さわって実感

リレーショナルデータベース設計 8

アプリケーション設計（3）

日本PostgreSQLユーザー会 理事長 石井達夫 ISHII Tatsuo
ishii@postgresql.jp

PostgreSQL 7.2.2リリース

PostgreSQL 7.2.2が8月末にリリースされました．PostgreSQL 7.2.1との違いはバグ修正です．機能の追加はありません．本連載では前回はPostgreSQL 7.2.1を使用しましたが，7.2.2は7.2.1に対してプログラムやデータベースの互換性がありますので，postmasterを停止後，単にプログラムを入れ替えてpostmasterを再起動するだけで7.2.2にアップグレードできます．

モデルクラスについて

前回はPHPで作成した雑誌データベース管理システム（図1，図2）のうち，コントローラクラスとビュークラスの解説を行いました．今回は最後に残った大物クラスであるモデルクラス（dbaccessクラス，ソースコードはlib/model.inc）の解説をします．また，複数テーブルを扱うarticlesクラスから呼ばれるメソッドについても解説します．

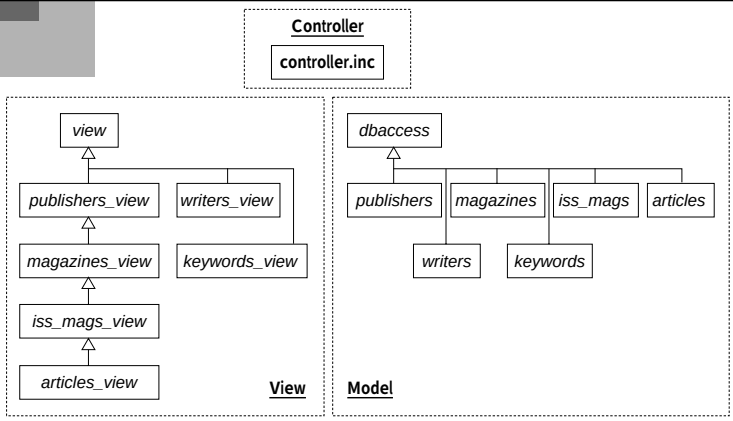
dbaccessクラス

dbaccessクラスでは，実際にDBの検索，更新処理を行います．

コンストラクタ

PHPLIBのDBアクセスクラスのオブジェクトを作成します（リスト1-①）．

図1 クラス図



get_meta

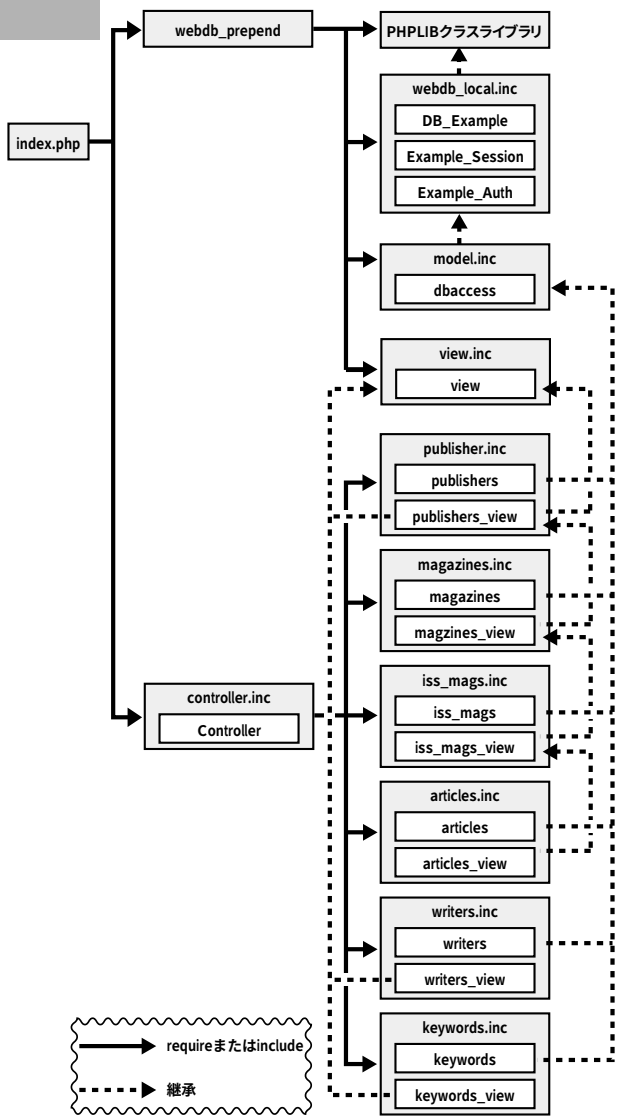
もともとはPHPLIBに付属する関数でしたが，ここで再定義しています（リスト1）．この関数はread_queryが返す列に関するデータ型などの情報を返します．関数の戻り値は連想配列で，

$return[列番号]["name"]　：列名

$res[列番号]["type"]　：データ型名
$res[列番号]["len"]　：列のデータ長

のようになっています．本スクリプトで使っているの

図2　ソースコードの構成



リスト1　dbaccessクラスの先頭部分

```
<?php
class dbaccess
{
  var $read_query = "";
  var $new_reg_query = "";
  var $write_table = "";
  var $write_cols = "";
  var $db = "";
  var $meta = "";
  var $key = "";
  var $foreign_key = "";
  var $parent_tables = "";
  var $multi_val_cols = "";
  var $multi_val_values = "";

  // コンストラクタ
  function dbaccess() {
    $this->db = new DB_Example;  ①
  }

  // メタデータの取得
  function get_meta() {
    if ($this->meta == "") {
      $count = 0;
      $id    = 0;
      $res   = array();

      $this->db->connect();
      $id = pg_exec($this->db->Link_ID,
      "SELECT * FROM (" . $this->read_query .
      ") as foo WHERE false");
      if ($id < 0) {
        $this->Error = pg_ErrorMessage($id);
        $this->Errno = 1;
        $this->halt("Metadata query failed.");
      }
      $count = pg_NumFields($id);

      for ($i=0; $i<$count; $i++) {
        $res[$i]["name"]  = pg_FieldName  ($id, $i);
        $res[$i]["type"]  = pg_FieldType  ($id, $i);
        $res[$i]["len"]   = pg_FieldSize  ($id, $i);
        $res[$i]["flags"] = "";
      }
      pg_FreeResult($id);
      $this->meta = $res;
    }
    return $this->meta;
  }
~以下省略~
```

は列名だけです．

exec_search

「検索」ボタンが押されたときに呼び出される関数で，実際に検索を行います（リスト2）．関数の戻り値は列名がキー，配列要素が検索値の連想配列です．

引数は4つあります．

- **第一引数（$first）**

TRUEならば，初回の検索実行．FALSEならページの「前」「次」のときに呼び出されたものと見なします．

- **第二引数（$offset）**

表示ページ．0スタートです．

- **第三引数（$def）**

表示されている列名の連想配列．通常viewクラスの$show_cols変数が渡されます．

- **第四引数（$vals）**

検索フォームで入力された値．キーが列名，配列要素値が入力値です．ページの「前」「次」のときはこの引数は省略します．

引数の他に以下のグローバル変数を使用します．いずれもcontrollerクラスで登録したセッション変数です．

- **$original_query**

最初に検索したときのSELECT文です．

- **$ctid**

現在表示中の行のTID（タプルID）です．

では処理を見ていきましょう．もし初回の表示であれば，$read_queryのWHERE句に「AND 列名='値'」

リスト2　dbaccessクラスのexec_search関数

```
function exec_search($first, $offset, $def,
  $vals = "") {
  global $original_query;
  global $ctid;

  if ($first) {
    $query = $this->read_query;
    if (ereg("WHERE",$query)) {
      $conj = " AND ";
    } else {
      $conj = " WHERE ";
    }
    foreach ($vals as $col => $var) {          ①
    if ($var != "") {
      $query .= $conj . "$col = " .  "'" .
        addslashes($var) . "'";
      }
      $conj = " AND ";
    }
    $original_query = $query;
  } else {
    if ($original_query == "") {
      // 次ページ/前ページ要求なのに，最初のページの表示がない
      return;
    }
    $query = $original_query;
  }

  $query .= " ORDER BY " . implode(",",
    $this->key) . " LIMIT 1 OFFSET $offset";   ②
  $defvals = array();
  $keyvals = array();
  $this->db->query($query);                    ③
  if ($this->db->next_record()) {              ④
    foreach ($def as $col => $var) {
      $defvals[$col] = $this->db->f($col);     ⑤
    }
    // 主キーの値を取り出す
    foreach ($this->key as $key) {
      $keyvals[$key] = $defvals[$key] =
        $this->db->f($key);                    ⑥
    }
    // 外部キーの値を取り出す                  ⑦
    if ($this->is_really_array(
      $this->foreign_key)) {
      foreach ($this->foreign_key as $key => $val) {
        $defvals[$key] = $this->db->f($key);
      }
    }

    // 現在のページのTIDを更新
    $ctid = $this->db->f("ctid");              ⑧
  }

  if ($this->is_really_array(
    $this->multi_val_cols)) {
    foreach ($this->multi_val_cols as $col) {
      $this->multi_val_values[$col] =
        $this->fetch_mvals($col, $keyvals);    ⑨
      $this->multi_val_candidates[$col] =
        $this->fetch_candidates($col);
    }
  }

  return $defvals;
}
```

の形で検索条件を追加していきます（リスト2-①）．実務のプログラムでは検索条件として，「列名 > '値'」や「列名 LIKE '値'」のような形式も使えるようにすべきですが，サンプルなのでそこまでしていません．

そして，検索結果の中から指定ページの1件を抜き出すために，「LIMIT 1 OFFSET $offset」を追加します（リスト2-②）．この構文はPostgreSQL固有のもので，このようにして検索結果の特定の部分を取り出せるので非常に便利です．また，検索結果をすべてDBサーバから転送する必要がないため，パフォーマンス的にも有利です．このときのポイントは，主キーでソートするためにORDER BYを追加していることです．LIMIT句を使用するときは，必ずORDER BYが必要です．ORDER BYがなくても動作しますが，実際問題として予測できない行が検索されるため，使いものになりません．詳しくはPostgreSQLのマニュアルselect（7）をご覧ください．

完成したSELECT文を使って検索を行います（リスト2-③）．SQL文の例を示します．

```
SELECT p.pb_name, m.mag_name, m.list_price,
i.issue_date, i.price, i.total_pages, a.title,
a.start_page,  a.num_pages,  a.articleid,
a.ctid, a.iss_magid
 FROM publishers AS p, magazines AS m,
iss_mags AS i, articles AS a
 WHERE p.publisherid = m.publisherid AND
m.magazineid = i.magazineid
  AND i.iss_magid = a.iss_magid ORDER BY
articleid LIMIT 1 OFFSET 0
```

query( )はPHPLIBが提供する検索の実行関数です．検索結果はPHPLIBのnext_record関数で取り出します（リスト2-④）．

検索結果は$defvalsと$keyvalsという連想配列に格納していきます．$defvalsはこの関数の戻り値になります．$keyvalsのほうは主キーの値を入れておきます．記事テーブルのときに使われるもので，あとで説明します．

まず，表示されている列名に対応した検索結果の値を$defvalsに格納します（リスト2-⑤）．これが直接画面に表示される値になります．f( )はPHPLIBの関数で，引数で指定された列名の値を取り出します．

次に主キーの値を取り出し，$defvalsに追加，$keyvalsにも格納します（リスト2-⑥）．

外部キーの値を取り出し，$defvalsに追加します（リスト2-⑦）．

現在のTIDを更新します（リスト2-⑧）．

リスト2-⑨は記事テーブル固有の処理ですので，後述します．

exec_new_search

exec_new_search（リスト3）はexec_searchと似ていますが，該当テーブルではなく，「親テーブル」を検索する点が違うだけで，ロジックはよく似ています．ここでは違いだけを説明します．

- **検索に使うSELECT文**

  read_queryではなく，new_reg_queryを使います．

- **更新対象の列**

  このテーブルで更新対象となっている列に関しては検索条件の対象としません（リスト3-①）．

- **記事テーブルに関する処理がない**

リスト3　dbaccessクラスのexec_new_search関数

```
function exec_new_search($first, $offset, $def,
  $vals = "") {
  global $original_query;
  global $ctid;

  if (!isset($this->new_reg_query) ||
    $this->new_reg_query == "") {
    print("この機能は有効ではありません<br>\n");
    return "";
  }

  if ($first) {
    $query = $this->new_reg_query;
    if (ereg("WHERE",$query)) {
      $conj = " AND ";
    } else {
      $conj = " WHERE ";
    }
    foreach ($vals as $col => $var) {
      if ($this->is_writable_col($col)) {
        continue;
      }                                    ①
: (以下省略)
```

8 アプリケーション設計（3）

exec_update

リスト4のexec_updateは，更新を実行します．引数は3つあります．

- **第一引数（$ctid）**
  更新対象のTID．
- **第二引数（$defvals）**
  更新前の値を格納した連想配列．「列名 => 値」のリストになっています．
- **第三引数（$val）**
  更新要求の値を格納した連想配列．「列名 => 値」のリストになっています．

では処理を見ていきましょう．まず，トランザクションの開始を宣言します（リスト4-①）．

次に関連するテーブルをロックします（リスト4-②）．本誌Vol.8連載第5回の「トランザクション設計」で説明したように，複数のテーブルをロックする場合はその順序に注意します．適当にロックすると，「デッドロック」という現象に悩まされることになります．ここでは，親テーブル，続いて本体テーブルをロックするという規約にしています．たとえば，「雑誌」テーブルなら，まず親テーブルである「出版社」テーブル，そして「雑誌」テーブルの順にロックします．親テーブルが複数ある場合（今回のサンプルでは「記事」テーブルの場合だけです）はparent_tablesに登録されている順番でロックします．したがって，parent_tablesにテーブルを書く順番は各クラスで統一しておかなければなりません．実際にロックを行っているのはlock_tablesですが，これは単にロックをするSQL文を発行しているだけなので，解説するまでもないでしょう．別関数にしているのは，LOCKコマンドのオプションを一カ所で管理したいからに過ぎません．

続くリスト4-③は，他のユーザによってデータが書き換えられていないかどうかのチェックです．

まずis_row_existsを使って主キーで指定される行のTIDを取り出します．もしTIDを取得できなければ，他のユーザによってこの行が削除されてしまったことになります．

次に覚えておいたTIDと取得したTIDが一致しているかどうかをチェックします．もし一致していなければ，この行は他のユーザによって更新されたことがわかります．ただし，このチェックでは，VACUUM FULLによって行が移動された場合もエラーとしてしまうことがあります．これを防ぐためには，行の内容そのものを覚えておいて比較する必要がありますが，

リスト4　dbaccessクラスのexec_update関数

```
function exec_update($ctid, $defvals, $val) {
  $this->db->query("BEGIN");                          ①
  if ($this->is_really_array(
    $this->parent_tables)) {
    $this->lock_tables($this->parent_tables);         ②
  }
  $table = $this->write_table;
  $this->lock_tables($table);

  $current_ctid = $this->is_row_exists($table,
    $this->key, $defvals);
    if ($current_ctid == FALSE) {
      print("該当行がありません<br>");
      $this->db->query("ROLLBACK");
      return FALSE;
    }

  // 他のトランザクションによってデータが変更されているか
  // どうかをctidでチェック
  if ($current_ctid != $ctid) {                       ③
    print("TIDが一致しません．他のユーザによってデータが更新されたか，VACUUMが実行された可能性があります．");
    $this->db->query("ROLLBACK");
    return FALSE;
  }

  $query = "UPDATE $table";
  $conj = " SET ";
  foreach ($val as $col => $var) {
    if ($var != "" AND isset(
      $this->write_cols[$col])) {
      $query .= $conj . "$col = " .  "'" .
        addslashes($var) . "'";                       ④
      $conj = " , ";
    }

  $query .= " WHERE ctid = '$ctid'";
  $this->db->query($query);
  $this->db->query("COMMIT");
  return TRUE;
}
```

まれなケースでもあり，今回はそこまでしていません．

以上でチェックが終わったので，UPDATE文を組み立て，更新を実行します（リスト4④）．

SQL文の例を示します．

```
UPDATE articles SET title = 'リレーショナルデータベース設計(1)' , start_page = '111' ,
num_pages = '12' WHERE ctid = '(0,22)'
```

exec_delete

削除を実行します（リスト5）．引数が2つあります．

- **第一引数（$ctid）**
  削除対象のTID．
- **第二引数（$defvals）**
  削除前の値を格納した連想配列．「列名 => 値」のリストになっています．

ロジックとしてはexec_updateと似ています．実際，トランザクションを開始してテーブルにロックをかけ，TIDのチェックを行うところまでは同じです．そして，DELETE文を組み立て，削除を実行します．

SQL文の例を示します．

リスト5　exec_delete関数

```
function exec_delete($ctid, $defvals) {
  $this->db->query("BEGIN");
  if ($this->is_really_array($this->parent_tables)) {
    $this->lock_tables($this->parent_tables);
  }
  $table = $this->write_table;
  $this->lock_tables($table);
〜省略〜
  $query =
    "DELETE FROM $table WHERE ctid = '$ctid'";
  $this->db->query($query);
  $this->db->query("COMMIT");
  return TRUE;
}
```

リスト6　dbaccessクラスのexec_insert関数

```
function exec_insert($defvals, $val) {
  // 外部キーを持っている場合，親テーブルが表示されていない
  // 場合は追加ができない
  if (isset($this->foreign_key) &&
    $this->is_really_array($this->foreign_key) &&
    !$this->is_really_array($defvals)) {
    return FALSE;
  }                                                    ①

  $this->db->query("BEGIN");
  if ($this->is_really_array($this->parent_tables)) {
    $this->lock_tables($this->parent_tables);
  }
  $table = $this->write_table;
  $this->lock_tables($table);                          ②

  $query = "INSERT INTO $table ";
  $target = "";
  $conj = "";
  $values = "";
  $checkvals = array();

  // 外部キーを持っている場合はそれを挿入データに追加   ③
  if (isset($this->foreign_key) &&
    $this->is_really_array($this->foreign_key)) {
    foreach ($defvals as $col => $var) {
      if ($var !=
        "" && isset($this->foreign_key[$col])) {
        $target .= $conj . $col;
        $values .= $conj . "'" .
          addslashes($var) . "'";
        $checkvals[$col] = $var;
        $conj = " , ";
      }
    }
  }

  foreach ($val as $col => $var) {                     ④
    if ($var != "" && isset(
      $this->write_cols[$col])) {
    $target .= $conj . $col;
    $values .= $conj . "'" . addslashes($var) . "'";
    $checkvals[$col] = $var;
    $conj = " , ";
    }
  }
  if ($conj != "") {
    if ($this->is_row_exists($table, $checkvals,
      $checkvals)) {
      print("すでに同じ行が存在しています<br>\n");      ⑤
      $this->db->query("COMMIT");
      return FALSE;
    }
    $query .= "(". $target . ") VALUES(" .
      $values . ")";
    $this->db->query($query);                          ⑥
  }
  $this->db->query("COMMIT");
  return TRUE;
}
```

```
DELETE FROM articles WHERE ctid = '(0,6)'
```

exec_insert

追加を実行します（リスト6）．引数が2つあります．

●第一引数（$defvals）

表示中の値を格納した連想配列．「列名 => 値」のリストになっています．この配列は外部キーを挿入するのに使います．

●第二引数（$val）

追加要求の値を格納した連想配列．「列名 => 値」のリストになっています．この配列は該当テーブルに値を挿入するのに使います．

まず，外部キーを持っているかどうかをチェックします．外部キーを持っている場合，その値が表示中でなければ処理ができないので，エラーとします（リスト6-①）．OKならば，テーブルをロックします（リスト6-②）．

続いてINSERT文の組み立てです．まず，外部キーがある場合は外部キーをINSERTするデータに追加します（リスト6-③）．その後，該当テーブルのデータを追加します（リスト6-④）．

SQL文の組み立てが終わったら，INSERTを実行する前にすでに同じ行が存在していないかどうかチェックします（リスト6-⑤）．OKなら，INSERTを実行します（リスト6-⑥）．

SQL文の例を示します．

```
INSERT INTO articles (iss_magid , title , start_page , num_pages) VALUES('1' , 'リレーショナルデータベース設計(2)' , '111' , '12')
```

その他の関数

model.incには他にも関数がありますが，次節「記事テーブル固有の処理」で解説します．

記事テーブル固有の処理

記事テーブルには，その記事を執筆した執筆者とキーワードが複数対応することがあります．また，執筆者や記事は別の記事と関連づけられることもあります．このような多対多の関係を表現するために，writingsとarticles_keywordsテーブルが設けられています．このような関連だけを持つテーブルをそのまま表示して更新しようとしても，人間にとってはわかりにくいものになってしまいます．そこで，本サンプルでは，記事テーブルの更新画面からこれらのテーブルを更新できるようにしてみました．

ここでもう一度，連載6回目の記事テーブルの画面を見てみましょう（図3）．ブラウザがMozillaになったので多少見た目が変っています．

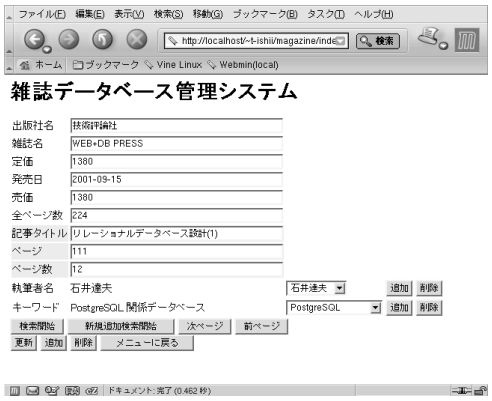
「執筆者名」と「キーワード」のところには，この記事に関連づけられている執筆者やキーワードが表示されています．また，右側のプルダウンメニューを使って新たに執筆者やキーワードを追加したり，削除できます．この部分は，データモデルのmultiで始まる変数で制御されます（表1）．

なお，multi_val_candidates_queries用のSELECT文では，HTMLのselectタグの<option value=のところをkey，>と</option>の間がvalという列名になるように設定する必要があります．

この他，表2のように動的に作られる配列があります．これらはモデルクラスで作られ，ビュークラスで利用されます．

たとえば，プルダウンメニューの部分のHTMLは以

図3　記事テーブル編集画面



下のようになります．

```
<select name="candidates[keyword]">
  <option value="2">PostgreSQL</option>
  <option value="3">関係データベース</option>
</select>
```

これらのデータはmulti_val_candidatesから持ってきています．この部分のHTMLを生成しているのは前号220ページのリスト4-④の部分です．

では，このデータを実際に作成しているmodel.incのexec_searchを見てみましょう．multi_val_valuesはfetch_mvals，multi_val_candidatesはfetch_candidatesが作成します．

fetch_mvals

●第一引数
列名．

●第二引数
キー列名とその値の連想リスト．

multi_val_queriesと引数からSELECT文を作り，検索結果を配列として返します．実行するSQL文の例を示します．

```
SELECT c.keyword FROM articles AS a,
articles_keywords AS b, keywords AS c
WHERE a.articleid = b.articleid AND
b.keywordid = c.keywordid AND a.articleid =
'1'
```

fetch_candidates

列名に対応したプルダウンメニューの表示用データを取得します．これは簡単で，単にmulti_val_candidates_queriesを実行し，結果を連想配列で返す

▼表1　静的変数の意味とデータ例

変数名	意味	データ例
multi_val_cols	処理対象となる列名リスト	array("wrt_name","keyword")
multi_val_tables	上記に関連するテーブル	array("wrt_name" => "writers"，"keyword" => "keywords")
multi_val_relations	多対多関連テーブル	array("wrt_name" => "writings"，"keyword" => "articles_keywords")
multi_val_relation_cols	列に関連するキー	"wrt_name" => array("wrt_name" => array("writerid", "articleid")，"keyword" => array("keywordid", "articleid"))
multi_val_queries	選択中の値を取得するためのSQL文リスト	array("wrt_name" => $q1, "keyword" => $q2)
multi_val_candidates_queries	プルダウンリストに表示するためのデータを取得するためのSQL文リスト	array("wrt_name" => $q3, "keyword" => $q4)

$q1：SELECT c.wrt_name FROM articles AS a,writings AS b, writers AS c
　　WHERE a.articleid = b.articleid AND b.writerid = c.writerid
$q2：SELECT c.keyword FROM articles AS a,articles_keywords AS b, keywords AS c
　　WHERE a.articleid = b.articleid AND b.keywordid = c.keywordid
$q3：SELECT writerid AS key, wrt_name AS val FROM writers ORDER BY wrt_name
$q4：SELECT keywordid AS key, keyword AS val FROM keywords ORDER BY keyword

▼表2　動的変数の意味

変数名	意味
multi_val_values	この記事で選択中の値（列名の右に表示されているもの）
multi_val_candidates	プルダウンメニューに表示される値

だけです．実行するSQL文の例を示します．

```
SELECT keywordid AS key, keyword AS val FROM
keywords ORDER BY keyword
```

exec_candidate_reg

プルダウンメニューで追加する著者やキーワードを選んだ後，「追加」ボタンを選択したときに呼ばれる関数です（リスト7）．引数は3つあります．

- **第一引数**
  列名．
- **第二引数**
  追加するキー値．
- **第三引数**
  現在表示中の列名=>値の連想配列．

テーブルをロックし（リスト7-①），すでに同じ値が関連テーブルに登録されていないかどうか調べます（リスト7-②）．SELECT文を生成するために検索対象のテーブルはmulti_val_relationsから，また検索対象列はmulti_val_relation_colsから持ってきます．multi_val_relation_colsの最初のデータは，プルダウンリストに表示される列名になっていることに注意してください．たとえば，次のようなSELECT文が実行されますが，

```
SELECT count(*) FROM articles_keywords WHERE
keywordid = '3' AND articleid = '1'
```

“keyword”が必ずmulti_val_relation_colsの先頭の要素である必要があります．また，'3'は第二引数から取得しています．

もしまだ同じ値が関連テーブルに登録されいなければ，INSERT文を組み立てて実行します（リスト7-③）．実行するSQL文の例としては，以下のようになります．

```
INSERT INTO articles_keywords (keywordid,articleid)
VALUES ('3', '1')
```

exec_candidate_del

プルダウンメニューで追加する著者やキーワードを選んだ後，「削除」ボタンを選択したときに呼ばれる関数です（リスト8）．引数はexec_candidate_regと同じです．テーブルをロックし（リスト8-①），DELETE

リスト7　dbaccessクラスのexec_candidate_reg関数

```
function exec_candidate_reg($col, $candidates,
  $defvals) {
  $this->db->query("BEGIN");

  // テーブルをロック
  $this->lock_tables($this->write_table);
  $this->lock_tables(                          ①
    $this->multi_val_relations[$col]);

  // すでに同じ値が登録済かどうかチェック          ②
  $query = "SELECT count(*) FROM
    {$this->multi_val_relations[$col]} WHERE";
  $first = TRUE;
  foreach ($this->multi_val_relation_cols[$col] as $c) {
    if ($first) {
      $query .= " $c = '$candidates'";
      $first = FALSE;
    } else {
      if (!isset($defvals[$c])) {
        $this->db->query("COMMIT");
        return;
      }
      $query .= " AND " . "$c = " . "'" .
          addslashes($defvals[$c]) . "'";
    }
  }
  $this->db->query($query);
  $this->db->next_record();
  if ($this->db->f("count") > 0) {
    // すでに登録済
    $this->db->query("COMMIT");
    return;
  }

  // 値を追加                                    ③
  $query =
    "INSERT INTO {$this->multi_val_relations[$col]} ("
    . implode(",",
    $this->multi_val_relation_cols[$col]) .
    ") VALUES ('$candidates', ";
  foreach ($this->key as $c) {
    $query .= "'" . $defvals[$c] . "'";
  }
  $query .= ")";
  $this->db->query($query);
  $this->db->query("COMMIT");
}
```

文を組み立てます（リスト8-②）.

実行するSQL文の例としては，以下のようになります.

```
DELETE FROM articles_keywords WHERE keywordid
= '3' AND articleid = '1'
```

次回予告

雑誌データベース管理システムを例にとり，データベース設計からアプリケーション設計，そして実装までをPostgreSQL＋PHPを使って解説しましたが，いかがだったでしょうか．1年以上に渡って続けてきた本連載もいよいよ大詰めです．データベースの基礎から始まり，ようやく実際に動くアプリケーションが完成したわけですが，アプリケーションを気持良く使い続けるためには，データベースの運用管理は避けて通れない話題です.

次回は連載の締めくくりとしてバックアップやチューニングなどのデータベースの運用管理技法について取り上げる予定です．WD

リスト8　dbaccessクラスのexec_candidate_del関数

```
function exec_candidate_del($col, $candidates,
    $defvals) {
  $this->db->query("BEGIN");

  // テーブルをロック
  $this->lock_tables($this->write_table);
  $this->lock_tables(                          ①
    $this->multi_val_relations[$col]);

                                               ②
  $query =
    "DELETE FROM {$this->multi_val_relations[$col]} WHERE";
  $first = TRUE;
  foreach ($this->multi_val_relation_cols[$col] as $c) {
    if ($first) {
      $query .= " $c = '$candidates'";
        $first = FALSE;
    } else {
    $query .= " AND " . "$c = " . "'" .
      addslashes($defvals[$c]) . "'";
    }
  }
  $this->db->query($query);
  $this->db->query("COMMIT");
}
```