DATABASE

第7章

Empress Embeddedの使用例

C言語によるプログラミング(2)
――レコード，バルク・データ

David Zhang

第6章に引き続き，C言語インターフェースを使ったデータベース・プログラミングの実例について解説する．ここでは，レコードに対する操作を解説したあと，データのロック，データの整合性を保証するためのトランザクション処理の実例，ストリーミング・データなどを扱えるバルク・データの扱いについて解説する．（編集部）

本稿では，レコードに対する操作，ロック，トランザクション，表現式の詳細，バルク・データの扱いについて解説します．

1 レコードに対する操作

● レコードの削除

mrdel関数とmrtdel関数は，引き数にレコード記述子を指定し，レコードを削除します．mrdel関数は，削除に失敗した場合に呼び出しプログラムを終了します．mrtdel関数は，削除に失敗した場合にfalseを，成功した場合にtrueを返します．通常，レコードは削除前に検索されます．対象レコードは，mrgetbegin関数と検索条件関数により初期化され，mrget関数で特定されます．すべての削除が完了した場合には，Empressの内部バッファからファイルへ書き込むために，mrdelend関数を呼び出す必要があります．

レコードを削除する一般的な形式を以下に示します．

```
mrdel (record_desc);
flag = mrtdel (record_desc);
mrdelend (record_desc);
```

リスト1は1992年5月より前に作成されたレコードを削除します．

● レコードの更新

レコードを更新するには，二つのレコード記述子を取得する必要があります．最初のレコード記述子（更新前用）は，mrmkrec関数によって取得します．次のレコード記述子（更新後用）は，mrcopyr関数によって，更新前用レコード記述子をコピーして作成します．この関数は，引き数に二つのレコード記述子を指定します．更新後用のレコード内のフィールド値は，レコードの追加処理と同じように，mrputvs関数，mrputvi関数などを使用して変更します．レコードの更新は，mrput関数を呼び出して行われます．この関数は，引き数に更新前，更新後のレコード記述子を指定します．

更新が失敗する可能性がある場合（ユニークなフィールドに重複値を入力したり，ロックがかけられる場合）には，mrtput関数を使用してください．この関数は，失敗したときにfalseを，成功したときにtrueを返します．

これらの関数の一般的な形式を以下に示します．

```
mrcopyr (new_rec_desc, old_rec_desc);
mrput (new_rec_desc, old_rec_desc);
flag = mrtput (new_rec_desc, old_rec_desc);
```

リスト2はloansテーブル内のローンについて2％の利子を計算し，借りている額と利子を表示します．また，現在額に利子を追加し，新しい合計を表示し，レコードを更新します．

● 後で使用するレコードの保存

アプリケーション・プログラムでは，異なる箇所で同一のレコードに異なった操作を行わなくてはならない場合があります．

リスト1　レコードを削除する（delete.c）

```
#include    <mscc.h>

#define     DATABASE   "repairs"

msmain ()
{
     addr     loans_tabdesc, date_attrdesc,
              loans_recdesc;
     addr     qual, retrieve_desc;
     char*    date_value;

     loans_tabdesc = mropen (DATABASE, "loans", 'u');

     loans_recdesc = mrmkrec (loans_tabdesc);
     date_attrdesc = mrngeta (loans_tabdesc, "date");
     qual = mrqcon ("<", date_attrdesc,
                    mrcvt (date_attrdesc, "1 May 1992"));

     retrieve_desc = mrgetbegin(qual, loans_recdesc, ADDRNIL);
     while (mrget (retrieve_desc))
          mrdel (loans_recdesc);

     mrgetend (retrieve_desc);

     mrdelend (loans_recdesc);

     mrfrrec (loans_recdesc);

     mrclose (loans_tabdesc);
}
```

日付との比較

KEYWORD ――レコード，バルク・データ，Perl，Tcl/Tk，トランザクション，レコード・ポインタ，レコード・レベル・ロック，データ・ストリーミング

リスト2　レコードの更新(interest.c)

```
 #include    <mscc.h>

#define     DATABASE   "repairs"

extern      double     dollcvt ();

msmain ()
{
     addr       loans_tabdesc, pers_tabdesc,
                loans_recdesc, pers_recdesc,
                new_recdesc;
     addr       pname_attrdesc, lname_attrdesc,
                date_attrdesc, amount_attrdesc;
     addr       qual, p_retrieve_desc,
                l_retrieve_desc;
     double     amount, sum, newamount, interest;
     char*      name;
     char*      pname_value;
     char*      date_value;
     char*      amount_value;
     char       value[20];

     loans_tabdesc = mropen (DATABASE, "loans", 'u');
     pers_tabdesc = mropen (DATABASE, "personnel", 'r');

     loans_recdesc = mrmkrec (loans_tabdesc);
     new_recdesc = mrmkrec (loans_tabdesc);
     pers_recdesc = mrmkrec (pers_tabdesc);

     pname_attrdesc = mrngeta (pers_tabdesc, "name");
     lname_attrdesc = mrngeta (loans_tabdesc, "name");
     amount_attrdesc = mrngeta (loans_tabdesc, "amount");
     date_attrdesc = mrngeta (loans_tabdesc, "date");

     amount_value = mrspv (amount_attrdesc);
     pname_value = mrspv (pname_attrdesc);
     date_value = mrspv (date_attrdesc);

     p_retrieve_desc = mrgetbegin (ADDRNIL, pers_recdesc,
                                   ADDRNIL);
     while (mrget (p_retrieve_desc))
     {
          mrcopyv (pers_recdesc, pname_attrdesc, pname_value);

          printf ("¥n¥nMonthly Statement for %s:¥n¥n",
                  pname_value);
          printf ("Loan      Date Made      Interest Total¥n");

          sum = 0;
          qual = mrqcon ("=", lname_attrdesc,
                         mrcvt (lname_attrdesc, pname_value));
          l_retrieve_desc = mrgetbegin (qual, loans_recdesc,
                                        ADDRNIL);
          while (mrget (l_retrieve_desc))
          {
               mrcopyv (loans_recdesc, amount_attrdesc,
                        amount_value);
               mrcopyv (loans_recdesc, date_attrdesc,
                        date_value);
               amount = dollcvt (amount_value);
               interest = amount * 0.02;
               newamount = amount + interest;
               sum = sum + newamount;
               printf ("$%-6.2f %s      $%-6.2f $%-6.2f¥n",
                       amount, date_value, interest, newamount);

               mrcopyr (new_recdesc, loans_recdesc);

               sprintf (value, "$%-6.2f", newamount);
               if (mrputvs (new_recdesc, amount_attrdesc,
                            value))
                    mrput (new_recdesc, loans_recdesc);
               else
                    fprintf (stderr, "Cannot convert¥
                         newamount '%-6.2f'¥n¥
                         Update not done for %s %s¥
                         %-6.2f¥n", newamount,
                         pname_value, date_value, amount);
          }
          mrgetend (l_retrieve_desc);

          if (sum > 0)
               printf ("¥nTotal now owing: $%-6.2f¥n", sum);
          else
               printf ("¥nNo loans outstanding.¥n");
     }
     mrgetend (p_retrieve_desc);

     mrfree (pname_value);
     mrfree (date_value);
     mrfree (amount_value);
     mrfrrec (loans_recdesc);
     mrfrrec (new_recdesc);
     mrfrrec (pers_recdesc);

     mrclose (loans_tabdesc);
     mrclose (pers_tabdesc);
}

/* strip $ , * space from dollar amounts */
double dollcvt (char* string)
{
     extern     double    atof ();
     char       c, buffer [20];
     char*      buf_ptr;

     for (buf_ptr = buffer ; (c = *string++) != '¥0' ; )
        if (c != '$' && c != '*' && c != ' ' && c != ',')
           *buf_ptr++ = c;
     *buf_ptr = '¥0';
     return (atof (buffer));
}
```


一度レコードを検索したら，以下に示すことが行えます．

- 直接操作できるように，そのレコードをメモリに保存する
- 以降の検索で，検索条件を構築しなくてもよいように，そのレコードへのポインタを保存する

レコード値が変更された場合，変更前の検索条件ではレコードを再検索できないので，レコードまたはポインタを保存することが非常に役にたちます．

● メモリへのレコード保存

レコードをメモリ内に保存する際には，そのレコードのレコード記述子をコピーすることになります．この場合，スペースを使い過ぎないように，数レコードにしぼって実行してください．レコードがロック状態で保たれている場合にも，メモリへのレコード保存は有用です．

レコード記述子をコピーするには，mrmkrec 関数を呼び出して新しいレコード記述子を取得し，レコードの更新と同じように，mrcopyr 関数によって新しいレコード記述子にコピーします．これによってレコードがメモリ内にコピーされ，テーブルから再検索することなく，いつでもアクセス可能となります(通常は，検索ループの中で現在のレコードが後で必要になると判断された場合に実行される)．保存されたレコード記述子は，後のプログラム内でいつでも，mrcopyv 関数などの検索関数の一つに受け渡すことが可能です．レコードに対する処理が完了したら，mrfrrec 関数を使用してスペースを解放してください．