Database access for illiterate programmers

K.B.Swiatlowski

Abstract

Writing an SQL statement can be difficult for people used to accessing data stored in flat files. Furthermore existing software code may already have a lots of places where flat file interfaces have been used. In order to avoid re-writing existing software and providing transparent, flat-file-like access to SQL database the functionality of file access words had to be extended. This approach to databases will be presented.

K.B.Swiatlowski B.Sc.
Micross Electronics Ltd.,
Units 4-5, Great Western Court,
Ross-on-Wye, Herefordshire.
HR9 7XP U.K.
Tel. +44 1989 768080
Fax. +44 1989 768163
Email. kbs@micross.co.uk

Flat file system and migration to DB

A Database table is a collection of records stored in a computer in a systematic way. A set of flat files can be a database too but in saying database [DB] we have in mind a set of tables with an access via some kind of a program allowing us to execute SQL commands.
Our old system used flat files to store and access records of information usually in the same sort of way.

As the first stage there is a declaration of a record structure:

```
STRUCT _STR1                        \ Structure name
  LENGHT1   FIELD STR_NAME1         \ Fields declaration
  LENGHT2   FIELD STR_NAME2
* * *
END-STRUCT

CREATE MEMPLACE _STR1 HOWMANY * ALLOT   \ File image place
PCB STR1PCB                             \ A path control block
```

In the simplest example of reading from the file we execute a word for a fixed size of file. It is possible to get the size of a file and allocate dynamic memory.

```
: READIN-STR1 ( -- ) \ Read in data for the structure 1
  Z"" C:\Filename.dat" STR1PCB SET-ZPATHNAME \ Provide the file name
  STR1PCB OPEN-PATH-PCB 0= IF               \ Open file
    MEMPLACE                                \ Provide mem address
    _STR1 HOWMANY *                         \ Requested size
    STR1PCB @ READ-PATH 2DROP               \ File handle, read!
  THEN
  STR1PCB CLOSE-PATH-PCB DROP               \ Close the file
;
```

All data interpretation used to be done inside the program which makes it hard to develop. Varying expectations from different customers pushed us into SQL DB with its flexible language; easy for data manipulation and retrieving information.

Migration from flat-file to SQL DB had to meet several restrictive criteria. Modification of the code should not influence already existing parts of the program and its functionality. At the beginning there was a need for an easy switch between flat file and DB method. Moving from flat file to new table in DB should be fast and easy, done in one place if possible. An ODBC interface has been used to execute DB commands on MySQL DB.

Change

Manipulation of data in file is divided into at least 3 stages:

- Opening the file.
- Reading or writing from the file.
- Closing the file.

If accessing corresponding DB table these operations can be mapped to:

- Acquiring access to DB (Opening connection or taking opened connection from the pool).
- Executing SQL command for reading or writing to DB.
- Freeing access to DB by closing connection or returning available connection to the pool.

Our company (Micross) uses the PFW compiler of MPE. In order to provide flat-file-like access to DB their words has been redefined and extended with a set of ODBC functions. The most important redefined FORTH words are presented below:

- OPEN-PATH-PCB, WINCREATEFILE - both words are use to ensure connection to DB.
- SET-PATHNAME - Indicates the table the programmer wants to access by providing the earlier registered file name (or file extension) - which is unique in the system. It allows finding the correct SELECT statement for read and INSERT statement for write operations and all settings for the table which corresponds to that file.
- CLOSE-PATH-PCB - Returns DB access point (connection handle) and frees allocated memory for temporary data.
- WINGETFILESIZE - Returns the size of the file in bytes. In fact, the number of records times the size of a record. This could be achieved by executing the query "SELECT COUNT(*) FORM table WHERE <<filter_conditions>>" but since in our system the next operation is always reading, this has been use to read the content of the table and return the size of data in bytes. This approach spares one query to DB.
- SEEK-PATH-PCB - Moves file pointer to selected record.
- WRITE-PATH - Executes INSERT query moving data from memory to DB.

- READ-PATH - Executes SELECT query and moves dataset to pointed memory.
- FIELD - Stores the size of a filed in the record structure. The type of the field is taken from DB.
- ARRAY-OF - Works similar to FIELD word.

A few new words and structures have been added, of which the most important are:

- STRUCT-SQL – Allocates memory for table configuration data such as: field offsets, column types – information gathered during declaration of the fields.
- PREPARETABLE (structsize --) – Ends gathering data at the structure definition stage. Allocates more memory for table name, associated SQL queries and various configuration parameters.
- SETSELECT (select$z where$z order$z --) - Allocates memory for SELECT statement and stores the query. INSERT query is generated and stored at the start of the program. DB interface provides functions to find out number of columns, its type, name and length.
- SETFILENAME (z$--) - Registers a file name or file extension, linking it with table configuration data.
- SETTABLENAME (z$ --) - Saves table name for that structure. Links DB table name with a file name.
- SETONE-TO-N (n --) - Informs that the first column in the table is not a part of the structure defined in FORTH, but N characters of the file name is stored in that column.

Two examples of tables working in our system "Tracknet".

1. Category table

   This holds data of linen processed by the laundry. It has many fields of which only a few are shown. The structure describes an offset in the record of each field i.e.: category name, id number, associated wash program and so on. With this information provided the program can access DB using standard flat file interface. It is the programmer's responsibility to design a table that structure matches the previously defined FORTH structure. Order

of columns, its type and data length should be kept in unison to avoid ambiguity.

```
STRUCT-SQL CATENTRY    \ Declaration of SQL table structure
  LENCATID       FIELD      CATID      \ Category ID
  LENCATNAME 1+  FIELD      CATNAME    \ Name
  LENCATBARCODE  FIELD      CATBARCODE \ Bar code
  * * *
END-STRUCT
CATENTRY PREPARETABLE

: CREATESTATMENT-CAT ( -- z$) \ Creates category table
  Z"" CREATE TABLE IF NOT EXISTS `category` ("
  Z"" `catid` varchar(4) NOT NULL default ''," Z+
  Z"" `catname` varchar(31) NOT NULL default ''," Z+
  * * *
  Z"" PRIMARY KEY (`catid`)" Z+
  Z"" ) ENGINE=MyISAM DEFAULT CHARSET=" Z+ SQLCHARSETZ$ Z+
 ;

: SELECTSTATMENT-CAT ( -- select$z where$z order$z ) \ Select for cat
  Z"" SELECT * FROM CATEGORY "    \ All fields match FORTH structure
  ^NULL                           \ No filter applied
  Z"" ORDER BY CATNAME"           \ Use alphabetic order
;

' CREATESTATMENT-CAT CREATETABLE  \ Saves a word with pointer to Z$
SELECTSTATMENT-CAT SETSELECT      \ Save select
Z"" CATEGORY" SETTABLENAME        \ Say what table it is in DB
Z"" TRCATS.TXT" SETFILENAME       \ Assign file name
```

2. Operator log file

   This holds data of events triggered by an employee. i.e.: logging on and logging off to different parts of the plant, or registering privileged actions. Each operator can have multiple (n) records in during a day. In the flat file system each day had a separate file with a distinctive file name in the format YYMMDD.LOG. This brought us to introduce an extra column called "filename" to separate records for each day/file.

```
STRUCT-SQL OPERATOR-LOG     \ Declaration of SQL table structure
  4 _BYTE        ARRAY-OF   OPLOG-LOCATION
  _INT           FIELD      OPLOG-ACTION
  SYSTEMTIME     FIELD      OPLOG-TIME
  _INT           FIELD      OPLOG-OPERKEY
  * * *
       \ * ! End of INSERT query fileds *     \
  NAMELEN 1+     FIELD      OPLOG-NAME       \ Name (zstring)
END-STRUCT
6 SETONE-TO-N \ First 6 chars of file name is a part of a table key
OPERATOR-LOG PREPARETABLE
```

```
: CREATESTATMENT-LOG ( -- z$)
  Z"" CREATE TABLE IF NOT EXISTS `operlog` ("
  Z""  `filename` int(10) NOT NULL default ''," Z+
  Z""  `replocation` tinyint(3) unsigned NOT NULL default '0'," Z+
  Z""  `repsubevent` tinyint(3) unsigned NOT NULL default '0'," Z+
  Z""  `repevent` tinyint(3) unsigned NOT NULL default '0'," Z+
  Z""  `represerved` tinyint(3) unsigned NOT NULL default '0'," Z+
  Z""  `opaction` int(10) unsigned NOT NULL default '0'," Z+
  Z""  `logsystime` datetime NOT NULL default CURRENT_TIME," Z+
  Z""  `opref` int(10) unsigned NOT NULL default '0'," Z+
  Z""  KEY `Index_1` (`filename`,`opref`)," Z+
  Z""  KEY `Index_2` (`logsystime`)," Z+
  Z""  ) ENGINE=MyISAM DEFAULT CHARSET=" Z+ SQLCHARSETZ$ Z+
;

: SELECTSTATMENT-LOG ( -- select$z where$z order$z )
  \ Select columns in order they appear in structure
  Z"" SELECT
  operlog.replocation,operlog.repsubevent,operlog.repevent,
  operlog.represerved,opaction,logsystime,operlog.opref,operator.name
  FROM operlog, operator "
  \ File name filter will be provided at opening the file
  Z"" WHERE operlog.opref=operator.opref AND filename="
  Z""  ORDER BY logsystime"
;

' CREATESTATMENT-LOG CREATETABLE  \ Save CREATE statement
SELECTSTATMENT-LOG SETSELECT      \ Save select statement
Z"" OPERLOG" SETTABLENAME         \ Say what table it is in DB
Z"" .LOG" SETFILENAME             \ Assign file name (extension)
```

Operator name field was present in the old version of the program. Since there is no need to store operator's name in the log file but only operator reference number, the value for that field is taken from OPERATOR table. During the generation or an INSERT query columns present only in OPERLOG table are taken as valid fields.

Conclusions

Changes done in one place only are invisible in other parts of a program. The programmer accessing the file cannot tell at first glance if it works with DB table or with a flat file. The execution sequence, words, parameters are the same for DB as for flat files. Moving to SQL has brought flexibility and speeded up development of customers reports.

In addition, not every word has been redefined. Delete from the DB table is to be done by an explicit SQL command, not as for flat files by setting an “end of file”.