STUDIA INFORMATICA 2003
Volume 24 Number 1 (52)

Lesław PAWLACZYK
Politechnika Śląska, Instytut Informatyki

NATYWNY DOSTĘP DO BAZ DANYCH W JĘZYKU C NA PLATFORMIE AS/400

Streszczenie. Natywny[1] dostęp do baz danych to stara i szybka metoda sprawdzona na przestrzeni lat. AS/400 jest platformą programistyczną stworzoną przez IBM. Artykuł ten prezentuje w kilku przykładach metodę dostępu do bazy danych oraz interfejs w języku C. Pokazano, jak czytać, dodawać oraz aktualizować rekordy oraz jak pisać programy transakcyjne.

Słowa kluczowe: język C, AS/400, baza danych, dostęp natywny.

NATIVE ACCESS TO BATABASES USING C LANGUAGE ON THE AS/400 PLATFORM

Summary. A native access to databases is an old and fast method verified during past years. AS/400 is a programming platform created by IBM. This article presents in few examples a method for accessing database files in C language. It is shown how to read, add and update records and how to write transaction programs.

Keywords: C language, AS/400, database, native access.

1. Wstęp

AS/400 jest platformą stworzoną przez firmę IBM jako kontynuacja systemów serii S/36. Skrót nazwy oznacza *Application Server / 400*, czyli serwer aplikacji. Opisywana platforma jest konstrukcją autorską koncernu IBM i nie ma żadnych swoich odpowiedników u innych producentów. Zastosowania, jakie znajduje AS/400, to przede wszystkim duże bazy danych w bankach, systemy informatyczne w dużych przedsiębiorstwach zatrudniających wiele tysięcy

[1] Natywny dostęp do bazy danych to operacje na pojedynczych rekordach.

pracowników. Najbardziej interesujące z punktu widzenia programisty tej platformy jest jej ukierunkowanie na wysoce wydajne operacje bazodanowe. AS/400 jest standardowo wyposażony w system operacyjny OS/400, którego najnowsza wersja 5.2 jest już wprowadzana na rynek. Opisywany system może zawierać kompilatory języków programowania, takich jak: C/C++, RPG, Java i Pascal. Do operacji bazodanowych można wykorzystywać język SQL dla zintegrowanej z systemem bazy DB2. W artykule zostanie jednak opisany dostęp do baz danych z wykorzystaniem dostępu natywnego i języka C.

2. Dostęp do bazy danych w języku C

DB2 jest bazą danych nierozerwalnie złączoną z systemem operacyjnym OS/400. Jest ona tak dobrze zintegrowana, iż praktycznie idea tego systemu operacyjnego opiera się na operacjach podobnych do operacji na bazie danych. System OS/400 jest systemem obiektowym, w którym każdy element systemu jest traktowany jako część wielkiej bazy danych. Najistotniejsze jest jednak to, że pliki bazy danych są traktowane jako część systemu operacyjnego, który sam w sobie posiada wsparcie do operacji na nich w postaci komend realizujących zapytania, wyświetlających oraz aktualizujących zawartość pliku. Każda tabela bazy danych jest osobnym plikiem nazywanym plikiem fizycznym. Wszelkie więzy referencyjne między tabelami oraz powiązania typu klucz obcy - klucz główny są możliwe do zdefiniowania na poziomie opisu tabel w plikach definicji tabel. Należy jednak pamiętać o tym, że w trybie dostępu natywnego do bazy danych, to programista musi zdefiniować algorytm wykonywania zapytania. Baza danych może być zdefiniowana na wiele różnych sposobów:

- język *SQL*,
- język *DDS – Data Description Specification* [1,2],
- narzędzia programistyczne firm trzecich, np. pakiet *Adelia*,
- narzędzie systemowe *IDDU – Interactive Data Specification Utility*.

Dostęp do bazy danych [3,4,5,6] opisany jest w pliku nagłówkowym *recio.h*, a funkcje operujące na plikach to: *_Rclose()*, *_Rcommit()*, *_Rdelete()*, *_Rfeod()*, *_Rformat()*, *_Rfiobk()*, *_Rlocate()*, *_Ropen()*, *_Ropnfbk()*, *_Rreadd()*, *_Rreadk()*, *_Rreadn()*, *_Rreadp()*, *_Rrlsck()*, *_Rufpb()*, *_Rupdate()*, *_Rwrite()*, *_Rwrited()*.

Aby otworzyć plik bazodanowy AS/400 jako plik rekordowy, należy użyć funkcji *_Ropen()* razem z jednym z podanych trybów: *rr*, *rr+*, *wr*, *wr+*, *ar*, *ar+*.

2.1. Odczyt danych z pliku

W poniższym przykładzie dane są odczytywane i wypisywane na ekran z pliku fizycznego (właściwego pliku bazy danych) *OSOBAF* o strukturze:

- klucz główny: kolumna *IMIE* – typ znakowy o długości 10,
- kolumna *NAZWISKO* – typ znakowy o długości 20,
- kolumna *MIASTO* – typ znakowy o długości 20.

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#define _RCDLEN 50
int main(void)
{
 _RFILE *in;
 _RIOFB_T *struktura;
 int warunek = 1;
 int licznik = 1;
 char record[_RCDLEN];
 if ( (in = _Ropen("*LIBL/OSOBAF", "rr, arrseq=Y")) == NULL )
 {
   printf("Otwarcie pliku wejściowego nie powiodło się\n");
   exit(1);
 }
 while (warunek)
 {
   char imie[10],nazwisko[20],miasto[20];
   struktura = _Rreadd(in, record, _RCDLEN, __NO_LOCK, licznik);
   if (struktura)
   {
       if (struktura->num_bytes < _RCDLEN) warunek = 0;
       else
       {
         sscanf(record,"%10s%20s%20s",imie,nazwisko,miasto);
         printf("\nCzytam rekord numer %d, gdzie imie = %s, nazwisko "
                " = %s, miasto = %s",licznik,imie,nazwisko,miasto);
         licznik++;
       }
   }
 }
 _Rclose(in);
 printf("\nKoniec wyświetlania rekordów");
 return 1;
}
```

Opisany powyżej sposób wyświetlania polega na czytaniu kolejnych rekordów z pliku *OSOBAF* według kolejności, w jakiej zostały zapisane, podając jedynie numer rekordu. Zmienna *struktura* jest to zmienna typu wskaźnik na *_RIOFB_T* (struktura opisująca plik bazy danych). W przypadku błędu przy odczycie rekordu – funkcja *Rreadd()* – pole *num_bytes* tej struktury zostaje zapisane wartością mniejszą niż długość rekordu. Dane w prostych plikach fizycznych są zapisywane jako łańcuchy tekstowe, dlatego jednym ze sposobów, by odczytać wartości kolumn dla danego rekordu, jest funkcja *sscanf()*.

2.2. Zapisywanie danych do pliku

Zapisanie danych do pliku można osiągnąć za pomocą funkcji *_Rwrite()* jak w poniższym przykładzie, gdzie zawartość jednego pliku jest kopiowana do drugiego:

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#define _RCDLEN 300
void main(void)
{
 _RFILE *in;
 _RFILE *out;
 _RIOFB_T *fb;
 char record[_RCDLEN];
 /* Otwarcie pliku wejściowego do przetwarzania w kolejności przychodzącej
*/
 if ( (in = _Ropen("*LIBL/ZRODLO", "rr, arrseq=Y")) == NULL )
 {
      printf("Otwarcie pliku wejściowego nie powiodło się\n");
      exit(1);
 };

 /* Otworzenie pliku wyjściowego. */
 if ( (out = _Ropen("*LIBL/CEL", "wr")) == NULL )
 {
      printf("Otwarcie pliku wyjściowego nie powiodło się \n");
      exit(2);
 };
 /* Kopiuj plik, aż pojawi się koniec pliku. */
 fb = _Rreadn(in, record, _RCDLEN, __DFT);
 while ( fb->num_bytes != EOF )
 {
      _Rwrite(out, record, _RCDLEN);
      fb = _Rreadn(in, record, _RCDLEN, __DFT);
 };
 _Rclose(in);
 _Rclose(out);
}
```

Zakłada się, że pliki *ZRODLO* i *CEL* to pliki fizyczne utworzone komendami:

1. *CRTPF FILE(ZRODLO) RCDLEN(300)*
2. *CRTPF FILE(CEL) RCDLEN(300)*

Pliki takie nie mają własnej definicji *DDS*, a postać rekordu to jedna kolumna o długości podanej w atrybucie *RCDLEN*, czyli 300 bajtów.

2.3. Aktualizacja

Kolejny przykład pozwala prześledzić aktualizację rekordów przy wyszukiwaniu według klucza głównego w tabeli:

```
/* Ten program ilustruje jak aktualizować rekordy funkcją _Rupdate() */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
void main(void)
{
      _RFILE *in;
      char stare_dane[50],nowe_dane[50];
      char nowe_imie[10] = "ADAM";
      char klucz[10]= "RYSIEK";
      char imie[10];
      char reszta_danych[40];
      /* Otwarcie pliku w kolejności według klucza. Plik jest tworzony z
      domyślnym indeksem*/
      if ( (in = _Ropen("*LIBL/OSOBAF", "rr+")) == NULL )
      {
            printf("Otwarcie nie powiodło się\n");
            exit(1);
      }
      //odczytanie właściwych danych rekordu
      fb = Rreadk(in, stare_dane, 50, __KEY_EQ, "RYSIEK", 4);
      if (fb->num_bytes < 50)
            ;
      else
      {
            //dane można zmieniać, bo odczytano rekord
            sscanf(stare_dane, "%10s%40s", imie, reszta_danych);
            sprintf(nowe_dane, "%10s%40s", nowe_imie, reszta_danych);
            /* Aktualizuj pierwszy rekord, gdzie IMIE = "RYSIEK" */
            /* _Rlocate blokuje rekordy. */
            _Rlocate(in, klucz, 0, __KEY_EQ);//ustawienie się na rekordzie
            _Rupdate(in, nowe_dane, 50);//można aktualizować
      }

      /* Wymuszenie końca danych. */
      _Rfeod(in);
      _Rclose(in);
}
```

Program wyszukuje rekord, gdzie klucz główny to *RYSIEK* i zamienia dane imię na *ADAM*. Do odczytania danych służy funkcja *_Rreadk()*. Następnie sprawdzany jest warunek poprawnego odczytania rekordu. W przypadku powodzenia zmieniana jest jego zawartość funkcjami *sscanf()* oraz *sprintf()*. Do ustawienia się na żądanej pozycji należy się posłużyć funkcją *_Rlocate()*, natomiast do aktualizacji funkcją *_Rupdate()*.

2.4. Opis składni przykładowej funkcji

Poniżej pokazana jest składnia przykładowej funkcji *_Rlocate()*, która posłużyła do umiejscowienia się na żądanym rekordzie:

_Rlocate() - Pozycjonowanie rekordu

```
#include <recio.h>
_RIOFB_T _Rlocate(_RFILE fp, void* key, int klen_rrn, int opts);
```

Opis:

Funkcja pozycjonuje rekord w pliku *fp* skojarzonym z kluczem *key* oraz parametrem *opts*. Rekord jest blokowany, chyba że parametr *opts* przyjmie wartość *_NO_LOCK*.

key - wskazuje na łańcuch zawierający pozycje kluczowe używane do pozycjonowania.

klen_rn - określa długość klucza służącego do pozycjonowania, jeżeli pozycjonowanie odbywa się po kluczu, w przypadku używania względnego numeru rekordu określa numer rekordu.

opts – specyfikuje opcje do operacji lokalizowania.

Wykaz możliwych wartości tego parametru :

- *__DFT* - domyślna wartość to *__KEY_EQ*, blokuje rekord do aktualizacji.
- *__END* - ustawia się tuż za ostatnim rekordem w pliku.
- *__END_FRC* - ustawia się tuż za ostatnim rekordem w pliku. Wszystkie buforowane zmiany są ostateczne. Nie ma rekordu złączonego z tą pozycją.
- *__FIRST* - ustawia się na pierwszy rekord w pliku. Parametr *key* jest ignorowany.
- *__KEY_EQ* - ustawia się na pierwszy rekord dla określonej wartości klucza.
- *__KEY_GE* - ustawia się na pierwszy rekord większy lub równy określonej wartości klucza.
- *__KEY_GT* - ustawia się na pierwszy rekord większy od określonej wartości klucza.
- *__KEY_LE* - ustawia się na pierwszy rekord mniejszy lub równy określonej wartości klucza.
- *__KEY_LT* - ustawia się na pierwszy rekord mniejszy od określonej wartości klucza.
- *__KEY_NEXTEQ* - ustawia się na następny rekord, który ma klucz równy wartości klucza o długości *klen_rn* na obecnej pozycji. Parametr *key* jest ignorowany.
- *__KEY_NEXTUNQ* - pozycjonuje się na następny rekord z unikalnym kluczem z obecnej pozycji w indeksie.
- *__KEY_PREVEQ* - pozycjonuje się na poprzedni rekord, który ma klucz równy wartości klucza o długości *klen_rn* na obecnej pozycji. Parametr *key* jest ignorowany.
- *__KEY_PREVUNQ* - pozycjonuje się na poprzedni unikalny rekord z aktualnego indeksu. Parametr *key* jest ignorowany.
- *__LAST* - pozycjonuje się na ostatni rekord w indeksie. Parametr *key* jest ignorowany.
- *__NEXT* - pozycjonuje się na następny rekord w indeksie. Parametr *key* jest ignorowany.
- *__PREVIOUS* - pozycjonuje się na poprzedni rekord w indeksie. Parametr *key* jest ignorowany.
- *__RRN_EQ* - ustawia się na rekordzie, którego numer względny jest równy parametrowi *klen_rn*.

- *__START* - ustawia się tuż przed pierwszym rekordem w pliku. Nie ma rekordu związanego z tą pozycją.
- *__START_FRC* - ustawia się tuż przed pierwszym rekordem w pliku. Nie ma rekordu związanego z tą pozycją. Wszystkie buforowane zmiany są trwałe.
- *__DATA_ONLY* - ustawia się tylko na rekordy z danymi. Rekordy usunięte będą ignorowane.
- *__KEY_NULL_MAP* - ustawienie parametru *key* na *NULL* wtedy, gdy szukane są rekordy według klucza.
- *__NO_LOCK* - rekord, na którym się pozycjonujemy, nie będzie blokowany.
- *__NO_POSITION* - pozycja w pliku pozostaje nie zmieniona, ale znalezione rekordy będą blokowane, jeżeli plik jest otwarty do aktualizacji.
- *__PRIOR* - ustawia się tuż przed żądanym rekordem .

Jeżeli określi się startowe i końcowe opcje *(__START, __START_FRC, __END lub__END_FRC)* z innymi opcjami, to inne opcje są pomijane. Funkcja *_Rlocate()* zwraca wskaźnik na strukturę *_RIOFB_T* skojarzoną z *fp*. Jeżeli operacja powiodła się, to pole *num_bytes* struktury zawiera 1. Jeżeli są użyte opcje *__START, __START_FRC, _END* lub *__END_FRC*, to pole *num_bytes* jest ustawione na *EOF*. W przypadku błędu pole to ma wartość 0. Pola *key* oraz *rm* są aktualizowane, a pole *key* będzie zawierało cały klucz, nawet jeśli jego część jest określona.

2.5. Transakcyjność w bazie DB2

Transakcyjność jest to grupowanie operacji na plikach w pojedyncze moduły, tak by synchronizować zmiany plików w ramach jednego zadania. Zanim się rozpocznie transakcję, należy się upewnić, że wszystkie pliki, które są przetwarzane, znajdują się w jednym środowisku transakcyjnym. Wszystkie pliki w takim środowisku muszą być śledzone z użyciem jednego dziennika. Do stworzenia takiego pliku dziennika należy się posłużyć komendą CL *CRTJRNRCV (Create Journal Receiver), CRTJRN(Create Journal)* i *STRJRNPF (Start Journal Physical File)*.

Gdy środowisko transakcyjne jest przygotowane, należy zastosować następującą sekwencję komend:

Start Commitment Control (STRCMTCTL)

CALL <nazwa_programu>

End Commitment Control (ENDCMTCTL)

W poniższym przykładzie pokazany zostanie sposób na wpisywanie do dzienników systemowych zmian zachodzących w pliku *OSOBAF*.

Przykład

Aby stworzyć obiekt odbiorcy dziennika *(journal receiver) JRNRCV*:

CRTJRNRCV JRNRCV(LABLIB/JRNRCV)

Aby stworzyć obiekt dziennika *JRN :*

CRTJRN JRN(LABLIB/JRN) JRNRCV(LABLIB/JRNRCV)

Rozpoczęcie zapisywania do dziennika zmian w pliku *OSOBAF*:

*STRJRNPF FILE(LABLIB/OSOBAF) JRN(MYLIB/JRN) IMAGES(*BOTH)*

Kod programu wywołujący operację *COMMIT* na pliku *OSOBAF* to:

```
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#define _RCDLEN 50
int main(void)
{
 _RFILE *in;
 _RIOFB_T *fb;
 int warunek = 1;
 char record[_RCDLEN];
 /* Otwarcie pliku wejściowego do zapisu*/
 if ( (in = _Ropen("LABLIB/OSOBAF", "wr")) == NULL )
 {
   printf("Otwarcie pliku wejściowego nie powiodło się\n");
   exit(1);
 };
 /*zapisanie do pliku danych podanych przez użytkownika*/
 while (warunek)
 {
   char imie[10],nazwisko[20],miasto[20];
   printf("\nProszę podać imię? ");
   scanf("%10s",imie);
   printf("\nProszę podać nazwisko? ");
   scanf("%20s",nazwisko);
   printf("\nProszę podać miasto? ");
   scanf("%20s",miasto);
   if (strlen(imie)>0)
   {
     sprintf(record,"%10s%20s%20s",imie,nazwisko,miasto);
     int wynik = _Rwrite(in, record, _RCDLEN);
     if (wynik)
        _Rcommit("Transakcja zakończona ");
     else
        _Rrollbck();
   }
   else warunek = 0;
 }
 _Rclose(in);
}
```

3. Zakończenie

AS/400 jest platformą przeznaczoną głównie do zastosowań bazodanowych. W artykule zaprezentowano sposób dostępu do bazy danych z użyciem języka C oraz funkcji bibliotecznych na kilku prostych przykładach. Przestawiony w pracy sposób dostępu to dostęp natywny do bazy danych bez wykorzystania języka SQL. Artykuł ten powstał jako rezultat pracy z komputerem AS/400, w celu lepszego poznania możliwości tej platformy w zastosowaniach bazodanowych. Użycie języka C jako narzędzia zostało podyktowane jego szeroką znajomością. Jest to jeden z najszybszych języków dla AS/400, jednak pod względem powszechności zastosowań ulega językowi RPG/400, który jest najczęściej stosowany w tworzeniu aplikacji dla systemu operacyjnego OS/400.

LITERATURA

1. IBM Redbooks: DB2 for AS/400 Database Programming.
2. IBM Redbooks: DDS Reference.
3. IBM Redbooks: ILE C/C++ for AS/400 Run-Time Library Reference.
4. IBM Redbooks: ILE C for AS/400 Programmer's Guide.
5. IBM Redbooks: ILE C for AS/400 Language Reference.
6. IBM Redbooks: ILE C/C++ for AS/400 Run-Time Library Reference.

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do Redakcji 23 stycznia 2003 r.

Abstract

AS/400 is a platform assigned mainly for database applications. In this article a method is shown for accessing database using C language and library functions in a few simple examples: adding, displaying and changing records. The method introduced in this article is a native access without using SQL language. This paper was written as a result of work with an AS/400 system for better recognition of its possibilities in database applications.

Adres

Lesław PAWLACZYK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, palles@polsl.gliwice.pl .