc）

```
 1  <?php
 2  /*
 3   * 検索結果をテーブル形式で表示する
 4   */
 5  require_once("dbconnect.inc");
 6
 7  class PgSelect {
 8    var $db;   // データベース接続ハンドル
 9
10    // カスタマイズ可能メンバ変数
11
12    var $mode_var = "mode"; // 表示モード変数名
13    var $next = "next";     // 次ページ
14    var $prev = "prev";     // 前ページ
15    var $maxl = 5;    // ページに一度に表示する行数
16
17    // 内部変数(カスタマイズ不可)
18    var $usersql = "";      // ユーザ入力SELECT文
19    var $offset = 0; // 表示オフセット
20
21    // コンストラクタ
22    function PgSelect() {
23      $this->db = new DbConnect();
24    }
25
26    // テーブル開始タグの印字
27    function printTableHeader() {
28      print("<table border>¥n");
29    }
30
31    // 列名の印字
32    function printHeader($i, $str) {
33      print("<th>$str</th>");
34    }
35
36    // データの印字
37    function printData($i, $str) {
38      print("<td>$str</td>");
39    }
40
41    // 検索結果の「前」「次」表示要求?
42    function IsPageChangeRequested() {
43      return(isset($_REQUEST[$this->mode_var]) &&
44                  ($_REQUEST[$this->mode_var] == $this->next ||
45                   $_REQUEST[$this->mode_var] == $this->prev));
46    }
47
48    // 表示モード変数の返却
```

```
 49    function modeVar() {
 50      return($_REQUEST[$this->mode_var]);
 51    }
 52
 53    // 検索結果の「前」表示要求?
 54    function IsPagePrevRequested() {
 55      return($this->IsPageChangeRequested() &&
 56             $this->modeVar() == $this->prev);
 57    }
 58
 59    // 検索結果の「次」表示要求?
 60    function IsPageNextRequested() {
 61      return($this->IsPageChangeRequested() &&
 62             $this->modeVar() == $this->next);
 63    }
 64
 65    // 「前」の表示
 66    function printPrev() {
 67      print <<<EOF
 68        <a href="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->prev}">
[前の $this->maxl 件に戻る]</a>
 69  EOF;
 70    }
 71
 72    // 「次」の表示
 73    function printNext($n) {
 74      print <<<EOF
 75        <a href="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->next}">
[次の $n 件に続く]</a>
 76  EOF;
 77    }
 78
 79    // 検索の実行
 80    function doSelect($sql = "") {
 81      if (!$this->IsPageChangeRequested()) {    // はじめての表示？
 82        if ($sql == "") {
 83          print("doSelect: SQL文が指定されていません");
 84          exit;
 85        }
 86        $this->usersql = $sql;
 87        $this->offset = 0;
 88      } else {
 89        if ($this->IsPageNextRequested()) {
 90          $this->offset += $this->maxl;
 91        } else {
 92          $this->offset -= $this->maxl;
 93        }
 94      }
```

```
 95
 96      // limit - offset 句の添付
 97      $sql = $this->usersql . " LIMIT $this->maxl OFFSET $this->offset";
 98
 99      $result = $this->db->doQuery($sql);          // select を実行
100      $rows = pg_num_rows($result);                // 行数を取得
101      $columns = pg_num_fields($result);           // 列数を取得
102
103      $this->printTableHeader();                   // テーブル開始タグの表示
104
105      for ($j = 0;$j < $rows;$j++) {
106        if ($j == 0) {
107          print("<tr>");
108          for ($i = 0;$i < $columns;$i++) {
109            $str = pg_field_name($result,$i);      // 列名の取り出し
110            $this->printHeader($i, $str);          // 列名の表示
111        }
112        print("</tr>¥n");
113        }
114
115        print("<tr>");
116        for ($i = 0;$i < $columns;$i++) {
117          $str = pg_fetch_result($result,$j,$i);  // データの取り出し
118          $this->printData($i, addslashes($str)); // データの表示
119        }
120        print("</tr>¥n");
121      }
122      pg_free_result($result);                     // 検索結果の解放
123
124      print("</table>¥n");
125
126      if ($this->offset > 0) {
127        $this->printPrev();
128      }
129
130      $offset = $this->offset + $this->maxl;
131      $sql = $this->usersql . " LIMIT $this->maxl OFFSET $offset";
132
133      $result = $this->db->doQuery($sql); // 次のページに表示するデータがあるか検索する
134      $n = pg_num_rows($result);
135
136      if ($n > 0) {   // 次のページのデータあり？
137        $this->printNext($n);
138      }
139    }
140  }
141  ?>
```

次に、実際に検索を行うメンバ関数doSelect（80～139行目）の変更点を見ていきましょう。

⑥ メンバ変数の初期化

まず、IsPageChangeRequested()を呼び出して、初めての表示かどうかチェックします。初めてだったらSQL文をメンバ変数に記憶するとともにオフセットを初期化します（86～87行目）。

そうでない場合は、IsPageNextRequested()を呼び出して表示要求を調べオフセットを増減します（89～93行目）。

⑦ LIMIT、OFFSET句の添付

ユーザから渡されたSELECT文の実行結果は、1ページに収まり切らない可能性があります。その中からこのページに必要な部分だけを切り出して使うこともできますが、データベースサーバから不必要なデータが転送される時間がもったいないですし、メモリの消費も気になります。そこで、ここではPostgreSQLに用意されている便利な機能を使って問題を解決することにします。

PostgreSQLでは、次のようにSELECT文の後ろにLIMIT、OFFSET句を追加すると、OFFSETで指定したデータから、LIMITで指定した件数分だけのデータがデータベースサーバから返却されます（下記の例では、10件目のデータから5件だけデータが返ります）。

```
SELECT ... LIMIT 5 OFFSET 10;
```

つまり必要なデータだけを直接指定して入手することができるわけで、データの転送時間、メモリが節約できます。今回の場合、OFFSET = $this->offset、LIMIT = $this->maxlとすればよいわけですから、目的のSQL文は97行目の

```
$sql = $this->usersql . " LIMIT $this->maxl OFFSET $this->offset";
```

で得ることができます。

⑧残りのデータのチェック

printNextメンバ関数を呼んで次のページへのリンクを表示する前に、次のページに表示できるデータがあるかどうかチェックします。これは簡単で、⑥⑦のSQL文と同じ方法で実現できます（130～138行目）。

3.5.2 サンプルプログラム４

では次に、改良版PgSelectクラスを使ったサンプルプログラムを見ていきましょう（**リスト2-7**）。

まず目につくのが22行目のsession_start()関数の呼び出しです。この関数を呼び出すことにより、このページがセッション管理の対象となることを宣言します。

次に目につくのがex3.php（**リスト2-5**）では一番最初に書かれていた<html>～

</body>の部分が下のほう（29～33行目）に移動していることです。これは、セッション管理を使う際の決まりごとで、セッション管理の関数群の呼び出しは、そのページに対して何か出力を行なう前でなければならないからです。

24行目で$_SESSION配列をチェックしています。session_start()を呼び出すと、セッションデータが格納されている$_SESSION配列にアクセスできるようになります。selは、myPgSelectクラスのオブジェクトですので、これが存在しないか、あるいは存在したとしても「前」「次」ページのリクエストがない場合は初回の表示ということになるので、26行目で新しくmyPgSelectのオブジェクトを作成します。

このあとはサンプルプログラム3（ex3/ex3.php：**リスト2-5**）と同じで、特に変更する必要はありません。**図2-5**がex4.phpを最初に表示した状態です。**図2-6**は、「次の5件に続く」を選択した状態です。

図2-5　サンプルプログラム4の実行結果

図2-6　次のページを表示した状態

リスト2-7　サンプルプログラム4（ex4/ex4.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3
 4  class myPgSelect extends PgSelect {
 5    // PgSelect クラスの printData()を override
 6    function printData($i, $str) {
 7      switch ($i) {
 8      case 2: // 温度は棒グラフで表示
 9        $width = (int)$str * 4;
10        print("<td><img src=¥"red.png¥" width=$width height=10></td>");
11        break;
12      case 3: // 雨量は右詰めで表示
13        printf("<td align=¥"right¥">$str</td>");
14        break;
15      default: // その他は左詰めで表示
16        print("<td>$str</td>");
17        break;
18      }
19    }
20  }
21
22  session_start(); // セッションの開始
23
24  if (!isset($_SESSION["sel"]) ||
25    !$_SESSION["sel"]->IsPageChangeRequested()) { // はじめての表示?
26    $_SESSION["sel"] = new myPgSelect; // myPgSelectオブジェクト作成
27  }
28
29  print <<<EOF
30  <html>
31  <head><title>Example 4</title></head>
32  <body>
33  EOF;
34
35   $_SESSION["sel"]->doSelect("SELECT day AS 年月日, tenki AS 天気, ondo AS 温
度, uryou AS 雨量 FROM otenki ORDER BY day");
36  ?>
37  </body>
38  </html>
```

3.6 検索フォーム

ここまでで紹介したサンプルでは、SELECT文はプログラム中に直接コーディングしていました。そのため、特定のデータを得るためには、WHERE句を手作業で作成しなければなりませんでした。しかし、検索のための条件は変わることが多く、そのたびにスクリプトを書き換えるというわけにはいきません。

そこで本節では、検索用のフォームを作って検索条件を柔軟に指定できるようにしてみます。また、検索用フォームと、検索時に実行するSQL文は自動生成しましょう。こうすればSQLを知らないユーザでも自由にデータベースの検索が行なえるようになります。

3.6.1 検索フォームのイメージ

図2-7 今回作成する検索フォーム

この検索フォームでは、検索対象となる列名に対して検索の対象となる値を入力することによって検索を行ないます。その際、演算子を指定することができます。たとえばotenkiテーブルのondo列はINTEGER型なので、=(等しい)、<>(等しくない)、>(より大きい)などの演算子から選択が可能です。

演算子と検索値が決まれば検索が実行できます。たとえば、ondo列で演算子として > 、検索値として25が入力されると温度が25度以上の行が検索の対象になります。値が入力されなかった列は検索対象になりません。複数の列が指定された場合はAND条件で検索します。**図2-4**の例では、天気が晴れかつ温度が25度以上のデータを検索します。

3.6.2 テーブルに関する情報の取得

検索用のフォームやSQL文を自動生成するためには、そのテーブルの列名、データ型、そのデータ型に許されているオペレータなどの情報が必要です。PostgreSQLでは、テーブルの定義（列名、データ型など）の情報もテーブルで管理されており、普通のSELECT文でこれらの情報を取得することができます。具体的には、以下のSQL文でotenkiテーブルに関する情報を取得できます。

```
foo=> SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
foo-> a.attnotnull, a.atthasdef
foo-> FROM pg_class c, pg_attribute a, pg_type t
foo-> WHERE c.relname = 'otenki'
foo-> AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
foo-> ORDER BY a.attnum;
 attnum | attname | typname | attlen | atttypmod | attnotnull | atthasdef
--------+---------+---------+--------+-----------+------------+-----------
      1 | day     | date    |      4 |        -1 | t          | t
      2 | tenki   | text    |     -1 |        -1 | f          | t
      3 | ondo    | int4    |      4 |        -1 | f          | t
      4 | uryou   | int4    |      4 |        -1 | f          | t
(4 rows)
```

このSELECT文の説明は省略しますが、ご覧のようにotenkiテーブルの列名（attname）、データ型（typname）、データのバイト長（attlen）などが取得されていることがわかります。同様の方法でオペレータに関する情報も取得できます。

3.6.3 検索フォームの例

検索フォームの生成、検索用のSELECT文を生成する機能を実現したのがPgMetaDataクラス（ex5/lib/pgmetadata.inc：**リスト2-9**）です。PgSelectクラスを併用することにより、テーブルを検索、表示するアプリケーションを簡単に作成することができます。

サンプルプログラム5

PgMetaDataクラスの説明に入る前に、これらのクラスを利用したサンプルプログラム5（**リスト2-8**）を見ながら全体の動きを把握しましょう。なお、プログラムを簡潔にするために、今回から温度の表示を棒グラフから元の数字の表現に戻しています。

リスト2-8 サンプルプログラム5（ex5/ex5.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3  require_once("pgmetadata.inc");
 4
 5  class myPgSelect extends PgSelect {
 6    var $mode_var = "mode";
 7  }
 8
 9  class myPgMetaData extends PgMetadata {
10    var $mode_var = "mode";  // 表示モード変数名
11    var $exec_select_command = "show";          // 検索開始コマンド名
12    var $table_name = "otenki";                 // テーブル名
13    var $sort_column = "day";                   // 日付でソート
14    var $aliases = array("day"=>"日付","tenki"=>"天気",
15                         "ondo"=>"温度","uryou"=>"雨量");
16  }
17
18  session_start();     // セッションの開始
19
20  if (!isset($_SESSION["sel"]) || !isset($_GET["mode"])) {
21    $_SESSION["sel"] = new myPgSelect;          // myPgSelectオブジェクト作成
22    $_SESSION["meta"] = new myPgMetaData;       // myPgmetaDataオブジェクト作成
23    $mode = "search";                           // 初期表示モード設定
24  } else {
25    $mode = (isset($_GET["mode"])?$_GET["mode"]:"show");
26  }
27
28  print <<<EOF
29  <html>
30  <head><title>Example 5</title></head>
31  <body>
32  EOF;
33
34  if ($mode == "search" ) {
35    $_SESSION["meta"]->printForm(); //検索フォームの表示
36  } else if ($mode == "show" ||
37      $_SESSION["sel"]->IsPageChangeRequested()) {
38    $_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL()); // 検索結果の表示
39    print <<<EOF
40    <br><a href="{$_SERVER['PHP_SELF']}?mode=search">検索フォームに戻る</a>
41  EOF;
42  }
43
44  ?>
45  </body>
46  </html>
```

例によって、PgMetaDataを継承することによって必要なパラメータを設定していきます（9～16行目）。テーブル名は$table_name、ソート対象の列名は$sort_columnに設定します。

$sort_columnの設定はオプションで、何も設定しなければ検索結果はソートされません。逆に、"uryou, ondo"のように複数列をカンマで区切って指定することもできます。この場合、まず雨量でソートし、次に同じ雨量の中では温度でソートします。ソート順はデフォルトで昇順（小さなものから大きいものへと並べる）ですが、逆（降順）にしたい場合はDESCというキーワードを追加します。いくつか例を示します。

uryou, ondo DESC	雨量、温度で降順にソート
uryou DESC, ondo	雨量で降順にソート、次に温度で昇順にソート

$aliasesは、検索フォームや検索結果リストで表示する見出し文字列を指定します。ご覧のように、テーブルの列名とペアで文字列を指定し、array関数で連想配列を作ってセットしておきます。$aliasesがセットされていると列名の代わりにここでセットした文字列が使われ、**図2-7**および**図2-8**のような表示になります。$aliasesがセットされていない場合は**図2-9**および**図2-10**のように列名がそのまま使われます。

●$mode変数

今までは単純にSQLを実行してその検索結果を表示するだけでしたが、今回は検索フォームの表示、検索結果の表示と異なる二つの処理を行ないます。これらを区別するために$_GET["mode"]を利用します。この変数の状態により、以下の処理を行ないます（**表2-4**）。

表2-4　$_GET["mode"]変数の値と処理

変数の値	処理
未定義	オブジェクトの作成 $mode変数の初期化
search	検索フォームの表示
show	検索結果の表示

まず$_GET["mode"]が未定義の場合は、myPgSelectオブジェクトとmyPgMetaDataオブジェクトを生成します。また、$mode変数を"search"に初期設定します（20～23行目）。

$_GET["mode"]がsearchの場合、検索条件入力用のフォームを表示します（35行目）。このためのメンバ関数がprintFormです。これについてはあとで述べます。

$_GET["mode"]がshowの場合は検索結果を表示します（38行目）。前回のサンプルと同様にdoSelectを使っていますが、今回はSQL文をmakeSQLというメンバ関数を使って生成しています。これについても後述します。

図2-8 $aliasesを設定した検索結果

図2-9 $aliasesを設定しない検索フォーム

図2-10 $aliasesを設定しない検索結果

3.6.4 PgMetaDataクラス

リスト2-9 PgMetaDataクラス（ex5/lib/pgmetadata.inc）

```
<?php
require_once("dbconnect.inc");

class PgMetaData {
  var $db;   // データベース接続ハンドル

  // カスタマイズ可能メンバ変数
  var $aliases = "";       // 列の別名連想配列([列名][別名])
  var $is_print_type_name = false;      // データ型名の表示有無
  var $is_print_opr_desc = false;       // オペレータの説明の表示有無
  var $table_name;         // テーブル名
  var $sort_column = "";  // ソートをする列名
  var $mode_var = "mode"; // 表示モード変数名
  var $exec_select_command = "show";   // 検索開始コマンド名

  // 内部変数(カスタマイズ不可)
  var $md;   // テーブルのメタデータ([列番号][メタデータ]の2次元配列)
  /* メタデータは以下：
     name: カラム名
     typename: データ型名
     len: 内部データ長(バイト数,可変長データは-1)
     modifier: 型修飾子
     notnull: NOT NULLなら t そうでないなら f
     defaultval: デフォルト値: なければf
     opinfo: オペレータ情報
     */

  var $op;   // オペレータ情報
  /* オペレータ情報は以下
     [型名][オペレータ情報番号][オペレータ名]
     [型名][オペレータ情報番号][オペレータに関する説明]
     */

  // コンストラクタ
  function PgMetaData() {
    $this->db = new DbConnect();
    $this->getMetaData();
    $this->getOperator();
  }

  // テーブルに関する情報の取得
  function getMetaData() {
    $sql = <<< EOF
```

```
44          SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
45          a.attnotnull, a.atthasdef
46          FROM pg_class c, pg_attribute a, pg_type t
47          WHERE c.relname = '$this->table_name'
48          AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
49          ORDER BY a.attnum
50  EOF;
51
52        $result = $this->db->doQuery($sql); // selectを実行
53        $rows = pg_num_rows($result);
54        for ($i=0;$i<$rows;$i++) {
55          $this->md[$i]["name"] = pg_fetch_result($result,$i,"attname");
56          $this->md[$i]["typename"] = pg_fetch_result($result,$i,"typname");
57          $this->md[$i]["len"] = pg_fetch_result($result,$i,"attlen");
58          $this->md[$i]["modifier"] = pg_fetch_result($result,$i,"atttypmod");
59          $this->md[$i]["notnull"] = pg_fetch_result($result,$i,"attnotnull");
60          //$this->md[$i]["defaultval"] = pg_fetch_result($result,$i,"atthasdef");
61        }
62      }
63
64      // 型に関するオペレータ情報の取得
65      function getOperator() {
66        $n = count($this->md);
67        for ($j=0;$j<$n;$j++) {
68          $typename = $this->md[$j]["typename"];
69          if (!isset($this->op["typename"])) {
70            $sql = <<< EOF
71            SELECT o.oprname AS op,t1.typname AS left_arg,
72            t2.typname AS right_arg, t0.typname AS result,
73            obj_description(p.oid) as description
74            FROM pg_proc p, pg_type t0, pg_type t1, pg_type t2,pg_operator o
75            WHERE t1.typname = '$typename'
76            AND p.prorettype = t0.oid AND RegprocToOid(o.oprcode) = p.oid
77            AND p.pronargs = 2 AND o.oprleft = t1.oid AND o.oprright = t2.oid
78            AND t0.typname = 'bool' AND t1.typname = t2.typname
79  EOF;
80            $result = $this->db->doQuery($sql); // selectを実行
81            $rows = pg_num_rows($result);
82            for ($i=0;$i<$rows;$i++) {
83              $this->op[$typename][$i]["name"] = pg_fetch_result($result,$i,"op");
84              $this->op[$typename][$i]["desc"] = pg_fetch_result($result,$i,"description");
85            }
86            pg_free_result($result);
87          }
88        }
89      }
90
```

```
 91    // <form action=...><tabl> の表示
 92    function printFormHeader() {
 93      $str = <<< EOF
 94        <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->
exec_select_command}" method="post">
 95        <table>
 96  EOF;
 97      print($str);
 98    }
 99
100    // </table><input type=submit...></form> の表示
101    function printFormFooter() {
102      $str = <<< EOF
103        </table>
104        <input type="submit" value="検索開始">
105        <input type="reset" value="クリア">
106        </form>
107  EOF;
108      print($str);
109    }
110
111    // 列名の表示
112    function printAttrName($atrnumber, $atrname) {
113      if ($this->is_print_type_name == false) {
114        print("<td>$atrname</td>");
115      } else {
116        $typename = $this->md[$atrnumber]["typename"];
117        print("<td>$atrname($typename)</td>");
118      }
119    }
120
121    // オペレータの表示
122    // 引数: カラム番号
123    function printOprName($n) {
124      $name = $this->md[$n]["name"];
125      $typename = $this->md[$n]["typename"];
126      print("<td><select name=¥"oplist[$name]¥">¥n");
127      printf("<option selected value=¥"%s¥">%s¥n",
128             $this->op[$typename][0]["name"],
129             htmlspecialchars($this->op[$typename][0]["name"]));
130
131      $n = count($this->op[$typename]);
132
133      for ($i=0;$i<$n;$i++) {
134        if ($this->is_print_opr_desc == true) {
135          printf("<option value=¥"%s¥">%s(%s)¥n",
136                 $this->op[$typename][$i]["name"],
```

```
137                 htmlspecialchars($this->op[$typename][$i]["name"]),
138                 htmlspecialchars($this->op[$typename][$i]["desc"]));
139       } else {
140         printf("<option value=¥"%s¥">%s¥n",
141                 $this->op[$typename][$i]["name"],
142                 htmlspecialchars($this->op[$typename][$i]["name"]));
143       }
144     }
145     print("</select></td>¥n");
146   }
147
148   // 検索フォームの生成
149   function printForm() {
150     if (is_array($this->md) == false) {
151       return;
152     }
153
154     // <form action=...><tabl> の表示
155     $this->printFormHeader();
156
157     $n = count($this->md);
158     for ($i=0;$i<$n;$i++) {
159
160       print("<tr>");
161
162       $name = $this->md[$i]["name"];
163       if (is_array($this->aliases)) {
164         $atrname = $this->aliases[$name];
165       } else {
166         $atrname = $name;
167       }
168       $this->printAttrName($i, $atrname);
169       $this->printOprName($i);
170
171       print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
172       print("</tr>¥n");
173     }
174     $this->printFormFooter();
175   }
176
177   // SQL文の生成
178   function makeSQL() {
179     if (!isset($_POST["atrlist"])) {
180       return;
181     }
182
183     $sql = "SELECT ";
```

```
184       $where = "WHERE";
185
186       $atrlist = $_POST["atrlist"];
187       $oplist = $_POST["oplist"];
188
189       $first = true;
190       $needComma = false;
191       foreach ($atrlist as $name => $val) {
192         if ($needComma == true) {
193           $sql .= ",";
194         }
195
196         $sql .= "$name ";
197         if (is_array($this->aliases)) {
198           $alias = $this->aliases[$name];
199           $sql .= "AS $alias";
200         }
201         if ($val != "") {
202           $op = $oplist[$name];
203           if ($first == false) {
204             $where .= " AND";
205           }
206           $where .= " $name $op '" . addslashes($val) . "'";
207           $first = false;
208         }
209         $needComma = true;
210       }
211
212       $sql .= " FROM $this->table_name ";
213
214       if ($where != "WHERE") {
215         $sql .= " ";
216         $sql .= $where;
217       }
218
219       if ($this->sort_column != "") {
220         $sql .= " ORDER BY ";
221         $sql .= $this->sort_column;
222       }
223       return($sql);
224     }
225   }
226   ?>
```

コンストラクタ（35～39行目）

コンストラクタはPostgreSQLのシステムテーブルにアクセスしてそのテーブルの列や、関連するオペレータに関する情報を取得します。実際の処理は下請け関数のgetMetaDataとgetOperatorが行ないます。まずgetMetaDataから見ていきましょう。

●getMetaData（42～62行目）

getMetaDataでは、PostgreSQLのシステムテーブルpg_class、pg_attribute、pg_typeから必要な情報を取り出します。PostgreSQL内部の構造に関することであり、本書の範囲を超えるのでSELECT文そのものの解説は省略しますが、pg_classがテーブル名などのテーブル本体の情報、pg_attributeが列に関する情報、pg_typeがデータ型に関する情報をそれぞれ管理していることだけを覚えておいてください。

ここで得られた情報は**表2-5**のもので、項目名をキーとする連想配列$mdに格納されます。このデータは列数分だけあるので、実際には**[列番号][項目名]**の二次元配列になっています。たとえば、列番号2のデータ型名は**$this->md[2]["typename"]**で取得できます。PgMetaDataクラスで使っているデータ項目は、このうちnameとtypename だけです。

表2-5　システムテーブルから取得できるテーブルの内部情報

項目名	意味
name	列名
typename	データ型名
len	内部データ長（単位バイト、可変長データは-1）
modifier	型修飾子
notnull	NULLを許さなければt、そうでなければf
defaultval	デフォルト値。なければf

●getOperator（64～89行目）

getOperatorは型に関するオペレータ情報をシステムテーブルから得て$opという連想配列に格納します。$opは三次元の連想配列になっており、一次元目が型名、二次元目がオペレータ情報番号、三次元目がnameまたはdescです。

一般に、ある型は複数のオペレータを持ちます。たとえば、INTEGERなら**<**、**>**、**=**、**<>**などになります。「オペレータ情報番号」はこれらに振られた通番です。nameはオペレータ名、descはオペレータの説明です。

●printForm（149～175行目）

コンストラクタを呼んだあとはメモリ上にそのテーブルの列、オペレータなどに関する情報が構築できているので、これを使ってPgMetaDataクラスはさまざまなサービスを提供できます。

printFormはテーブルタグを使って見栄えのよい検索フォームを表示します。その際、表示のカスタマイズが可能なように、フォームを構成するパーツを下請け関数に担当させてい

ます。したがって、PgMetaDataを継承したクラスでこれらの関数をカスタマイズすることによってフォームの見栄えなどを変更できます。

表2-6　下請けメンバ関数の一覧とその役割

関数名	役割
printFormHeader	form開始タグとテーブル開始タグの表示
printFormFooter	テーブル終了タグとsubmitタグなどの表示
printAttrName	列名の表示
printOprName	オペレータの表示

では、printFormが生成する検索フォームを見ていきましょう。

● フォーム開始文（155行目）

<form action=... の部分です。ここはprintFormHeaderというメンバ関数になっているので、必要に応じて継承先で書き換えることにより表示形式などを変更できます。標準ではactionの指定が自分自身、かつ枠なしのテーブル開始タグを表示します。

以後、テーブルの各行に列名、オペレータ、データ入力欄を表示していきます。

● 列名（168行目）

列名はprintAttrNameにより表示します。最初の引数は列番号です。二番目の引数は列名ですが、別名が設定されている場合は別名になります。

● オペレータ（169行目）

オペレータはprintOprNameにより表示します。引数は列番号です。

オペレータは複数あり得ますから、<select>タグを使って選択リストを表示するようにしています。printfの書式がちょっとわかりにくいので、実際に生成されたhtmlを示します。

```
<td><select name="oplist[day]">
<option selected value="=">=
<option value="=">=
<option value="<>">&lt;&gt;
<option value="<">&lt;
<option value="<=">&lt;=
<option value=">">&gt;
<option value=">=">&gt;=
</select></td>
```

<select>タグで指定するnameはoplist[day]としています。これにより、起動された側のPHPスクリプトでは$oplist[day]としてオペレータに指定された値を受け取ることができます。選択リストに表示するオペレータ名には < や > などのhtmlの特殊記号が含まれるため、htmlspecialcharsにより変換を行なっています。

● **データ入力欄（171行目）**

これは特にカスタマイズの必要性を感じなかったので、メンバ関数を設けず直接printFormの中で実行しています。ユーザが入力した検索値は**$atrlist[列名]**として起動されたPHPスクリプト側で受け取るようになっています。

● **フォーム終了文（174行目）**

printFormFooterで表示します。これをカスタマイズすれば、「検索開始」の文字を変更したり、検索開始のボタンをグラフィックスを使って表示できるようになるでしょう。

● **makeSQL（178～224行目）**

検索フォームに検索値を入力して検索開始を選択すると、自分自身（ex5/ex5.php：**リスト2-8**）がもう一度起動されます。このとき$_POST["artlist"]が設定されていることを利用し、最初のページを表示する
処理を実行します。

サンプルプログラム4（ex4/ex4.php：**リスト2-7**）でも使ったdoSelectを使って検索を行ないますが、その前にSELECT文を作らなければなりません。それを実行するのがmakeSQLです。必要な情報は検索フォームから渡されます。それが$_POST["atrlist"]、$_POST["oplist"]です。

生成されるSELECT文は、以下のようになっていなければなりません。

```
SELECT 列名1 [AS 別名1], 列名2 [AS 別名2]...列名n [AS 別名n] FROM テーブル名
[WHERE 列名a オペレータa '検索値a'[ AND 列名b オペレータb '検索値b']...]
[ORDER BY 列名]
```

列名1～列名nの部分を選択項目またはターゲットリストと呼びます。ターゲットリストを構成する列名は$atrlistから取得できます。別名が設定されている場合は、**AS 別名**を付加します。検索フォームに何か検索値が入力された場合は、WHERE以降（WHERE句）を付加します。検索値が複数項目指定された場合はANDでつないでいきます。検索値がまったく指定されない場合はWHEREも含め、WHERE以降がないことに注意してください。

以上を念頭に置いて、makeSQLを見ていきましょう。少々複雑に見えますが、文字列処理を行なっているだけです。

スクリプトでは、**SELECT＋ターゲットリスト**の部分（$sql）とWHERE句（$WHERE）を別々に作り、あとで合成します。

191行目からのforeachループをまわるたびに列1個が処理されます。列名は$name、検索値は$valに設定されています。

ターゲットリストに列名を追加したあと（196行目）、別名が設定されていれば**AS 別名**を付加します（197～200行目）。検索値が設定されていれば、オペレータリスト（$oplist）からオペレータを取り出し、WHERE句を作成します（202～209行目）。

最後にターゲットリストに**FROM テーブル名**を追加し（212行目）、必要ならばWHERE句を添付します（214～217行目）。このとき、ユーザが入力した検索データをそのまま使わ

ず、addslashes関数を適用しているのがポイントです。addslashesは' などの特殊文字に¥を追加し、¥' とします。これによって不正なSQLが実行されるのを防ぐことができます。たとえば、天気の検索データとして、

```
晴れ';DELETE FROM otenki
```

のようなものを入力されたことを考えましょう。もしaddslashesを使わなければテーブルからすべてのデータが削除されてしまうことでしょう。

次に、ソート指定がある場合はORDER BY句を追加して（219～222行目）SELECT文のできあがりです。

3.7 データ入力フォーム

前節では、テーブル検索フォームを自動生成し、SQLを意識することなくデータ検索ができるようになりました。次はデータを追加入力できるようにしましょう。

3.7.1 データチェックはしっかり

データ入力において最も重要なことは、データベースに誤ったデータが入力されないようにすることです。そのためには、以下の方法があります。

① データベース自体が持つデータチェック機能を活用する
② PHPスクリプトにデータをチェックするロジックを組み込む

①は、**制約**（constraint）と呼ばれるデータベースの機能を使います。制約を使うことで、データの未入力（NULL）や、重複データ、さらに値の範囲チェックなどが可能になります。また、制約を使えばデータベース自体が誤データの入力を防止するので、アプリケーションにチェック洩れがあっても安全です。したがって、まず制約をしっかり設定することが重要です。

3.7.2 CREATE TABLEに制約を追加する

制約は、CREATE TABLE文で指定します。

制約で実現できるチェックには以下のものがあります。

① テーブル中の同じ列の中で、その列の値がユニークである

otenkiテーブルではday列が相当します。同じ日のデータが二つあっては困るからです。

実はこれはすでに

```
day date PRIMARY KEY,    -- 日付(主キー)
```

で実現されています。PRIMARY KEY宣言をしておくと、同時にデータの唯一性が保証されるようになります。PRIMARY KEYでない列の値をユニークにしたい場合は、UNIQUEを使います。たとえば、同じ温度の日がないと仮定する場合は（おかしな仮定ですが）ondo列を次のように定義します。

```
ondo INTEGER UNIQUE
```

② NOT NULL

NULLは、「値が未入力」「値が定まらない」状態を表します。デフォルトではどの列もNULLが許されていますが、日付のようにかならずデータが入力されていないと困る列にはNOT NULL制約を追加します。たとえば、雨量をかならず入力する場合には

```
uryou INTEGER NOT NULL
```

とします。なお、PRIMARY KEYはNOT NULL制約を含んでいるため、あらためてNOT NULLを書く必要はありません。

③ そのほかの制約

これ以外に、任意の論理値をとるSQL文を書いて制約とすることができます。たとえば、負の雨量があり得ないことは次のように表現できます。

```
uryou INTEGER CHECK(uryou >= 0)
```

複数の列にまたがる制約を指定することもできます。たとえば（これまたナンセンスな例で恐縮ですが）温度と雨量の合計が1000以下であることを指定するには

```
CONSTRAINT otenki_check CHECK((ondo + uryou) <= 1000)
```

を列定義のあとに追加します。ここで、CONSTRAINTは予約語で、otenki_checkは制約の名前です。カンマ（,）で区切って複数のCONSTRAINT文を書くこともできます。

3.7.3 デフォルト値

INSERT文で値が指定されない列に対して、適当な**デフォルト値**（初期値）を設定することができます。デフォルト値により、入力の手間が省けるだけでなく、誤りの防止につながります。

たとえば、天気のデフォルト値を「晴れ」にしたい場合には以下のようにします。

```
tenki TEXT DEFAULT '晴れ',
```

デフォルト値が適用されるのは、INSERT文においてその列に対する値が指定されていない場合です。たとえば、

```
INSERT INTO otenki(day,ondo,uryou) VALUES('2000/9/1',28,0);
```

とすれば、tenki列には値が指定されていないので、デフォルト値が適用されます。

全列にデフォルト値を適用するためには、以下のようにします。

```
INSERT INTO otenki DEFAULT VALUES;
```

ところで、SQLでは、値がセットされていない状態に対してNULLという特別な記号を割り当てます。たとえば、その日の天気、温度は計ったのに雨量だけを計り忘れたときは、雨量だけをNULLとすることで「計り忘れ」を表現できます。しかし、このNULLとINSERT文において「値が指定されていない」こととは別問題であることに注意してください。たとえば、

```
INSERT INTO otenki(day,tenki,ondo,uryou) VALUES('2000/9/1',NULL,28,0);
```

としたときには、tenki列にデフォルト値が入るのではなく、NULLがセットされます。

また、ある列に対して値の範囲を指定する制約とデフォルト値を併用する場合、その制約はNULLには適用されないということを含めておかないと、決してデフォルト値を入力することができなくなります。たとえば

```
CHECK(ondo > -50 AND ondo < 50)
```

のままではNULLもエラーになってしまうので、以下のようにします。

```
CHECK(ondo IS NULL OR (ondo > -50 AND ondo < 50))
```

3.7.4 otenkiテーブルでの制約などの定義について

otenkiテーブルの定義の制約をあらためて見てみましょう。

```
day DATE DEFAULT 'today' PRIMARY KEY
```

PRIMARY KEYで主キー制約を表します。デフォルト値はtoday、すなわち本日です。

```
tenki TEXT DEFAULT '晴れ',
```

制約はありませんが、デフォルト値に「晴れ」が設定されます。

```
ondo INTEGER DEFAULT 25 CHECK(ondo IS NULL OR ONDO > -50 AND ONDO <50)
```

温度は-50から50度のあいだでなければならないという制約が設定されています。また、デフォルト値は25度です。

```
uryou INTEGER DEFAULT 0 CHECK(uryou IS NULL OR URYOU >= 0)
```

雨量は0以上でなければならないという制約があります。デフォルト値は0です。

このほか、以下のようなテーブル制約があります。

```
CHECK(tenki IS NULL OR
  (tenki = '晴れ' AND uryou = 0 OR tenki = '曇' OR tenki = '雨'))
);
```

これは、天気が晴れなら雨量は0でなければならない。そうでなければ、天気は曇か雨でなければならないという制約を表します。

以上をまとめると、**表2-7**のようになります。

表2-7　今回設定した制約とデフォルト値

列	制約	デフォルト値
day	主キー	'today'（本日）
tenki	晴れ、曇、雨のどれか	'晴れ'
ondo	-50～50	25
uryou	0以上	0
テーブル制約	天気が晴れなら雨量は0でなければならない。 そうでなければ、天気は曇りか雨でなければならない	

3.7.5 データ入力フォームのイメージ

基本的には検索フォーム（**図2-7**）と同じように、各列に対して値を入力します。「登録」ボタンを押すとデータが追加されます。何もデータを入力せずに登録すると、その列の値はNULLかデフォルト値になります（デフォルト値が設定されている場合）。NULLのチェックボックスをオンにすると、NULLが入力されます。また、制約やデフォルト値を表示してユーザがデータを入力する際の助けにしています。

図2-11　データ入力フォームのイメージ

PgMetaDataの改良

制約やデフォルト値などの情報を取得するために、PgMetaDataクラスを強化しました。また、データ入力フォームを表示するメンバ関数も追加しています。**リスト2-11**に変更後のPgMetaDataを示します。

リスト2-11　変更後のPgMetaData（ex6/lib/pgmetadata.inc）

```
<?php
require_once("dbconnect.inc");

class PgMetaData {
  var $db;                                  // データベース接続ハンドル

  // カスタマイズ可能メンバ変数
  var $aliases = "";                        // 列の別名連想配列([列名][別名])
  var $is_print_type_name = false;          // データ型名の表示有無
  var $is_print_opr_desc = false;           // オペレータの説明の表示有無
  var $table_name;                          // テーブル名
  var $sort_column = "";                    // ソートをする列名
  var $mode_var = "mode";                   // 表示モード変数名
  var $exec_select_command = "show";        // 検索開始コマンド名
  var $exec_insert_command = "insertExec"; // 挿入コマンド名

  // 内部変数(カスタマイズ不可)
  var $md;  // テーブルのメタデータ([列番号][メタデータ]の二次元配列)
  /* メタデータは以下：
     name: カラム名
     typename: データ型名
     len: 内部データ長(バイト数,可変長データは-1)
     modifier: 型修飾子
     notnull: NOT NULLなら t そうでないなら f
     defaultval: デフォルト値: なければf
     opinfo: オペレータ情報
     rcsrc: 列制約
     */

  var $op;  // オペレータ情報
  /* オペレータ情報は以下
     [型名][オペレータ情報番号][オペレータ名]
     [型名][オペレータ情報番号][オペレータに関する説明]
     */

  var $table_oid;                           // テーブルの oid
```

```
37    var $table_constraints = "";              // テーブル制約
38
39    // コンストラクタ
40    function PgMetaData() {
41      $this->db = new DbConnect();
42      $this->getMetaData();    // テーブルに関する情報の取得
43      $this->getOperator();    // オペレータに関する情報の取得
44      $this->getConstraint();  // 制約に関する情報の取得
45      $this->getDefault();     // デフォルト値に関する情報の取得
46    }
47
48    // テーブルに関する情報の取得
49    function getMetaData() {
50      $sql = <<< EOF
51        SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
52        a.attnotnull, a.atthasdef, c.oid
53        FROM pg_class c, pg_attribute a, pg_type t
54        WHERE c.relname = '$this->table_name'
55        AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
56        ORDER BY a.attnum
57  EOF;
58
59      $result = $this->db->doQuery($sql);  // selectを実行
60      $this->table_oid = pg_fetch_result($result,0,"oid");
61      $rows = pg_num_rows($result);
62      for ($i=0;$i<$rows;$i++) {
63        $this->md[$i]["name"] = pg_fetch_result($result,$i,"attname");
64        $this->md[$i]["typename"] = pg_fetch_result($result,$i,"typname");
65        $this->md[$i]["len"] = pg_fetch_result($result,$i,"attlen");
66        $this->md[$i]["modifier"] = pg_fetch_result($result,$i,"atttypmod");
67        $this->md[$i]["notnull"] = pg_fetch_result($result,$i,"attnotnull");
68        //$this->md[$i]["defaultval"] = pg_fetch_result($result,$i,"atthasdef");
69      }
70    }
71
72    // 型に関するオペレータ情報の取得
73    function getOperator() {
74      $n = count($this->md);
75      for ($j=0;$j<$n;$j++) {
76        $typename = $this->md[$j]["typename"];
77        if (!isset($this->op["typename"])) {
78          $sql = <<< EOF
```

```
 79            SELECT o.oprname AS op,t1.typname AS left_arg,
 80            t2.typname AS right_arg, t0.typname AS result,
 81            obj_description(p.oid) as description
 82            FROM pg_proc p, pg_type t0, pg_type t1, pg_type t2,pg_operator o
 83            WHERE t1.typname = '$typename'
 84            AND p.prorettype = t0.oid AND RegprocToOid(o.oprcode) = p.oid
 85            AND p.pronargs = 2 AND o.oprleft = t1.oid AND o.oprright = t2.oid
 86            AND t0.typname = 'bool' AND t1.typname = t2.typname
 87  EOF;
 88          $result = $this->db->doQuery($sql);  // selectを実行
 89          $rows = pg_num_rows($result);
 90          for ($i=0;$i<$rows;$i++) {
 91            $this->op[$typename][$i]["name"] = pg_fetch_result($result,$i,"op");
 92            $this->op[$typename][$i]["desc"] = pg_fetch_result($result,$i,"description");
 93          }
 94          pg_free_result($result);
 95        }
 96      }
 97    }
 98
 99    // 制約に関する情報の取得
100    function getConstraint() {
101      $sql = <<< EOF
102        SELECT rcname,rcsrc FROM pg_relcheck WHERE rcrelid = '$this->table_oid'
103  EOF;
104      $result = $this->db->doQuery($sql); // selectを実行
105      $rows = pg_num_rows($result);
106
107      // 制約無し
108      if ($rows == 0) {
109        pg_free_result($result);
110        return;
111      }
112
113      // 連想配列に制約名と制約値を格納
114      for ($i=0;$i<$rows;$i++) {
115        $constraints[$i]["rcname"] = pg_fetch_result($result,$i,"rcname"); // 制約名
116        $constraints[$i]["rcsrc"] = pg_fetch_result($result,$i,"rcsrc");   // 制約
117      }
118      pg_free_result($result);
119
120      for ($i=0;$i<count($this->md);$i++) {
```

```
121         // 列制約の名前は「テーブル名_列名」
122         $colname = $this->table_name . "_" . $this->md[$i]["name"];
123         for ($j=0;$j<count($constraints);$j++) {
124           if ($constraints[$j]["rcname"] == $colname) {
125             this->md[$i]["constraint"] = $constraints[$j]["rcsrc"];
126             $constraints[$j]["rcname"] = "";  // チェック済の制約名を無効にする
127             break;
128           }
129         }
130       }
131
132       // 列制約をすべて無効にし、残ったのがテーブル制約
133       for ($j=0;$j<count($constraints);$j++) {
134         $rcname = $constraints[$j]["rcname"];
135         if ($rcname != "") {
136           $this->table_constraints[$rcname] = $constraints[$j]["rcsrc"];
137         }
138       }
139     }
140
141     // デフォルト値の取得
142     function getDefault() {
143       $sql = <<< EOF
144         SELECT adsrc, adnum FROM pg_attrdef WHERE adrelid = $this->table_oid
145   EOF;
146       $result = pg_query($sql);
147       if ($result == false) {
148         print("テーブル $this->table_name のデフォルト値が取れません。");
149         return false;
150       }
151       $rows = pg_num_rows($result);
152
153       // デフォルト値なし
154       if ($rows == 0) {
155         pg_free_result($result);
156         return;
157       }
158
159       for ($i=0;$i<$rows;$i++) {
160         // adnumは列番号+1
161         $adnum = pg_fetch_result($result,$i,"adnum");
162         $this->md[$adnum-1]["defaultval"] = pg_fetch_result($result,$i,"adsrc");
```

```
163       }
164       pg_free_result($result);
165     }
166
167     // <form action=...><tabl> の表示
168     function printFormHeader() {
169       $str = <<< EOF
170         <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->exec_select_command}" method="post">
171         <table>
172   EOF;
173       print($str);
174     }
175
176     // </table><input type=submit...></form> の表示
177     function printFormFooter() {
178       $str = <<< EOF
179         </table>
180         <input type="submit" value="検索開始">
181         <input type="reset" value="クリア">
182         </form>
183   EOF;
184       print($str);
185     }
186
187     // 列名の表示
188     function printAttrName($atrnumber, $atrname) {
189       if ($this->is_print_type_name == false) {
190         print("<td>$atrname</td>");
191       } else {
192         $typename = $this->md[$atrnumber]["typename"];
193         print("<td>$atrname($typename)</td>");
194       }
195     }
196
197     // オペレータの表示
198     // 引数: カラム番号
199     function printOprName($n) {
200       $name = $this->md[$n]["name"];
201       $typename = $this->md[$n]["typename"];
202       print("<td><select name=¥"oplist[$name]I">¥n");
203       printf("<option selected value=¥"%s¥">%s¥n",
204       $this->op[$typename][0]["name"],
```

```
205       htmlspecialchars($this->op[$typename][0]["name"]));
206
207       $n = count($this->op[$typename]);
208
209       for ($i=0;$i<$n;$i++) {
210         if ($this->is_print_opr_desc == true) {
211           printf("<option value=¥"%s¥">%s(%s)¥n",
212           $this->op[$typename][$i]["name"],
213           htmlspecialchars($this->op[$typename][$i]["name"]),
214           htmlspecialchars($this->op[$typename][$i]["desc"]));
215         } else {
216           printf("<option value=¥"%s¥">%s¥n",
217           $this->op[$typename][$i]["name"],
218           htmlspecialchars($this->op[$typename][$i]["name"]));
219         }
220       }
221       print("</select></td>¥n");
222     }
223
224     // 検索フォームの生成
225     function printForm() {
226       if (is_array($this->md) == false) {
227         return;
228       }
229
230       // <form action=...><tabl> の表示
231       $this->printFormHeader();
232
233       $n = count($this->md);
234       for ($i=0;$i<$n;$i++) {
235
236         print("<tr>");
237
238         $name = $this->md[$i]["name"];
239         if (is_array($this->aliases)) {
240           $atrname = $this->aliases[$name];
241         } else {
242           $atrname = $name;
243         }
244         $this->printAttrName($i, $atrname);
245         $this->printOprName($i);
246
```

```
247         print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
248         print("</tr>¥n");
249       }
250       $this->printFormFooter();
251     }
252
253     // SQL文の生成
254     function makeSQL() {
255       if (!isset($_POST["atrlist"])) {
256         return;
257       }
258
259       $sql = "SELECT ";
260       $where = "WHERE";
261
262       $atrlist = $_POST["atrlist"];
263       $oplist = $_POST["oplist"];
264
265       $first = true;
266       $needComma = false;
267       foreach ($atrlist as $name => $val) {
268         if ($needComma == true) {
269           $sql .= ",";
270         }
271
272         $sql .= "$name ";
273         if (is_array($this->aliases)) {
274           $alias = $this->aliases[$name];
275           $sql .= "AS $alias";
276         }
277         if ($val != "") {
278           $op = $oplist[$name];
279           if ($first == false) {
280             $where .= " AND";
281           }
282           $where .= " $name $op '" . addslashes($val) . "'";
283           $first = false;
284         }
285         $needComma = true;
286       }
287
288       $sql .= " FROM $this->table_name ";
```

```
289
290       if ($where != "WHERE") {
291         $sql .= " ";
292         $sql .= $where;
293       }
294
295       if ($this->sort_column != "") {
296         $sql .= " ORDER BY ";
297         $sql .= $this->sort_column;
298       }
299       return($sql);
300     }
301
302     // ---------------- データ登録用メンバ関数 ----------------------------
303     // <form action=...><tabl> の表示
304     function printDataInputFormHeader() {
305       print<<<EOF
306       <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->exec_insert_command}" method="post">
307       <table border>
308       <tr><th>カラム</th><th>データ型</th><th>データ</th>
309       <th>NULL</th><th>初期値</th><th>制約</th>
310   EOF;
311     }
312
313     // </table><input type=submit...></form> の表示
314     function printDataInputFormFooter() {
315       print("</table>¥n");
316
317       if (is_array($this->table_constraints)) {
318         print("<table border>¥n");
319         print("<tr><th>テーブル制約名</th><th>制約</th></tr>¥n");
320
321         foreach ($this->table_constraints as $name => $val) {
322           print("<tr><tr><td>$name</td><td>$val</td></tr>¥n");
323         }
324         print("</table>¥n");
325       }
326       print("<input type=¥"submit¥" value=¥"登録¥">¥n");
327       print("<input type=¥"reset¥" value=¥"クリア¥">¥n");
328       print("</form>¥n");
329     }
330
```

```
331    // データ入力フォームの生成
332    function printDataInputForm() {
333      if (is_array($this->md) == false) {
334        return;
335      }
336
337      // <form action=...><tabl> の表示
338      $this->printDataInputFormHeader();
339
340      $n = count($this->md);
341      for ($i=0;$i<$n;$i++) {
342
343        print("<tr>");
344
345        $name = $this->md[$i]["name"];
346        if (is_array($this->aliases)) {
347          $atrname = $this->aliases[$name] . "(" . $name . ")";
348        } else {
349          $atrname = $name;
350        }
351        print("<td>$atrname</td>¥n");
352
353        $typename = $this->md[$i]["typename"];
354        print("<td align=center>$typename</td>");
355
356        print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
357
358        if ($this->md[$i]["notnull"] == "t") {
359          print("<td align=center>不可</td>¥n");
360        } else {
361          print("<td align=center><input type=¥"checkbox¥" name=¥"nulllist[$name]¥" ");
362          print("></td>¥n");
363        }
364
365        if (isset($this->md[$i]["defaultval"])) {
366          $def = $this->md[$i]["defaultval"];
367          print("<td align=center>$def</td>¥n");
368        } else {
369          print("<td align=center>なし</td>¥n");
370        }
371
372        if (isset($this->md[$i]["constraint"])) {
```

```
373           $constraint = $this->md[$i]["constraint"];
374         } else {
375           $constraint = "なし";
376         }
377         print("<td align=center>$constraint</td>¥n");
378
379         print("</tr>¥n");
380       }
381       $this->printDataInputFormFooter();
382     }
383
384     // ---------------- SQL文生成関数 -----------------------------
385     // データ登録SQLの作成、実行
386     function insertSQL($user_check = false, $debug = false) {
387       $sql = "INSERT INTO $this->table_name (";
388       $atrs = "";
389       $vals = "";
390       $needComma = false;
391       $has_someval = false;
392
393       foreach ($_POST["atrlist"] as $name => $val) {
394         if (!isset($_POST["nulllist"][$name]) && $val == "") {
395           continue;  // 何も値が指定されていない
396         }
397
398         $has_someval = true;
399
400         if ($needComma == true) {
401           $atrs .= ",";
402           $vals .= ",";
403         }
404         $needComma = true;
405
406         $atrs .= "$name ";
407
408         if (isset($_POST["nulllist"][$name])) {
409           $vals .= "NULL";  // NULLが指定された
410         } else if ($val != "") {
411           $vals .= "'" . addslashes($val) . "'";  // 何か値が入力された
412         }
413       }
414
```

```
415      if ($user_check) {       // データチェックユーザ関数あり?
416        if ($user_check($_POST["atrlist"]) == false) {
417          return(false);
418        }
419      }
420
421      // 何も値が指定されていない場合はデフォルト値を入力する
422      if ($has_someval == false) {
423        $sql = "INSERT INTO $this->table_name DEFAULT VALUES";
424      } else {
425        $sql .= "$atrs) values ($vals)";
426      }
427
428      if ($debug == true) {
429        print($sql);
430      }
431      $rtn = $this->db->doQuery($sql);
432      return($rtn);
433    }
434  }
435  ?>
```

■コンストラクタの変更

制約を取得するメンバ関数getConstraintやデフォルト値を取得するメンバ関数getDefaultがコンストラクタから呼び出されるようになっています(44～45行目)。まずここから見ていきましょう。

●getConstraint(100～139行目)

制約はシステムテーブルpg_relcheckから以下のSQL文で取得できます。

```
SELECT rcname,rcsrc FROM pg_relcheck WHERE rcrelid = テーブルのOID;
```

ここで、rcnameは制約名、rcsrcは制約の内容です。

制約名の決め方には内部的な約束があり、**列制約**(uryou check(uryou >=0)のように、列定義のときに指定するもの)には**テーブル名_列名**という名前がシステムによって自動的に割り当てられます。これに対して**テーブル制約**にはユーザが指定した名前がつけられます。ここでは、まず列制約を取り出したあと(120～130行目)、もし残ったものがあればテーブル制約とみなしています(133～138行目)。

列制約は、テーブルのメタデータ$mdにconstraintという連想配列名で格納されます。テーブル制約のほうは$table_constraintsという配列に入れておきます。

● getDefault（142～165行目）

ある列に対応するデフォルト値はpg_attrdefから以下のSQL文で取得できます。

```
SELECT adsrc FROM pg_attrdef WHERE adrelid = テーブルOID;
```

ここで、列番号は1から始まることに注意してください。デフォルト値はテーブルのメタデータ$mdに、defaultvalという連想配列名で格納されます（162行目）。

データ入力フォーム用メンバ関数の追加

以下のメンバ関数が追加されています。

● printDataInputFormHeader（304～311行目）

フォームに表示する項目が増えたため、検索用のフォームヘッダ表示printFormHeaderとは別関数にしました。また、デザイン上テーブルを枠つきにしています。

● printDataInputFormFooter（314～329行目）

テーブル制約を表示する必要上、検索用のprintFormFooterに変更を加えて別メンバ関数にしています。また、検索フォームのときは「検索」ボタンが表示されていましたが、これを「登録」ボタンに変えました。

● printDataInputForm（332～382行目）

入力フォームを表示する本体です。PgSelectクラスの検索フォーム表示（printForm）とよく似たロジックになっています。

まずprintDataInputFormHeaderでフォーム開始文を表示します。次に、各列に対応した列名、データ入力エリアを表示します。printFormと違ってオペレータを表示する必要がないため、printOprNameは呼び出しません。

各列の処理では、以下の項目を表示します。

表2-8　printDataInputFormの表示項目とデータ取得先

表示項目	データの取得先
列別名	aliases[列番号]
列名	md[列番号][name]
データ型	md[列番号][typename]
データ入力エリア	
NOT NULLの可／不可	md[列番号][notnull]
デフォルト値	md[列番号][defaultval]
制約	md[列番号][constraint]

データ入力SQLの実行

データ入力フォームで入力したデータは「登録」ボタンを選択したときに起動されるスクリプトで処理されます。この時insertSQLを呼び出せば、ほとんどの処理がこの中で自動的に行なわれます。

● insertSQL（386～433行目）

insertSQLの引数は二つあり、最初の引数は入力データの整合性チェックのためのユーザ関数です。データベースの制約機能で対応できないものは、このようにユーザ関数を作成することでチェックすることができます。ユーザ関数には、列名とそれに対応するフォームの入力値が格納されている連想配列が引数として渡ります。二番目の引数はデバッグフラグで、これがtrueの場合にはSQL文を表示します。

これ以外にinsertSQLに渡る情報としては、フォームからの入力があります。ひとつは$_POST["atrlist"]（393行目）で、前節のサンプル同様連想配列になっており、**$_POST["atrlist"][列名]**のかたちで入力されたデータを得ることができます。もうひとつは$_POST["nulllist"]です（394行目）。これは**$nulllist[列名]**がセットされているならばNULLのチェックボックスが押されたことを意味します。

insertSQLの最初の仕事は、フォームから入力されたデータからSQL文（INSERT）を作ることにあります。

ここで作成するINSERT文は次の形式です。

```
INSERT INTO テーブル名(列名1, 列名2,...) VALUES(値1, 値2,...)
```

入力フォームで値が入力されなかった列は列名リストから外されます。ただし、まったく値が入力されない場合はこのかたちのSQL文は使えないので、

```
INSERT INTO テーブル名 DEFAULT VALUES
```

という特別な形式のSQL文を作るようにしています（423行目）。

insertSQLでは、SELECT文を作成するスクリプト同様foreachループ（393～419行目）をまわるたびに列1個が処理されます。列名は$name、検索値は$valに設定されています。列名リストは$arts、値のリストは$valsにSQL文が完成したら、doQueryを使ってINSERT文を実行します。doQueryの戻り値がそのままinsertSQLの戻り値になります。戻りがfalseなら、なんらかの理由でINSERTが失敗したことになります。

サンプルプログラム6

改良版のPgMetaDataクラスを使って検索とデータ入力を実現したサンプルプログラム6を**リスト2-12**に示します。

今回のサンプルプログラムは、検索機能とデータ入力機能を持っているので、最初に機能を選ぶためのメニュー画面を設けることにしました（40～43行目）。本書の例でいえば、

```
http://localhost/~foo/examples/ex6/ex6.php
```

というURLで表示される画面になります（**図2-12**）。

図2-12 検索/データ入力選択メニュー

リスト2-12 サンプルプログラム6（ex6/ex6.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3  require_once("pgmetadata.inc");
 4
 5  class myPgSelect extends PgSelect {
 6    var $mode_var = "mode";  // 表示モード変数名
 7    var $next = "next";      // 次ページ
 8    var $prev = "prev";      // 前ページ
 9  }
10
11  class myPgMetaData extends PgMetadata {
12    var $mode_var = "mode";  // 表示モード変数名
13    var $exec_select_command = "show";    // 検索開始コマンド名
14    var $exec_insert_command = "insertExec";    // 挿入コマンド名
15    var $table_name = "otenki";    // テーブル名
16    var $sort_column = "day";      // 日付でソート
17    var $aliases = array("day"=>"日付","tenki"=>"天気",
18   "ondo"=>"温度","uryou"=>"雨量");
19  }
20
21  session_start();    // セッションの開始
22
23  if (!isset($_SESSION["sel"]) || !isset($_GET["mode"])) {   // はじめての表示?
24    $_SESSION["sel"] = new myPgSelect;    // myPgSelectオブジェクト作成
25    $_SESSION["meta"] = new myPgMetaData;// myPgmetaDataオブジェクト作成
26    $mode = "top";           // 初期表示モード設定
27  } else {
28    $mode = (isset($_GET["mode"])?$_GET["mode"]:"top");
29  }
30
31  print <<<EOF
```

```
<html>
<head><title>Example 6</title></head>
<body>
EOF;

switch($mode) {
 case "top":
   print <<<EOF
    <ul>
    <li><a href="{$_SERVER["PHP_SELF"]}?mode=search">検索フォーム</a>
    <li><a href="{$_SERVER["PHP_SELF"]}?mode=insert">登録フォーム</a>
    </ul>
EOF;
   break;
 case "search":
   $_SESSION["meta"]->printForm();       //検索フォームの表示
   break;
 case "show":
   $_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL());  // 検索結果の表示
   break;
 case "next":
 case "prev":
   $_SESSION["sel"]->doSelect();  // 検索結果の前/次表示
   break;
 case "insert":
   $_SESSION["meta"]->printDataInputForm();    //登録フォームの表示
   break;
 case "insertExec":
   $sts = $_SESSION["meta"]->insertSQL();              //登録の実行
   if ($sts != false) {
     print("正常にデータを登録しました。<br>¥n");
   }
   break;
}

if ($mode != "top") {
  print <<< EOF
<br><a href="{$_SERVER["PHP_SELF"]}?mode=search">検索フォームに戻る</a>
<br><a href="{$_SERVER["PHP_SELF"]}?mode=insert">登録フォームに戻る</a>
<br><a href="{$_SERVER["PHP_SELF"]}?mode=top">トップメニューに戻る</a>
EOF;
}
?>
</body>
</html>
```

ご覧のようにサンプルなので非常に質素なものですが、必要ならばもっと見栄えを整えればよいでしょう。このメニュー画面は、$mode変数が未設定、もしくはtopという値を持つときに表示されるようになっています。$mode変数はこれ以外にもいろいろな値を持ちます。表にまとめたので、37～65行目と見比べて使い方を理解してください（**表2-9**）。

表2-9　$mode変数の値と使い方

$mode変数の値	動作
未設定またはtop	トップレベルメニュー
search	検索フォームの表示
show	検索結果表示
next	次ページ表示
prev	前ページ表示
insert	登録フォームの表示
insertExec	登録の実行

3.8 データ修正/削除

データの検索と入力ができるようになったので、次はデータの修正と削除です。基本的なアイデアとしては、検索画面を使って修正したいデータを表示し、修正したいデータを選択することによってデータ更新/削除画面に移動できるようにします。データ更新/削除画面は、データ入力画面と似ており、前節で作成したスクリプトをかなり流用できます。

更新対象行の特定方法

ここでのポイントは、選択された行をテーブルの中から**唯一の**ものとして特定することです。データの修正にあたっては、どの行が更新/削除の対象であるのか曖昧であってはいけないからです。行を厳密に特定する方法としては、関係データベースでは通常主キーを用います。しかし、どのテーブルにも常に主キーが設定されているとはかぎらないため、この方法が適用できない場合があります。そこでここでは**TID**を使うことにします。TIDとは「タプルID」のことで、行の物理的な位置を表し、PostgreSQL固有のものです。TIDは**ctid**という特別な列名でアクセスできます。

```
foo=> SELECT ctid FROM otenki LIMIT 1;
 ctid
-------
 (0,2)
(1 row)
```

ここで、(0,2)の"0"はその行が所属する**ブロック番号**を表します。ブロックはPostgreSQLがディスクIO（入出力）を行なう単位で、通常8192バイトの固定長の領域です。一方 "2" はそのブロックの中における行の位置番号です。

とりあえずこういった細かなことは忘れていただいても結構ですが、とにかくあるテーブルの中ではどの行にもユニークなTIDが割り当てられていることだけは覚えておいてください。

TIDを使用することの問題点は、TIDが不変ではないということです。PostgreSQLでは、行を更新すると古い行を不可視の状態にし、更新された新しい行データをテーブルに追加します。つまり更新後の行は物理的な位置が変るので、TIDも変ってしまいます。また、行を更新しない場合でも、VACUUMという特別なコマンドを実行して不可視になっている古い行を削除し、行を前詰めにしてテーブルをコンパクトにする操作がPostgreSQLでは行なわれます。この場合もTIDが変化します。つまり、更新対象の行を表示しておいてから、ブラウザ上でデータを変更して更新操作を行なうまでの間にTIDが変化するかもしれないのです。最悪の場合は、違う行を更新してしまうかもしれません。

そこで本スクリプトでは、検索時の

①更新対象の行のTID
②更新対象行の内容 *6

を覚えておき、更新削除時に同じデータを取り出して以前と変っていないことを確認してから更新処理に入るようにしています。また、データの確認処理を行なう前に行にロックをかけ、データの確認処理と更新処理の間のわずかな時間のあいだにデータを変更されてしまわないように排他制御を行ないます。

ただし以上の方法でも防ぎきれないケースがあります。それは、まったく同じ内容の行が複数ある場合です。この場合、上に挙げた方法では行が更新されたことを検知できない可能性がありますが、そもそもまったく同じ内容の行が複数あること自体が関係データベースとして異常なことなので、無視することにします。

PostgreSQLには、**OID**（オブジェクトID）というシステムが列があり、これはTIDと違って更新やVACUUMでも変化しないのでOIDを使うことも考えられるのですが、PostgreSQL 7.2以降ではかならずしもすべてのテーブルにOIDが存在するとはかぎらないので、本書では使用しません。

*6
実際には、②で行内容をそのまま覚えておくわけではなく、MD5ハッシュ関数を使ってハッシュ値だけを記憶するようにしています。これによって大きな行データを記憶する必要がないのでメモリを節約できます。

更新可能な検索結果とは？

次に注意することとしては、検索結果として表示されているデータが、かならずしもすべて編集可能とはかぎらないことです。たとえば

```
SELECT 1;
```

のような検索結果に対応するテーブルの実体が存在しないため、修正変更できません。

また

```
SELECT COUNT(*),tenki FROM otenki GROUP BY tenki;
```

は、天気によって行を分類し、それぞれのグループの出現頻度を検索するものですが、検索結果には演算の結果（COUNT(*)）が使われるため、結果の行に対応するOIDというものが定義されません。したがってやはり修正の対象にはなりません。このように、検索結果に対応するデータの実体が存在するかどうかを自動的に判定するのはかなりむずかしいため、利用者に明示的にそのことを指定してもらうことにしましょう。

3.8.1 更新可能であることを明示的に指定

PgSelectクラスには、検索を実行するdoSelectという関数があり、引数として検索用のSQL文を渡していました。その方法には二通りあって、

```
$sel->doSelect("SELECT day AS 年月日, tenki AS 天気, ondo AS 温
度, uryou AS 雨量 FROM otenki");
```

のように直接SELECT文を指定する方法と、

```
$sel->doSelect($m->makeSQL("day"));
```

のようにPgMetaDataクラスのmakeSQL関数を利用してSELECT文を生成して渡す方法がありました。このどちらの方法でも使えるようにしたいので、今回は次のように考えます。

doSelect("SELECT day as 年月日, ...) のように直接SQL文を指定する場合

SQL文の中で、ターゲットリスト（SELECTのあとにくる部分）の中に次の項目を含めてもらいます。

```
ctid AS __target_tid__
```

たとえば、前述の例は、

```
$sel->doSelect("SELECT ctid AS __target_tid__, day AS 年月日,
tenki AS 天気, ondo AS 温度, uryou AS 雨量 FROM otenki");
```

となります。

この__target_tid__という別名が現れた場合、この列は検索結果には表示させず、更新時のキーとして内部的に使用します。[*7]

*7 このように__target_tid__という別名は特殊な意味を持たせているので、アプリケーション側でこの別名を使ってはいけません。

makeSQLを使用している場合

PgMetaDataクラスのmakeSQLに引数を追加し、ctid AS __target_tid__を生成する指示が可能になるようにします。関数仕様は以下のようになります。

```
function makeSQL($updatable = false)
     $updatable    trueならばctid as __target_tid__を追加する
     $updatable …… 省略可能な引数
```

makeSQLでは、引数$updatableがtrueのときにSELECTのターゲットリストにctid AS __ target_tid__が加わるようにします。

3.8.2 PgSelectクラスの修正

リスト2-13を見ながらPgSelectクラスの修正箇所を確認していきましょう。

メンバ関数doSelectを修正し、SELECT文を実行した結果返ってきた列に__target_tid__が含まれていたら(136行目)列名を表示せず、その代わりに何番目の列が__target_tid__だったかを記憶しておきます(137行目)。

そして、表示行の最後に「更新/削除」と書かれたアンカーを表示し、クリックされたら更新/削除用のフォームが表示されるようにします(155行目)。実際にこのアンカー部分を表示するのは新しく追加したメンバ関数 printUpdateTagです(92～99行)。中身は非常に簡単で、

```
function printUpdateTag($tid) {
   print<<<EOF
     <td>
       <a  href="{$_SERVER["PHP_SELF"]}?{$this->mode_var}=
{$this->update}&tid=$tid">[更新]</a>
       <a  href="{$_SERVER["PHP_SELF"]}?{$this->mode_var}=
{$this->delete}&tid=$tid">[削除]</a>
     </td>
EOF;
}
```

のように定義されています。この関数はユーザにオーバライドさせることを想定していますが、デフォルトでは「更新」「削除」という文字のところにアンカーが打たれ、そこを選択することによって自分自身($_SERVER["PHP_SELF"])を起動するようになっています。そのときに、mode変数にupdateをセットすることにします。また、更新すべき行が特定できるように、tidも渡すようにしておきます。

図2-13 更新機能を持つ検索結果

以上で**図2-13**のように、ex7/ex7.phpによって更新可能な検索結果が表示されるようになります。なお、この段階でのex6.php（**リスト2-12**）とex7.php（**リスト2-14**）の違いは、ex6.phpの50行目で、

```
$_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL());
// 検索結果の表示
```

だったのが、ex7.phpの60行目では

```
$_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL(true));
                                                    // 検索結果の表示
```

になっているだけです。

リスト2-13 PgSelectクラス（ex7/lib/pgselect.inc）

```
 1  <?php
 2  /*
 3   * 検索結果をテーブル形式で表示する
 4   */
 5  require_once("dbconnect.inc");
 6
 7  class PgSelect {
 8    var $db;   // データベース接続ハンドル
 9
10    // カスタマイズ可能メンバ変数
11
12    var $mode_var = "mode"; // 表示モード変数名
13    var $next = "next";     // 次ページ
14    var $prev = "prev";     // 前ページ
15    var $update = "update"; // 更新要求
```

```
16    var $delete = "delete"; // 削除要求
17    var $max1 = 5;    // ページに一度に表示する行数
18
19    // 内部変数(カスタマイズ不可)
20    var $usersql = "";      // ユーザ入力SELECT文
21    var $offset = 0; // 表示オフセット
22
23    // コンストラクタ
24    function PgSelect() {
25      $this->db = new DbConnect();
26    }
27
28    // DBへの接続
29    function doConnect() {
30      $this->db = new DbConnect();
31    }
32
33    // DBへの接続を閉じる
34    function doClose() {
35      $this->db->doClose();
36    }
37
38    // テーブル開始タグの印字
39    function printTableHeader() {
40      print("<table border>¥n");
41    }
42
43    // 列名の印字
44    function printHeader($i, $str) {
45      print("<th>$str</th>");
46    }
47
48    // データの印字
49    function printData($i, $str) {
50      print("<td>$str</td>");
51    }
52
53    // 検索結果の「前」「次」表示要求?
54    function IsPageChangeRequested() {
55      return(isset($_GET[$this->mode_var]) &&
56           ($_GET[$this->mode_var] == $this->next ||
57            $_GET[$this->mode_var] == $this->prev));
58    }
59
60    // 表示モード変数の返却
61    function modeVar() {
62      return($_GET[$this->mode_var]);
```

```
 63     }
 64
 65     // 検索結果の「前」表示要求?
 66     function IsPagePrevRequested() {
 67       return($this->IsPageChangeRequested() &&
 68            $this->modeVar() == $this->prev);
 69     }
 70
 71     // 検索結果の「次」表示要求?
 72     function IsPageNextRequested() {
 73       return($this->IsPageChangeRequested() &&
 74            $this->modeVar() == $this->next);
 75     }
 76
 77     // 「前」の表示
 78     function printPrev() {
 79       print <<<EOF
 80   <a href="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->prev}">[前の
$this->maxl 件に戻る]</a>
 81   EOF;
 82     }
 83
 84     // 「次」の表示
 85     function printNext($n) {
 86       print <<<EOF
 87   <a href="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->next}">[次の
$n 件に続く]</a>
 88   EOF;
 89     }
 90
 91     // 更新/削除タグの表示
 92     function printUpdateTag($tid) {
 93       print<<<EOF
 94         <td>
 95         <a href="{$_SERVER["PHP_SELF"]}?{$this->mode_var}={$this->update}
&tid=$tid">[更新]</a>
 96         <a href="{$_SERVER["PHP_SELF"]}?{$this->mode_var}={$this->delete}
&tid=$tid">[削除]</a>
 97         </td>
 98   EOF;
 99     }
100
101     // 検索の実行
102     function doSelect($sql = "") {
103       $tid = false;
104
105       if (!$this->IsPageChangeRequested()) {    // はじめての表示？
```

```
106         if ($sql == "") {
107         print("doSelect: SQL文が指定されていません");
108         exit;
109         }
110         $this->usersql = $sql;
111         $this->offset = 0;
112       } else {
113         if ($this->IsPageNextRequested()) {
114           $this->offset += $this->maxl;
115         } else {
116           $this->offset -= $this->maxl;
117         }
118       }
119
120       // limit - offset 句の添付
121       $sql = $this->usersql . " LIMIT $this->maxl OFFSET $this->offset";
122
123       $result = $this->db->doQuery($sql);            // selectを実行
124       $rows = pg_num_rows($result);                  // 行数を取得
125       $columns = pg_num_fields($result);             // 列数を取得
126
127       $this->printTableHeader();                     // テーブル開始タグの表示
128
129       $is_updatable = -1;
130
131       for ($j = 0;$j < $rows;$j++) {
132         if ($j == 0) {
133           print("<tr>");
134           for ($i = 0;$i < $columns;$i++) {
135             $str = pg_field_name($result,$i);        // 列名の取り出し
136             if ($str == "__target_tid__") {          // 更新キー？
137               $is_updatable = $i;
138             } else {
139               $this->printHeader($i, $str);          // 列名の表示
140             }
141           }
142           print("</tr>¥n");
143         }
144
145         print("<tr>");
146         for ($i = 0;$i < $columns;$i++) {
147           $str = pg_fetch_result($result,$j,$i);     // データの取り出し
148           if ($is_updatable == $i) {
149             $tid = $str;
150           } else {
151             $this->printData($i, addslashes($str)); // データの表示
152           }
```

```
153         }
154         if ($tid != false) {
155           $this->printUpdateTag($tid);
156         }
157
158         print("</tr>¥n");
159       }
160       pg_free_result($result);            // 検索結果の解放
161
162       print("</table>¥n");
163
164       if ($this->offset > 0) {
165         $this->printPrev();
166       }
167
168       $offset = $this->offset + $this->maxl;
169       $sql = $this->usersql . " LIMIT $this->maxl OFFSET $offset";
170
171       $result = $this->db->doQuery($sql);// 次のページに表示するデータがあるか検索する
172       $n = pg_num_rows($result);
173
174       if ($n > 0) {   // 次のページのデータあり？
175         $this->printNext($n);
176       }
177     }
178   }
179   ?>
```

リスト2-14　サンプルプログラム7（ex7/ex7.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3  require_once("pgmetadata.inc");
 4
 5  class myPgSelect extends PgSelect {
 6    var $mode_var = "mode";  // 表示モード変数名
 7    var $next = "next";      // 次ページ
 8    var $prev = "prev";      // 前ページ
 9  }
10
11  class myPgMetaData extends PgMetadata {
12    var $mode_var = "mode";  // 表示モード変数名
13    var $exec_select_command = "show";    // 検索開始コマンド名
14    var $insert_command = "insert"; // 挿入リクエストコマンド名
15    var $update_command = "update"; // 更新リクエストコマンド名
```

```
16    var $delete_command = "delete";            // 削除リクエストコマンド名
17    var $exec_insert_command = "insertExec";   // 挿入コマンド名
18    var $exec_update_command = "updateExec";   // 更新コマンド名
19    var $exec_delete_command = "deleteExec";   // 削除コマンド名
20    var $table_name = "otenki";                // テーブル名
21    var $sort_column = "day";                  // 日付でソート
22    var $aliases = array("day"=>"日付","tenki"=>"天気",
23                         "ondo"=>"温度","uryou"=>"雨量");
24  }
25
26  session_start();  // セッションの開始
27
28  if (!isset($_SESSION["sel"]) || !isset($_GET["mode"])) {  // はじめての表示?
29    $_SESSION["sel"] = new myPgSelect;    // myPgSelectオブジェクト作成
30    $_SESSION["meta"] = new myPgMetaData; // myPgmetaDataオブジェクト作成
31    $mode = "top";  // 初期表示モード設定
32  } else {
33    $mode = (isset($_GET["mode"])?$_GET["mode"]:"top");
34  }
35
36  print<<<EOF
37  <html>
38  <head>
39  <title>Example 7</title>
40  EOF;
41  include_once("jscripts.inc");
42  print<<<EOF
43  </head>
44  <body>
45  EOF;
46
47  switch($mode) {
48   case "top":
49     print <<<EOF
50      <ul>
51      <li><a href="{$_SERVER["PHP_SELF"]}?mode=search">検索フォーム</a>
52      <li><a href="{$_SERVER["PHP_SELF"]}?mode=insert">登録フォーム</a>
53      </ul>
54  EOF;
55     break;
56   case "search":
57     $_SESSION["meta"]->printForm();                    //検索フォームの表示
58     break;
59   case "show":
60     $_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL(true));
                                                        // 検索結果の表示
61     break;
```

```
62    case "next":
63    case "prev":
64      $_SESSION["sel"]->doSelect();                    // 検索結果の前/次表示
65      break;
66    case "insert":
67    case "update":
68    case "delete":
69      $_SESSION["meta"]->printDataInputForm($mode);   //登録フォームの表示
70      break;
71    case "insertExec":
72      if ($_SESSION["meta"]->insertSQL()) {           //登録の実行
73        print("登録処理が正常に終了しました。<br>¥n");
74      }
75      break;
76    case "updateExec":                                // 更新の実行
77      if ($_SESSION["meta"]->updateSQL()) {           // 更新処理
78        print("更新処理が正常に終了しました。<br>¥n");
79      }
80      break;
81    case "deleteExec":                                // 削除の実行
82      if ($_SESSION["meta"]->deleteSQL()) {           // 削除処理
83        print("削除処理が正常に終了しました。<br>¥n");
84      }
85      break;
86  }
87
88  if ($mode != "top") {
89    print<<<EOF
90  <br><a href="{$_SERVER["PHP_SELF"]}?mode=search">検索フォームに戻る</a>
91  <br><a href="{$_SERVER["PHP_SELF"]}?mode=insert">登録フォームに戻る</a>
92  <br><a href="{$_SERVER["PHP_SELF"]}?mode=top">トップメニューに戻る</a>
93  EOF;
94  }
95  ?>
96  </body>
97  </html>
```

3.8.3 更新/削除フォーム

次に**図2-13**で更新/削除を選択したときに表示される更新/削除用のフォーム画面を作りましょう（**図2-14**）。基本的にはデータ入力フォーム（**図2-11**）とほぼ同じですが、以下のような違いがあります。

・「登録」ボタンではなく、「更新」または「削除」ボタンを表示する
・データ入力エリアにあらかじめ現在のデータを表示しておき、ユーザの便宜をはかる

データ入力フォームは、PgMetaDataクラスのprintDataInputFormというメンバ関数で表示していましたが、これをコピーして改造するか、なんらかの方法で更新/削除用のフォームの処理機能も持たせるかは判断に迷うところですが、今回は後者の方法でいくことにしましょう。

図2-14 今回作成する更新フォーム

PgMetaDataクラスのprintDataInputFormの改造

それでは**リスト2-15**（ex7/lib/pgmetadata.inc）を見ながら改造点を確認していきましょう。

リスト2-15 更新/削除機能を追加したpgmetadata.inc（ex7/lib/pgmetadata.inc）

```
<?php
require_once("dbconnect.inc");

class PgMetaData {
  var $db;   // データベース接続ハンドル

  // カスタマイズ可能メンバ変数
  var $aliases = "";       // 列の別名連想配列([列名][別名])
```

```
 9    var $is_print_type_name = false;      // データ型名の表示有無
10    var $is_print_opr_desc = false;       // オペレータの説明の表示有無
11    var $table_name;          // テーブル名
12    var $sort_column = "";  // ソートをする列名
13    var $mode_var = "mode"; // 表示モード変数名
14    var $exec_select_command = "show";    // 検索開始コマンド名
15    var $insert_command = "insert";       // 挿入リクエストコマンド名
16    var $exec_insert_command = "insertExec";    // 挿入コマンド名
17    var $update_command = "update";       // 更新リクエストコマンド名
18    var $exec_update_command = "updateExec";    // 更新コマンド名
19    var $delete_command = "delete";       // 削除リクエストコマンド名
20    var $exec_delete_command = "deleteExec";    // 削除コマンド名
21
22    // 内部変数(カスタマイズ不可)
23    var $md;   // テーブルのメタデータ([列番号][メタデータ]の二次元配列)
24    /* メタデータは以下：
25       name: カラム名
26       typename: データ型名
27       len: 内部データ長(バイト数,可変長データは-1)
28       modifier: 型修飾子
29       notnull: NOT NULLなら t そうでないなら f
30       defaultval: デフォルト値: なければf
31       opinfo: オペレータ情報
32       rcsrc: 列制約
33       */
34
35    var $op;   // オペレータ情報
36    /* オペレータ情報は以下
37       [型名][オペレータ情報番号][オペレータ名]
38       [型名][オペレータ情報番号][オペレータに関する説明]
39       */
40
41    var $table_oid;          // テーブルの Oid
42    var $table_constraints = "";   // テーブル制約
43    var $tid;  // 現在行のTID
44    var $hash; // 現在行値のMD5 hash(verify用)
45
46    // コンストラクタ
47    function PgMetaData() {
48      $this->db = new DbConnect();
49      $this->getMetaData(); // テーブルに関する情報の取得
50      $this->getOperator(); // オペレータに関する情報の取得
51      $this->getConstraint();      // 制約に関する情報の取得
52      $this->getDefault();  // デフォルト値に関する情報の取得
53    }
54
55    // テーブルに関する情報の取得
```

```
  56    function getMetaData() {
  57      $sql = <<< EOF
  58        SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
  59        a.attnotnull, a.atthasdef, c.oid
  60        FROM pg_class c, pg_attribute a, pg_type t
  61        WHERE c.relname = '$this->table_name'
  62        AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
  63        ORDER BY a.attnum
  64  EOF;
  65
  66      $result = $this->db->doQuery($sql);       // selectを実行
  67      $this->table_oid = pg_fetch_result($result,0,"oid");
  68      $rows = pg_num_rows($result);
  69      for ($i=0;$i<$rows;$i++) {
  70        $this->md[$i]["name"] = pg_fetch_result($result,$i,"attname");
  71        $this->md[$i]["typename"] = pg_fetch_result($result,$i,"typname");
  72        $this->md[$i]["len"] = pg_fetch_result($result,$i,"attlen");
  73        $this->md[$i]["modifier"] = pg_fetch_result($result,$i,"atttypmod");
  74        $this->md[$i]["notnull"] = pg_fetch_result($result,$i,"attnotnull");
  75        //$this->md[$i]["defaultval"] = pg_fetch_result($result,$i,"atthasdef");
  76      }
  77    }
  78
  79    // 型に関するオペレータ情報の取得
  80    function getOperator() {
  81      $n = count($this->md);
  82      for ($j=0;$j<$n;$j++) {
  83        $typename = $this->md[$j]["typename"];
  84        if (!isset($this->op["typename"])) {
  85        $sql = <<< EOF
  86          SELECT o.oprname AS op,t1.typname AS left_arg,
  87          t2.typname AS right_arg, t0.typname AS result,
  88          obj_description(p.oid) as description
  89          FROM pg_proc p, pg_type t0, pg_type t1, pg_type t2,pg_operator o
  90          WHERE t1.typname = '$typename'
  91          AND p.prorettype = t0.oid AND RegprocToOid(o.oprcode) = p.oid
  92          AND p.pronargs = 2 AND o.oprleft = t1.oid AND o.oprright = t2.oid
  93          AND t0.typname = 'bool' AND t1.typname = t2.typname
  94  EOF;
  95        $result = $this->db->doQuery($sql);     // selectを実行
  96        $rows = pg_num_rows($result);
  97        for ($i=0;$i<$rows;$i++) {
  98          $this->op[$typename][$i]["name"] = pg_fetch_result($result,$i,"op");
  99          $this->op[$typename][$i]["desc"] = pg_fetch_result($result,$i,"description");
 100        }
 101        pg_free_result($result);
 102        }
```

```
103       }
104     }
105
106     // 制約に関する情報の取得
107     function getConstraint() {
108       $sql = <<< EOF
109         SELECT rcname,rcsrc FROM pg_relcheck WHERE rcrelid = '$this->table_oid'
110   EOF;
111       $result = $this->db->doQuery($sql);       // selectを実行
112       $rows = pg_num_rows($result);
113
114       // 制約無し
115       if ($rows == 0) {
116         pg_free_result($result);
117         return;
118       }
119
120       // 連想配列に制約名と制約値を格納
121       for ($i=0;$i<$rows;$i++) {
122         $constraints[$i]["rcname"] = pg_fetch_result($result,$i,"rcname");   // 制約名
123         $constraints[$i]["rcsrc"] = pg_fetch_result($result,$i,"rcsrc");     // 制約
124       }
125       pg_free_result($result);
126
127       for ($i=0;$i<count($this->md);$i++) {
128         // 列制約の名前は「テーブル名_列名」
129         $colname = $this->table_name . "_" . $this->md[$i]["name"];
130         for ($j=0;$j<count($constraints);$j++) {
131         if ($constraints[$j]["rcname"] == $colname) {
132           $this->md[$i]["constraint"] = $constraints[$j]["rcsrc"];
133           $constraints[$j]["rcname"] = "";      // チェック済の制約名を無効にする
134           break;
135         }
136         }
137       }
138
139       // 列制約をすべて無効にし、残ったのがテーブル制約
140       for ($j=0;$j<count($constraints);$j++) {
141         $rcname = $constraints[$j]["rcname"];
142         if ($rcname != "") {
143           $this->table_constraints[$rcname] = $constraints[$j]["rcsrc"];
144         }
145       }
146     }
147
148     // デフォルト値の取得
149     function getDefault() {
```

```
150       $sql = <<< EOF
151         SELECT adsrc, adnum FROM pg_attrdef WHERE adrelid = $this->table_oid
152   EOF;
153       $result = $this->db->doQuery($sql);
154       if ($result == false) {
155         print("テーブル $this->table_name のデフォルト値が取れません。");
156         return false;
157       }
158       $rows = pg_num_rows($result);
159
160       // デフォルト値無し
161       if ($rows == 0) {
162         pg_free_result($result);
163         return;
164       }
165
166       for ($i=0;$i<$rows;$i++) {
167         // adnumは列番号+1
168         $adnum = pg_fetch_result($result,$i,"adnum");
169         $this->md[$adnum-1]["defaultval"] = pg_fetch_result($result,$i,"adsrc");
170       }
171       pg_free_result($result);
172     }
173
174     // <form action=...><tabl> の表示
175     function printFormHeader() {
176       print<<< EOF
177         <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->
exec_select_command}" method="post">
178         <table>
179   EOF;
180     }
181
182     // </table><input type=submit...></form> の表示
183     function printFormFooter() {
184       print<<< EOF
185         </table>
186         <input type="submit" value="検索開始">
187         <input type="reset" value="クリア">
188         </form>
189   EOF;
190     }
191
192     // 列名の表示
193     function printAttrName($atrnumber, $atrname) {
194       if ($this->is_print_type_name == false) {
195         print("<td>$atrname</td>");
```

```
196       } else {
197         $typename = $this->md[$atrnumber]["typename"];
198         print("<td>$atrname($typename)</td>");
199       }
200     }
201
202     // オペレータの表示
203     // 引数: 列番号
204     function printOprName($n) {
205       $name = $this->md[$n]["name"];
206       $typename = $this->md[$n]["typename"];
207       print("<td><select name=¥"oplist[$name]¥">¥n");
208       printf("<option selected value=¥"%s¥">%s¥n",
209            $this->op[$typename][0]["name"],
210            htmlspecialchars($this->op[$typename][0]["name"]));
211
212       $n = count($this->op[$typename]);
213
214       for ($i=0;$i<$n;$i++) {
215         if ($this->is_print_opr_desc == true) {
216         printf("<option value=¥"%s¥">%s(%s)¥n",
217                $this->op[$typename][$i]["name"],
218                htmlspecialchars($this->op[$typename][$i]["name"]),
219                htmlspecialchars($this->op[$typename][$i]["desc"]));
220         } else {
221         printf("<option value=¥"%s¥">%s¥n",
222                $this->op[$typename][$i]["name"],
223                htmlspecialchars($this->op[$typename][$i]["name"]));
224         }
225       }
226       print("</select></td>¥n");
227     }
228
229     // 検索フォームの生成
230     function printForm() {
231       if (is_array($this->md) == false) {
232         return;
233       }
234
235       // <form action=...><tabl> の表示
236       $this->printFormHeader();
237
238       $n = count($this->md);
239       for ($i=0;$i<$n;$i++) {
240
241         print("<tr>");
242
```

```
243         $name = $this->md[$i]["name"];
244         if (is_array($this->aliases)) {
245          $atrname = $this->aliases[$name];
246         } else {
247          $atrname = $name;
248         }
249         $this->printAttrName($i, $atrname);
250         $this->printOprName($i);
251
252         print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
253         print("</tr>¥n");
254       }
255       $this->printFormFooter();
256     }
257
258
259     // --------------- データ登録用メンバ関数 ----------------------------
260
261     // 現在の行を取り出す
262     function getRow($lock = false) {
263       $sql = "SELECT * FROM $this->table_name WHERE ctid = '$this->tid'";
264       if ($lock) {
265         $sql .= " FOR UPDATE";
266       }
267       $result = $this->db->doQuery($sql);
268       if ($result == false || pg_num_rows($result) != 1) {
269         return false;
270       }
271       return $result;
272     }
273
274     function getHash($result) {
275       // 行の値のMD5ハッシュ値を得る
276       $hash = "";
277       $row = pg_fetch_array($result, 0, PGSQL_ASSOC);
278       foreach ($row as $val) {
279         $hash .= $val;
280       }
281       return(md5($hash));
282     }
283
284     // <form action=...><tabl> の表示
285     // データ入力フォームの生成
286     function printDataInputFormHeader($mode) {
287       switch ($mode) {
288       case $this->insert_command:
289         $op= $this->exec_insert_command;
```

```
290         break;
291       case $this->update_command:
292         $op= $this->exec_update_command;
293         break;
294       case $this->delete_command:
295         $op= $this->exec_delete_command;
296         break;
297       }
298
299       print <<<EOF
300         <form action="{$_SERVER["PHP_SELF"]}?{$this->mode_var}=$op&tid=
{$this->tid}" method="post">
301         <table border>
302         <tr><th>カラム</th><th>データ型</th><th>データ</th>
303         <th>NULL</th><th>初期値</th><th>制約</th></tr>
304   EOF;
305     }
306
307     // </table><input type=submit...></form> の表示
308
309     function printDataInputFormFooter($mode) {
310       print("</table>¥n");
311
312       if (is_array($this->table_constraints)) {
313         print("<table border>¥n");
314         print("<tr><th>テーブル制約名</th><th>制約</th></tr>¥n");
315
316         foreach ($this->table_constraints as $name => $val) {
317           print("<tr><tr><td>$name</td><td>$val</td></tr>¥n");
318         }
319       }
320       print("</table>¥n");
321
322       switch ($mode) {
323       case $this->insert_command:
324         print<<<EOF
325         <input type="submit" value="登録">
326   EOF;
327         break;
328       case $this->update_command:
329         print<<<EOF
330         <input type="submit" name="{$this->update_command}" value="更新"
onClick="return formConfirm('update')">
331   EOF;
332         break;
333       case $this->delete_command:
334         print<<<EOF
```

```
335         <input type="submit" name="{$this->delete_command}" value="削除"
onClick="return formConfirm('delete')">
336   EOF;
337       }
338       print<<<EOF
339         <input type="reset" value="クリア">
340         </form>
341   EOF;
342     }
343
344     // データ入力フォームの生成
345     function printDataInputForm($mode) {
346       if (is_array($this->md) == false) {
347         return;
348       }
349
350       // <form action=...><tabl> の表示
351       $this->printDataInputFormHeader($mode);
352
353       // 更新フォームのときはtidを記憶し，現在のテーブルの値を取得する
354       if ($mode != $this->insert_command) {
355         if (!isset($_REQUEST["tid"])) {
356           print("更新対象行を特定するデータがありません．");
357           exit;
358         }
359         // 処理対象の行を記憶
360         $this->tid = addslashes($_REQUEST["tid"]);
361
362         // 行の値のMD5ハッシュ値を記憶
363         $result = $this->getRow();
364         if ($result == false) {
365           print("現在行を取得できません．");
366           return false;
367         }
368         $this->hash = $this->getHash($result);
369       }
370
371       $n = count($this->md);
372
373       for ($i=0;$i<$n;$i++) {
374         $name = $this->md[$i]["name"];
375         if (is_array($this->aliases)) {
376           $atrname = $this->aliases[$name] . "(" . $name . ")";
377         } else {
378           $atrname = $name;
379         }
380
```

```
381         // 挿入フォーム以外のときは現在のテーブルの値を取得する
382         if ($mode != $this->insert_command) {
383           $str = htmlspecialchars(pg_fetch_result($result,0,$this->md[$i]["name"]));
384         } else {
385           $str = "";
386         }
387
388         if ($this->md[$i]["notnull"] == "t") {
389           $nullset = "不可";
390         } else {
391           $nullset =<<<EOF
392           <input type="checkbox" name="nulllist[$name]"
393   EOF;
394         }
395
396         if (isset($this->md[$i]["defaultval"])) {
397           $def = $this->md[$i]["defaultval"];
398         } else {
399           $def = "なし";
400         }
401
402         if (isset($this->md[$i]["constraint"])) {
403           $constraint = $this->md[$i]["constraint"];
404         } else {
405           $constraint = "なし";
406         }
407
408         print<<<EOF
409         <tr>
410         <td>$atrname</td>
411         <td align=center>{$this->md[$i]["typename"]}</td>
412         <td><input type="text" value="$str" name="atrlist[$name]"></td>
413         <td align=center>$nullset</td>
414         <td align=center>$def</td>
415         <td align=center>$constraint</td>
416         </tr>
417   EOF;
418       }
419       $this->printDataInputFormFooter($mode);
420     }
421
422     // --------------- SQL文生成関数 ----------------------------
423     // 検索SQL文の生成
424     function makeSQL($updatable = false) {
425       if (!isset($_POST["atrlist"])) {
426         return;
427       }
```

```
428
429       $sql = "SELECT ";
430       $where = "WHERE";
431
432       $atrlist = $_POST["atrlist"];
433       $oplist = $_POST["oplist"];
434
435       $first = true;
436
437       if ($updatable == true) {
438         $sql .= "ctid AS __target_tid__ ";
439         $needComma = true;
440       } else {
441         $needComma = false;
442       }
443
444       foreach ($atrlist as $name => $val) {
445         if ($needComma == true) {
446           $sql .= ",";
447         }
448
449         $sql .= "$name ";
450         if (is_array($this->aliases)) {
451           $alias = $this->aliases[$name];
452           $sql .= "AS $alias";
453         }
454         if ($val != "") {
455           $op = $oplist[$name];
456           if ($first == false) {
457             $where .= " AND";
458           }
459           $where .= " $name $op '" . addslashes($val) . "'";
460           $first = false;
461         }
462         $needComma = true;
463       }
464
465       $sql .= " FROM $this->table_name ";
466
467       if ($where != "WHERE") {
468         $sql .= " ";
469         $sql .= $where;
470       }
471
472       if ($this->sort_column != "") {
473         $sql .= " ORDER BY ";
474         $sql .= $this->sort_column;
```

```
475       }
476       return($sql);
477     }
478
479     // データ登録SQLの作成、実行
480     function insertSQL($user_check = false, $debug = false) {
481       $sql = "INSERT INTO $this->table_name (";
482       $atrs = "";
483       $vals = "";
484       $needComma = false;
485       $has_someval = false;
486
487       foreach ($_POST["atrlist"] as $name => $val) {
488         if (!isset($_POST["nulllist"][$name]) && $val == "") {
489           continue;   // 何も値が指定されていない
490         }
491
492         $has_someval = true;
493
494         if ($needComma == true) {
495           $atrs .= ",";
496           $vals .= ",";
497         }
498         $needComma = true;
499
500         $atrs .= "$name ";
501
502         if (isset($_POST["nulllist"][$name])) {
503           $vals .= "NULL";  // NULLが指定された
504         } else if ($val != "") {
505           $vals .= "'" . addslashes($val) . "'"; // 何か値が入力された
506         }
507       }
508
509       if ($user_check) { // データチェックユーザ関数あり?
510         if ($user_check($_POST["atrlist"]) == false) {
511           return(false);
512         }
513       }
514
515       // 何も値が指定されていない場合はデフォルト値を入力する
516       if ($has_someval == false) {
517         $sql = "INSERT INTO $this->table_name DEFAULT VALUES";
518       } else {
519         $sql .= "$atrs) values ($vals)";
520       }
521
```

```
522       if ($debug == true) {
523         print($sql);
524       }
525       $rtn = $this->db->doQuery($sql);
526       return($rtn);
527     }
528
529     // データ変更SQLの作成、実行
530     function updateSQL($user_check = false) {
531       $this->db->doQuery("BEGIN"); // トランザクションの開始
532
533       $result = $this->getRow(TRUE);
534       if ($result == false ||
535         $this->getHash($result) != $this->hash) {
536         print("他のユーザによってデータが更新されています．検索からやり直してください．");
537         $this->db->doQuery("END"); // トランザクションの終了
538         return false;
539       }
540
541       $sql = "UPDATE $this->table_name SET ";
542       $atrs = "";
543       $vals = "";
544       $needComma = false;
545       $has_someval = false;
546
547       foreach ($_POST["atrlist"] as $name => $val) {
548         $has_someval = true;
549
550         if ($needComma == true) {
551           $sql .= ",";
552         }
553         $needComma = true;
554         if (isset($_POST["nulllist"][$name])) {
555           $sql .= "$name = null";
556         } else {
557           $sql .= "$name = '" . addslashes($val) . "'";
558         }
559       }
560
561       if ($has_someval == false) {
562         print("データが入力されていません。");
563         $this->db->doQuery("END"); // トランザクションの終了
564         return(false);
565       }
566
567       if ($user_check) {
568         if ($user_check($_POST["atrlist"]) == false) {
```

```
569         $this->db->doQuery("END");// トランザクションの終了
570         return(false);
571         }
572       }
573
574       $sql .= " WHERE ctid = '$this->tid'";
575       $rtn = $this->db->doQuery($sql);
576       $this->db->doQuery("END");   // トランザクションの終了
577       return $rtn;
578     }
579
580     // データ削除SQLの作成、実行
581     function deleteSQL() {
582       $this->db->doQuery("BEGIN");// トランザクションの開始
583
584       $result = $this->getRow(TRUE);
585       if ($result == false ||
586          $this->getHash($result) != $this->hash) {
587         print("他のユーザによってデータが更新されています．検索からやり直してください．");
588         $this->db->doQuery("END");// トランザクションの終了
589         return false;
590       }
591       $sql = "DELETE FROM $this->table_name WHERE ctid = '$this->tid' ";
592       $rtn = $this->db->doQuery($sql);
593       $this->db->doQuery("END");   // トランザクションの終了
594       return $rtn;
595     }
596   }
597   ?>
```

● 引数の追加

データ登録用のフォームなのか、それとも更新/削除用のフォームなのかを区別するために$modeという引数を設け、これが$this->insert_commandならばデータ登録、$this->update_commandならば更新、$this->delete_commandなら削除用のフォームを表示することにします（345行目）。

● 更新または削除の時は現在のTIDを記憶する（354～369行目）

$this->tidという変数を追加し、そこに現在のTIDを記憶します。また、TIDが指す行の値をgetRow()で取得し（363行目）、そのハッシュ値を$this->hashに入れておきます（368行目）。

● データ入力エリアに現在値を表示

データ入力エリアに現在の値を表示し、更新の際に使いやすくします（383行目）。

● submit ボタン

「登録」ボタン、「更新」ボタンあるいは「削除」ボタンを表示します（419行目）。実際に処理を行なうのはprintDataInputFormFooterです（309～342行目）。

JavaScriptによる確認ダイアログ表示

これでひととおり検索、データ登録、更新、削除の処理が可能になりましたが、このままだと、うっかり「更新」や「削除」のボタンを押したときに、いきなりデータベースを更新するのでいささか危険です。特に削除は間違えると取り返しがつかないので、確認のダイアログを出すことを考えましょう。

確認用のページをPHPで生成することもできますが、このような場合にはJavaScriptを使うのが最適です。PHPと違ってサーバ側ではなく、ブラウザ側で実行されます。そのため、ダイアログを出すたびにサーバにアクセスすることもなく、負荷がかかりません。逆に、PHPのようにデータベースにアクセスするようなことは不可能で、あくまでもブラウザ画面に密着した簡単な処理をするためのもの、と考えたほうがよいでしょう。

確認ダイアログをJavaScriptで作る

さっそくJavaScriptを使ってダイアログ表示機能を組み込んでみましょう。まず、submit文を以下のように変更します（**リスト2-15**：ex7/lib/pgmetadata.incの330行目と335行目）。

```
<input type="submit" name="{$this->update_command}" value="更
新" onClick="return formConfirm('update')">

<input type="submit" name="{$this->delete_command}" value="削
除" onClick="return formConfirm('delete')">
```

ここで、

```
onClick="return formConfirm('update')"
```

などが新たに追加された部分です。onClickはJavaScriptの予約語で、ここでは「更新」ボタンが押されたらreturn以降を実行する、という意味になります。formConfirmは今回作成したJavaScriptの関数で、文字列を引数にとってtrueかfalseを返します。formConfirm()がfalseを返すとsubmit処理は中断され、更新/削除処理は行なわれません。

では、formConfirmはどこで定義するのでしょうか。HTMLでページを記述する際には、

```
<html>
<head>
<title>タイトル名</title>
</head>
```

のような部分をつけることになっていますが、この<head>～</head>の部分で定義するのです。

Column

JavaScript

▶ JavaScriptとは

JavaScriptはNetscape NavigatorやInternet Explorerなどのブラウザで使用できるスクリプト言語で、PHPと同じようにHTMLに直接埋め込んで使用します。JavaScriptは名称こそJavaに似ていますが、まったく別の言語です。

JavaScriptはNetscape社とSun Microsystems社が共同で開発した言語で、バージョン1.0からはじまり、現在は1.5というバージョンが最新になっています。JavaScriptのバージョンによってはブラウザが対応していないこともあるので注意してください。

JavaScriptバージョン	対応ブラウザバージョン
JavaScript 1.1	Netscape Navigator 3.0, Internet Explorer 4.0 以降
JavaScript 1.2	Netscape Navigator 4.0, Internet Explorer 4.0 以降
JavaScript 1.3	Netscape Navigator 4.06, Internet Explorer 5.0 以降
JavaScript 1.4	Netscape Navigator 6.0
JavaScript 1.5	Netscape Navigator 6.0

Netscape NavigatorとInternet ExplorerではJavaScriptの動作が異なることがあります。また、日本語の動作や細かな点でうまく動かなかったりすることも多く、最新の機能や、はじめて使うJavaScriptの機能には十分注意する必要があります（本書で使う程度の基本的な機能ならば問題ありません）。本書では、JavaScript1.1[*8]を前提に説明します。

▶ JavaScriptでできること

前述したように、JavaScriptでできることはブラウザの中の世界に限られますが、制御構造も記述できますし、普通のHTMLではできないきめの細かい処理が可能です。たとえば次のようなことが可能です。

- ウィンドウの生成 / 消滅
- アニメーションの表示
- マウスポインタなどのイベントの取得とそれに対応するアクションの定義
- フレームの制御
- ダイアログの表示

▶ JavaScriptとPHPの関係

すでに述べたように、PHPがサーバ側で動作するスクリプトであるのに対し、JavaScriptはブラウザ側で動作します。したがって、PHPとJavaScriptを組み合わせて使う場合、次のようになります。

PHPがJavaScriptを含むHTMLを生成
　--> ブラウザが生成されたHTMLを表示、JavaScriptを実行

逆にJavaScriptからPHPを呼び出すようなことはできません。

*8
現在はJavaScriptをベースに「ECMAScript」（http://www.ecma.ch/）という標準仕様が策定されています。今後はECMAScriptをサポートするブラウザが増えてくるものと思われます。

JavaScriptの記述は一般に以下のようになります。

```
<script language="JavaScript1.1">
   [スクリプト本体]
</script>
```

ただし、このままだとJavaScriptに対応していないブラウザでは、[スクリプト本体]が見えてしまいます。そこで、HTMLのコメント記号を使って

```
<script language="JavaScript1.1">
<!--
   [スクリプト本体]
//-->
</script>
```

とします。なお、スクリプト本体を直接書かずに

```
<script src="スクリプトファイル名" language="JavaScript1.1"></script>
```

とすることもできますが、今回はあえてPHPのincludeを使って読み込むようにしてみました。たとえば

```
<html>
<head>
<title>Example 7</title>
<?php include("jscripts.ini");?>
</head>
<body>
```

のようになります。jscripts.iniの中身は以下のとおりです。

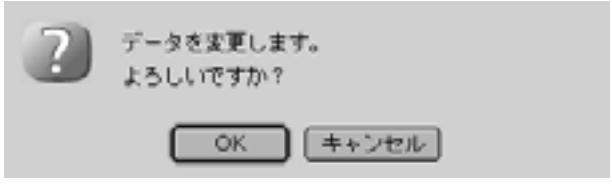
```
<script language="JavaScript1.1">
<!--
function formConfirm(type) {
  if (type == "update") {
    rtn = confirm("データを変更します。よろしいですか？");
  } else {
    rtn = confirm("データを削除します。よろしいですか？");
  }

  if (rtn) {
```

```
      return true;
    }
    return false;
  }
  //-->
  </script>
```

変数名に$がつかないことを除けば、PHPとよく似ています。やっていることはごく単純で、引数の種別によって表示メッセージを変えながらJavaScriptの組み込み関数confirmを呼んでダイアログを表示するだけです。

confirmは、OKを押されるとtrueを返し、キャンセルを押されるとfalseを返します*9（図2-15）。

図2-15　データ変更確認ダイアログ



更新処理

PgMetaDataクラスに更新フォーム（図2-14）で「更新」を選択した場合の処理を行なう関数をupdateSQLとして追加します（リスト2-15：581～595行目）。

updateSQLの引数はinsertSQLと同じで、データチェック用ユーザ定義関数およびデバッグフラグです。

フォームから渡されたデータは、以下の変数で取得しています。

$_POST["atrlist"]	列名、値のリスト
$_POST["nulllist"]	NULLかどうかのフラグのリスト

以上のデータから、update用のSQL文を作って実行するのが本関数の役目です。update用のSQL文は以下のパターンになります。

```
UPDATE テーブル名 SET 列名1 = '値1', 列名2 = '値2', ...
WHERE ctid = TID
```

その列の値として NULL が指定されることもあるので、NULLフラグのリスト（$_POST["nulllist"]）をチェックしています。

*9 ブラウザによってボタンに表示される文字列は異なります。

この関数でのポイントは以下のようになります。

①データを更新する前に現在の行の値を取得し、検索時の行の値と一致しているかどうかチェックする。一致していなければ、ほかのユーザによって更新が行なわれた可能性があるのでその旨を表示して更新処理を中止する（533～539行目）。
②①のチェックがOKでも、そのあと実際にUPDATEを発行するまでのわずかな時間のあいだにほかのユーザに行を更新される可能性があるので、検索された行にロックをかけています（533行目）。

getRow()は引数$lockがTRUEのときに、SELECT文の最後に"FOR UPDATE"を付加します（265行目）。FOR UPDATEがついていると、検索された行はトランザクションの終了までロックがかかり、ほかのユーザ（トランザクション）はその行を更新を待たされることになります。

なお、このような処理をするために、明示的にトランザクションを開始しています（531行目）。トランザクションについては、コラムを参照してください。

Column トランザクション

トランザクションとは、データベースに対して行なう、ある一連の処理のまとまりを指します。たとえば人事データのテーブルと給与台帳のテーブルがあり、社員の退職処理を行なうものとします。そのためにはまず人事テーブルから該当社員の行を削除し、続いて給与台帳のテーブルからも該当社員を削除します。

この二つの処理は不可分であり、もしも人事テーブルからは削除したのに給与台帳テーブルにはその社員のデータが残ってしまうとおかしなことが起きるのは想像に難くありません。そこでデータベースシステムではこれらの処理をトランザクションとして宣言します。PostgreSQLのSQL文では、次のように記述します。

```
BEGIN;      -- トランザクションの開始
DELETE FROM 人事テーブル WHERE 社員番号 = 12345;
DELETE FROM 給与台帳テーブル WHERE 社員番号 = 12345;
COMMIT;     -- トランザクションの終了
```

こうしておけば、人事テーブルの更新に成功したあと、なんらかの理由で給与台帳テーブルの更新に失敗した際にも自動的に人事テーブルの内容を更新前の状態に戻してくれるので、この二つのテーブルの整合性がとれない状態が発生することを防ぐことができます。

このようにBEGIN; COMMIT;を使わずに単独でSQL文を発行した場合には、PostgreSQLはこのSQL文があたかもそれ自体1個のトランザクションとして扱われます。

削除処理

削除フォームで「削除」を選択した場合の処理を、PgMetaDataクラスにdeleteSQLというメンバ関数として追加します（**リスト2-15**：550～560行目）。deleteSQLの引数はデバッグフラグだけです。

更新のときと同様、削除対象の行のTIDは$this->tidで識別します。delete用のSQL文は以下のパターンになります。

```
DELETE FROM テーブル名 WHERE ctid = TID;
```

削除前に行内容を読み出して確認すること、削除対象の行をロックすることなどは更新処理と変りません。

サンプルプログラム7

以上でライブラリクラスの準備ができたので、実際にそれらを使うサンプルプログラム7（**リスト2-14**：ex7/ex7.php）を紹介します。

基本的に、サンプルプログラム7はサンプルプログラム6（**リスト2-12**：ex6/ex6.php）をベースにしているので、変更点のみ説明します。

●JavaScriptファイルの読み込み（41行目）

前述したように <head>～</head> の中で

```
include_once("jscripts.inc");
```

によってJavaScriptスクリプトを取り込んでいます。

●$mode変数

表2-9に加え、いくつか$mode変数の値が追加されています。ここではそれらをまとめて**表2-10**に示します。

表2-10　$mode変数の値と使い方

$mode変数の値	動作
未設定またはtop	トップレベルメニュー
search	検索フォームの表示
show	検索結果表示
next	次ページ表示
prev	前ページ表示
insert	登録フォームの表示
insertExec	登録の実行
update	更新フォームの表示
updateExec	更新/削除処理
delete	削除フォームの表示
deleteExec	削除の実行

● **更新/削除フォームの表示（66～70行目）**

printDataInputFormで更新 / 削除フォームを表示します。

● **データ更新/削除SQLの実行（76～85行目）**

$mode変数に応じて、更新ならばupdateSQL、削除ならばdeleteSQLを呼び出すだけです。

3.9 フレームの利用

フレームは、ブラウザのページの中の表示領域を複数に分割するテクニックです。うまく使えば同時に複数のページを表示できるので操作性が向上します。逆に使い方を誤ると、非常に使いにくいものになるもあります。できるだけシンプルな使い方を心がけましょう。

ここでは、**図2-16**のように、ページの左側にメインメニュー、ページ右上に検索キー入力/データ追加入力、そしてページ右下には検索結果/データ更新/削除の入力画面が表示されるようにしてみました。

図2-16 フレームを利用したサンプルプログラム8（ex8/index.html）

3.9.1 フレームの枠組み

フレームを使う場合、まずフレームの「枠組み」を規定するページを作ります（**リスト2-16**：ex8/index.html）。

このページ自体は何も表示しません。単にページの中のフレーム分割を規定するだけです（<body>タグがないことに注意してください）。

リスト2-16 フレームの枠組み（ex8/index.html）

```
 1  <html>
 2  <!doctype html public "-//w3c//dtd html 3.2//ja">
 3  <html>
 4  <head>
 5  <title>Table Editor Powered by PostgreSQL/PHP</title>
 6  <meta http-equiv=content-type content="text/html; charset=x-euc-jp">
 7  </head>
 8
 9  <frameset cols="30%,70%">
10   <frame src="ex8.php">
11   <frameset rows="50%,50%">
12         <frame src="init_image1.html" name="key">
13         <frame src="init_image2.html" name="result">
14   </frameset>
15  </frameset>
16  </html>
```

フレーム分割の指定は<frameset>タグで行ないます。たとえば

```
<frameset cols="30%,70%">
```

は、フレームを左右にそれぞれ30%と70%の比率で分割することを指定しています。この30%、70%という値はあくまで初期表示の比率であり、枠をマウスでドラッグすることによってフレームの表示幅を変えることができます。枠をドラッグできないようにすることもできますが、利用する側からするとはなはだ不便であり、ストレスがたまります。よほどの理由がないかぎりこのような設定をすべきではないでしょう。

フレームの中に表示する実際の内容の指示は<frame>タグで行ないます。

```
<frame src="ex8.php">
```

上の例ではフレームの中にex8.phpを表示することを指示しています。複数の<frame>タグがある場合は、<frameset>の指示にしたがって順番に<frame src=…>で指定した内容が表示されます。ここでは<frameset cols="30%,70%">の指定を行なっているので、左側のフレームから<frame src=…>で指定したページが表示されていきます。

<frame>タグの代わりに入れ子で<frameset>タグを指定することもできます。

```
<frameset rows="50%,50%">
```

ここでは、右側のフレームをさらに上下に分割しています。そして

```
<frame src="init_image1.html" name="key">
<frame src="init_image2.html" name="result">
```

とすることにより、上のフレームにinit_image1.html、下のフレームにinit_image2.htmlが

表示されるようにしています。これらのページは初期状態でのみ表示されるページです（**図2-16**）。単に穴埋めのために表示するページですから中身はなんでもよいのですが、今回はgimpで作ったイメージを適当に貼りつけたページを用意しました。

<frame>タグにname=が指定されていますが、これによってフレームに名前をつけ、あとからこの名前を使って指定のフレームにページ内容を表示することができます。

リスト2-17　サンプルプログラム8（ex8/ex8.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3  require_once("pgmetadata.inc");
 4
 5  class myPgSelect extends PgSelect {
 6    var $mode_var = "mode";                          // 表示モード変数名
 7    var $next = "next";                              // 次ページ
 8    var $prev = "prev";                              // 前ページ
 9  }
10
11  class myPgMetaData extends PgMetadata {
12    var $mode_var = "mode";  // 表示モード変数名
13    var $exec_select_command = "show";               // 検索開始コマンド名
14    var $insert_command = "insert";                  // 挿入リクエストコマンド名
15    var $update_command = "update";                  // 更新リクエストコマンド名
16    var $delete_command = "delete";                  // 削除リクエストコマンド名
17    var $exec_insert_command = "insertExec";         // 挿入コマンド名
18    var $exec_update_command = "updateExec";         // 更新コマンド名
19    var $exec_delete_command = "deleteExec";         // 削除コマンド名
20    var $table_name = "otenki";                      // テーブル名
21    var $sort_column = "day";                        // 日付でソート
22    var $aliases = array("day"=>"日付","tenki"=>"天気",
23   "ondo"=>"温度","uryou"=>"雨量");
24
25    function printFormHeader() {
26      print<<< EOF
27        <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->
exec_select_command}" method="post" target="result">
28        <table>
29  EOF;
30    }
31
32  }
33
34  session_start();     // セッションの開始
35
36  if (!isset($_SESSION["sel"]) || !isset($_GET["mode"])) { // はじめての表示?
```

```
37    $_SESSION["sel"] = new myPgSelect;     // myPgSelectオブジェクト作成
38    $_SESSION["meta"] = new myPgMetaData; // myPgmetaDataオブジェクト作成
39    $mode = "top";                         // 初期表示モード設定
40  } else {
41    $mode = (isset($_GET["mode"])?$_GET["mode"]:"top");
42  }
43
44  print<<<EOF
45  <html>
46  <head>
47  <title>Example 8</title>
48  EOF;
49  include_once("jscripts.inc");
50  print<<<EOF
51  </head>
52  <body>
53  EOF;
54
55  switch($mode) {
56   case "top":
57     print <<<EOF
58       <ul>
59       <li><a href="{$_SERVER["PHP_SELF"]}?mode=search" target="key">検索フォーム</a>
60       <li><a href="{$_SERVER["PHP_SELF"]}?mode=insert" target="key">登録フォーム</a>
61       </ul>
62  EOF;
63     break;
64   case "search":
65     $_SESSION["meta"]->printForm();                    //検索フォームの表示
66     break;
67   case "show":
68     $_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL(true)); // 検索結果の表示
69     break;
70   case "next":
71   case "prev":
72     $_SESSION["sel"]->doSelect(); // 検索結果の前/次表示
73     break;
74   case "insert":
75   case "update":
76   case "delete":
77     $_SESSION["meta"]->printDataInputForm($mode);      //登録フォームの表示
78     break;
79   case "insertExec":
80     if ($_SESSION["meta"]->insertSQL()) {              //登録の実行
81       print("登録処理が正常に終了しました。<br>¥n");
82     }
83     break;
```

```
 84    case "updateExec":                                  // 更新の実行
 85      if ($_SESSION["meta"]->updateSQL()) {             // 更新処理
 86          print("更新処理が正常に終了しました。<br>¥n");
 87      }
 88      break;
 89    case "deleteExec":                  // 削除の実行
 90      if ($_SESSION["meta"]->deleteSQL()) {             // 削除処理
 91          print("削除処理が正常に終了しました。<br>¥n");
 92      }
 93      break;
 94  }
 95  ?>
 96  </body>
 97  </html>
```

左側のフレームにメニューを表示しているサンプルプログラム8（**リスト2-17**）の59行目をご覧ください。

```
<li><a  href="{$_SERVER["PHP_SELF"]}?mode=search"  target=
"key">検索フォーム</a>
```

このように<a>タグでtargetを指定することによりhref=で指定されたページがkeyという名前のついたフレーム（右上のフレーム）に表示されます。

target属性が使えるタグとしては、このほかに<form>タグと<img>タグがあります。たとえば

```
<form action="test.php3" target="test_frame" method="post">
<img src="test.gif" target="test_frame">
```

のような使い方ができます。

3.9.2 検索結果をresultフレームに表示

何もしないと、右上のkeyフレームで検索を行なうと検索結果が同じkeyフレームに表示されます。ここで検索結果をその下のresultフレームに表示することを考えましょう。

前述したように、検索結果をresultフレームに表示するためには、<form>タグでtarget属性を指定すればよいのです。この部分は、PgMetaDataクラスのメンバ関数printFormHeader()で処理していました（**リスト2-15**：175～180行目）。

```
function printFormHeader() {
  print<<< EOF
```

```
        <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}
={$this->exec_select_command}" method="post">
        <table>
EOF;
  }
```

このような画面デザインの問題のために、汎用的なクラスであるPgMetaDataクラスを変更するのは考えものです。そこで、PgMetaDataを継承したクラスの中でメンバ関数をオーバライドします（**リスト2-17**：25～30行目）。

```
  function printFormHeader() {
    print<<< EOF
        <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}
={$this->exec_select_command}" method="post" target="result">
        <table>
EOF;
  }
```

オリジナル（**リスト2-15**）との違いは**target="result"**を追加しているだけです。これで検索結果がresultフレームに表示されるようになります。

3.10 セキュリティ

ここまでに作成したスクリプトを使えば、ひととおりデータベースの検索、データ入力、更新、削除が可能で、個人向けの用途ならこのままでも十分実用的です。ただし、インターネット上にこのシステムを設置したり、企業のような組織の中で運用しようとすると、問題が起こるおそれがあります。

・誰でもデータを閲覧できる
・誰でもデータを登録できる
・誰でもデータを変更、削除できる

つまり、故意あるいは手違いから、データを好ましくない状態に変更したり、抹消したりすることができるのです。このような問題は一般に「セキュリティ問題」と呼ばれ、さまざまな解決策があります。

3.10.1 REVOKE

インターネット上でのデータ公開では「閲覧、登録はできるが、データの変更や削除は一切できないようにしておく」という方法が使われることがあります。この場合、PHPからデータの変更、削除ができないように、nobodyユーザに対してSQLのREVOKEコマンドによって更新、削除権限を剥奪しておき、データの変更、削除はPostgreSQLのスーパユーザがpsqlから実施するなどの方法をとります。

REVOKEの使い方は、

```
REVOKE 権限1[, 権限2,...]
       ON テーブル1[,テーブル2,...]
       FROM [PUBLIC | GROUP グループ名 | ユーザ名]
```

となります。ここで権限は

```
ALL
SELECT
INSERT
UPDATE
DELETE
RULE
```

のいずれかです。ALLはすべての権限を表し、それ以外はSQLのSELECT、INSERT、UPDATE、DELETE、RULE[*10]に対応しています。ONで対象テーブルを指定します。

FROM以降は権限を剥奪するユーザを指定します。PUBLICはすべてのユーザ、GROUPはある特定のグループ[*11]、それ以外はユーザ名として解釈されます。

たとえば、otenkiテーブルに対してPHPから更新や削除を許さないようにする場合は、以下のように実行します。

```
REVOKE UPDATE ON otenki FROM nobody;
REVOKE DELETE ON otenki FROM nobody;
```

*10
RULEとは、あるSQL文を実行したときに、そのSQL文に加えてあるSQL文を実行する、あるいは代わりに別のSQL文を実行するというPostgreSQL独特の機能です。RULEの定義はCREATE RULE文で行います。詳細についてはPostgreSQLの付属のドキュメントか参考文献[1]を参照してください。

*11
groupはUNIXのgroupに似た概念で、複数のユーザをまとめて権限の管理を行うことができます。

3.10.2 認証

セキュリティを高める別の方法として、**ユーザ認証**と呼ばれるものがあります。これは、何らかの方法で個々のユーザを識別し、特定のユーザにのみ操作を許すものです。ICカードや網膜、指紋などを使った物理的な認証方法もありますが、ここでは一般に使われているパスワードによる認証をとり上げます。

パスワードによる認証は、UNIXにログインするときに使われているのですでにおなじみだと思います。システムはそのユーザしか知らないと思われる文字列（パスワード）の入力を

促し、ユーザがパスワードを正しく入力できたかどうかで、そのユーザが本当にその人自身であるかどうかを識別するというものです。逆にいえばパスワードを知っていさえすれば認証をパスできるため、グループでパスワードを共有するような使い方も可能です。

WWWシステム上でユーザ認証を行なう方法は大きく分けて二つあります。

①HTTPプロトコルで定義された認証プロトコルを使用する。
②独自にユーザ認証機構を実装する。

①はさらにパスワードの管理方法により、Apache自身の認証機構を使うもの、dbmやBerkeley DB fileを使うもの、PostgreSQLなどのデータベースを使うものなどに分かれますが、すべてBasic認証と呼ばれるプロトコルに準拠していることには変わりありません。

Basic認証では、認証が必要なURI（URL）がアクセスされた場合、サーバは

```
WWW-Authenticate: Basic realm="somerealm"
```

というメッセージをクライアントに送り認証の開始を要求します。ここでsomerealmは、認証の対象となるURIを識別するための任意の文字列で、認証用のダイアログにも表示されます。

メッセージを受け取ったクライアントは、認証ダイアログを表示し、ユーザがユーザ名とパスワードを入力するのを待ちます。

パスワードとユーザ名が入力されると、ブラウザはそれをサーバに送信します。

サーバは受け取ったユーザ名とパスワードの正当性をなんらかの方法でチェックします。もしOKならばURIの内容をクライアントに送信し、認証済のページがブラウザの画面に表示されます。

サーバが認証を拒否する場合は

```
HTTP/1.0 401 Unauthorized
```

というメッセージをクライアントに送信します。

これを見てもわかるように、Basic認証では基本的にネットワーク上をパスワード文字列がそのまま流れるため、パケットモニタなどを使ってパスワードを盗聴される可能性があり、あまり安全とはいえません。*12

そこで考え出されたのが、ダイジェストアクセス認証（Digest Access Authentication）と呼ばれるプロトコルです。ダイジェストアクセス認証では、ユーザ名とパスワードなどから非常に推測しにくい文字列を生成して認証を行ないます。生のパスワードをネットワークに流さないのでBasic認証に比べると安全ですが、すべてのブラウザが対応しているわけではないのが難点です。

Basic認証はいくつか問題はあるものの、手軽に使えるために広く普及しています。あまり重要でないデータを扱う場合、あるいはイントラネットなどではこれで十分なケースがほとんどです。本書でもBasic認証を使った実例を紹介します。

*12
SSL（Secure Socket Layer）と呼ばれる技術を使えばパスワードの盗聴を防止できますが、証明書の入手や費用の問題が発生します。

これに対し、②はすべてを自分で実装するわけですから、ユーザ認証のダイアログを独自のものにできるし、より強固なセキュリティを構築することができます。たとえば、PHP Base Library（http://phplib.shonline.de/）はその一例で、ダイジェストアクセス認証と類似のユーザ認証機構を独自に実装しています。また、Basic認証では困難な「ログアウト」[*13]を実現しています。興味のある人は使ってみてください。

*13
Basic認証では、一度認証されたページは、ブラウザを抜けるまで認証が有効になります。したがって、もうそのページへのアクセスが必要なくなったらかならずブラウザを終了させなければなりません。そうしないと、席を離れた隙に、ほかの人がパスワードなしで認証済みのページを閲覧できてしまいます。

3.10.3 ApacheのBasic認証

ApacheにはBasic認証が実装されており、特にPHPスクリプトを書かなくてもすぐにユーザ認証が使えます。ただし、デフォルトでは、Apacheは任意のディレクトリに対してBasic認証を有効にしていません。たとえば、/usr/local/apache/htdocs/以下でBasic認証を有効にするためには、/usr/local/apache/conf/httpd.confの、

```
DocumentRoot "/usr/local/apache/htdocs"
```

の下のほうにある

```
AllowOverride None
```

を

```
AllowOverride AuthConfig
```

に変更してください。

fooというユーザにだけアクセスを許可するには、まずスクリプトのあるディレクトリに以下の内容を持つ.htaccessという名前のファイルを作ります。

```
AuthType Basic
AuthName test
AuthUserFile /home/postgres/password
<Limit GET POST>
require user foo
</Limit>
```

/home/postgres/passwordにはユーザ名と暗号化されたパスワードが入ります。Apache付属のhtpasswdというツールを使って、以下の方法で作成できます。

```
$ /usr/local/apache/bin/htpasswd -c /home/postgres/password foo
```

なお、-cはパスワードファイルを生成するオプションで、初回のみ使用します。

複数のユーザを認証するには、

```
AuthType Basic
AuthName test
AuthUserFile /home/postgres/password
<Limit GET POST>
require user foo
require user bar
</Limit>
```

のように.htaccessにユーザを追加し、またパスワードファイルにも

```
$ /usr/local/apache/bin/htpasswd /home/postgres/password bar
```

で追加します。

ユーザ数が増えると.htaccessに多くのrequire user…… の行を書かなければならなくなりますが、ユーザの集合である**グループ**という概念を使うことでこれを回避できます。

たとえば上の例をグループを使って書き直すと

```
AuthType Basic
AuthName test
AuthUserFile /home/postgres/password
AuthGroupFile /home/postgres/group
<Limit GET POST>
require group mygroups
</Limit>
```

となります。/home/postgres/groupはグループの定義が書かれたファイルで

```
mygroups: foo, bar
```

という内容になっています。

Basic認証が完了すると、認証されたユーザ名が環境変数REMOTE_USERにセットされます。PHPでは環境変数を**$_SERVER[環境変数名]**としてアクセスできるので

```
<?php print("ようこそ {$_SERVER["REMOTE_USER"]} さん！");?>
```

というスクリプトを実行すると、認証済みのユーザ名が表示されます。

3.10.4 PHPによるBasic認証

Apache組み込みのユーザ認証では、ユーザ名やパスワードは外部ファイルに格納されていました。これをデータベースで管理するにはPHPによる認証を使います。PHPでは、Header()という関数を使ってhttpのプロトコルを直接扱うことができ、これを使ってHTTPのBasic認証を容易に実装できます。

PHPによる認証では、**PHP_AUTH_USER**と**PHP_AUTH_PW**という変数が利用できます。PHP_AUTH_USERとPHP_AUTH_PWには、Basic認証が完了している場合にユーザ名とパスワードがセットされます。ただし、PHPによる認証を行なっている場合は、セキュリティ上の理由により、REMOTE_USER変数はセットされません。

では、具体的な手順を見ていきましょう。**リスト2-18**(ex9/php_auth.php)をご覧ください。

リスト2-18 PHPによる認証サンプルプログラム(ex9/php_auth.php)

```
 1  <?php
 2  if (!isset($_SERVER["PHP_AUTH_USER"])) {
 3    Header("HTTP/1.0 401 Unauthorized");
 4    Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
 5    print("ユーザ認証はキャンセルされました。");
 6    exit;
 7  } else {
 8    if ($_SERVER["PHP_AUTH_PW"] != "secret") {
 9      Header("HTTP/1.0 401 Unauthorized");
10      Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
11      print("パスワードが違います。");
12      exit;
13    } else {
14      print("ようこそ、{$_SERVER["PHP_AUTH_USER"]} さん！");
15    }
16  }
17  ?>
```

まず気をつけなければならないのは、これらのスクリプトがHTMLのコンテンツに先立って先頭になければならないことです。つまり、<html>タグよりも前でなければなりません。

2行目で$_SERVER["PHP_AUTH_USER"]がセットされているかどうかチェックします。この変数がセットされていない場合、このページはまだ認証されていないので3～6行目を実行し、認証要求をブラウザに送信します。すると、ブラウザは**図2-14**のようにユーザ名とパスワードを要求するダイアログを表示し、ユーザがユーザ名とパスワードを入力するのを待ちます。

図2-17 認証ダイアログ

ここで「キャンセル」を選ぶと、ブラウザは処理を打ちきり、ダイアログを閉じたあとで5行目のメッセージを表示して終了します（**図2-18**）。

図2-18 認証キャンセル画面

では、ユーザ名とパスワードを入力して確認ボタンを押すとどうなるでしょう？

実は、ブラウザは再度同じURLにアクセスを行ないます。入力されたユーザ名とパスワードもサーバに送られます。今度は$_SERVER["PHP_AUTH_USER"]がセットされているので、8行目に移ります。単純にパスワードと固定文字列を比較していますが、ここでユーザ名とパスワードのテーブルを検索するようにすれば、データベースを使った認証ができます。

もし入力されたパスワードがsecret以外なら、9行目で

```
Header("HTTP/1.0 401 Unauthorized");
```

をクライアントに送信しています。すると、ブラウザは再び**図2-17**のダイアログを表示します。

パスワードが一致していれば、13行目から実行するので「ようこそ……」のあとにダイアログで入力したユーザ名を表示します。

3.10.5 データベースを使ったBasic認証

PHPによる認証で、パスワードチェックの部分にデータベースを使ってみましょう。今回はごくシンプルに、PostgreSQL自体が持つユーザ認証を活用することにします。

PostgreSQLでは、データベースへの接続時にパスワード認証を行なうことができます。デフォルトでは認証を行なわないようになっているので、まずそれを有効にします。

pg_hba.confの設定

/usr/local/pgsql/data/pg_hba.confの下のほうに、

```
local       all                                             trust
host        all         127.0.0.1     255.255.255.255     trust
```

という行があります。上の行は、UNIXドメインのソケットで接続するときの設定です。pg_connectで接続するときのホスト名を空文字列（""）にすると、UNIXドメインソケットを使った接続になります。その下の行は、INETドメインで自ホスト、すなわちlocalhostに接続するときの設定です。

本章ではUNIXドメインでの接続を使っているので、上の行を変更します。各項目は以下のような意味になります。

- **local**

  UNIXドメインソケットを表すキーワード。

- **all**

  制限の対象となるデータベース名。allとすると、すべてのデータベースが対象となる。

- **trust**

  認証の種類。

  - trust　：認証なし
  - md5　：md5認証（パスワードは暗号化されてから送信。データベース内に暗号化してパスワードを管理）
  - crypt　：crypt認証（パスワードは暗号化されてから送信）
  - password　：パスワード認証（クリアテキスト）

  など。

今回は、データベースfooに対してmd5認証を適用することにします。さきほどの2行の上に

```
local       foo                                             md5
```

という行を追加してください（認証は、foo以外のデータベースには適用されません）。

このままではlocalhostを指定して接続すると認証なしでfooデータベースに接続できてしまうので、実際に使う場合には

```
host        foo         127.0.0.1     255.255.255.255   md5
```

という行も追加して

```
local       foo                                         md5
host        foo         127.0.0.1     255.255.255.255   md5
local       all                                         trust
host        all         127.0.0.1     255.255.255.255   trust
```

としておいたほうがよいでしょう。

PostgreSQLのパスワードの設定

次に、パスワードの設定を行ないます。今回は、fooというユーザを作り、パスワードを設定します。PostgreSQLのスーパユーザであるpostgresでログインし、createuserコマンドでfooユーザを作成します。

```
$ createuser foo
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

次にパスワードを設定します。以下の例では、barというパスワードを設定しています。

```
$ psql foo
[中略]
foo=> ALTER USER foo WITH ENCRYPTED PASSWORD 'bar';
ALTER USER
```

そしてpostmasterにpg_hba.confファイルの再読み込みをリクエストします。

```
$ pg_ctl reload
```

このコマンドはかならずPostgreSQLのスーパユーザ（postgres）で実行します。

では、設定したパスワードが有効かどうか一応確認してみましょう。psqlでユーザ名を指定してログインするために、-Uオプションを使います。

```
$ psql -U foo foo
Password: bar

Welcome to psql, the PostgreSQL interactive terminal.

Type:  copyright for distribution terms
       h for help with SQL commands
       ? for help on internal slash commands
       g or terminate with semicolon to execute query
       q to quit

Using pager is off.
foo=>
```

次に、わざと違うパスワードを入力してみます。

```
$ psql -U foo foo
Password: foo

psql: Password authentication failed for user 'foo'
```

このように、パスワードが働いていることが確認できました。

GRANTによるアクセス権の設定

いままではnobodyユーザでPostgtreSQLに接続していましたが、ユーザ認証を行なう場合には認証されたユーザ名で接続することになります。そこで、ユーザfooがotenkiテーブルにアクセスできるようにGRANTを使ってアクセス権を与えます。psqlから以下を実行してください。

```
GRANT ALL ON otenki TO foo;
```

ユーザfooに検索だけを許可する場合は

```
GRANT SELECT ON otenki TO foo;
```

とします。このように、SELECT、INSERT、UPDATE（と、DELETE）ごとに権限を設定することも可能です。

認証サンプルプログラム

では、パスワード認証を実際にPHPスクリプトの中で使ってみましょう。前節で紹介した、フレームを使ったサンプルをベースに修正を加えます。

前回のサンプルプログラム8（**リスト2-17**：ex8/ex8.php）と今回のサンプルプログラム（**リスト2-19**：ex9/ex9.php）を比較しながら説明します。

リスト2-19 認証サンプルプログラム（ex9/ex9.php）

```
 1  <?php
 2  require_once("pgselect.inc");
 3  require_once("pgmetadata.inc");
 4
 5  class myPgSelect extends PgSelect {
 6    var $mode_var = "mode"; // 表示モード変数名
 7    var $next = "next";     // 次ページ
 8    var $prev = "prev";     // 前ページ
 9  }
10
11  class myPgMetaData extends PgMetadata {
12    var $mode_var = "mode"; // 表示モード変数名
13    var $exec_select_command = "show";          // 検索開始コマンド名
14    var $insert_command = "insert";             // 挿入リクエストコマンド名
15    var $update_command = "update";             // 更新リクエストコマンド名
16    var $delete_command = "delete";             // 削除リクエストコマンド名
17    var $exec_insert_command = "insertExec";    // 挿入コマンド名
18    var $exec_update_command = "updateExec";    // 更新コマンド名
19    var $exec_delete_command = "deleteExec";    // 削除コマンド名
20    var $table_name = "otenki";    // テーブル名
21    var $sort_column = "day";      // 日付でソート
22    var $aliases = array("day"=>"日付","tenki"=>"天気",
23        "ondo"=>"温度","uryou"=>"雨量");
24
25    function printFormHeader() {
26      print<<< EOF
27        <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}={$this->
exec_select_command}" method="post" target="result">
28        <table>
29  EOF;
30    }
31
32  }
33
34  if (!isset($_SERVER["PHP_AUTH_USER"])) {
35    Header("HTTP/1.0 401 Unauthorized");
36    Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
37    print("ユーザ認証はキャンセルされました。");
38    exit;
39  } else {
40    $db = new DbConnect();
41    if ($db->getConnection() == false) {
42      Header("HTTP/1.0 401 Unauthorized");
43      Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
44      print("ユーザ名かパスワードが違います。");
45      exit;
46    }
47  }
```

```
48
49  session_start();    // セッションの開始
50
51  if (!isset($_SESSION["sel"]) || !isset($_GET["mode"])) { // はじめての表示?
52    $_SESSION["sel"] = new myPgSelect;   // myPgSelectオブジェクト作成
53    $_SESSION["meta"] = new myPgMetaData;// myPgmetaDataオブジェクト作成
54    $mode = "top";           // 初期表示モード設定
55  } else {
56    $mode = (isset($_GET["mode"])?$_GET["mode"]:"top");
57  }
58
59  print<<<EOF
60  <html>
61  <head>
62  <title>Example 9</title>
63  EOF;
64  include_once("jscripts.inc");
65  print<<<EOF
66  </head>
67  <body>
68  EOF;
69
70  switch($mode) {
71   case "top":
72     print <<<EOF
73       <ul>
74       <li><a href="{$_SERVER["PHP_SELF"]}?mode=search" target="key">検索フォーム</a>
75       <li><a href="{$_SERVER["PHP_SELF"]}?mode=insert" target="key">登録フォーム</a>
76       </ul>
77  EOF;
78     break;
79   case "search":
80     $_SESSION["meta"]->printForm();                    //検索フォームの表示
81     break;
82   case "show":
83     $_SESSION["sel"]->doSelect($_SESSION["meta"]->makeSQL(true)); // 検索結果の表示
84     break;
85   case "next":
86   case "prev":
87     $_SESSION["sel"]->doSelect(); // 検索結果の前/次表示
88     break;
89   case "insert":
90   case "update":
91   case "delete":
92     $_SESSION["meta"]->printDataInputForm($mode);     //登録フォームの表示
93     break;
94   case "insertExec":
```

```
 95      if ($_SESSION["meta"]->insertSQL()) {      //登録の実行
 96        print("登録処理が正常に終了しました。<br>¥n");
 97      }
 98      break;
 99    case "updateExec":                           // 更新の実行
100      if ($_SESSION["meta"]->updateSQL()) {      // 更新処理
101        print("更新処理が正常に終了しました。<br>¥n");
102      }
103      break;
104    case "deleteExec":                           // 削除の実行
105      if ($_SESSION["meta"]->deleteSQL()) {      // 削除処理
106        print("削除処理が正常に終了しました。<br>¥n");
107      }
108      break;
109  }
110  ?>
111  </body>
112  </html>
```

34行目からPHPによる認証を行ないます。40行目で、データベースへの接続に失敗した場合は、認証に失敗したとみなして「ユーザ名かパスワードが違います。」というメッセージを表示しています。

ところで、従来のDbConnectクラスのままではPHPが設定するユーザ名やパスワードを使って認証してくれないので、DbConnectクラスを改造します。

DbConnectクラスの改造

修正したのはDbConnectコンストラクタ関数だけなので、その部分を**リスト2-20**に示します。

リスト2-20　DbConnect関数（ex9/lib/dbconnect.iniより）

```
15    function DbConnect() {  // コンストラクタ
16      if (isset($_SERVER["PHP_AUTH_USER"])) {
17        $this->user = $_SERVER["PHP_AUTH_USER"];       // ユーザ名
18      }
19      if (isset($_SERVER["PHP_AUTH_PW"])) {
20        $this->password = $_SERVER["PHP_AUTH_PW"];     // パスワード
21      }
22      $this->getConnection();
23    }
```

Hypertext Preprocessor

Chapter-4 まとめ

第2部では、PHPとPostgreSQLによる、データベースを使った実用的なアプリケーションの作り方を説明しました。業務での使用に耐えるアプリケーションを作るためには、いろいろな問題をクリアしていかなければなりません。そこで、簡単な検索スクリプトから始まり、セッション管理の導入、データ更新機能、フレーム、セキュリティなどの機能を段階的に追加しながらこれらの問題に対する解決策を提案しました。その成果はクラスライブラリのかたちになっており、容易に拡張が可能です。

最後にこれらクラスライブラリの仕様をまとめ、第2部の締めくくりとします。ここで説明する仕様は、最終段階のもの（ex9/lib/dbconnect.inc、ex9/lib/pgselect.inc、ex9/lib/pgmetadata.inc）に基づいています。

4.1 DbConnect

このクラスは PostgreSQL データベースシステムへの接続を管理します。接続情報のデフォルト値はdef.incで定義されています。

def.incの内容

```
<?php
define("DBNAME", "foo");// データベース名
define("HOST", "");     // ホスト名
define("PORT", "");     // ポート番号
define("USER", "");     // ユーザ名
define("PASSWORD", ""); // パスワード
?>
```

コンストラクタ

■DbConnect()

引数はありません。

メンバ関数

■doConnect()

PostgreSQLデータベースに接続します。$_SERVER["PHP_AUTH_USER"]、$_SERVER["PHP_AUTH_PW"]が設定されている場合はそれらをユーザ名、パスワードとして認証つきで接続します。正常に接続できたかどうかは、後述するインスタンス変数conで確認できます。

■doClose()

PostgreSQLデータベースとの接続を切断します。接続されていない場合は何もしません。

参照インスタンス変数

■con

PostgreSQLデータベースとの接続ハンドル。falseの場合は接続に失敗したことを示します。

設定可能インスタンス変数

なし。

4.2 PgSelect

指定テーブルに関する検索処理を行ないます。

コンストラクタ

■PgSelect

引数はありません。DbConnectクラスを使い、データベースへの接続処理を行ないます。

メンバ関数

■doSelect($sql = "")

SELECT文を実行し、HTMLのテーブル形式で表示します。引数$sqlは実行するSELECT文です。

doSelectは、後述するように2ページ目以降では引数でSELECT文をもらう必要がありません。

データ件数がインスタンス変数のmaxl（デフォルト5行）を超える場合は自動的に「次ページ」に表示を持ち越します。

なお、テーブルの表示詳細は次に示すメンバ関数をオーバライドすることによって変更できます。

● printTableHeader()

テーブル開始タグを印字します。

● printHeader($i, $str)

テーブルの見出し$i列目に$strで指定される列名を印字します。

● printData($i, $str)

テーブルの$i列目のデータを印字します。

● printPrev()

「前」ページ用のアンカーを表示します。

● printNext($n)

「次」ページ用のアンカーを表示します。$nは次ページの件数です。

● printUpdateTag($tid)

検索結果が「更新可能」の場合、テーブル内に「更新/削除」用のアンカーを表示します。$tidは該当行のTID（タプルID）です。

設定可能インスタンス変数

■ **max**

ページに一度に表示する行数。

■ **mode_var**

表示モード変数名。初期値は"mode"。

■ **next**

次ページを表示要求するコマンド。初期値は"next"。

■ **prev**

前ページを表示要求するコマンド。初期値は"prev"。

■ **update**

更新要求するコマンド。初期値は"update"。

■ **delete**

削除要求するコマンド。初期値は"delete"。

4.3 PgMetaData

コンストラクタ

■ **PgMetaData**

引数はありません。

メンバ関数

■getMetaData()

該当テーブルの情報を取得します。

■printForm()

検索用のフォームを生成します。以下のメンバ関数をオーバライドすることによって表示の外観を変更できます。

- printFormHeader()

検索フォームの<form..>と<table>を表示します（フォームの表示には<table>を使っています）。デフォルトでは以下のように定義されています。

```
function printFormHeader() {
  print<<< EOF
    <form action="{$_SERVER['PHP_SELF']}?{$this->mode_var}
={$this->exec_select_command}" method="post">
    <table>
EOF;
  }
```

- printFormFooter()

printFormHeader()とペアになる関数です。デフォルトでは以下のように定義されています。

```
function printFormFooter() {
  print<<< EOF
    </table>
    <input type="submit" value="検索開始">
    <input type="reset" value="クリア">
    </form>
EOF;
  }
```

- printAttrName($atrnumber, $atrname)

$atrnumber番目の列の名前を$atrnameとして表示します。

- printOprName($n)

オペレータを<select>タグを使って表示します。$nは列番号です。

■printDataInputForm($mode)

データ入力（新規追加、変更）のフォームを表示します。以下のメンバ関数をオーバライドすることによって表示の外観を変更できます。

●printDataInputFormHeader()

検索フォームの<form..>と<table>を表示します(フォームの表示には<table>を使っています)。デフォルトでは以下のように定義されています。

```
function printDataInputFormHeader($mode) {
  switch ($mode) {
  case $this->insert_command:
    $op= $this->exec_insert_command;
    break;
  case $this->update_command:
    $op= $this->exec_update_command;
    break;
  case $this->delete_command:
    $op= $this->exec_delete_command;
    break;
  }

  print <<<EOF
     <form action="{$_SERVER["PHP_SELF"]}?{$this->mode_var}
=$op&tid={$this->tid}" method="post">
      <table border>
      <tr><th>カラム</th><th>データ型</th><th>データ</th>
      <th>NULL</th><th>初期値</th><th>制約</th></tr>
EOF;
  }
```

●printDataInputFormFooter($mode)

printDataInputFormHeader()とペアになる関数です。$modeで処理の区別を行ないます.$modeがインスタンス変数のinsert_commandと等しければ登録、update_commandと等しければ更新、delete_commandと等しければ削除となります。デフォルトでは以下のように定義されています。

```
function printDataInputFormFooter($mode) {
  print("</table>¥n");

  if (is_array($this->table_constraints)) {
    print("<table border>¥n");
```

```
      print("<tr><th>テーブル制約名</th><th>制約</th></tr>¥n");

      foreach ($this->table_constraints as $name => $val) {
     print("<tr><tr><td>$name</td><td>$val</td></tr>¥n");
      }
    }
    print("</table>¥n");

    switch ($mode) {
    case $this->insert_command:
      print<<<EOF
     <input type="submit" value="登録">
EOF;
      break;
    case $this->update_command:
      print<<<EOF
     <input  type="submit"  name="{$this->update_command}"
value="更新" onClick="return formConfirm('update')">
EOF;
      break;
    case $this->delete_command:
      print<<<EOF
     <input  type="submit"  name="{$this->delete_command}"
value="削除" onClick="return formConfirm('delete')">
EOF;
    }
    print<<<EOF
      <input type="reset" value="クリア">
      </form>
EOF;
  }
```

■ insertSQL($user_check = false, $debug = false) {

データ登録用のinsert文を作成、実行します。

$user_checkはユーザが定義するデータチェック用の関数です。この関数には$atrlistが引数として渡ります。$atrlistは列名をキーとする連想配列で、ユーザ入力のデータが格納されています。データチェックの結果がOKならtrue、NGならfalseを返すことが求められます。

$debugがtrueならば生成されたSQL文を表示します。

■**updateSQL($user_check = false, $debug = false)**

データ更新用のupdate文を作成、実行します。引数の意味はinsertSQLと同じです。

■**deleteSQL($debug = false)**

データ削除用のdelete文を作成、実行します。引数の意味はinsertSQLと同じです。

設定可能インスタンス変数

■**table_name**

テーブル名。

■**sort_column**

ソートを行なう列名。

■**aliases**

列の別名連想配列 ([列名][別名])。

■**is_print_type_name**

データ型名の表示の有無。初期値は表示なし。

■**is_print_opr_desc**

オペレータの説明の表示の有無。初期値は表示なし。

■**mode_var**

表示モード変数名。初期値は"mode"。

■**exec_select_command**

検索開始コマンド名。初期値は"show"。

■**insert_command**

挿入リクエストコマンド名。初期値は"insert"。

■**exec_insert_command**

挿入コマンド名。初期値は"insertExec"。

■**exec_update_command**

挿入コマンド名。初期値は"updateExec"。

■**exec_delete_command**

挿入コマンド名。初期値は"deleteExec"。

参考文献

[1]「改訂第3版 PostgreSQL完全攻略ガイド」石井達夫著、技術評論社、2001

[2]「はじめてのPostgreSQL」Bruce Momjian著、ピアソンエデュケーション、2000

[3]「標準SQLガイド 改訂第4版」C.J.Date,Hugh Darwen共著、アスキー、1998

[4]「プログラマのためのSQL第2版」J.Celko著、ピアソンエデュケーション、2001

[5]「PostgreSQLオフィシャルマニュアル」PostgreSQL Global Development Group著、インプレス、2002