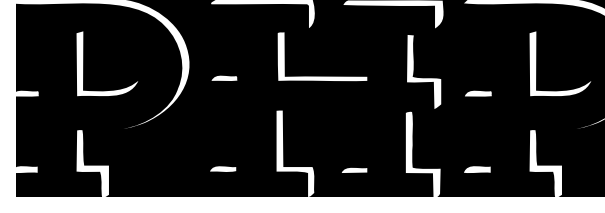
Hypertext Preprocessor

Part-2

PHPと
データベースの連携

第２部



Hypertext Preprocessor

HP

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に限らず、どんな言語を使ったシステムでも同様です。

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、本書で紹介するPostgreSQLをはじめとするフリーのデータベースが手に入るようになりました。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さがしたり、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？」とディスクの中をさが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＆オープンソフト

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いません。PostgreSQL付属のマニュアルや部末の参考文献[1]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文字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。空白やハイフン（-）などの記号、日本語は使えません。

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。

これもまたUNIXと同じですが、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限り、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
  Database  |  Owner   | Encoding
------------+----------+----------
 foo        | foo      | EUC_JP
 regression | postgres | EUC_JP
 template1  | postgres | EUC_JP
(3 rows)
```

Database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。

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言語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。それに対して行の方は、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、列名は数字以外ではじまる31文字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。日本語も使えます。空白やハイフン（-）などの記号、および**表2-2**にある予約語は使えません[*3]。

*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」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は「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（"）を用いた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バイトの文字列（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バイトの可変長文字列
TEXT	可変長文字列
SMALLINT	2バイト整数
INTEGER	4バイト整数
INT	INTEGERの別名
INT8	8バイト整数
NUMERIC	多倍長整数（有効桁数1000桁まで）
DECIMAL	多倍長整数（有効桁数1000桁まで）
FLOAT	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
POINT	点
BOX	矩形
OID	オブジェクトID *4

表 2-2 予約語

ABORT	ALL	ANALYZE	AND
ANY	AS	ASC	BETWEEN
BINARY	BIT	BOTH	CASE
CAST	CHAR	CHARACTER	CHECK
CLUSTER	COALESCE	COLLATE	COLUMN
CONSTRAINT	COPY	CROSS	
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER
DEC	DECIMAL	DEFAULT	DEFERRABLE
DESC	DISTINCT	DO	ELSE
END	EXCEPT	EXISTS	EXPLAIN
EXTEND	EXTRACT	FALSE	FLOAT
FOR	FOREIGN	FROM	FULL
GLOBAL	GROUP	HAVING	IN
INITIALLY	INNER	INTERSECT	INTO
IS	ISNULL	JOIN	LEADING
LEFT	LIKE	LIMIT	LISTEN
LOAD	LOCAL	LOCK	MOVE
NATURAL	NCHAR	NEW	NONE
NOT	NOTNULL	NULL	NULLIF
NUMERIC	OFFSET	OLD	ON
OR	ORDER	OUTER	OVERLAPS
POSITION	PRECISION	PRIMARY	PROCEDURE
PUBLIC	REFERENCES	RESET	RIGHT
SELECT	SESSION_USER	SETOF	SHOW
SOME	SUBSTRING	TABLE	THEN
TO	TRAILING	TRANSACTION	TRIM
TRUE	UNION	UNIQUE	USER
USING	VACUUM	VARCHAR	VERBOSE
WHEN	WHERE		

*4 PostgreSQLでは、OID（Object Id）というものですべての行やテーブルを識別できるようになっています。

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。

図2-1　otenki.sql

```
-- お天気日記テーブルの作成
DROP TABLE otenki;
CREATE TABLE otenki (
     day DATE PRIMARY KEY,-- 日付(主キー)
     tenki TEXT,  -- 天気(晴れ、曇...)
     ondo INTEGER,-- 温度
     ryou INTEGER -- 雨量
);
```

⇩ **実行結果**

```
DROP TABLE otenki;
psql:otenki.sql:2: ERROR:  Relation 'otenki' does not exist
CREATE TABLE otenki (
     day DATE PRIMARY KEY,
     tenki TEXT,
     ondo INTEGER,
     ryou INTEGER
);
psql:otenki.sql:8: 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。

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。

```
foo=> ¥d otenki
         Table "otenki"
 Attribute |  Type   | Modifier
-----------+---------+----------
 day       | date    | not null
 tenki     | text    |
 ondo      | integer |
 ryou      | integer |
Index: otenki_pkey
```

Attribute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。

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 (l)」か、User's Guide「19. SQLCommands」の「CREATE TABLE」を参照してください。

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、create tableの使用方法は、以下のようにして調べることができます。

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。

```
foo=> ¥h create table
Command: create table
Description: create a new table
Syntax:
CREATE [ TEMPORARY | TEMP ] TABLE table (
    column type
    [ NULL | NOT NULL ] [ UNIQUE ] [ DEFAULT value ]
    [column_constraint_clause | PRIMARY KEY } [ ... ] ]
    [, ... ]
    [, PRIMARY KEY ( column [, ...] ) ]
    [, CHECK ( condition ) ]
    [, table_constraint_clause ]
    ) [ INHERITS ( inherited_table [, ...] ) ]
```

2.5 データの登録

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。

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

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。

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。

```
INSERT INTO otenki VALUES('2000-8-1','晴れ',30,0);
INSERT INTO otenki VALUES('2000-8-3','曇',27,10);
INSERT INTO otenki VALUES('2000-8-10','雨',25,100);
```

```
DROP TABLE otenki;
DROP
CREATE TABLE otenki (
     day DATE PRIMARY KEY,
     tenki TEXT,
     ondo INTEGER,
     uryou INTEGER
);
psql:otenki.sql:8: NOTICE:  CREATE TABLE/PRIMARY KEY will create
implicit index 'otenki_pkey' for table 'otenki'
CREATE
INSERT INTO otenki VALUES('2000-8-1','晴れ',30,0);
INSERT 655151 1
INSERT INTO otenki VALUES('2000-8-3','曇',27,10);
INSERT 655152 1
INSERT INTO otenki VALUES('2000-8-10','雨',25,100);
INSERT 655153 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、**-n**オプションを追加してpsqlを起動してください。

INSERTについての詳細はオンラインマニュアル「insert (1)」か、User'sGuide「19. 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][5]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
------------+-------+------+-------
 2000-08-01 | 晴れ  |   30 |     0
 2000-08-03 | 曇    |   27 |    10
 2000-08-10 | 雨    |   25 |   100
(3 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
------------+-------+------+-------
 2000-08-10 | 雨    |   25 |   100
(1 row)
```

例 温度が29度以下の日のデータを取り出す

```
foo=> SELECT * FROM otenki WHERE ondo <= 29;
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2000-08-03 | 曇    |   27 |    10
 2000-08-10 | 雨    |   25 |   100
(2 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 平均気温 FROM otenki;
 平均気温
----------
       27
(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
----------
      110
(1 row)
```

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通りです。

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

たとえば、8月3日の雨量を20にするには以下を実行します。

```
foo=> UPDATE otenki SET uryou = 20 WHERE day = '2000-8-3';
UPDATE 1
test=> SELECT * FROM otenki;
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2000-08-01 | 晴れ  |   30 |     0
 2000-08-10 | 雨    |   25 |   100
 2000-08-03 | 曇    |   27 |    20
(3 rows)
```

値は相対値で指定することもできます。たとえば、8月10日の雨量を10増やすには以下を実行します。

```
foo=> UPDATE otenki SET uryou = uryou + 10 WHERE day = '2000-8-10';
UPDATE 1
foo=> SELECT * FROM otenki;
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2000-08-01 | 晴れ  |   30 |     0
 2000-08-03 | 曇    |   27 |    20
 2000-08-10 | 雨    |   25 |   110
(3 rows)
```

UPDATEについての詳細は、オンラインマニュアル「update (1)」か、User's Guide「19. 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**文を使います[*5]。DELETE文の基本形は以下の通りです。

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

たとえば、8月1日のデータを含む行を削除するには以下のように実行します。

```
foo=> DELETE FROM otenki WHERE day = '2000-8-1';
DELETE 1
foo=> SELECT * FROM otenki;
    day     | tenki | ondo | uryou
------------+-------+------+-------
 2000-08-03 | 曇    |   27 |    20
 2000-08-10 | 雨    |   25 |   110
(2 rows)
```

DELETEについての詳細はオンラインマニュアル「delete (1)」か、User'sGuide「19. 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しょう。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;
```

データが3件だけでは少ないので、データを追加します（追加データは本書添付のCD-ROMに**articles/2/examples/add.sql**というファイル名で収録してあります）。これもfooユーザで以下のように実行してください。

```
$ psql -f add.sql
```

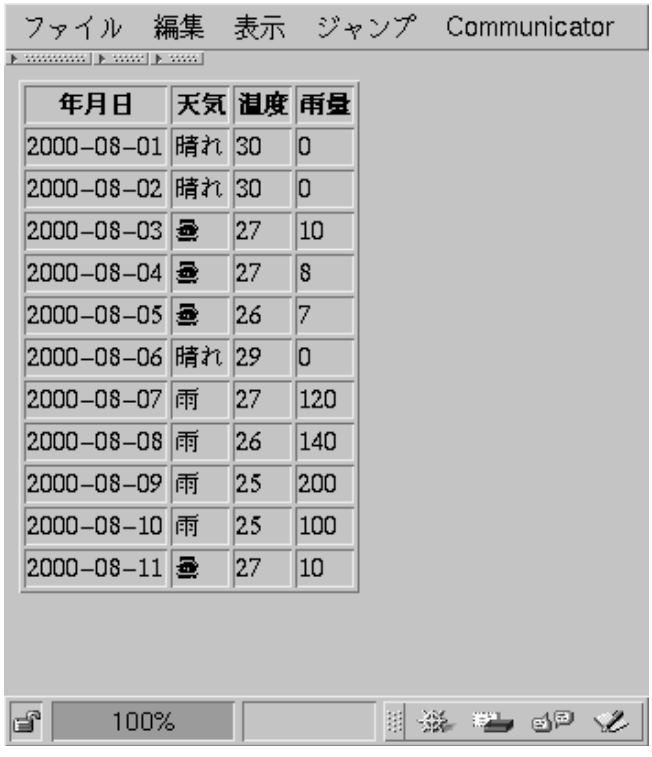
これで準備ができました。

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2-2**)。

図2-2　サンプルプログラム1の実行例



年月日	天気	温度	雨量
2000-08-01	晴れ	30	0
2000-08-02	晴れ	30	0
2000-08-03	曇	27	10
2000-08-04	曇	27	8
2000-08-05	曇	26	7
2000-08-06	晴れ	29	0
2000-08-07	雨	27	120
2000-08-08	雨	26	140
2000-08-09	雨	25	200
2000-08-10	雨	25	100
2000-08-11	曇	27	10

本章で紹介するサンプルプログラムはすべてCD-ROMの**articles/2/examples/**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
$ cp -r /mnt/cdrom/articles/2/examples .
```

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、ソースプログラムの先頭に書いてあります。何も書いていないプログラムは著作権を主張するほどのものではないということで、煮るなり焼くなりお好きなようにどうぞ。

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。

また、サンプルプログラムはブラウザに次のURLを入力して表示することができます。**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。

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  @$con = pg_connect("","","foo");        // データベースに接続する
 7  if ($con == false) {
 8    print("データベースに接続できませんでした。");
 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  $result = pg_exec($sql);                // selectを実行
14  if ($result == false) {
15    printf("SQL:¥"$sql¥"の実行に失敗しました。");
16    exit;
17  }
18  $rows = pg_numrows($result);            // 行数を取得
19  $columns = pg_numfields($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         $str = pg_fieldname($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       $str = pg_result($result,$j,$i);   // データの取り出し
36       print("<td>$str</td>");
37     }
38     print("</tr>¥n");
39   }
40
41   pg_freeresult($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。

1 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。pg_connectには複数の呼び出し形式がありますが、ここで使っているのはそのひとつで、

```
pg_connect($hostname, $port, $dbname)
```

の形式です。$hostname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。サンプルのように空文字列（""）を与えると、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。

$portはpostmasterが接続を受けつけているポート番号です。postmasterを起動するときに**-p**オプションを使って明示的にポート番号を指定している場合、その番号に合わせます。**-p**なしでpostmasterを起動している場合はデフォルトのポート番号（5432）になるので、その場合はこの引数に**5432**と指定するか、サンプルのように**""**を与えておきます。

$dbnameはデータベース名です。""と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。

これ以外のpg_connectの呼び出し形式には、

```
① pg_connect($hostname, $port, $options, $dbname)
② pg_connect($hostname, $port, $options, $tty, $dbname)
③ pg_connect($connect_string)
```

の三種類があります。$optionsはバックエンド（postgresというプログラム）を起動するときに渡すオプションを指定します。指定可能なオプションについてはPostgreSQL付属のオンラインマニュアル「postgres（1）」を参照してください。通常は""で問題ないはずです。

$ttyはバックエンドのログを取るときに使います。たとえば /tmp/pg_logというファイルを作り、$ttyに/tmp/pg_logを与えると、エラーが発生したときにこのファイルにエラーメッセージやトレースログが書き込まれます。

③の形式は、パスワード認証を行うときに使用するものです。これについては後述します。

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。

2 pg_exec（13行目）

pg_execは引数で与えられたSQL文を実行します。pg_execの呼び出し形式には二通りあり、一方は

```
pg_exec($con, SQL文字列)
```

のかたちをとります。$conはpg_connectの戻り値です。この形式は、同じスクリプトの中でpg_connectを複数呼び出し、異なるデータベースを同時に扱うときに使います。

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。

3 pg_numrows（18行目）

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

4 pg_numfields（19行目）

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

5 pg_fieldname（27行目）

pg_fieldnameは、pg_exec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。

6 pg_result（35行目）

pg_resultはpg_exec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。

7 pg_freeresult（41行目）

pg_freeresultはpg_exec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_exec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われるのでpg_freeresultをかならずしも呼ばなくてもよいのですが、続けてpg_execを使用する場合はかならずpg_freeresult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。

8 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呼ばなくても実害はないのですが、処理の終了を明確にするためにもpg_closeを使うように心がけましょう。

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。

今回はデータベースへの接続を管理するクラス**DbConnect**(ex1/dbconnect.ini)と、検索と結果の表示を行うクラス**PgSelect**(ex1/pgselect.ini)に分けてみました。

3.3.1 DbConnectクラス

DbConnectクラス(**リスト2-2**)はデータベースへの接続を管理します。コンストラクタ(7行目)が呼ばれると、getConnection->doConnectの順に呼び出しを行い、結局pg_connectを使ってPostgreSQLデータベースへの接続を行います。仕事が終わればdoClose(30行目)でデータベースへの接続を解除します。

データベース名、接続先ホスト名、ポート番号などのデータベースへの接続に必要なパラメータは、このクラスを継承した先で3～5行目のクラス変数を置き換えることによって指定します。

リスト2-2 DbConnectクラス(ex2/dbconnect.ini)

```
 1  <?php
 2  class DbConnect {
 3    var $dbname = "";        // データベース名(必要に応じて継承先でoverride)
 4    var $hostname = "";      // ホスト名(必要に応じて継承先でoverride)
 5    var $port = "";          // ポート番号(必要に応じて継承先でoverride)
 6    var $con = false;        // コネクションハンドル
 7
 8    function DbConnect() {  // コンストラクタ
 9         $this->getConnection();
10    }
11
12    // コネクションハンドルを返す
13    function getConnection() {
14      if ($this->con == false) {
15        return($this->doConnect());
16      }
17      return($this->con);
18    }
19
20    function doConnect() {
21      // データベースに接続する
22      @$this->con = pg_connect($this->hostname,$this->port,$this->dbname);
23      if ($this->con == false) {
24        print("データベース $this->dbname に接続できませんでした。");
25        exit;
```

```
26        }
27        return($this->con);
28      }
29
30      // データベースとの接続切断
31      function doClose() {
32        if ($this->con != false) {
33          pg_close($this->con);
34          $this->con = false;
35        }
36      }
37    }
38    ?>
```

3.3.2 PgSelectクラス

PgSelectクラス（**リスト2-3**）はデータベースに対して検索を行い、結果をHTMLのテーブルの形で表示します。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**）。

検索を実行するのはdoSelectです（18行目）。引数はSQL文です。

リスト2-3　PgSelectクラス(ex2/pgselect.ini)

```
 1   <?php
 2   class PgSelect {
 3     // テーブル開始タグの印字
 4     function printTableHeader() {
 5       print("<table border>¥n");
 6     }
 7
 8     // 列名の印字
 9     function printHeader($i, $str) {
10       print("<th>$str</th>");
11     }
12
13     // データの印字
14     function printData($i, $str) {
15       print("<td>$str</td>");
16     }
17
18     // 検索の実行
```

```
19     function doSelect($sql) {
20       @$result = pg_exec($sql);    // selectを実行
21       if ($result == false) {
22         printf("SQL:¥"$sql¥"の実行に失敗しました。");
23         exit;
24       }
25       $rows = pg_numrows($result);                    // 行数を取得
26       $columns = pg_numfields($result);               // 列数を取得
27
28       $this->printTableHeader();
29
30       for ($j = 0;$j < $rows;$j++) {
31         if ($j == 0) {
32                print("<tr>");
33                for ($i = 0;$i < $columns;$i++) {
34                  $str = pg_fieldname($result,$i);     // 列名の取り出し
35                  $this->printHeader($i, $str);
36                }
37                print("</tr>¥n");
38         }
39         print("<tr>");
40         for ($i = 0;$i < $columns;$i++) {
41                $str = pg_result($result,$j,$i);       // データの取り出し
42                $this->printData($i, $str);
43         }
44         print("</tr>¥n");
45       }
46       pg_freeresult($result);                         // 検索結果の解放
47       print("</table>¥n");
48     }
49   }
50   ?>
```

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

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**）。

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
 5   <?php
 6   //$include_path="/usr/local/apache/sample_php_lib";
 7   $include_path=".";
 8   require("$include_path/dbconnect.ini");
 9   require("$include_path/pgselect.ini");
10
11   class myDbConnect extends DbConnect {
12     var $dbname = "foo";           // データベース名
13   }
14
15   $d = new myDbConnect;
16   $sel = new PgSelect;
17   $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");
18   $d->doClose();
19   ?>
20   </body>
21   </html>
```

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。

7行目でクラスライブラリのファイルを格納したディレクトリをカレントディレクトリに指定しています。クラスライブラリファイルは拡張子が「.php」ではないため、普通にアクセスできる場所に置いておくとプログラムの内容が丸見えになり、セキュリティ上好ましくありません。かといって、拡張子を「.php」にすると不用意にプログラムが実行されてしまう可能性があります。したがって、実際には6行目のようにして、ブラウザがアクセスできない場所にクラスライブラリを置いておくほうがよいでしょう。

11行目でDbConnectを継承したmyDbConnectクラスを定義しています。ここではデータベース名を指定するだけで、そのほかの接続パラメータはデフォルトのままでよいので、データベース名だけを指定しています。

以上で準備ができたので、あとはインスタンスの生成（15行目～16行目）、検索の実行と結果表示（17行目）を行い、最後にデータベースへの接続を切断します（18行目）。

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.gif" 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。

ご覧のように、15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2-3**に示します。

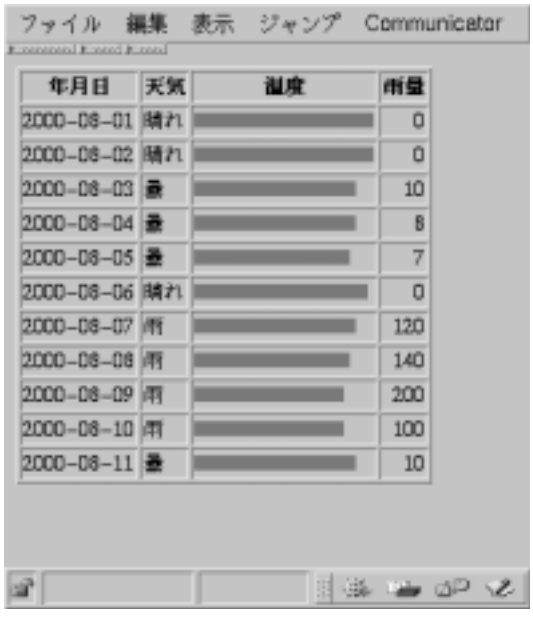
このように、クラスと継承を利用すれば、汎用性を失うことなく少ない変更でカスタマイズが可能になります。

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  //$include_path="/usr/local/apache/sample_php_lib";
 7  $include_path=".";
 8  require("$include_path/dbconnect.ini");
 9  require("$include_path/pgselect.ini");
10
11  class myDbConnect extends DbConnect {
12    var $dbname = "foo";                  // データベース名
13  }
14
```

```
15  class myPgSelect extends PgSelect {
16    // PgSelect クラスの printData()を override
17    function printData($i, $str) {
18      switch ($i) {
19      case 2:  // 温度は棒グラフで表示
20        $width = (int)$str * 4;
21        print("<td><img src=¥"red.gif¥" width=$width height=10></td>");
22        break;
23      case 3:  // 雨量は右詰めで表示
24        printf("<td align=¥"right¥">$str</td>");
25        break;
26      default: // その他は左詰めで表示
27        print("<td>$str</td>");
28        break;
29      }
30    }
31  }
32
33  $d = new myDbConnect;
34  $sel = new myPgSelect;
35  $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  $d->doClose();
37  ?>
38  </body>
39  </html>
```

図2-3　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難しいことなのです。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pg_closeを呼び出してバックエンド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メンバ変数$maxlを追加（3行目）。1ページに一度に表示する行数です。これを変更するときは、継承先のクラスで書き換えます。

②前のページ、次のページのどちらを表示するかを表す大域（グローバル）変数**$direction**を追加します。**prev**なら前、**next**なら次のページを表示する要求を表します。また、この変数自体が定義されていない場合は、初回の表示を表すものとします。

③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（6～7行目）。

④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（24～34行目）。この関数の中では、「前」や「次」を押したときに変数$directionに表示方向がセットされるように、URLに引数をつけています。リンク先は自分自身で、PHPでは**$PHP_SELF**が自分自身のURLを表す大域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/pgselect.ini)

```
 1  class PgSelect {
 2    // カスタマイズ可能メンバ変数
 3    var $maxl = 5;           // ページに一度に表示する行数
 4
 5    // 内部変数(カスタマイズ不可)
 6    var $usersql = "";       // ユーザ入力SELECT文
 7    var $offset = 0;         // 表示オフセット
 8
 9    // テーブル開始タグの印字
10    function printTableHeader() {
11      print("<table border>¥n");
12    }
13
14    // 列名の印字
15    function printHeader($i, $str) {
16      print("<th>$str</th>");
17    }
18
19    // データの印字
20    function printData($i, $str) {
21      print("<td>$str</td>");
22    }
23
```

```
24    //「前」の表示
25    function printPrev() {
26      global $PHP_SELF;
27      print("<a href=¥"$PHP_SELF?direction=prev¥">[前の $this->maxl 件に戻る]</a>");
28    }
29
30    //「次」の表示
31    function printNext($n) {
32      global $PHP_SELF;
33      print("<a href=¥"$PHP_SELF?direction=next¥">[次の $n 件に続く]</a>");
34    }
35
36    // 検索の実行
37    function doSelect($sql = "") {
38      global $direction;
39
40      if (!isset($direction)) {                    // はじめての表示？
41        $this->usersql = $sql;
42        $this->offset = 0;
43      } else {
44        if ($direction == "next") {
45         $this->offset += $this->maxl;
46        } else {
47         $this->offset -= $this->maxl;
48        }
49      }
50
51      // limit - offset 句の添付
52      $sql = $this->usersql . " LIMIT $this->maxl OFFSET $this->offset";
53
54      @$result = pg_exec($sql);                    // selectを実行
55      if ($result == false) {
56        printf("SQL:¥"$sql¥"の実行に失敗しました。");
57        exit;
58      }
59      $rows = pg_numrows($result);                 // 行数を取得
60      $columns = pg_numfields($result);            // 列数を取得
61
62      $this->printTableHeader();                   // テーブル開始タグの表示
63
64      for ($j = 0;$j < $rows;$j++) {
65        if ($j == 0) {
66         print("<tr>");
67         for ($i = 0;$i < $columns;$i++) {
68           $str = pg_fieldname($result,$i);        // 列名の取り出し
```

```
 69          $this->printHeader($i, $str);          // 列名の表示
 70        }
 71        print("</tr>¥n");
 72       }
 73
 74       print("<tr>");
 75       for ($i = 0;$i < $columns;$i++) {
 76        $str = pg_result($result,$j,$i);        // データの取り出し
 77        $this->printData($i, $str);             // データの表示
 78       }
 79       print("</tr>¥n");
 80     }
 81     pg_freeresult($result);                    // 検索結果の解放
 82
 83     print("</table>¥n");
 84
 85     if ($this->offset > 0) {
 86       $this->printPrev();
 87     }
 88
 89     $offset = $this->offset + $this->maxl;
 90     $sql = $this->usersql . " LIMIT $this->maxl OFFSET $offset";
 91
 92     @$result = pg_exec($sql);    // 次のページに表示するデータがあるか検索する
 93     if ($result == false) {
 94       printf("SQL:¥"$sql¥"の実行に失敗しました。");
 95       exit;
 96     }
 97     $n = pg_numrows($result);
 98
 99     if ($n > 0) {   // 次のページのデータあり？
100       $this->printNext($n);
101     }
102     pg_freeresult($result);                    // 検索結果の解放
103   }
104 }
105 ?>
```

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

⑤ メンバ変数の初期化

まず、② で述べた大域変数$direction をチェックし、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（40～42行目）。

そうでない場合は、$directionに設定された表示要求の種類にしたがってオフセットを増減します（44～48行目）。

⑥ LIMIT、OFFSET句の添付

ユーザから渡されたSELECT文の実行結果は、1ページに収まりきれ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52行目の

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

で得ることができます。

⑦ 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、⑥のSQL文と同じ方法で実現できます（89～101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（40～42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う前でなければならないからです。

次に、29行目で変数$directionをチェックして初回の起動かどうか調べます。PgSelectオブジェクトを生成するタイミングは初回だけだからです。そして初回ならばmyPgSelectオブジェクトを作成します。このとき念のためにsession_destroyを呼び出して不要なセッショ

ン情報を破棄します。2回目以降の起動、すなわち「前」あるいは「次」を押して表示される場合にはmyPgSelectクラスによって$directionがセットされるため、2回目以降はここを通りません。

37行目のsession_registerは、引数で与えた変数名をセッション管理の対象とすることを宣言します。なお、session_registerの呼び出しは初回だけでなく、毎回必要です。

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。

リスト2-7　サンプルプログラム4(ex4/ex4.php)

```
 1  <?php
 2  //$include_path="/usr/local/apache/sample_php_lib";
 3  $include_path=".";
 4  require("$include_path/dbconnect.ini");
 5  require("$include_path/pgselect.ini");
 6
 7  class myDbConnect extends DbConnect {
 8    var $dbname = "foo";           // データベース名
 9  }
10
11  class myPgSelect extends PgSelect {
12    // PgSelect クラスの printData()を override
13    function printData($i, $str) {
14      switch ($i) {
15      case 2:  // 温度は棒グラフで表示
16        $width = (int)$str * 4;
17        print("<td><img src=¥"red.gif¥" width=$width height=10></td>");
18        break;
19      case 3:  // 雨量は右詰めで表示
20        printf("<td align=¥"right¥">$str</td>");
21        break;
22      default: // その他は左詰めで表示
23        print("<td>$str</td>");
24        break;
25      }
26    }
27  }
28
29  if (!isset($direction)) {        // はじめての表示?
30    @session_destroy();            // セッション情報を破棄
31    $sel = new myPgSelect;         // myPgSelectオブジェクト作成
32  }
33
34  // myPgSelectオブジェクトをセッション管理することを宣言
```

```
35   // まだセッションが開始されていなければ、暗黙的にセッションを
36   // 開始する
37   session_register("sel");
38   ?>
39
40   <html>
41   <head><title>Example 4</title></head>
42   <body>
43
44   <?php
45   $d = new MyDbConnect;
46   $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");
47   $d->doClose();
48   ?>
49   </body>
50   </html>
```

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こ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、以下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。

なお、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であるかのように扱います。

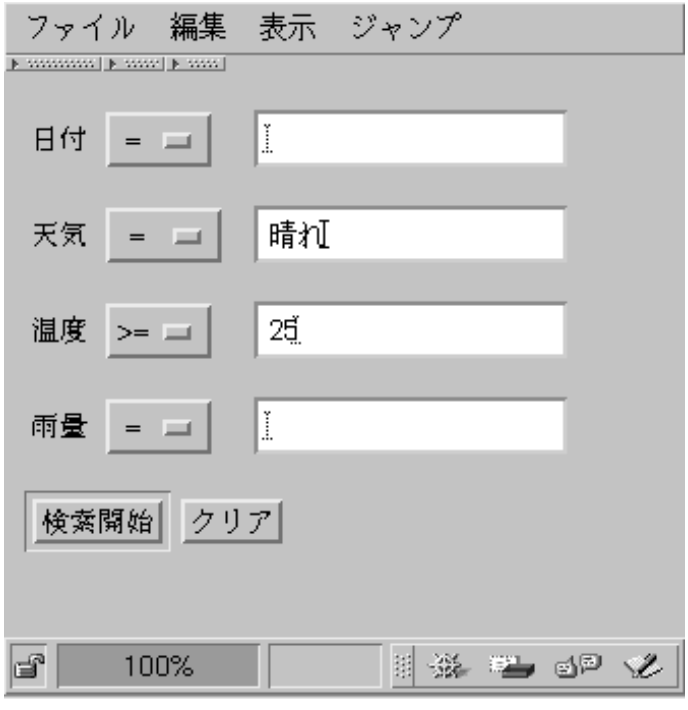
3.6 検索フォーム

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2-4 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。

```
SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
a.attnotnull, a.atthasdef FROM pg_class c, pg_attribute a, pg_type t
WHERE c.relname = 'otenki'
AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
ORDER BY attnum;

 attname | typname | attlen | atttypmod | attnotnull | atthasdef
---------+---------+--------+-----------+------------+-----------
 uryou   | int4    |      4 |        -1 | f          | f
 ondo    | int4    |      4 |        -1 | f          | f
 tenki   | text    |     -1 |        -1 | f          | f
 day     | date    |      4 |        -1 | t          | f
(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/pgmetadata.ini：**リスト2-9**）です。DbConnectクラス、PgSelectクラスと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)

```
<?php
//$include_path="/usr/local/apache/sample_php_lib";
$include_path=".";
require("$include_path/dbconnect.ini");
require("$include_path/pgselect.ini");
require("$include_path/pgmetadata.ini");

class myDbConnect extends DbConnect {
  var $dbname = "foo";           // データベース名
}

class myPgSelect extends PgSelect {
}

class myPgMetaData extends PgMetaData {
  var $table_name = "otenki";    // テーブル名
  var $sort_column = "day";      // 日付でソート
  var $aliases = array("day"=>"日付","tenki"=>"天気",
       "ondo"=>"温度","uryou"=>"雨量");
}

$d = new myDbConnect;

if (!isset($mode)) {             // はじめての表示?
  @session_destroy();            // セッション情報を破棄
  $sel = new myPgSelect;         // myPgSelectオブジェクト作成
  $meta = new myPgMetaData;      // myPgmetaDataオブジェクト作成
  $mode = "search";              // 初期表示モード設定(検索モード表示)
}

```

```
31   // セッション変数を登録
32   session_register("sel");
33   session_register("meta");
34   ?>
35
36   <html>
37   <head><title>Example 5</title></head>
38   <body>
39
40   <?php
41   if ($mode == "search" ) {
42     $meta->printForm();                    //検索フォームの表示
43   } else if ($mode == "show") {
44     $sel->doSelect($meta->makeSQL());     // 検索結果の表示
45     print("<br><a href=¥"$PHP_SELF?mode=search¥">検索フォームに戻る</a>¥n");
46   }
47
48   $d->doClose();
49   ?>
50   </body>
51   </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（15～20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2-4**および**図2-5**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2-6**および**図2-7**のように列名がそのまま使われます。

図 2-5　$aliasesを設定した検索結果

図 2-6　$aliasesを設定しない検索フォーム

図 2-7　$aliasesを設定しない検索結果

表 2-4　$mode変数の値と処理

$mode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

● $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います。これらを区別するために**$mode**という大域変数を導入します。この変数の状態により、以下の処理を行います（**表2-4**）。

まず$modeが未定義の場合は、myPgSelectオブジェクトとmyPgMetaDataオブジェクトを生成します。また、$mode変数を初期設定します（24～29行目）。

$modeが**search**の場合、検索条件入力用のフォームを表示します（42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。

$modeが**show**の場合は検索結果を表示します（44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。

3.6.4 PgMetaDataクラス

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

```
 1  <?php
 2  class PgMetaData {
 3    // カスタマイズ可能メンバ変数
 4    var $aliases = "";                    // カラムの別名連想配列([列名][別名])
 5    var $is_print_type_name = false;      // データ型名の表示有無
 6    var $is_print_opr_desc = false;       // オペレータの説明の表示有無
 7    var $table_name;                      // テーブル名
 8    var $sort_column = "";                // ソートをする列名
 9
10    // 内部変数(カスタマイズ不可)
11    var $md;   // テーブルのメタデータ([カラム番号][メタデータ]の2次元配列)
12    /* メタデータは以下：
13       name: 列名
14       typename: データ型名
15       len: 内部データ長(バイト数,可変長データは-1)
16       modifier: 型修飾子
17       notnull: NOT NULLなら t そうでないなら f
18       defaultval: デフォルト値: なければf
19       opinfo: オペレータ情報
20       */
21
22    var $op;   // オペレータ情報
23    /* オペレータ情報は以下
24       [型名][オペレータ情報番号][オペレータ名]
25       [型名][オペレータ情報番号][オペレータに関する説明]
26       */
27
28    // コンストラクタ
29    function PgMetaData() {
30      $this->getMetaData();
31      $this->getOperator();
32    }
33
```

```
34    // テーブルに関する情報の取得
35    function getMetaData() {
36      $sql = <<< EOF
37        SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
38        a.attnotnull, a.atthasdef
39        FROM pg_class c, pg_attribute a, pg_type t
40        WHERE c.relname = '$this->table_name'
41        AND a.attnum > 0 AND a.attrelid = c.oid AND a.atttypid = t.oid
42        ORDER BY a.attnum
43  EOF;
44
45      $result = pg_exec($sql);
46      if ($result == false) {
47        exit;
48      }
49      $rows = pg_numrows($result);
50      for ($i=0;$i<$rows;$i++) {
51        $this->md[$i][name] = pg_result($result,$i,"attname");
52        $this->md[$i][typename] = pg_result($result,$i,"typname");
53        $this->md[$i][len] = pg_result($result,$i,"attlen");
54        $this->md[$i][modifier] = pg_result($result,$i,"atttypmod");
55        $this->md[$i][notnull] = pg_result($result,$i,"attnotnull");
56        $this->md[$i][defaultval] = pg_result($result,$i,"atthasdef");
57      }
58      pg_freeresult($result);
59    }
60
61    // 型に関するオペレータ情報の取得
62    function getOperator() {
63      $n = count($this->md);
64      for ($j=0;$j<$n;$j++) {
65        $typename = $this->md[$j][typename];
66        if (!isset($this->op[typename])) {
67         $sql = <<< EOF
68           SELECT o.oprname AS op,t1.typname AS left_arg,
69           t2.typname AS right_arg, t0.typname AS result,
70           obj_description(p.oid) as description
71           FROM pg_proc p, pg_type t0, pg_type t1, pg_type t2,pg_operator o
72           WHERE t1.typname = '$typename'
73           AND p.prorettype = t0.oid AND RegprocToOid(o.oprcode) = p.oid
74           AND p.pronargs = 2 AND o.oprleft = t1.oid AND o.oprright = t2.oid
75           AND t0.typname = 'bool' AND t1.typname = t2.typname
76  EOF;
77         $result = pg_exec($sql);
78         if ($result == false) {
79           exit;
80         }
81         $rows = pg_numrows($result);
82         for ($i=0;$i<$rows;$i++) {
```

```
 83            $this->op[$typename][$i][name] = pg_result($result,$i,"op");
 84            $this->op[$typename][$i][desc] = pg_result($result,$i,"description");
 85          }
 86          pg_freeresult($result);
 87         }
 88       }
 89     }
 90
 91     // <form action=...><tabl> の表示
 92     function printFormHeader() {
 93       global $PHP_SELF;
 94       $str = <<< EOF
 95         <form action="$PHP_SELF?mode=show" method="post">
 96         <table>
 97 EOF;
 98       print($str);
 99     }
100
101     // </table><input type=submit...></form> の表示
102     function printFormFooter() {
103       $str = <<< EOF
104         </table>
105         <input type="submit" value="検索開始">
106         <input type="reset" value="クリア">
107         </form>
108 EOF;
109       print($str);
110     }
111
112     // 列名の表示
113     function printAttrName($atrnumber, $atrname) {
114       if ($this->is_print_type_name == false) {
115         print("<td>$atrname</td>");
116       } else {
117         $typename = $this->md[$atrnumber][typename];
118         print("<td>$atrname($typename)</td>");
119       }
120     }
121
122     // オペレータの表示
123     // 引数: カラム番号
124     function printOprName($n) {
125       $name = $this->md[$n][name];
126       $typename = $this->md[$n][typename];
127       print("<td><select name=¥"oplist[$name]¥">¥n");
128       printf("<option selected value=¥"%s¥">%s¥n",
129            $this->op[$typename][0][name],
130            htmlspecialchars($this->op[$typename][0][name]));
131
```

```
132       $n = count($this->op[$typename]);
133
134       for ($i=0;$i<$n;$i++) {
135         if ($this->is_print_opr_desc == true) {
136          printf("<option value=¥"%s¥">%s(%s)¥n",
137                 $this->op[$typename][$i][name],
138                 htmlspecialchars($this->op[$typename][$i][name]),
139                 htmlspecialchars($this->op[$typename][$i][desc]));
140         } else {
141          printf("<option value=¥"%s¥">%s¥n",
142                 $this->op[$typename][$i][name],
143                 htmlspecialchars($this->op[$typename][$i][name]));
144         }
145       }
146       print("</select></td>¥n");
147     }
148
149     // 検索フォームの生成
150     function printForm() {
151       if (is_array($this->md) == false) {
152         return;
153       }
154
155       // <form action=...><tabl> の表示
156       $this->printFormHeader();
157
158       $n = count($this->md);
159       for ($i=0;$i<$n;$i++) {
160
161         print("<tr>");
162
163         $name = $this->md[$i][name];
164         if (is_array($this->aliases)) {
165          $atrname = $this->aliases[$name];
166         } else {
167          $atrname = $name;
168         }
169         $this->printAttrName($i, $atrname);
170         $this->printOprName($i);
171
172         print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
173         print("</tr>¥n");
174       }
175       $this->printFormFooter();
176     }
177
178     // SQL文の生成
179     function makeSQL() {
180       global $atrlist;      // フォームからの入力データ(検索キー)
```

```
181      global $oplist; // フォームからの入力データ(オペレータ)
182
183      if (!isset($atrlist)) {
184        return;
185      }
186
187      $sql = "SELECT ";
188      $where = "WHERE";
189
190      $first = true;
191      $needComma = false;
192      while (list($name, $val) = each($atrlist)) {
193        if ($needComma == true) {
194         $sql .= ",";
195        }
196
197        $sql .= "$name ";
198        if (is_array($this->aliases)) {
199         $alias = $this->aliases[$name];
200         $sql .= "AS $alias";
201        }
202        if ($val != "") {
203         $op = $oplist[$name];
204         if ($first == false) {
205           $where .= " AND";
206         }
207         $where .= " $name $op '$val'";
208         $first = false;
209        }
210        $needComma = true;
211      }
212
213      $sql .= " FROM $this->table_name ";
214
215      if ($where != "WHERE") {
216        $sql .= " ";
217        $sql .= $where;
218      }
219
220      if ($this->sort_column != "") {
221        $sql .= " ORDER BY ";
222        $sql .= $this->sort_column;
223      }
224      return($sql);
225    }
226  }
227  ?>
```

コンストラクタ(28～32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が行います。まずgetMetaDataから見ていきましょう。

● getMetaData(34～57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。

36行目の書式は「ヒアドキュメント」と呼ばれるもので、EOF～EOFの文字列を$sqlに代入します。1行の長い文字列で書いたり、文字列連結演算子を使うよりも見やすくなります。

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。なお、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true、そうでなければfalse
defaultval	デフォルト値。なければfalse

● getOperator(61～89行目)

getOperatorは型にかん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(149～176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（156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（169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（170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により変換を行っています。

●データ入力欄（172行目）

これはとく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（175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(178～226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$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global宣言された$atrlist、$oplistです（180～181行目）。

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。

192行目からのwhile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（197行目）、別名が設定されていれば**AS 別名**を

付加します（198～201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（213行目）、必要ならばWHERE句を添付します（215～218行目）。ソート指定がある場合はORDER BY句を追加して（220～223行目）SELECT文のできあがりです。

3.6.5 PgSelectクラスの変更

ところで、PgSelecクラスにも若干の変更が必要です。次ページや前ページを表示したときに、$mode変数を引き渡す必要があるからです。といっても変更はわずかで、「前」を表示するprintPrevと「次」を表示するprintNextをそれぞれ1行変えて$mode変数をセットするだけです。

リスト2-10 PgSelectクラスの変更部分(ex5/pgselect.ini)

```
44     // 「前」の表示
45     function printPrev() {
46       global $PHP_SELF;
47       print("<a href=¥"$PHP_SELF?mode=show&direction=prev¥">
         [前の $this->maxl 件に戻る]</a>");
48     }
49
50     // 「次」の表示
51     function printNext($n) {
52       global $PHP_SELF;
53       print("<a href=¥"$PHP_SELF?mode=show&direction=next¥">
         [次の $n 件に続く]</a>");
54     }
```

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。これまで使ってきたotenkiテーブルの定義には制約が含まれていませんでした。ここで制約を盛り込んで、改めてotenkiテーブルを定義し直しましょう。

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、負の雨量はあり得ませんが、これ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

値のチェックと直接の関連はありませんが、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テーブルの定義例(改訂版)

では、以上を考慮してotenkiテーブルを定義し直してみましょう。

```
-- 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'))
);
```

以下の制約とデフォルト値が設定されています。

表2-7 今回定義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
その他	天気が晴れなら雨量は0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2-4**）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2-8 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PgMetaDataの変更、追加部分を示します。

リスト2-11 PgMetaDataクラスの変更部分(ex6/pgmetadata.ini)

```
28     var $table_oid;                // テーブルの Oid
29     var $table_constraints = "";   // テーブル制約


34     function PgMetaData() {
35       $this->getMetaData();        // テーブルに関する情報の取得
36       $this->getOperator();        // オペレータに関する情報の取得
37       $this->getConstraint();      // 制約に関する情報の取得
38       $this->getDefault();         // デフォルト値に関する情報の取得
39     }
40
41     // テーブルに関する情報の取得
42     function getMetaData() {
43       $sql = <<< EOF
44         SELECT a.attnum, a.attname, t.typname, a.attlen, a.atttypmod,
45         a.attnotnull, a.atthasdef, c.oid
46         FROM pg_class c, pg_attribute a, pg_type t
47         WHERE c.relname = '$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;
```

```
 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      $result = pg_exec($sql);
105      if ($result == false) {
106        $msg = sprintf("テーブル %s の制約値が取れません。",$table);
107        print("$msg");
108        exit;
109      }
110      $rows = pg_numrows($result);
111
112      // 制約無し
113      if ($rows == 0) {
114        pg_freeresult($result);
115        return;
116      }
117
118      // 連想配列に制約名と制約値を格納
119      for ($i=0;$i<$rows;$i++) {
120        $constraints[$i][rcname] = pg_result($result,$i,"rcname"); // 制約名
121        $constraints[$i][rcsrc] = pg_result($result,$i,"rcsrc");   // 制約
122      }
123      pg_freeresult($result);
124
125      for ($i=0;$i<count($this->md);$i++) {
126        // 列制約の名前は「テーブル名_列名」
127        $colname = $this->table_name . "_" . $this->md[$i][name];
128        for ($j=0;$j<count($constraints);$j++) {
129         if ($constraints[$j][rcname] == $colname) {
130           $this->md[$i][constraint] = $constraints[$j][rcsrc];
131           $constraints[$j][rcname] = ""; // チェック済の制約名を無効にする
132           break;
133         }
134        }
135      }
136
137      // 列制約をすべて無効にし、残ったのがテーブル制約
138      for ($j=0;$j<count($constraints);$j++) {
139        $rcname = $constraints[$j][rcname];
140        if ($rcname != "") {
141         $this->table_constraints[$rcname] = $constraints[$j][rcsrc];
142        }
143      }
144    }
145
```

```
146    // デフォルト値の取得
147    function getDefault() {
148      $sql = <<< EOF
149        SELECT adsrc, adnum FROM pg_attrdef WHERE adrelid = $this->table_oid
150  EOF;
151      $result = pg_exec($sql);
152      if ($result == false) {
153        print("テーブル $this->table_name のデフォルト値が取れません。");
154        exit;
155      }
156      $rows = pg_numrows($result);
157
158      // デフォルト値無し
159      if ($rows == 0) {
160        pg_freeresult($result);
161        return;
162      }
163
164      for ($i=0;$i<$rows;$i++) {
165        // adnumは列番号+1
166        $adnum = pg_result($result,$i,"adnum");
167        $this->md[$adnum-1][defaultval] = pg_result($result,$i,"adsrc");
168      }
169      pg_freeresult($result);
170    }


262    // --------------- データ登録用メンバ関数 ----------------------------
263    // <form action=...><tabl> の表示
264    function printDataInputFormHeader() {
265      global $PHP_SELF;
266      print("<form action=¥"$PHP_SELF?mode=insertExec¥" method=¥"post¥">¥n");
267      print("<table border>¥n");
268      print("<tr><th>カラム</th><th>データ型</th><th>データ</th>");
269      print("<th>NOT NULL</th><th>初期値</th><th>制約</th>¥n");
270    }
271
272    // </table><input type=submit...></form> の表示
273    function printDataInputFormFooter() {
274      print("</table>¥n");
275
276      if (is_array($this->table_constraints)) {
277        print("<table border>¥n");
278        print("<tr><th>テーブル制約名</th><th>制約</th></tr>¥n");
279        while (list($name, $val) = each($this->table_constraints)) {
280         print("<tr><tr><td>$name</td><td>$val</td></tr>¥n");
281        }
```

```
282       }
283       print("</table>¥n");
284       print("<input type=¥"submit¥" value=¥"登録¥">¥n");
285       print("<input type=¥"reset¥" value=¥"クリア¥">¥n");
286       print("</form>¥n");
287     }
288
289     // データ入力フォームの生成
290     function printDataInputForm() {
291       if (is_array($this->md) == false) {
292         return;
293       }
294
295       // <form action=...><tabl> の表示
296       $this->printDataInputFormHeader();
297
298       $n = count($this->md);
299       for ($i=0;$i<$n;$i++) {
300
301         print("<tr>");
302
303         $name = $this->md[$i]["name"];
304         if (is_array($this->aliases)) {
305          $atrname = $this->aliases[$name] . "(" . $name . ")";
306         } else {
307          $atrname = $name;
308         }
309         print("<td>$atrname</td>¥n");
310
311         $typename = $this->md[$i][typename];
312         print("<td align=center>$typename</td>");
313
314         print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
315
316         if ($this->md[$i]["notnull"] == "t") {
317          print("<td align=center>不可</td>¥n");
318         } else {
319          print("<td align=center>
                 <input type=¥"checkbox¥" name=¥"nulllist[$name]¥" ");
320          print("></td>¥n");
321         }
322
323         if (isset($this->md[$i][defaultval])) {
324          $def = $this->md[$i][defaultval];
325           print("<td align=center>$def</td>¥n");
326         } else {
327           print("<td align=center>なし</td>¥n");
```

```
328         }
329
330         $constraint = $this->md[$i][constraint];
331         if ($constraint == false) {
332          $constraint = "なし";
333         }
334         print("<td align=center>$constraint</td>¥n");
335
336         print("</tr>¥n");
337       }
338       $this->printDataInputFormFooter();
339     }
340
392     // データ登録SQLの作成、実行
393     function insertSQL($user_check = false, $debug = false) {
394       global $atrlist;
395       global $nulllist;
396
397       $sql = "INSERT INTO $this->table_name (";
398       $atrs = "";
399       $vals = "";
400       $needComma = false;
401       $has_someval = false;
402
403       while (list($name, $val) = each($atrlist)) {
404         if ($nulllist[$name] == "" && $val == "") {
405          continue;    // 何も値が指定されていない
406         }
407
408         $has_someval = true;
409
410         if ($needComma == true) {
411          $atrs .= ",";
412          $vals .= ",";
413         }
414         $needComma = true;
415
416         $atrs .= "$name ";
417
418         if ($nulllist[$name] != "") {
419          $vals .= "NULL";    // NULLが指定された
420         } else if ($val != "") {
421          $vals .= "'$val'";  // 何か値が入力された
422         }
423       }
```

```
424
425     if ($user_check) {    // データチェックユーザ関数あり?
426       if ($user_check($atrlist) == false) {
427        return(false);
428       }
429     }
430
431     // 何も値が指定されていない場合はデフォルト値を入力する
432     if ($has_someval == false) {
433       $sql = "INSERT INTO $this->table_name DEFAULT VALUES";
434     } else {
435       $sql .= "$atrs) values ($vals)";
436     }
437
438     if ($debug == true) {
439       print($sql);
440     }
441     $rtn = pg_exec($sql);
442     return($rtn);
443   }
444 }
445 ?>
```

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(37～38行目)。まずここから見ていきましょう。

●getConstraint(99～144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。テーブルのOIDはgetMetaDataで発行するSQL文を変更し、$table_oidというメンバ変数に入れるようにしています(44～49行目)。

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(125～135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(137～143行目)。

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

● **getDefault(147～170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(167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(263～270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。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。

● **printDataInputFormFooter(272～287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。

● **printDataInputForm(289～339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われます。

●insertSQL（392～444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$atrlist（394行目）で、前節のサンプル同様連想配列になっており、$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$nulllistです（395行目）。これは$nulllist[列名]が""（空文字列）以外ならば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文ではだめなので、

```
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（433行目）。

insertSQLでは、SELECT文を作成するスクリプト同様while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。列名リストは$arts、値のリストは$valsにSQL文が完成したら、pg_execを使ってINSERT文を実行します。pg_exec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。

リスト2-12 サンプルプログラム6(ex6/ex6.php)

```
<?php
//$include_path="/usr/local/apache/sample_php_lib";
$include_path=".";
require("$include_path/dbconnect.ini");
require("$include_path/pgselect.ini");
require("$include_path/pgmetadata.ini");

class myDbConnect extends DbConnect {
  var $dbname = "foo";           // データベース名
}

class myPgSelect extends PgSelect {
}

class myPgMetaData extends PgMetadata {
  var $table_name = "otenki";    // テーブル名
  var $sort_column = "day";      // 日付でソート
  var $aliases = array("day"=>"日付","tenki"=>"天気",
       "ondo"=>"温度","uryou"=>"雨量");
}

$d = new myDbConnect;

if (!isset($mode)) {             // はじめての表示?
  @session_destroy();            // セッション情報を破棄
  $sel = new myPgSelect;         // myPgSelectオブジェクト作成
  $meta = new myPgMetaData;      // myPgmetaDataオブジェクト作成
}

// セッション変数を登録
session_register("sel");
session_register("meta");
?>

<html>
<head><title>Example 6</title></head>
<body>

<?php
if (!isset($mode) || $mode == "top") {
  $str = <<<EOF
<ul>
<li><a href="$PHP_SELF?mode=search">検索フォーム</a>
<li><a href="$PHP_SELF?mode=insert">登録フォーム</a>
</ul>
EOF;
```

```
47    print($str);
48
49  } else {
50    switch($mode) {
51    case "search":
52      $meta->printForm();                  //検索フォームの表示
53      break;
54    case "show":
55      $sel->doSelect($meta->makeSQL());  // 検索結果の表示
56      break;
57    case "insert":
58      $meta->printDataInputForm();         //登録フォームの表示
59      break;
60    case "insertExec":
61      $sts = $meta->insertSQL();           //登録の実行
62      if ($sts != false) {
63        print("正常にデータを登録しました。<br>¥n");
64      }
65      break;
66    }
67    $str = <<< EOF
68      <br><a href="$PHP_SELF?mode=search">検索フォームに戻る</a>
69      <br><a href="$PHP_SELF?mode=insert">登録フォームに戻る</a>
70      <br><a href="$PHP_SELF?mode=top">トップメニューに戻る</a>
71  EOF;
72    printf($str);
73  }
74
75  $d->doClose();
76  ?>
77  </body>
78  </html>
```

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（**図2-9**）。

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、50～66行目と見比べて使い方を理解してください（**表2-9**）。

図 2-9　サンプルプログラム 6

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
insert	登録フォームの表示
insertExec	登録の実行

3.8 データ更新／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。

OID による行の唯一性

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限らないため、この方法が適用できない場合があります。そこで **OID** を使うことにします。OID はどの行にもかならず設定され、しかもその唯一性が保証されているため、今回のような目的には最適です。

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限らないことです。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難しいため、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。

```
oid AS __target_oid__
```

たとえば、前述の例は、

```
$sel->doSelect("SELECT oid AS __target_oid__, 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_oid__**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[*5]。

●makeSQLを使用している場合

PgMetaDataクラスのmakeSQLに引数を追加し、**oid AS __target_oid__**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oid as __target_oid__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**oid AS __ target_oid__**が加わるようにします。

*5 このように__target_oid__という列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列名を使ってはいけません。

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_oid__**が含まれていたら（97行目）列名を表示せず、その代わりに何番目の列が**__target_oid__**だったかを記憶しておきます（98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（116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（56～60行）。中身は非常に簡単で、

```
function printUpdateTag($oid) {
  global $PHP_SELF;
  print("<td><a href=¥"$PHP_SELF?mode=update&oid=$oid¥">
         更新/削除</a></td>¥n");
}
```

のように定義されています。この関数はユーザがオーバライドでき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（$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、oidも渡すようにしておきます。

図2-10　更新機能を持つ検索結果

以上で**図2-10**のように、ex7/ex7.php（**リスト2-12**）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とex7.php（**リスト2-14**）の違いは、ex6.phpの55行目で、

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

だったのが、ex7.phpの587行目では

```
$sel->doSelect($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/pgselect.ini)

```
 1  <?php
 2  /*
 3   * Copyright (C) 2000  Tatsuo Ishii
 4   *
 5   * Permission to use, copy, modify, and distribute this software and
 6   * its documentation for any purpose and without fee is hereby
 7   * granted, provided that the above copyright notice appear in all
 8   * copies and that both that copyright notice and this permission
 9   * notice appear in supporting documentation, and that the name of the
10   * author not be used in advertising or publicity pertaining to
11   * distribution of the software without specific, written prior
12   * permission. The author makes no representations about the
13   * suitability of this software for any purpose.  It is provided "as
14   * is" without express or implied warranty.
15   */
16
17  /*
18   * 検索結果をテーブル形式で表示する
19   */
20
21  class PgSelect {
22    // カスタマイズ可能メンバ変数
23    var $maxl = 5;           // ページに一度に表示する行数
24
25    // 内部変数(カスタマイズ不可)
26    var $usersql = "";       // ユーザ入力SELECT文
27    var $offset = 0;         // 表示オフセット
28
29    // テーブル開始タグの印字
30    function printTableHeader() {
31      print("<table border>¥n");
32    }
33
```

```
34      // 列名の印字
35      function printHeader($i, $str) {
36        print("<th>$str</th>");
37      }
38
39      // データの印字
40      function printData($i, $str) {
41        print("<td>$str</td>");
42      }
43
44      // 「前」の表示
45      function printPrev() {
46        global $PHP_SELF;
47        print("<a href=¥"$PHP_SELF?mode=show&direction=prev¥">
                [前の $this->maxl 件に戻る]</a>");
48      }
49
50      // 「次」の表示
51      function printNext($n) {
52        global $PHP_SELF;
53        print("<a href=¥"$PHP_SELF?mode=show&direction=next¥">
                [次の $n 件に続く]</a>");
54      }
55
56      // 更新/削除タグの表示
57      function printUpdateTag($oid) {
58        global $PHP_SELF;
59        print("<td><a href=¥"$PHP_SELF?mode=update&oid=$oid¥">
                更新/削除</a></td>¥n");
60      }
61
62      // 検索の実行
63      function doSelect($sql = "") {
64        global $direction;
65
66        if (!isset($direction)) {    // はじめての表示？
67          $this->usersql = $sql;
68          $this->offset = 0;
69        } else {
70          if ($direction == "next") {
71           $this->offset += $this->maxl;
72          } else {
73           $this->offset -= $this->maxl;
74          }
75        }
```

```
 76
 77      // limit - offset 句の添付
 78      $sql = $this->usersql . " LIMIT $this->maxl OFFSET $this->offset";
 79
 80      @$result = pg_exec($sql);                  // selectを実行
 81      if ($result == false) {
 82        printf("SQL:¥"$sql¥"の実行に失敗しました。");
 83        exit;
 84      }
 85      $rows = pg_numrows($result);               // 行数を取得
 86      $columns = pg_numfields($result);          // 列数を取得
 87
 88      $this->printTableHeader();                 // テーブル開始タグの表示
 89
 90      $is_updatable = -1;
 91
 92      for ($j = 0;$j < $rows;$j++) {
 93        if ($j == 0) {
 94         print("<tr>");
 95         for ($i = 0;$i < $columns;$i++) {
 96           $str = pg_fieldname($result,$i);      // 列名の取り出し
 97           if ($str == "__target_oid__") {       // 更新キー？
 98             $is_updatable = $i;
 99           } else {
100             $this->printHeader($i, $str);       // 列名の表示
101           }
102         }
103         print("</tr>¥n");
104        }
105
106        print("<tr>");
107        for ($i = 0;$i < $columns;$i++) {
108         $str = pg_result($result,$j,$i);        // データの取り出し
109         if ($is_updatable == $i) {
110           $oid = $str;
111         } else {
112           $this->printData($i, $str);           // データの表示
113         }
114        }
115        if ($oid != false) {
116         $this->printUpdateTag($oid);
117        }
118        print("</tr>¥n");
119      }
120      pg_freeresult($result);                    // 検索結果の解放
```

```
121
122      print("</table>¥n");
123
124      if ($this->offset > 0) {
125        $this->printPrev();
126      }
127
128      $offset = $this->offset + $this->maxl;
129      $sql = $this->usersql . " LIMIT $this->maxl OFFSET $offset";
130
131      @$result = pg_exec($sql);    // 次のページに表示するデータがあるか検索する
132      if ($result == false) {
133        printf("SQL:¥"$sql¥"の実行に失敗しました。");
134        exit;
135      }
136      $n = pg_numrows($result);
137
138      if ($n > 0) {                // 次のページのデータあり？
139        $this->printNext($n);
140      }
141      pg_freeresult($result);      // 検索結果の解放
142    }
143  }
144  ?>
```

3.8.3 更新／削除フォーム

次にアンカーを選択され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2-11**）。基本的にはデータ入力フォーム（**図2-8**）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」「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2-11 今回作成する更新/削除フォーム

PgMetaDataクラスのprintDataInputFormの改造

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

●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**input**ならばデータ登録、**update**ならば更新／削除用のフォームを表示することにします(322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(363～368行目)。そのために333～344行目で前もってテーブルから現在行を取得しています。

●submit ボタン

更新ボタンと削除ボタンを表示します(392行目)。実際に処理を行うのはprintDataInputFormFooterです(297～317行目)。

どちらかのボタンを押されたときは$mode変数に**updateExec**という値がセットされますが、このままでは更新ボタンなのか削除ボタンなのか区別がつかないので、更新のときは$update_request、削除のときは$delete_requestという変数が渡るようにしておきます。

JavaScriptによる確認ダイアログ表示

これで一通り検索、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。

▶ 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はネットスケープ社とサン・マイクロシステムズ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1.3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 1.0	Netscape Navigator 2.0, Internet Explorer 3.0 以降
JavaScript 1.1	Netscape Navigator 3.0, Internet Explorer 4.0 以降
JavaScript 1.2	Netscape Navigator 4.0
JavaScript 1.3	Netscape Navigator 4.06 以降

なお、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を前提に説明します[*6]。

▶ 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

*6 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。

確認ダイアログをJavaScriptで作る

早速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/pgmetadata.iniの309行目付近）。

```
if ($mode == "inputForm") {
  print("<input type=¥"submit¥" value=¥"登録¥">¥n");
} else {
  print("<input type=¥"submit¥" name=¥"update¥" value=¥"更新¥"
     onClick=¥"return formConfirm('update')¥">¥n");
  print("<input type=¥"submit¥" name=¥"delete¥" value=¥"削除¥"
     onClick=¥"return formConfirm('delete')¥">¥n");
}
```

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、文字列を引数に取って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。

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の中身は以下の通りです。

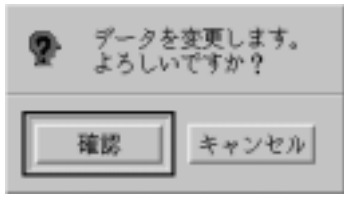
```
<script language="JavaScript1.1">
<!--
function formConfirm(type) {
  if (type == "update") {
    rtn = confirm("データを変更します。¥nよろしいですか？");
  } else {
    rtn = confirm("データを削除します。¥nよろしいですか？");
  }

  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*7（**図2-12**）。

*7 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。

図2-12　データ変更確認ダイアログ



更新処理

PgMetaDataクラスに更新／削除フォーム（**図2-11**）で「更新」を選択した場合の処理を行う関数を**updateSQL**として追加します（**リスト2-15**：506～547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、global宣言された以下の変数で取得しています。

$atrlist …… 列名、値のリスト
$nulllist …… NULLかどうかのフラグのリスト
$oid …… 更新対象の行のOID

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 oid = オブジェクトID;
```

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（$nulllist）をチェックしています。

削除処理

更新／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。

フォームから渡ってきたデータは、global宣言された以下の変数で取得しています。

$oid　　更新対象の行のOID

以上のデータから、delete用のSQL文を作って実行します。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 oid = オブジェクトID;
```

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（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(38行目)

前述したように <head>～</head> の中で

```
<?php include("jscripts.ini");?>
```

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
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

●更新／削除フォームの表示(69～71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(72～85行目)

$update_request変数が設定されていれば更新処理なので、updateSQLを実行します。そうでなければ削除処理deleteSQLを実行します。この$update_request変数は、pgmetadata.ini（**リスト2-15**）の312行目で設定されており、「更新」ボタンを押したときに<submit>タグが変数を作成します。

リスト2-14 サンプルプログラム7(ex7/ex7.php)

```
<?php
//$include_path="/usr/local/apache/sample_php_lib";
$include_path=".";
require("$include_path/dbconnect.ini");
require("$include_path/pgselect.ini");
require("$include_path/pgmetadata.ini");

class myDbConnect extends DbConnect {
  var $dbname = "foo";                    // データベース名
}

class myPgSelect extends PgSelect {
}

class myPgMetaData extends PgMetadata {
  var $table_name = "otenki";             // テーブル名
  var $sort_column = "day";               // 日付でソート
  var $aliases = array("day"=>"日付","tenki"=>"天気",
       "ondo"=>"温度","uryou"=>"雨量");
}

$d = new myDbConnect;

if (!isset($mode)) {                      // はじめての表示?
  @session_destroy();                     // セッション情報を破棄
  $sel = new myPgSelect;                  // myPgSelectオブジェクト作成
  $meta = new myPgMetaData;               // myPgmetaDataオブジェクト作成
}

// セッション変数を登録
session_register("sel");
session_register("meta");
?>

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

<?php
if (!isset($mode) || $mode == "top") {
  $str = <<<EOF
<ul>
<li><a href="$PHP_SELF?mode=search">検索フォーム</a>
<li><a href="$PHP_SELF?mode=insert">登録フォーム</a>
</ul>
EOF;
```

```
50      print($str);
51
52    } else {
53      switch($mode) {
54      case "search":
55        $meta->printForm();                          //検索フォームの表示
56        break;
57      case "show":
58        $sel->doSelect($meta->makeSQL(true));        // 検索結果の表示
59        break;
60      case "insert":
61        $meta->printDataInputForm("insert");         //登録フォームの表示
62        break;
63      case "insertExec":
64        $sts = $meta->insertSQL();                   //登録の実行
65        if ($sts != false) {
66          print("正常にデータを登録しました。<br>¥n");
67        }
68        break;
69      case "update":                                 // 更新フォームの表示
70        $meta->printDataInputForm("update");
71        break;
72      case "updateExec":                             // 更新/削除の実行
73        if (isset($update_request)) {
74          $sts = $meta->updateSQL();                 // 更新処理
75        } else {
76          $sts = $meta->deleteSQL();                 // 削除処理
77        }
78        if ($sts != false) {
79          if (isset($update_request)) {
80           print("更新処理が正常に終了しました。<br>¥n");
81          } else {
82           print("削除処理が正常に終了しました。<br>¥n");
83          }
84        }
85        break;
86      }
87      $str = <<< EOF
88        <br><a href="$PHP_SELF?mode=search">検索フォームに戻る</a>
89        <br><a href="$PHP_SELF?mode=insert">登録フォームに戻る</a>
90        <br><a href="$PHP_SELF?mode=top">トップメニューに戻る</a>
91    EOF;
92      printf($str);
93    }
94
95    $d->doClose();
96    ?>
97    </body>
98    </html>
```

リスト2-15 PgMetaDataクラス(ex7/pgmetadata.ini)

```
<?php
/*
 * Copyright (C) 2000  Tatsuo Ishii
 *
 * Permission to use, copy, modify, and distribute this software and
 * its documentation for any purpose and without fee is hereby
 * granted, provided that the above copyright notice appear in all
 * copies and that both that copyright notice and this permission
 * notice appear in supporting documentation, and that the name of the
 * author not be used in advertising or publicity pertaining to
 * distribution of the software without specific, written prior
 * permission. The author makes no representations about the
 * suitability of this software for any purpose.  It is provided "as
 * is" without express or implied warranty.
 */
class PgMetaData {
  // 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

  // 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: 列名
     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
  var $table_constraints = "";          // テーブル制約


  // --------------- コンストラクタ関連関数 ----------------------------
  // コンストラクタ
```

```
48     function PgMetaData() {
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       $result = pg_exec($sql);
67       if ($result == false) {
68         exit;
69       }
70       $rows = pg_numrows($result);
71       $this->table_oid = pg_result($result,0,"oid");
72       for ($i=0;$i<$rows;$i++) {
73         $this->md[$i][name] = pg_result($result,$i,"attname");
74         $this->md[$i][typename] = pg_result($result,$i,"typname");
75         $this->md[$i][len] = pg_result($result,$i,"attlen");
76         $this->md[$i][modifier] = pg_result($result,$i,"atttypmod");
77         $this->md[$i][notnull] = pg_result($result,$i,"attnotnull");
78         $this->md[$i][defaultval] = pg_result($result,$i,"atthasdef");
79       }
80       pg_freeresult($result);
81     }
82
83     // 型に関するオペレータ情報の取得
84     function getOperator() {
85       $n = count($this->md);
86       for ($j=0;$j<$n;$j++) {
87         $typename = $this->md[$j][typename];
88         if (!isset($this->op[typename])) {
89          $sql = <<< EOF
90           SELECT o.oprname AS op,t1.typname AS left_arg,
91           t2.typname AS right_arg, t0.typname AS result,
92           obj_description(p.oid) as description
93           FROM pg_proc p, pg_type t0, pg_type t1, pg_type t2,pg_operator o
94           WHERE t1.typname = '$typename'
```

```
 95          AND p.prorettype = t0.oid AND RegprocToOid(o.oprcode) = p.oid
 96          AND p.pronargs = 2 AND o.oprleft = t1.oid AND o.oprright = t2.oid
 97          AND t0.typname = 'bool' AND t1.typname = t2.typname
 98  EOF;
 99        $result = pg_exec($sql);
100        if ($result == false) {
101          exit;
102        }
103        $rows = pg_numrows($result);
104        for ($i=0;$i<$rows;$i++) {
105          $this->op[$typename][$i][name] = pg_result($result,$i,"op");
106          $this->op[$typename][$i][desc] = pg_result($result,$i,"description");
107        }
108        pg_freeresult($result);
109       }
110     }
111   }
112
113   // 制約に関する情報の取得
114   function getConstraint() {
115     $sql = <<< EOF
116       SELECT rcname,rcsrc FROM pg_relcheck WHERE rcrelid = '$this->table_oid'
117  EOF;
118     $result = pg_exec($sql);
119     if ($result == false) {
120       $msg = sprintf("テーブル %s の制約値が取れません。",$table);
121       print("$msg");
122       exit;
123     }
124     $rows = pg_numrows($result);
125
126     // 制約無し
127     if ($rows == 0) {
128       pg_freeresult($result);
129       return;
130     }
131
132     // 連想配列に制約名と制約値を格納
133     for ($i=0;$i<$rows;$i++) {
134       $constraints[$i][rcname] = pg_result($result,$i,"rcname"); // 制約名
135       $constraints[$i][rcsrc] = pg_result($result,$i,"rcsrc");   // 制約
136     }
137     pg_freeresult($result);
138
139     for ($i=0;$i<count($this->md);$i++) {
140       // 列制約の名前は「テーブル名_列名」
141       $colname = $this->table_name . "_" . $this->md[$i][name];
```

```
142         for ($j=0;$j<count($constraints);$j++) {
143          if ($constraints[$j][rcname] == $colname) {
144            $this->md[$i][constraint] = $constraints[$j][rcsrc];
145            $constraints[$j][rcname] = ""; // チェック済の制約名を無効にする
146            break;
147          }
148         }
149       }
150
151       // 列制約をすべて無効にし、残ったのがテーブル制約
152       for ($j=0;$j<count($constraints);$j++) {
153         $rcname = $constraints[$j][rcname];
154         if ($rcname != "") {
155          $this->table_constraints[$rcname] = $constraints[$j][rcsrc];
156         }
157       }
158     }
159
160     // デフォルト値の取得
161     function getDefault() {
162       $sql = <<< EOF
163         SELECT adsrc, adnum FROM pg_attrdef WHERE adrelid = $this->table_oid
164   EOF;
165       $result = pg_exec($sql);
166       if ($result == false) {
167         print("テーブル $this->table_name のデフォルト値が取れません。");
168         exit;
169       }
170       $rows = pg_numrows($result);
171
172       // デフォルト値無し
173       if ($rows == 0) {
174         pg_freeresult($result);
175         return;
176       }
177
178       for ($i=0;$i<$rows;$i++) {
179         // adnumは列番号+1
180         $adnum = pg_result($result,$i,"adnum");
181         $this->md[$adnum-1][defaultval] = pg_result($result,$i,"adsrc");
182       }
183       pg_freeresult($result);
184     }
185
186
187     // --------------- データ検索用フォーム関数 ----------------------------
188     // <form action=...><tabl> の表示
```

```
189   function printFormHeader() {
190     global $PHP_SELF;
191     $str = <<< EOF
192       <form action="$PHP_SELF?mode=show" method="post">
193       <table>
194 EOF;
195     print($str);
196   }
197
198   // </table><input type=submit...></form> の表示
199   function printFormFooter() {
200     $str = <<< EOF
201       </table>
202       <input type="submit" value="検索開始">
203       <input type="reset" value="クリア">
204       </form>
205 EOF;
206     print($str);
207   }
208
209   // 列名の表示
210   function printAttrName($atrnumber, $atrname) {
211     if ($this->is_print_type_name == false) {
212       print("<td>$atrname</td>");
213     } else {
214       $typename = $this->md[$atrnumber][typename];
215       print("<td>$atrname($typename)</td>");
216     }
217   }
218
219   // オペレータの表示
220   // 引数: カラム番号
221   function printOprName($n) {
222     $name = $this->md[$n][name];
223     $typename = $this->md[$n][typename];
224     print("<td><select name=¥"oplist[$name]¥">¥n");
225     printf("<option selected value=¥"%s¥">%s¥n",
226          $this->op[$typename][0][name],
227          htmlspecialchars($this->op[$typename][0][name]));
228
229     $n = count($this->op[$typename]);
230
231     for ($i=0;$i<$n;$i++) {
232       if ($this->is_print_opr_desc == true) {
233        printf("<option value=¥"%s¥">%s(%s)¥n",
234               $this->op[$typename][$i][name],
235               htmlspecialchars($this->op[$typename][$i][name]),
```

```
236                 htmlspecialchars($this->op[$typename][$i][desc]));
237         } else {
238          printf("<option value=¥"%s¥">%s¥n",
239                 $this->op[$typename][$i][name],
240                 htmlspecialchars($this->op[$typename][$i][name]));
241         }
242       }
243       print("</select></td>¥n");
244     }
245
246     // 検索フォームの生成
247     function printForm() {
248       if (is_array($this->md) == false) {
249         return;
250       }
251
252       // <form action=...><tabl> の表示
253       $this->printFormHeader();
254
255       $n = count($this->md);
256       for ($i=0;$i<$n;$i++) {
257
258         print("<tr>");
259
260         $name = $this->md[$i][name];
261         if (is_array($this->aliases)) {
262          $atrname = $this->aliases[$name];
263         } else {
264          $atrname = $name;
265         }
266         $this->printAttrName($i, $atrname);
267         $this->printOprName($i);
268
269         print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
270         print("</tr>¥n");
271       }
272       $this->printFormFooter();
273     }
274
275
276     // --------------- データ登録用メンバ関数 ----------------------------
277     // <form action=...><tabl> の表示
278     function printDataInputFormHeader($mode) {
279       global $PHP_SELF;
280       global $oid;
281
282       if ($mode == "insert") {
```

```
283         $op="insertExec";
284       } else {
285         $op = "updateExec";
286       }
287       $str = <<<EOF
288         <form action="$PHP_SELF?mode=$op&oid=$oid" method="post">
289         <table border>
290         <tr><th>カラム</th><th>データ型</th><th>データ</th>
291         <th>NULL</th><th>初期値</th><th>制約</th></tr>
292   EOF;
293       printf($str);
294     }
295
296     // </table><input type=submit...></form> の表示
297     function printDataInputFormFooter($mode) {
298       print("</table>¥n");
299
300       if (is_array($this->table_constraints)) {
301         print("<table border>¥n");
302         print("<tr><th>テーブル制約名</th><th>制約</th></tr>¥n");
303         while (list($name, $val) = each($this->table_constraints)) {
304          print("<tr><tr><td>$name</td><td>$val</td></tr>¥n");
305         }
306       }
307       print("</table>¥n");
308
309       if ($mode == "insert") {
310         print("<input type=¥"submit¥" value=¥"登録¥">¥n");
311       } else {
312         print("<input type=¥"submit¥" name=¥"update_request¥"
                    value=¥"更新¥" onClick=¥"return formConfirm('update')¥">¥n");
313         print("<input type=¥"submit¥" name=¥"delete_request¥" value=¥"削除¥"
                    onClick=¥"return formConfirm('delete')¥">¥n");
314       }
315       print("<input type=¥"reset¥" value=¥"クリア¥">¥n");
316       print("</form>¥n");
317     }
318
319     // データ入力フォームの生成
320     // mode: "input" : データ登録
321     //       "update": 変更/削除
322     function printDataInputForm($mode) {
323       global $oid;           // 現在行のOID。updateモードの時のみ使用
324
325       if (is_array($this->md) == false) {
326         return;
327       }
```

```
328
329       // <form action=...><tabl> の表示
330       $this->printDataInputFormHeader($mode);
331
332       // 更新フォームのときは現在のテーブルの値を取得する
333       if ($mode == "update") {
334         $sql = "SELECT * FROM $this->table_name WHERE oid = $oid";
335         @$result = pg_exec($sql);
336         if ($result == false) {
337          printf("SQL:¥"$sql¥"の実行に失敗しました。");
338          exit;
339         }
340         if (pg_numrows($result) != 1) {
341          print("テーブルの現在の値を取得できません。");
342          exit;
343         }
344       }
345
346       $n = count($this->md);
347       for ($i=0;$i<$n;$i++) {
348
349         print("<tr>");
350
351         $name = $this->md[$i]["name"];
352         if (is_array($this->aliases)) {
353          $atrname = $this->aliases[$name] . "(" . $name . ")";
354         } else {
355          $atrname = $name;
356         }
357         print("<td>$atrname</td>¥n");
358
359         $typename = $this->md[$i][typename];
360         print("<td align=center>$typename</td>");
361
362         // 更新フォームのときは現在のテーブルの値を取得する
363         if ($mode == "update") {
364          $str = htmlspecialchars(pg_result($result,0,$this->md[$i][name]));
365          print("<td><input type=¥"text¥" value=¥"$str¥" name=¥"atrlist[$name]¥"></td>¥n");
366         } else {
367          print("<td><input type=¥"text¥" name=¥"atrlist[$name]¥"></td>¥n");
368         }
369
370         if ($this->md[$i]["notnull"] == "t") {
371          print("<td align=center>不可</td>¥n");
372         } else {
373          print("<td align=center><input type=¥"checkbox¥" name=¥"nulllist[$name]¥" ");
374          print("></td>¥n");
```

```
375         }
376
377         if (isset($this->md[$i][defaultval])) {
378          $def = $this->md[$i][defaultval];
379            print("<td align=center>$def</td>¥n");
380         } else {
381            print("<td align=center>なし</td>¥n");
382         }
383
384         $constraint = $this->md[$i][constraint];
385         if ($constraint == false) {
386          $constraint = "なし";
387         }
388         print("<td align=center>$constraint</td>¥n");
389
390         print("</tr>¥n");
391       }
392       $this->printDataInputFormFooter($mode);
393     }
394
395     // -------------- SQL文生成関数 ---------------------------
396
397     // データ検索用SQL文の生成
398     function makeSQL($updatable = false) {
399       global $atrlist;      // フォームからの入力データ(検索キー)
400       global $oplist;       // フォームからの入力データ(オペレータ)
401
402       if (!isset($atrlist)) {
403         return;
404       }
405
406       $sql = "SELECT ";
407       $where = "WHERE";
408
409       $first = true;
410       $needComma = false;
411
412       if ($updatable == true) {
413         $sql .= "oid AS __target_oid__ ";
414         $needComma = true;
415       }
416
417       while (list($name, $val) = each($atrlist)) {
418         if ($needComma == true) {
419          $sql .= ",";
420         }
421
```

```
422         $sql .= "$name ";
423         if (is_array($this->aliases)) {
424          $alias = $this->aliases[$name];
425          $sql .= "AS $alias";
426         }
427         if ($val != "") {
428          $op = $oplist[$name];
429          if ($first == false) {
430            $where .= " AND";
431          }
432          $where .= " $name $op '$val'";
433          $first = false;
434         }
435         $needComma = true;
436       }
437
438       $sql .= " FROM $this->table_name ";
439
440       if ($where != "WHERE") {
441         $sql .= " ";
442         $sql .= $where;
443       }
444
445       if ($this->sort_column != "") {
446         $sql .= " ORDER BY ";
447         $sql .= $this->sort_column;
448       }
449       return($sql);
450     }
451
452     // データ登録SQLの作成、実行
453     function insertSQL($user_check = false, $debug = false) {
454       global $atrlist;
455       global $nulllist;
456
457       $sql = "INSERT INTO $this->table_name (";
458       $atrs = "";
459       $vals = "";
460       $needComma = false;
461       $has_someval = false;
462
463       while (list($name, $val) = each($atrlist)) {
464         if ($nulllist[$name] == "" && $val == "") {
465          continue;           // 何も値が指定されていない
466         }
467
468         $has_someval = true;
```

```
469
470       if ($needComma == true) {
471        $atrs .= ",";
472        $vals .= ",";
473       }
474       $needComma = true;
475
476       $atrs .= "$name ";
477
478       if ($nulllist[$name] != "") {
479        $vals .= "NULL";          // NULLが指定された
480       } else if ($val != "") {
481        $vals .= "'$val'";        // 何か値が入力された
482       }
483     }
484
485     if ($user_check) {           // データチェックユーザ関数あり?
486       if ($user_check($atrlist) == false) {
487        return(false);
488       }
489     }
490
491     // 何も値が指定されていない場合はデフォルト値を入力する
492     if ($has_someval == false) {
493       $sql = "INSERT INTO $this->table_name DEFAULT VALUES";
494     } else {
495       $sql .= "$atrs) values ($vals)";
496     }
497
498     if ($debug == true) {
499       print($sql);
500     }
501     $rtn = pg_exec($sql);
502     return($rtn);
503   }
504
505   // データ変更SQLの作成、実行
506   function updateSQL($user_check = false, $debug = false) {
507     global $atrlist;
508     global $nulllist;
509     global $oid;
510
511     $sql = "update $this->table_name SET ";
512     $atrs = "";
513     $vals = "";
514     $needComma = false;
515     $has_someval = false;
```

```
516
517       while (list($name, $val) = each($atrlist)) {
518         $has_someval = true;
519
520         if ($needComma == true) {
521          $sql .= ",";
522         }
523         $needComma = true;
524         if ($nulllist[$name] != "") {
525          $sql .= "$name = null";
526         } else {
527          $sql .= "$name = '$val'";
528         }
529       }
530
531       if ($has_someval == false) {
532         print("データが入力されていません。");
533         return(false);
534       }
535
536       if ($user_check) {
537         if ($user_check($atrlist) == false) {
538          return(false);
539         }
540       }
541
542       $sql .= " WHERE oid = $oid";
543       if ($debug == true) {
544         print($sql);
545       }
546       return(@pg_exec($sql));
547     }
548
549     // データ削除SQLの作成、実行
550     function deleteSQL($debug = false) {
551       global $oid;
552
553       $sql = "DELETE FROM $this->table_name WHERE oid = $oid ";
554       if ($debug == true) {
555         print($sql);
556       }
557       return(pg_exec($sql));
558     }
559
560   }
561   ?>
```

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恐れもあります。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2-13**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2-13　フレームの利用例

3.9.1 フレームの枠組

フレームを使う場合、まずフレームの「枠組」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）。

フレーム分割の指定は<frameset>タグで行います。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

リスト2-16 フレームの枠組(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>
```

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限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>タグで行います。

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

はフレームの中にex8.phpを表示することを指示します。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。

なお、<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  //$include_path="/usr/local/apache/sample_php_lib";
 3  $include_path=".";
 4  require("$include_path/dbconnect.ini");
 5  require("$include_path/pgselect.ini");
 6  require("$include_path/pgmetadata.ini");
 7
 8  class myDbConnect extends DbConnect {
 9    var $dbname = "foo";                  // データベース名
10  }
11
12  class myPgSelect extends PgSelect {
13  }
14
15  class myPgMetaData extends PgMetadata {
16    var $table_name = "otenki";           // テーブル名
17    var $sort_column = "day";             // 日付でソート
18    var $aliases = array("day"=>"日付","tenki"=>"天気",
19         "ondo"=>"温度","uryou"=>"雨量");
20
21    function printFormHeader() {
22      global $PHP_SELF;
23      $str = <<< EOF
24        <form action="$PHP_SELF?mode=show" method="post" target="result">
25        <table>
26  EOF;
27      print($str);
28    }
29  }
30
31  $d = new myDbConnect;
32
33  if (!isset($mode)) {                    // はじめての表示?
34    @session_destroy();                   // セッション情報を破棄
35    $sel = new myPgSelect;                // myPgSelectオブジェクト作成
36    $meta = new myPgMetaData;             // myPgmetaDataオブジェクト作成
37  }
38
39  // セッション変数を登録
40  session_register("sel");
41  session_register("meta");
42  ?>
43
44  <html>
45  <head>
46  <title>Example 8</title>
47  <?php include("$include_path/jscripts.ini");?>
48  </head>
49  <body bgcolor="white">
50
```

```
 51  <?php
 52  if (!isset($mode) || $mode == "top") {
 53    $str = <<<EOF
 54  <ul>
 55  <li><a href="$PHP_SELF?mode=search" target="key">検索フォーム</a>
 56  <li><a href="$PHP_SELF?mode=insert" target="key">登録フォーム</a>
 57  </ul>
 58  EOF;
 59    print($str);
 60
 61  } else {
 62    switch($mode) {
 63    case "search":
 64      $meta->printForm();                          //検索フォームの表示
 65      break;
 66    case "show":
 67      $sel->doSelect($meta->makeSQL(true));      // 検索結果の表示
 68      break;
 69    case "insert":
 70      $meta->printDataInputForm("insert");       //登録フォームの表示
 71      break;
 72    case "insertExec":
 73      $sts = $meta->insertSQL();                 //登録の実行
 74      if ($sts != false) {
 75        print("正常にデータを登録しました。<br>¥n");
 76      }
 77      break;
 78    case "update":                               // 更新フォームの表示
 79      $meta->printDataInputForm("update");
 80      break;
 81    case "updateExec":                           // 更新/削除の実行
 82      if (isset($update_request)) {
 83        $sts = $meta->updateSQL();               // 更新処理
 84      } else {
 85        $sts = $meta->deleteSQL();               // 削除処理
 86      }
 87      if ($sts != false) {
 88        if (isset($update_request)) {
 89         print("更新処理が正常に終了しました。<br>¥n");
 90        } else {
 91         print("削除処理が正常に終了しました。<br>¥n");
 92        }
 93      }
 94      break;
 95    }
 96  }
 97
 98  $d->doClose();
 99  ?>
100  </body>
101  </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55行目をご覧ください。

```
<a href="$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うと、検索結果はやはり同じ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。

前述のように、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**：189～196行目）。

```
  function printFormHeader() {
    global $PHP_SELF;
    $str = <<< EOF
      <form action="$PHP_SELF?mode=show" method="post">
      <table>
EOF;
    print($str);
  }
```

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**：21～28行目）。

```
  function printFormHeader() {
    global $PHP_SELF;
    $str = <<< EOF
      <form action="$PHP_SELF?mode=show" method="post" target="result">
```

```
        <table>
EOF;
    print($str);
  }
```

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、一通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。

① 誰でもデータを閲覧できる
② 誰でもデータを登録できる
③ 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

たとえば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／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に対応しています。ONで対象テーブルを指定します[*8]。

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[*9]、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／削除を許さないようにする場合は、

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

を実行します。

*8
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。

*9
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う方法は大きく分けて二つあります。

① HTTPプロトコルで定義された認証プロトコルを使用する
② まったく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。

これを見ても分かるように、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[*10]。

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。

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」を実現しています[*11]。興味のある人は使ってみてください。

*10
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、標準のApacheではサポートされていません。

*11
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することができてしまいます。

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"
```

の下の方にある

```
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では環境変数をそのままの名前で通常の変数としてアクセスできるので

```
<?php print("ようこそ $REMOTE_USER さん！");?>
```

というスクリプトを実行すると、認証済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。

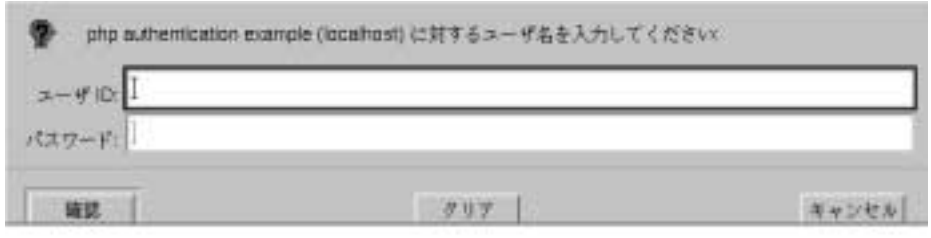
リスト2-18 PHPによる認証サンプルプログラム(ex9/php_auth.php)

```
 1  <?php
 2  if (!$PHP_AUTH_USER) {
 3    Header("HTTP/1.0 401 Authorized Request");
 4    Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
 5    print("ユーザ認証はキャンセルされました。");
 6    exit;
 7  } else {
 8    if ($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、$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$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2-14 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2-15**)。

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にアクセスを行います。今度は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$PHP_AUTH_USERがセットされていますので、8行目に移ります。ここでは、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。

図2-15 認証失敗後の画面

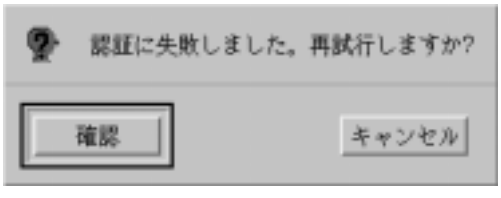
ここでは、もし入力されたパスワードがsecret以外なら9行目で

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

をクライアントに送信しています。すると、ブラウザには**図2-16**のダイアログが表示されます。ここで「確認」を選ぶと再びユーザ名とパスワードの入力ダイアログが表示されますが、キャンセルを選ぶと「パスワードが違います。」というメッセージを表示して終了します。

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。

図2-16 再認証を要するダイアログの例



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うことができます。デフォルトでは認証を行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

  - 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crypt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                                         crypt
```

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）。

なお、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    crypt
```

という行も追加して

```
local       foo                                         crypt
host        foo        127.0.0.1    255.255.255.255    crypt
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 password 'bar';
ALTER USER
```

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;
```

なお、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.php3）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  //$include_path="/usr/local/apache/sample_php_lib";
 3  $include_path=".";
 4  require("$include_path/dbconnect.ini");
 5  require("$include_path/pgselect.ini");
 6  require("$include_path/pgmetadata.ini");
 7
 8  class myDbConnect extends DbConnect {
 9    var $dbname = "foo";                  // データベース名
10  }
11
12  class myPgSelect extends PgSelect {
13  }
14
15  class myPgMetaData extends PgMetadata {
16    var $table_name = "otenki";           // テーブル名
17    var $sort_column = "day";             // 日付でソート
18    var $aliases = array("day"=>"日付","tenki"=>"天気",
19         "ondo"=>"温度","uryou"=>"雨量");
20
21    function printFormHeader() {
```

```
22      global $PHP_SELF;
23      $str = <<< EOF
24        <form action="$PHP_SELF?mode=show" method="post" target="result">
25        <table>
26  EOF;
27      print($str);
28    }
29  }
30
31  if (!$PHP_AUTH_USER) {
32    Header("HTTP/1.0 401 Authorized Request");
33    Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
34    print("ユーザ認証はキャンセルされました。");
35    exit;
36  } else {
37    $d = new myDbConnect;
38    if ($d->con == false) {
39      Header("HTTP/1.0 401 Unauthorized");
40      Header("WWW-authenticate: basic realm=¥"php authentication example¥"");
41      print("パスワードが違います。");
42      exit;
43    }
44  }
45
46  if (!isset($mode)) {              // はじめての表示?
47    @session_destroy();             // セッション情報を破棄
48    $sel = new myPgSelect;          // myPgSelectオブジェクト作成
49    $meta = new myPgMetaData;       // myPgmetaDataオブジェクト作成
50  }
51
52  // セッション変数を登録
53  session_register("sel");
54  session_register("meta");
55  ?>
56
57  <html>
58  <head>
59  <title>Example 9</title>
60  <?php include("$include_path/jscripts.ini");?>
61  </head>
62  <body bgcolor="white">
63
64  <?php
65  if (!isset($mode) || $mode == "top") {
66    $str = <<<EOF
67  <ul>
68  <li><a href="$PHP_SELF?mode=search" target="key">検索フォーム</a>
```

```
 69  <li><a href="$PHP_SELF?mode=insert" target="key">登録フォーム</a>
 70  </ul>
 71  EOF;
 72    print($str);
 73
 74  } else {
 75    switch($mode) {
 76    case "search":
 77      $meta->printForm();                          // 検索フォームの表示
 78      break;
 79    case "show":
 80      $sel->doSelect($meta->makeSQL(true));        // 検索結果の表示
 81      break;
 82    case "insert":
 83      $meta->printDataInputForm("insert");         // 登録フォームの表示
 84      break;
 85    case "insertExec":
 86      $sts = $meta->insertSQL();                   // 登録の実行
 87      if ($sts != false) {
 88        print("正常にデータを登録しました。<br>¥n");
 89      }
 90      break;
 91    case "update":                                 // 更新フォームの表示
 92      $meta->printDataInputForm("update");
 93      break;
 94    case "updateExec":                             // 更新/削除の実行
 95      if (isset($update_request)) {
 96        $sts = $meta->updateSQL();                 // 更新処理
 97      } else {
 98        $sts = $meta->deleteSQL();                 // 削除処理
 99      }
100      if ($sts != false) {
101        if (isset($update_request)) {
102         print("更新処理が正常に終了しました。<br>¥n");
103        } else {
104         print("削除処理が正常に終了しました。<br>¥n");
105        }
106      }
107      break;
108    }
109  }
110
111  $d->doClose();
112  ?>
113  </body>
114  </html>
```

31行目からPHPによる認証を行います。37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「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クラスのままではパスワードチェックができません。これは使用しているpg_connect関数にその機能がないからです。幸いpg_connetにはパスワードチェックを行う呼び出し形式もあるのでそれを使うようにDbConnectクラスを改造します。

DbConnectクラスの改造

修正したのはdoConnectメンバ関数だけなので、その部分を**リスト2-20**に示します。

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

```
40     function doConnect() {
41       global $PHP_AUTH_USER, $PHP_AUTH_PW;
42
43       // データベースに接続する
44       if ($PHP_AUTH_USER) {
45         if ($this->hostname != "") {
46          $constr .= "host=$this->hostname ";
47         }
48         if ($this->dbname != "") {
49          $constr .= "dbname=$this->dbname ";
50         }
51         if ($this->port != "") {
52          $constr .= "port=$this->port ";
53         }
54         @$this->con = pg_connect("$constr user=$PHP_AUTH_USER password=$PHP_AUTH_PW");
55       } else {
56         @$this->con = pg_connect($this->hostname,$this->port,$this->dbname);
57         if ($this->con == false) {
58          print("データベース $this->dbname に接続できませんでした。");
59          exit;
60         }
61       }
62       return($this->con);
63     }
```

いままで紹介した呼び出し形式では

```
pg_connect(ホスト名,ポート番号,データベース名)
```

となっており、ユーザ名やパスワードを渡すことができません。ユーザ名やパスワードを渡すには、次の形式を使います。

```
pg_connect(接続文字列)
```

接続文字列は、**キーワード=値**をいくつかつなげたものです。キーワードには**表2-11**のものを指定できます。

表2-11　接続文字列で使用できるキーワード

キーワード	意　味
host	ホスト名
dbname	データベース名
port	ポート番号
user	ユーザ名
password	パスワード

ほかにもありますが、普通使うのはこのくらいです。

たとえば、ホスト名は省略（すなわちUNIXドメインソケットで接続）、ポート番号は省略（デフォルトの5432を使用）、データベース名としてfooを指定、ユーザ名、パスワードはそれぞれfoo、barであるとします。この場合のpg_connectの接続文字列は次のようになります。

```
dbname=foo user=foo password=bar
```

このように、使わないキーワードは省略できます。

dbConnectでは、$PHP_AUTH_USERが存在するときに、接続文字列を使ったpg_connectの呼び出しを行っています（**リスト2-20**：44～54行目）。

Hypertext Preprocessor

HP

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/dbconnect.ini、ex9/pgselect.ini、ex9/pgmetadata.ini)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。

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。$PHP_AUTH_USER、$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

■ dbname

データベース名。

■ hostname

ホスト名。

■ port

ポート番号。

4.2 PgSelect

指定テーブルに関する検索処理を行います。本クラスではPostgreSQLにアクセスするので、DbConnetクラスを使ってあらかじめデータベースへの接続を確立しておく必要があります。

コンストラクタ

■ PgSelect

引数はありません。

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、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($oid)**

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。$oidは該当行のOID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。

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）。

● **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 printFormHeader() {
    global $PHP_SELF;
    $str = <<< EOF
      <form action="$PHP_SELF?mode=show" method="post">
      <table>
EOF;
    print($str);
  }
```

● **printAttrName($atrnumber, $atrname)**

$atrnumber番目の列の名前を$atrnameとして表示します。

● **printOprName($name, $op)**

オペレータを<select>タグを使って表示します。$opは二次元の連想配列で、$nameを列名として**$op[$i][$name]**でオペレータが取り出せます。

■ 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）。

● **printDataInputFormFooter($mode)**

printDataInputFormHeader()とペアになる関数です。$modeはデータ新規追加と変更を区別するためのものです。**insert**なら新規追加とします。デフォルトでは以下のように定義されています。

```
  function printDataInputFormHeader($mode) {
    global $PHP_SELF;
    global $oid;

    if ($mode == "insert") {
      $op="insertExec";
    } else {
      $op = "updateExec";
    }
    $str = <<<EOF
```

```
        <form action="$PHP_SELF?mode=$op&oid=$oid" method="post">
        <table border>
        <tr><th>カラム</th><th>データ型</th><th>データ</th>
        <th>NULL</th><th>初期値</th><th>制約</th></tr>
EOF;
    printf($str);
  }
```

■ **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**

データ型名の表示有無。

■ **is_print_opr_desc**

オペレータの説明の表示有無。

参考文献

[1]「改訂版 PostgreSQL完全攻略ガイド」石井達夫著、技術評論社、1999

[2]「SQLプログラミング入門」R.K.スティーブンス他著、ソフトバンク、1998

[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]「SQL92完全ガイド」J.メルトン／A.R.サイモン著、トッパン、1998

[5]「プログラマのためのSQL」J.Celko著、トッパン、1996