第 6 章

リレーショナルデータベースと 3 層 Web プログラミング

本章では関係データベース (Relational DataBase) の入門編として，シンプルな構造を持つ RDBMS(RDB Management System) である SQLite を使っていく。次章では，SQLite と PHP を使って 3 層 Web プログラミングを体験するが，そのための準備として SQL 文の扱い方にも触れる。

6.1　RDB と SQLite

データを保存するためにはファイルを使う。しかし単純にファイルの読み書きをするというだけでは，保存したデータの利用と言う点では不都合が多い。ファイル内の任意の位置でデータの追記，削除を行うのは結構面倒である。

そこで，ファイル内のデータの形式をきっちり決め，利便性を高めたソフトウェアとしてデータベースを使うことが近年では一般的になっている。ことに，関係データベース（Relational DataBase)，RDB を使うと，一度保存したデータを**テーブル**というカタマリで再利用しやすくなり，データ量が増えても整理がしやすくなるという利点がある。

6.1.1　RDB の基本用語

RDB の本格的な内容については立ち入らず，ここではごく基本的な事項のみを扱う。まず RDB を使うにあたって最低限知っておくべき用語を図 6.1 に示す。

■データベース，テーブル，フィールド　RDB において，**データベース**とは狭い意味での用語であり，データのカタマリである。**テーブル**を複数格納することができるディレクトリのようなものを意味する。データベースもテーブルにもそれぞれ名前（データベース名，テーブル名）が与えられる。

データはすべてテーブルの中に格納される。テーブル内に格納されるデータは項目として**フィールド**が設定され，それぞれのフィールドには名前 (フィールド名) が与えられる。フィールド名ごとにデータ型を決めて置く必要があり，例えば int 型 (整数)，varchar(文字数) 型 (指定文字数の文字列) 等が設定できる。

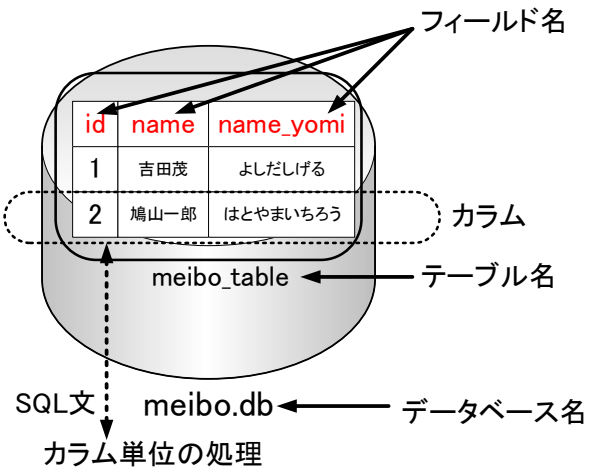

図 6.1　RDB の用語

■SQL 文 (言語)　SQL は RDB を操作する言語体系として国際規格として制定されているものであるが，RDBMS ごとに細かい「方言」が存在しており，骨格は同じだが具体的な操作となると RDBMS ごとに異なる SQL 文を発行しなければらないのが現状である。本書では言語としての利用は行わず，CREATE, SELECT, INSERT, DELETE, UPDATE, DROP といった要素文のごく基本的な利用に留める。

6.1.2　SQLite

本書では，パブリックドメインな RDBMS である SQLite[6] を使う[*1]。現在商業利用もされている代表的な RDBMS，例えば MySQL, PostgreSQL, SQL Server, Oracle は，RDB を保存するデータベースサーバと，RDB にアクセスしてユーザとの直接的なやりとりを行うクライアントに分離している，クライアント・サーバ (C/S) 型のシステム構成をとるのが普通である。C/S 型の RDBMS の利点は，サーバがネットワーク的に分離されたマシンに設置されていても，ネットワーク経由でクライアントからアクセスできるというメリットがある。

SQLite は単なる RDB ファイルを扱うソフトウェアライブラリであるので，データベースはデータベース名と同名の 1 ファイルとしてローカルマシンに保存されるだけである。C/S 型の本格的な RDBMS に比べて管理ツール等が充実しているとは言えないが，RDB の基本操作を行うだけの比較的小規模のデータ量しか扱わないようであれば，十分実用になるものである。

6.2　名簿データベース

以降はホームディレクトリの直下に”unix10”ディレクトリを作成し，そこで作業を行う。次回以降で，今回作成したデータベースを Web 経由で使用するので，ここでの作業は必ず行っておくこと。

[*1] 現在は Version 3 系が用いられる。

なお，Web 経由でデータベースファイルを扱うときには，セキュリティ確保のため，データベースファイルは絶対に public_html ディレクトリには置いてはいけない！データベースごと盗まれる可能性があるからである。

データベース名 ・・・meibo.db
テーブル名 ・・・meibo_table

フィールド名とデータ型は次のとおりである。

id ・・・番号 (int 型)
name ・・・氏名 (varchar(32) 型)
name_yomi ・・・よみがな (varchar(64) 型)

6.2.1 SQLite の起動と終了，ヘルプ

SQLite は 1 データベース＝ 1 ファイルという単位でデータを扱う。従って，起動時にはデータベースファイル名を指定する。

```
$ sqlite3 meibo.db ←meibo.db(データベース名)を指定(新規の場合は作成)
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> ←sqliteのプロンプト
```

SQLite を終了したい時は，SQLite のコマンド (ドットを接頭辞とする) で

```
sqlite> .quit
```

または

```
sqlite> .exit
```

とする。

SQLite のコマンドが分からなくなったら，SQLite のプロンプトからヘルプコマンドで確認してみよう。

```
sqlite> .help
```

6.2.2 主要な SQL 文: CREATE, INSERT, SELECT, UPDATE, DELETE, DROP

次に主要な SQL 文の意味と扱い方を見ておこう。

■データベースの作成，テーブルの作成・・・CREATE 文 前述したように，SQLite は 1 データベース＝ 1 ファイルなので，データベースは sqlite3 コマンドから生成する。データベース内のテーブルは以下に示すように CREATE 文を使って作成する。フィールド名やデータ型はその際にまとめて指定していることがわかる。

```
sqlite> CREATE TABLE meibo_table (id int, name varchar(32), name_yomi varcha
r(64));  ←meibo_table(テーブル名)をフィールド名とデータ型を指定して作成
sqlite> .dump meibo_table  ←  作成したテーブルの内容を確認
BEGIN TRANSACTION;
CREATE TABLE meibo_table (id int, name varchar(32), name_yomi varchar(64));
COMMIT;
```

■データの追加・・・INSERT 文　データの追記は下記のように INSERT 文を用い，行単位で一括して行う。

```
sqlite> INSERT INTO meibo_table VALUES(1, "吉田茂", "よしだしげる");  ←データの
追加
sqlite> INSERT INTO meibo_table VALUES(2, "鳩山一郎", "鳩山一郎");
```

■データの検索表示・・・SELECT 文　テーブルに保存されたデータの検索表示のためには SELECT 文を使う。フィールド名の指定にはワイルドカードが使用でき，この場合はすべてのフィールドを検索対象としている。

```
sqlite> SELECT * FROM meibo_table;  ← データの検索
1|吉田茂|よしだしげる
2|鳩山一郎|鳩山一郎
sqlite> SELECT id, name_yomi from meibo_table;
1|よしだしげる
2|鳩山一郎
```

■データの更新・・・UPDATE 文　すでにテーブルに存在しているデータの更新には UPDATE を用いる。更新すべきデータの場所を WHERE 節で示し (この場合は id 番号が 2 の行を対象としている)，その行内のどのフィールドのデータを更新するかを指定できる (この場合はよみがなを変更)。

```
sqlite> UPDATE meibo_table SET name_yomi="はとやまいちろう" WHERE id = 2;
sqlite> SELECT * FROM meibo_table;
1|吉田茂|よしだしげる
2|鳩山一郎|はとやまいちろう
```

■データの削除・・・DELETE 文　データの削除には DELETE 文を用いる。削除は行単位で一括して行われる。特定のフィールドだけを消去したいときには，空白文字列か NULL で UPDATE すればよい。

```
sqlite> DELETE FROM meibo_table WHERE id=2;
sqlite> SELECT * FROM meibo_table;
1|吉田茂|よしだしげる
```

■テーブル，データベースの削除　テーブルを削除したい時は DROP 文を用いる。

```
sqlite> DROP TABLE テーブル名;
```

データベースを消したい時にはデータベースファイルを rm コマンドで消去すればよい。

6.2.3 データベースのバックアップとリストア

SQLite に限らず，RDB では作成したデータベースを SQL 文の形でテキストファイルとして保存 (バックアップ) することが普通に行われる。こうしておくと，他のデータベースにも流用しやすくなる。

```
$ sqlite3 meibo.db  ←meibo.dbを開く
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> .dump ←データベースの内容をSQL文の形で画面に出力
BEGIN TRANSACTION;
CREATE TABLE meibo_table (id int, name varchar(32), name_yomi varchar(64));
INSERT INTO "meibo_table" VALUES(1, '吉田茂', 'よしだしげる');
INSERT INTO "meibo_table" VALUES(2, '鳩山一郎', 'はとやまいちろう');
INSERT INTO "meibo_table" VALUES(3, '岸信介', 'きしのぶすけ');
COMMIT;
sqlite> .output meibo-backup.txt ←出力先をファイル(meibo-backup.txt)に変更
sqlite> .dump ←出力(画面には出ず，meibo-backup.txtに保存される)
sqlite> .quit
```

バックアップしたデータを確認してみると，SQL 形式の内容がそのままテキストファイルとして保存されていることが分かる。

```
$ cat meibo-backup.txt ←.dumpの結果と同じものが保存されていることを確認
BEGIN TRANSACTION;
CREATE TABLE meibo_table (id int, name varchar(32), name_yomi varchar(64));
INSERT INTO "meibo_table" VALUES(1, '吉田茂', 'よしだしげる');
INSERT INTO "meibo_table" VALUES(2, '鳩山一郎', 'はとやまいちろう');
INSERT INTO "meibo_table" VALUES(3, '岸信介', 'きしのぶすけ');
COMMIT;
```

バックアップデータを復活（リストア）させるためには次のようにすれば良い。

```
$ sqlite3 meibo_new.db ←空の新規データベース(meibo_new.db)を作成
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> .dump ←空であることを確認
BEGIN TRANSACTION;
COMMIT;
sqlite> .read meibo-backup.txt ←SQL形式のテキストファイルを読み込み
sqlite> .dump ←meibo.dbと同じデータが復活(リストア)されていることが分かる
BEGIN TRANSACTION;
CREATE TABLE meibo_table (id int, name varchar(32), name_yomi varchar(64));
INSERT INTO "meibo_table" VALUES(1, '吉田茂', 'よしだしげる');
INSERT INTO "meibo_table" VALUES(2, '鳩山一郎', 'はとやまいちろう');
INSERT INTO "meibo_table" VALUES(3, '岸信介', 'きしのぶすけ');
COMMIT;
sqlite>
```

課題 A

meibo.db に次の 6 人のデータを正確に入力してみよ。

1. 吉田茂, よしだしげる
2. 鳩山一郎, はとやまいちろう

3. 岸信介, きしのぶすけ
4. 石橋湛山, いしばしたんざん
5. 池田勇人, いけだはやと
6. 佐藤栄作, さとうえいさく

打ち込みを失敗した時には DELETE, UPDATE 文をうまく組み合わせて自分でフォローせよ。

6.3　3 層 Web プログラミングとは？

以降では，SQLite と PHP スクリプトを組み合わせて 3 層 Web システムを構築する。現在のショッピングサイトや大規模 Web サイトでは，データの保存・検索のために RDBMS を用いることが普通に行われている。本書では極小規模ではあるが，データの登録，更新，削除，表示が一通り可能な Web サイトを，今まで学んできた知識を総動員して構築していく。

3 層 Web システム (図 6.2 を構築するためのプログラミングを 3 層 Web プログラミングと呼ぶことにしよう。

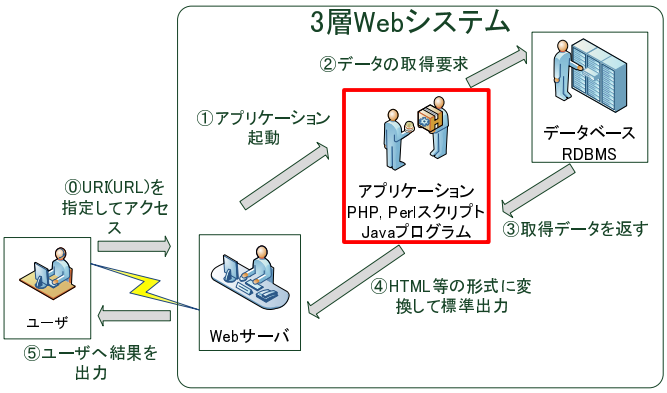

図 6.2　3 層 Web システムの概念図

3 層とは，下記の 3 要素を組み合わせていることを意味する。

1. Web サーバ・・・今回は Apache を使用する
2. RDBMS(Relational DataBase Management System)・・・いわゆるデータベースソフトウェア (RDB)。今回は SQLite を使用する
3. アプリケーション・・・Web サーバ上で動作するプログラムを記述・・・今回は PHP を使用する
   - 入出力 I/F には Web(HTML) を使用
   - データの管理は RDBMS に任せる

6.4 SQLite ファイルを使用してみる

では，次の手順に従って，http://自分の UNIX マシンの IP アドレス/~自分のユーザ ID/meibo/ に簡単な名簿データベースを構築していこう。

1. public_html/meibo/以下に index.html と PHP スクリプトを作成
2. 下記の機能を作成し，index.html からリンク
   - show_all.php ・・・ 全データ表示
   - insert.php ・・・ データ追加
   - delete_update.php ・・・ データ削除・更新
     → delete_exec.php, update_exec.php を呼び出す

[注意！] データベースファイルは相対パス指定すること！ (unix10/meibo.db にあるものとする)

前述した通り，Web ページ内の CGI/PHP スクリプトは Web サーバから起動され，スクリプトが存在するディレクトリがカレントディレクトリとなる。そのため相対パスしてを多用するので，ディレクトリの位置関係を頭に入れた上でプログラミングを行って頂きたい。

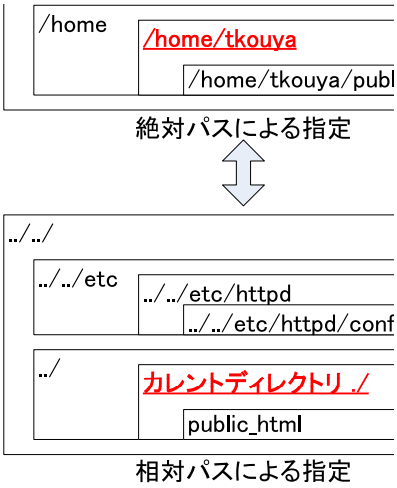

図 6.3 絶対パス指定 (上) と相対パス指定 (下，/home/tkouya/public_html をカレントディレクトリとした場合)

6.5 index.html の作成

では，完成時にはこの index.html からすべての機能が呼び出せるよう，リンクを張っておこう。実際にはリンク先の PHP スクリプトが完成してからリンクを生かすようにして

おくことが望ましい。でないと，どこまで作ったか，動かしてみるまでは分からないと言う状態になる。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>名簿データベース</title>
  </head>
<body>
<h1>名簿データベース(SQLite + PHP)</h1>
  <ul>
    <li><a href="show_all.php">全データ表示</a></li>
    <li><a href="insert.php">データ追加</a></li>
    <li><a href="delete_update.php">データ削除・更新</a></li>
</ul>
</body>
</html>
```

6.6　PHP のテンプレートファイル:meibo_template.php

以降では，PHP スクリプトからデータベースにアクセスする，骨組みだけのテンプレート PHP スクリプト (HTML 埋め込み型) として meibo_template.php を作成する。

6.6.1　HTML 部分

HTML 部分はごく簡素にしておく。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>タイトル</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
<body>
   <?php
   ・・・       ← PHPスクリプトをここに埋め込む
   ?>
  <a href="index.html">戻る</a>
</body>
</html>
```

6.6.2　PHP スクリプト部分

上記の HTML ファイルの PHP スクリプト部分は下記のようになる。SQLite ファイルを PHP から使用するためにはいくつかの方法があるが，ここでは PDO クラスを使っている。データベースを指定し，開いて閉じるというだけのものである。

```
$dbname = "../../unix10/meibo.db"; ←データベース（ファイル）の指定（相対パス）
$table  = "meibo_table";  ←テーブル名指定
$dsn = "sqlite:" . $dbname;  ←SQLiteを使用
try
```

```
{
  $connect = new PDO($dsn, null, null);←PDOクラスを初期化
}
catch (PDOException $error)←データベースを開けなかったときの処理
{
   echo '接続に失敗しました: ' . $error->getMessage();
   exit; ←PHPスクリプト強制終了
}

・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
・・・・・・・・・・ SQL文実行とデータ処理・・・・・・・・・・
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

unset($connect);
```

課題 B

meibo_template.php をコピーして，全てのデータを表示する show_all.php スクリプトを作れ。

show_all.php の全ソースコードは行番号つきで 97 ページから掲載してあるが，このうち，変更箇所のみ以下に抜粋しておく。

```
25:    // SQL文: SELECT * FROM テーブル名
26:    $sql = sprintf("SELECT * FROM %s", $table);
27:
28:    // SQL実行
29:    $query_ret = $connect->query($sql);
30:
31:    // 全データ表示
32:    while($data = $query_ret->fetch(PDO::FETCH_ASSOC))
33:    {
34:      printf("id: %s, name: %s, name_yomi: %s<br />¥n",
35:        $data['id'],
36:        $data['name'],
37:        $data['name_yomi']
38:      );
39:    }
```

6.7 名簿データベースの完成

meibo_template.php を insert.php にコピーして以降はこれを編集する。すべてのスクリプトが完成したら，index.html からのリンクをたどってすべての機能が正しく動作することを確認しよう。なお全ソースコードは付録に掲載してあるので，必要があれば適宜参照されたい。

6.7.1 insert.php の作成

「データ処理」部分を下記のように書き換え，入力データがフォームに書き込まれた後に，INSERT 文を発行してデータを追加できるようにする。

```
26:    // SQL文: SELECT max(id) FROM テーブル名
```

```
27:    $sql = sprintf("SELECT max(id) FROM %s", $table);
28:
29:    // SQL実行
30:    $query_ret = $connect->query($sql);
31:
32:    // 次のidを決定
33:    $next_id = $query_ret->fetchColumn() + 1;
34:
35:    // idが入力されているか?
36:    if(isset($_GET['id']))
37:    {
38:      // id >= 1 ? nameとname_yomiが入力されているか?
39:      if(($_GET['id'] >= 1) && ($_GET['name'] != "") && ($_GET['name_yomi']
 != ""))
40:      {
41:        $sql = sprintf("INSERT INTO %s VALUES(%s, '%s', '%s')",
42:          $table,
43:          $_GET['id'], $_GET['name'], $_GET['name_yomi']
44:        );
45:        $query_ret = $connect->query($sql);
46:        if($query_ret == FALSE)
47:        {
48:          echo "データ登録失敗!:" . $sql . "<br />¥n";
49:        }
50:        else
51:        {
52:          echo "データ登録成功!:" . $sql . "<br />¥n";
53:        }
54:      }
55:    }
```

追加のためのデータ入力フォームは<?php ・・・ ?>の下に，下記のように追加しておく。

```
<form>  ←ACTION=〝・・・〞の省略時は自分自身を呼出
  id : <input type="text" name="id" value="<?php echo $next_id ?>"><br>
  Name: <input type="text" name="name" width="32"><br>
  Name_yomi: <input type="text" name="name_yomi" width="64"><br>
  <input type="submit" value="追加"> <input type="reset" value="消去">
</form>
```

動作チェックのために，unix10/と unix10/meibo.db の others に対して書き込み権限を与え，ブラウザから insert.php を呼び出して動作確認を行う。

6.7.2　delete_update.php の作成

ここでは全データを表示し，各行毎に「削除」ボタンか「更新」ボタンを押して機能を選択するためのフォームを提供する。

```
25:    // SQL文: SELECT * FROM テーブル名
26:    $sql = sprintf("SELECT * FROM %s", $table);
27:
28:    // SQL実行
29:    $query_ret = $connect->query($sql);
30:
31:    // 全データ表示
32:    echo "<table>¥n";
```

```
33:    while($data = $query_ret->fetch(PDO::FETCH_ASSOC))
34:    {
35:      // テーブルの行
36:      echo "<tr>";
37:
38:      // delete用のform
39:      echo '<td><form action="delete_exec.php">';
40:      printf('<input type="submit" value="消去" />');
41:      printf('<input type="hidden" name="id" value="%d" />', $data['id']);
42:      echo '</form></td>';
43:
44:      // update用のform
45:      echo '<td><form action="update_exec.php">';
46:      printf('<input type="submit" value="更新" />');
47:      printf('<input type="hidden" name="id" value="%d" />', $data['id']);
48:      printf('<input type="hidden" name="name" value="%s" />', $data['name']
);
49:      printf('<input type="hidden" name="name_yomi" value="%s" />', $data['n
ame_yomi']);
50:      echo '</form></td>';
51:
52:      // データの表示
53:      printf("<td>id: %d, name: %s, name_yomi: %s</td>", $data['id'], $data[
'name'], $data['name_yomi']);
54:
55:      // テーブル行終了
56:      echo "</tr>¥n";
57:    }
58:    echo "</table>¥n";
```

6.7.3 delete_exec.php の作成

delete_update.php から「削除」ボタンが押されたら，この delete_exec.php スクリプトで指定された id 番号のデータを DELETE 文で削除する。

```
23:    // DELETE FROM table WHERE id = id
24:    if($_GET['id'] >= 1)
25:    {
26:      $sql = sprintf("DELETE FROM %s WHERE id = %d", $table, $_GET['id']);
27:      $query_ret = $connect->query($sql);
28:      if($query_ret == FALSE)
29:      {
30:        echo "削除失敗!: " . $sql . "<br />¥n";
31:      }
32:      else
33:      {
34:        echo "削除成功!: " . $sql . "<br />¥n";
35:      }
36:    }
```

6.7.4 update_exec.php の作成

delete_update.php から「更新」ボタンが押されたら，この update_exec.php スクリプトで指定された id 番号のデータの更新データをユーザに入力させ，その後で UPDATEE 文で更新を行う。

```
23:    // UPDATE table SET .... WHERE id = id
24:    if($_REQUEST['id'] >= 1)
25:    {
26:      if($_REQUEST['update_exec'] == 'on')
27:      {
28:        $sql = sprintf("UPDATE %s SET name='%s', name_yomi='%s' WHERE
id = %d",
29:          $table,
30:          $_REQUEST['name'],
31:          $_REQUEST['name_yomi'],
32:          $_REQUEST['id']
33:        );
34:        $query_ret = $connect->query($sql);
35:        if($query_ret == FALSE)
36:        {
37:          echo "更新失敗!: " . $sql . "<br />¥n";
38:        }
39:        else
40:        {
41:          echo "更新成功!: " . $sql . "<br />¥n";
42:        }
43:      }
44:    }
```

INSERT 同様，UPDATE のためにも更新データの入力用フォームが必要になるので，PHP スクリプトの下にフォームをつくる。下記では更新用に予め既存のデータをフォームのテキストボックスに表示する処理を行い，データの修正の手助けを行うようにしてある。

```
<!-- 更新用フォーム -->
<form>
  <input type="hidden" name="update_exec" value="on" />
  <input type="text" name="id" value="<?php echo $_REQUEST['id']; ?>" />
  <input type="text" name="name" value="<?php echo $_REQUEST['name']; ?>" />
  <input type="text" name="name_yomi" value="<?php echo $_REQUEST['name_yomi
']; ?>" />
  <input type="submit" value="更新" />
</form>
```

第 6 章の学習チェックリスト

- □ RDBMS における「データベース」と「テーブル」を説明することができる。
- □ 基本的な SQL 文 (create, select, insert, delete, update, drop 等) の意味を理解し，CUI から適切に使用することができる。
- □ PHP から SQLite で作成したデータベースファイルを操作することができる。