84

# DB2 update

*October 1999*

## In this issue

# DB2 Update

**Published by**
Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38030
From USA: 01144 1635 38030
E-mail: info@xephon.com

**North American office**
Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

## Contributions

Articles published in *DB2 Update* are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*.

## *DB2 Update* on-line

Code from *DB2 Update* can be downloaded from our Web site at http://www.xephon.com/db2update.html; you will need the user-id shown on your address label.

**Editor**
Robert Burgess

## Disclaimer

## Subscriptions and back-issues

A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1994 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

# Java meets DB2: get there from here – SQLJ

*Editor's note: the author's first article on database connectivity from Java explored JDBC, Java's answer to ODBC. In this second article, the author covers a new technology, SQLJ.*

As explained in *Java meets DB2: get there from here – JDBC*, published in *DB2 Update*, Issue 83, September 1999, JDBC is a low-level, dynamic call-level interface predicated upon the X/Open CLI specification. It's a mature robust technology that can be used in mission-critical applications today. However, JDBC has a number of built-in problems and limitations, most of which can be solved with a standard for embedded static SQL. This is where SQLJ comes in. SQLJ offers the following advantages over JDBC:

- Development-time error checking.
- Smaller source programs.
- Enhanced run-time performance with statically bound SQL.
- Eliminates the need to learn a complicated new API.
- Leverages existing knowledge of embedded SQL programming.

SQLJ was developed by a consortium of major players in the database industry, including IBM, Oracle, Tandem (Compaq), and Sybase. The specification for SQLJ contains three parts:

- Part 0 – describes specifications for embedding SQL statements in Java methods.
- Part 1 – describes specifications for calling Java static methods as SQL stored procedures and user-defined functions.
- Part 2 – describes specifications for using Java classes as SQL user-defined datatypes.

Part 0 was approved by the ANSI X3H2 committee in early 1999, while Parts 1 and 2 are still being finalized. This article will cover only Part 0.

## DEVELOPMENT PROCESS

Because JDBC is part of the core Java API, and hence is written in 100% pure Java, JDBC calls within a Java program are reduced to bytecodes just like the rest of the program – through the invocation of a standard Java compiler. At run-time, these bytecodes are executed interpretively within a Java Virtual Machine. But SQLJ calls are non-standard Java and somehow must be disposed of prior to invoking the Java compiler. This is accomplished via an extra preparation step in which the SQLJ calls are searched for and replaced with some other database access API, whether it be JDBC, ODBC, or a native API such as DSNHLI. (Long-time mainframe DB2-COBOL programmers should be very familiar with the concept.) Which database access API is actually used at run-time is not formally defined in the ANSI specification; it depends upon the implementation of SQLJ that is being used. Figure 1 compares the two processes.

## SQLJ SYNTAX

Static SQL in Java appears in SQLJ clauses. Each SQLJ clause begins with the token #sql, which is not a legal Java identifier, and so makes the clause recognizable in Java programs. The simplest SQLJ executable clauses consist of the token #sql followed by an SQL statement enclosed in curly brackets, followed by a semi-colon. See the following code:

```
import sqlj.runtime.*;
public class SQLJExample1 {
  public static void main(String args[]) throws Exception {
    Class.forName("COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver");
    String v, m;
    int y;
    v = new String("BEA16TYE4HGD99");
    #sql {SELECT MAKE, YEAR FROM INV_TAB
                 INTO :m, :y WHERE VIN = :v };
    System.out.println("Make is " + m + " year is " + y);
    y = 1997;
    #sql { UPDATE INV_TAB SET YEAR = :y WHERE VIN = :v };
    m = new String("Mercedes-Benz");
    y = 1990;
    v = new String("VHS3Y7EWQS8M22");
    #sql { INSERT INTO INV_TAB VALUES (:v, :y, :m) };
```

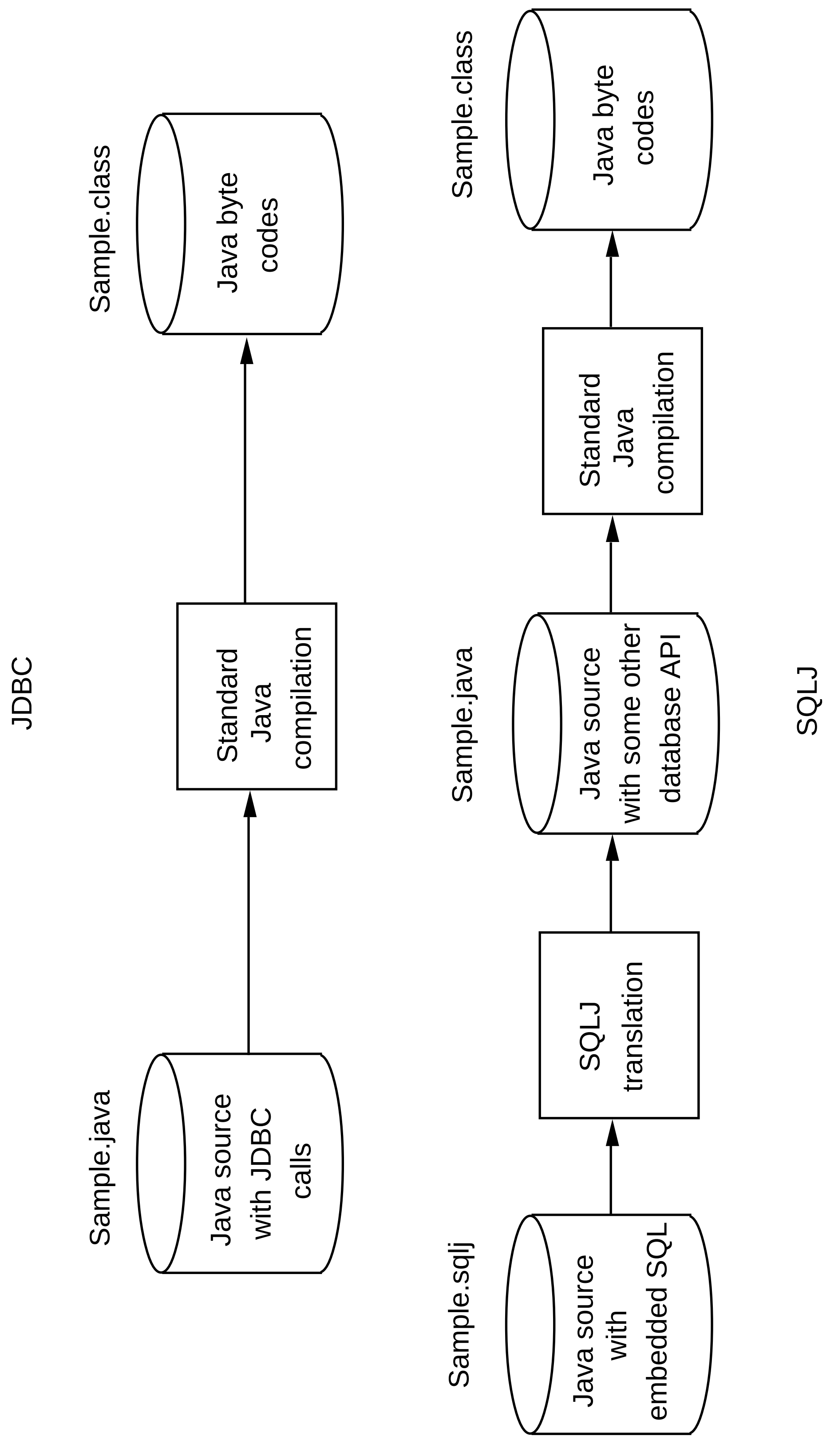
JDBC
Sample.java
Java source with JDBC calls
Standard Java compilation
Sample.class
Java byte codes
SQLJ
Sample.sqlj
Java source with embedded SQL
SQLJ translation
Sample.java
Java source with some other database API
Standard Java compilation
Sample.class
Java byte codes

*Figure 1: JDBC versus SQLJ development steps*

```
    #sql { DELETE FROM INV_TAB };
  }
}
```

In an SQLJ clause, the tokens inside the curly brackets are SQL tokens, except for the host-variables, which are marked by colon characters. SQL tokens never occur outside of the single pair of curly brackets in an SQLJ statement.

You'll notice a driver being loaded, just as with JDBC. In the example above, the driver being loaded is the SQLJ driver developed by IBM specifically for use with DB2 for OS/390.

Assume that the table used in the example has a primary key of Vehicle ID Number (VIN). That would imply that the SELECT statement shown above would never return more than one row. But suppose we had a different WHERE clause, or none at all? How would a multi-row result set be processed in SQLJ?

The answer is through the use of something called an SQLJ result-set iterator. A result-set iterator is a Java object from which the data returned by an SQL query can be retrieved. In that role, it corresponds to the cursors used in standard embedded SQL, from which data is fetched. Unlike the cursor, however, an iterator is a first-class Java object. An iterator can be passed as a parameter to a method, and can be used outside the SQLJ translation unit that creates it, without losing its static type for the purposes of type checking of component interfaces.

An iterator has one or more columns with associated Java types. Names that are Java identifiers can optionally be provided for the iterator columns. The columns of an iterator (which have Java types) are conceptually distinct from the columns of a query (which have SQL types), and therefore, a means of matching one to the other must be chosen. Two mechanisms are supported for matching iterator columns to query columns:

- Bind-by-position
- Bind-by-name.

The iterator declaration clause indicates whether the iterator is using bind-by-position or bind-by-name. (The two styles of access to result set data are mutually exclusive; an iterator class supports either bind-by-position or bind-by-name, but not both.)

Bind-by-position means that the left-to-right order of declaration of the iterator columns places them in correspondence with the expressions selected in an SQL query. Traditional FETCH...INTO syntax is used to retrieve data from the iterator object into Java variables, as shown in the following code:

```
import sqlj.runtime.*;
public class SQLJExample2 {
  public static void main(String args[]) throws Exception {
    Class.forName("COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver");
    #sql iterator ByPos (String, int);
    ByPos result;  // declare object of type ByPos
    String m;
    int y;
    #sql result = {SELECT MAKE, YEAR FROM INV_TAB};
    while (true) {
       #sql { FETCH :result INTO :m, :y };
       if (result.endFetch())
               break;   // exit the infinite loop
       System.out.println("Make is " + m + " year is " + y);
    }
  }
}
```

Bind-by-name means that the name of each iterator column is matched to the name of a column returned by the SQL query, independent of the order in which they appear in the query. Named no-arg accessor methods are automatically generated for the columns of the iterator in this case. The name of an accessor method matches the name of a column returned by a query. Its return type is the Java type of the iterator column. FETCH may not be used with iterators of bind-by-name types. See the following code:

```
import sqlj.runtime.*;
public class SQLJExample3 {
  public static void main(String args[]) throws Exception {
    Class.forName("COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver");
    #sql iterator ByName (int YEAR, String MAKE);
    ByName result;  // declare object of type ByName
    String m;
    int y;
    #sql result = {SELECT MAKE, YEAR FROM INV_TAB};
    while (result.next()) {
        m = result.MAKE();  // standard method call
        y = result.YEAR();  // standard method call
        System.out.println("Make is " + m + " year is " + y);
    }
  }
}
```

## CONNECT WHERE?

The one critical piece missing from the code examples shown so far is the DB2 subsystem on which the SQL is going to run.

In SQLJ terminology, all statements execute within a connection context. And it’s the connection context that specifies the location and, by extension, the subsystem.

There are four different ways to establish a connection context:

- Using a command line parameter:

  ```
  $ sqlj -url=jdbc:db2os390:ORLANDO SQLJExample1.sqlj
  ```

- Coding an SQLJ properties file and using it at translation time:

  ```
  $ sqlj -props=sqlj.properties SQLJExample1.sqlj
  ```

  where ‘sqlj.properties’ is an ASCII text file that might look like this:

```
# specify jdbc-style URL
sqlj.url=jdbc:db2os390:ORLANDO
# specify a user name and password
sqlj.user=whgates
sqlj.password=borg
```

- Establishing a default connection context programmatically:

```
import java.sql.*;
import sqlj.runtime.*;
public class SQLJExample4 {
  public static void main(String args[]) throws Exception {
    Class.forName("COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver");
    #sql context Ctx1;
    Ctx1 prod = new Ctx1(DriverManager.getConnection(
                "jdbc:db2os390:ORLANDO"));
    DefaultContext.setDefaultContext(prod);
    String v, m;
    int y;
    v = new String("BEA16TYE4HGD99");
    #sql {SELECT MAKE, YEAR FROM INV_TAB
```

```
            INTO :m, :y WHERE VIN = :v };
    System.out.println("Make is " + m + " year is " + y);
    y = 1997;
    #sql { UPDATE INV_TAB SET YEAR = :y WHERE VIN = :v };
  }
}
```

A default connection context is established by calling the method setDefaultContext() defined on the DefaultContext class. When calling this method you will need to pass along a valid context object; this is done in the previous code example with the variable called 'prod'. This is created as an instance of Ctx1, which in turn was declared in a prior #sql clause as a type of context. Note the DriverManager class is being used just as was seen in the first article in this series.

- Use an explicit connection context on a per statement basis:

```
import java.sql.*;
import sqlj.runtime.*;
public class SQLJExample5 {
  public static void main(String args[]) throws Exception {
    Class.forName("COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver");
    #sql context Ctx1;
    Ctx1 prod = new Ctx1(DriverManager.getConnection(
              "jdbc:db2os390:ORLANDO"));
    Ctx1 test = new Ctx1(DriverManager.getConnection(
              "jdbc:db2os390:GLENVIEW"));
    String v, m;
    int y;
    v = new String("BEA16TYE4HGD99");
    #sql [prod] {SELECT MAKE, YEAR FROM INV_TAB
                  INTO :m, :y WHERE VIN = :v };
    System.out.println("Make is " + m + " year is " + y);
    y = 1997;
    #sql [test] { UPDATE INV_TAB SET YEAR = :y WHERE VIN = :v };
  }
}
```

In this code, no default connection context is established. Instead, two explicit contexts are created, one for the production subsystem and the other for test. These objects are referenced within square brackets immediately after the #sql delimiter.

There is no limit on the number of contexts that can be created and used within a single application or applet. However, beyond a certain limit things might get confusing for the maintenance programmer.

Note that if a default context has been established, that default can be referenced within square brackets, but it is obviously not required – it’s your choice.

## CONCLUSIONS

SQLJ offers a robust alternative to JDBC. Although JDBC has a head start in the marketplace, look for SQLJ to close the gap rapidly as more of the database vendors climb on board the bandwagon.

IBM has already established itself as a leader in the Java market and is moving aggressively to bring Java-based solutions into the world of DB2 databases.

Java is the future of computing. With JDBC and SQLJ, DB2 will occupy an important place in that future.

## FURTHER READING

- *Application Programming Guide For Java*, from IBM at http://www.software.ibm.com/data/db2/os390/pdf/javadb5.pdf.
- *DB2 UDB SQLJ Support*, from IBM at http://www.software.ibm.com/data/db2/java/sqlj/index.html.
- IBM’s Java Enablement page at http://www.software.ibm.com/data/db2/java.
- SQLJ Consortium Web site at http://www.sqlj.org.
- *Essential SQLJ Programming* by J Basu and J Shome, John Wiley & Sons; ISBN: 0471349208.
- *Understanding the New SQLJ* by J Melton and A Eisenberg, Ap Professional; ISBN: 1558605622.

*Editor’s note: if you would like to discuss this article further, the author can be contacted at jbradford@gr.com.*

*John T Bradford*
*Greenbrier and Russel (USA)*

## Taming the traces

DB2 traces are an excellent way of analysing DB2 systems and application performance objectives, and diagnosing problems. The analysis of these traces is complicated. While there are numerous products available on the market offering on-line monitoring, batch report, exception analysis, etc, they do not necessarily provide the desired reports. Getting to know how to format, analyse, and generate custom reports from DB2 traces using one's own programs gives great power and flexibility to a DBA.

I have found that writing programs (especially using REXX) to format and analyse traces, for certain customized needs, is very helpful. This article presents various techniques I have used to achieve this. It only contains information about SMF and OP destinations. All the REXX code presented is for sample purposes only and could be non-optimized.

### DB2 TRACE

DB2's Instrumentation Facility Component (IFC) provides a trace facility that is used to record DB2 data and events. The trace records can be written to SMF, GTF, SRV, or On-line Performance (OP) buffer (provided by DB2) destinations. To analyse and format these records, you need to retrieve them from SMF, GTF, or OP buffers and format them outside DB2.

DB2 traces are categorized into six types, based on the information they retrieve:

- Statistics
- Accounting
- Audit
- Monitor
- Performance
- Global.

Each of these categories is further divided into various classes. For example, in order to know wait times for a thread, you need to start ACCOUNT CLASS(3) trace. The trace is started using the START TRACE command. Starting a trace for any particular class triggers DB2 to write particular types of information requested. DB2 writes various types of record and they are identified by many Instrumentation Facility Component Identifiers (IFCIDs). The details of IFCIDs are described, along with the comments in their mapping macros, contained in prefix.SDSNMACS, which is shipped with DB2. These mapping macros are written in Assembler and PL/I, making it difficult to format for anyone who knows only REXX or COBOL. This article provides helpful tips to format DB2 traces without knowing Assembler.

The table in Figure 1 describes various IFCIDs that are triggered when the START TRACE command is issued. For simplicity, the table contains only a few common events. For a complete description of all the trace types and classes, please refer to the *DB2 Administration Guide*.

## TRACE DESTINATION

The destination of trace records is found using the DB2 command START TRACE(?) DEST(SMF). The most commonly used destinations are SMF, GTF, and OP buffers.

SMF collects and records system and job-related information that your installation can use in billing users, reporting reliability, analysing the configuration, scheduling jobs, summarizing direct access volume activity, evaluating dataset activity, etc. Because this facility is available for various subsystems in the installation, this destination should be used with care. Heavy volume trace data should avoid using SMF as a destination.

Normally, SMF is used for statistics and accounting traces, because some installations prefer running these traces all the time. There could be two or more datasets where SMF writes the records. SMF data-gathering routines fill predefined datasets one at a time. While the gathering routines write records on one dataset, SMF can write out or clear the others. SMF continues to write records for as long as it can find an empty inactive dataset when the active dataset becomes full.

| *Type* | *Class* | *Data collected* | *IFCIDS activated* |
|---|---|---|---|
| Statistics | 1 | Statistical data | 1-2, 105, 106, 202 |
| | 3 | Deadlock and timeout, connect or disconnect from a group buffer, long running tasks | 152, 172, 196, 250 261, 262, 313 |
| Accounting | 1 | Accounting data | 3, 106, 239 |
| | 2 | In DB2 time | 232 |
| | 3 | Various waits | 6-7, 8-9, 32-33, 44-45 , (51-52, 56-57),117-118 127-128, 170, 171, 174-175, 213-214, 215-216, 226-227, 242-243 |
| | 7 | Package level accounting | 232,240 |
| | 8 | Package level wait time | Same as in Accounting Class 3 |
| Audit | 1 | Authorization failures | 140 |
| Performance | 30 | No specific events | This triggers the IFCID mentioned in the IFCID() keyword of START TRACE command |

*Figure 1: IFCIDs triggered by START TRACE command*

SMF dataset definition:

```
DEFINE CLUSTER  (NAME(SYS1.MAN1) VOLUME(xxxxxx)  NONINDEXED          +
CYLINDERS(1ØØ)   REUSE   RECORDSIZE(4Ø86,32767)  SPANNED   SPEED      +
CONTROLINTERVALSIZE(4Ø96)  SHAREOPTIONS(2))
```

All SMF records written have an SMF Type to identify them. DB2 records have SMF Type 100,101, and 102. SMF Record type 100 is for statistics trace, 101 for accounting, and 102 is for monitor, performance, and audit traces.

## SMF DUMP PROGRAM

When one SMF dataset becomes full, the SMF writer switches to the next dataset and the previous one is ready to be cleared. The program IFASMFDP is used to clear out the filled SMF VSAM dataset and dump the contents into sequential datasets on either tape or DASD. This entire process is automated in most installations.

The output sequential datasets (where the SMF Dump program has dumped the data) can be retained for a long time and from here we can easily extract the DB2 trace records. These sequential datasets are defined with a record format of Variable Block Spanned (VBS) and record length of 32,760 or 32,767. The block size is installation-dependent and can be 4,096.

Most DBAs are probably not used to handling VBS record format and, moreover, VBS format datasets cannot be read/browsed directly using REXX or ISPF services. The easiest way to format DB2 records is to extract them from SMF dump output and write them into another dataset with VB or FB format. For example:

```
/STEP1 EXEC PGM=SORT
//SORTIN   DD    DSN=<mention SMF Dump dataset name>,DISP=SHR
//SORTOUT  DD DSN=<trace data, input to rexx>,DISP=(NEW,CATLG,DELETE),
//         UNIT=SYSDA,SPACE=(CYL,(2,15),RLSE),
//         LRECL=8188,BLKSIZE=8192,RECFM=VB,DSORG=PS
//SORTWK1  DD DISP=(NEW,DELETE,DELETE),
//         SPACE=(CYL,(55,5Ø)),UNIT=SYSDA
//SORTWK2  DD DISP=(NEW,DELETE,DELETE),
//         SPACE=(CYL,(55,5Ø)),UNIT=SYSDA
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD  *
  SORT FIELDS=(7,4,CH,A)
  INCLUDE COND=(6,1,CH,EQ,X'65')
/*
```

This JCL not only writes the content from SMF Dump output to variable block datasets for a REXX program to read, but also eliminates all unnecessary records from SMF and writes only DB2 record type 101 to SORTOUT in an efficient way.

The parameter INCLUDE COND=(6,1,CH,EQ,X'65') in SYSIN is used to choose only SMF 101 (accounting trace data). The SMF record is positioned on the sixth byte in the record and it is represented in binary form. Record type 101 is represented as X'65'. The SMF record header is discussed later in this article.

Please note that the records in the dataset mentioned under DDname SORTOUT will appear shifted to the left by four bytes. This means the SMF record type X'65' will appear in the second column instead of the sixth. Similarly, all the fields will appear at a position which is 4 bytes earlier than you would expect.

Another point is that the shortest records in the SMF DUMP output (which are the header and the trailer records) are 18 bytes long. Therefore, you should not try to sort on a field which is beyond 18 bytes. For example, the following SYSIN card is wrong, because it is trying to sort on a field which is beyond the shortest record in the input dataset.

```
//SYSIN DD *
SORT FIELDS=(2Ø,4,CH,A)  ——WRONG———
```

So how do you extract SMF record type 100 for DB2 subsystem DB2D?

SMF record type 100 can be represented as X'64'. The subsystem name appears on position nineteen in the SMF header. So the SORT control card will appear as follows:

```
INCLUDE COND=(6,1,CH,EQ,X'64',AND,19,4,CH,EQ,C'DB2D')
```

## OTHER WAYS TO READ DATA FROM SMF DUMP

You could consider writing a COBOL program to read the SMF Dump dataset. The FILE-CONTROL SECTION appears as follows, to read a QSAM dataset with VBS format:

```
FD      SMFDATA
        LABEL RECORDS ARE STANDARD
        RECORDING MODE IS S
        RECORD CONTAINS 18 TO 32767 CHARACTERS
        BLOCK CONTAINS Ø CHARACTERS
Ø1      SMFDATA-REC                    PIC X(32767).
```

All other considerations remain the same as when you read any other dataset through your COBOL program. Here again the record read into your program will not show Record Descriptor Word (RDW). This means all the fields will be at a position four bytes earlier than you would expect. For example, the SMF header date field, which starts at the eleventh byte, would appear inside the COBOL record at the seventh byte.

An Assembler programmer could use the BFTEK=A option to read a VBS QSAM dataset and the DCB will appear as below:

```
INDATA  DCB   DSORG=PS,MACRF=GL,DDNAME=SMFIN,EODAD=RETURN,             X
              RECFM=VBS,LRECL=32767,BLKSIZE=4Ø96,BFTEK=A
```

Assembler programmers do not have to compensate for 4 bytes RDW and can easily map any field using mapping macros provided by IBM.

## ON-LINE PERFORMANCE BUFFERS

DB2 provides eight performance monitor destinations – OPn, where n is equal to a value from 1 to 8. Typically, the destination of OPn is only used with commands issued from a monitor program. For example, the monitor program can pass a specific on-line performance monitor destination (eg OP1) on the START TRACE command to start asynchronous trace data collection.

The OP destination is very useful for any kind of trace, either low volume or high volume data, but at the same time it is more complicated to handle. As mentioned earlier, in most cases you need to use some kind of monitor program. However, DB2’s stand-alone utility, DSN1SDMP, is a very handy-to-use OP destination. For example, if you want to start trace performance class 30 (which is a default category) with IFCID(22,59,60,61,63,58), and want to collect up to 5,000 records into an output dataset defined by you, then you could use the following JCL:

```
//OPTRACE    EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
//STEPLIB  DD DISP=SHR,DSN=DSN51Ø.SDSNLOAD
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SDMPPRNT DD SYSOUT=*
//SDMPTRAC DD DISP=(NEW,CATLG,CATLG),DSN=<output trace dataset>,
//            UNIT=SYSDA,SPACE=(8192,(1ØØ,1ØØ)),DCB=(DSORG=PS,
//            LRECL=8188,RECFM=VB,BLKSIZE=8192)
//SDMPIN   DD *
  START TRACE(P) CLASS(3Ø) IFCID(22,59,6Ø,61,63,58) DESTINATION(OPX)
FOR(5ØØØ)
/*
```

```
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DSN)
 RUN PROG(DSN1SDMP) PLAN(<any valid plan name>)
 END
/*
//*
```

In this JCL, the DSN1SDMP needs a plan to run; although it does not execute any SQL, it needs the plan name to establish a thread with DB2. The DDname SDMPIN contains a valid start trace command without the hyphen (-). The keyword FOR is used to specify how many records DSN1SDMP should look for. Here, the ACTION keyword is missing because we are only interested in writing the trace records into the output dataset mentioned in DDname DSMPTRAC. This dataset can be used as input to a REXX or COBOL program, which will then format the traces and produce the desired reports.

Those familiar with Assembler can write their own monitor program that establishes a thread with DB2, issues the start trace command using DSNWLI2, and waits until buffers are filled to a specified threshold and DB2 posts the ECB. This program will then need to issue a READA command to read trace data from OP buffers and write into an output dataset.

To keep this article simple, I am not listing the Assembler code for a monitor program. The *DB2 Administration Guide* has good guidelines and useful information on how to do this. I was able to make my monitor program work with very little effort.

The job priority plays an important role here. The user-written monitor program, or DSN1SDMP, should run with equal or higher priority than DB2 itself, otherwise there is a possibility of losing data from the OP destination. The trace data is written in wrap-around fashion in OP buffers and can get overwritten if buffers are full. Alternatively, you could use third-party monitors, which allow you to start traces on the OP destination and download into user-specified datasets – for example, the Supertrace utility of TMON, etc.

## DESTINATION OPN VERSUS OPX

When you specify any trace destination from OP1 to OP8, DB2 starts a trace on that particular OP buffer. However, if there is already a trace running on that destination, there is the possibility of over-writing someone else’s trace data. Therefore, it is best to specify a generic destination OP*x*, in which case DB2 will find the first empty buffer slot in one of the eight OP buffers and start the trace there.

## FORMAT OF TRACE RECORDS

A typical trace record (accounting trace), together with the exploded record, is shown in Figure 2. The trace record contains the following:

- Header – the header of the trace record depends on the destination type and could contain the SMF header, the IFI header, or the GTF header based on the destination specified in the start trace command.
- Self-defining section – this section resides next to the header and contains a pointer to the product section followed by as many

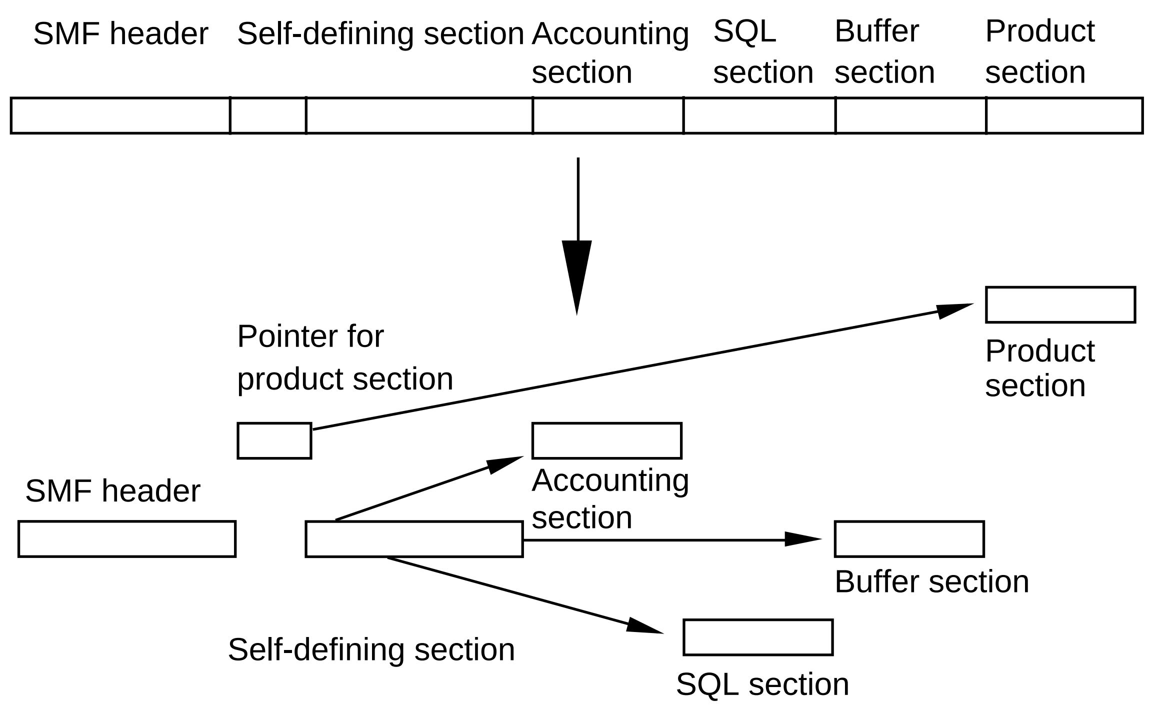

*Figure 2: A typical trace record*

pointers as data sections. The pointer contains triplet fields – offset of the section, length of the section, and number of times the section is repeated.

- Data section – one or more data sections follow the self-defining section and they have to be tracked down using pointers provided in the self-defining section. These data sections contain the actual trace data, requested through the START TRACE command.
- Product section – the last section of the trace record is the product section and the pointer provided in the self-defining section tracks its position. The product section for all record types contains the standard header. The standard header contains information like IFCID, number of data sections (self-defining sections), Subsystem-id, etc. Other headers – correlation, CPU, distributed, and data sharing data – may also be present. The information in the product section header is controlled through TDATA options of the START TRACE command. TDATA specifies the product section headers to be placed into the product section of each trace record. If you do not specify TDATA, then the type of trace determines the type of product section header. The product section of a trace record can contain multiple headers.

All IFC records have a standard IFC header. The correlation header is added for accounting, performance, audit, and monitor records. The trace header is added for serviceability records.

## HOW TO FORMAT TRACE RECORDS

Having extracted trace records from SMF or OP into a specified dataset, you now need to format them. Writing REXX code is the easiest way to do this; alternatively, you could use COBOL code.

### REXX

Using REXX, it is easy to format your trace data and generate customized reports. The description of each IFCID is available in dataset prefix.DSNSAMP(DSNWMSGS) (provided by IBM) and the mapping macros are also available in the dataset prefix.DSNMACS(DSNDQ*).

Understanding these macros requires Assembler or PL/I programming knowledge. However, a person with no prior knowledge of Assembler can understand this with a little practice.

For those not familiar with Assembler programming, taking help from mapping macros provided by IBM is the easiest way to format traces on your own. These macros start with a DSECT, followed by data definition of each variable in the IFCID, and end with MEND.

Here is an example of an Assembler macro to define the self-defining section DSNDQWA0:

```
QWAØ     DSECT
*
*  /*  DB2 SELF DEFINING SECTION MACRO FOR ACCOUNTING IFCID=ØØØ3     */
*  /*  PRODUCT SECTION FOR ACCOUNTING CONTAINS TWO HEADERS
*
QWAØ1PSO DS    AL4           /*OFFSET TO THE PRODUCT SECTION         */
QWAØ1PSL DS    XL2           /*LENGTH OF THE PRODUCT SECTION         */
QWAØ1PSN DS    XL2           /*NUMBER OF PRODUCT SECTIONS            */
```

This is how a REXX programmer could use it:

```
/*  REXX   */
Length_of_SMF_Header = 28 - 4 /*4 bytes reduced to compensate for RDW*/
Offset_self_def_sect = Length_of_SMF_Header
/* position at which self-defining section starts */
cursor_pos = Offset_self_def_sect + 1
QWAØ1PSO = C2D(SUBSTR(input_record, cursor_pos,4))
/* Full word occupies 4 bytes */
cursor_pos  = cursor_pos + 4
Offset_product_sect = (QWAØ1PSO - 4 )
/* Subtract 4 bytes for RDW and add 1 for position of product section */
QWAØ1PSL = C2D(SUBSTR(input_record,cursor_pos,2))
/* halfword occupies 2 bytes, contains length of product section*/
QWAØ1PSL = C2D(SUBSTR(input_record,cursor_pos,2))
/* halfword occupies 2 bytes - times product section is repeated*/
```

Some of the common data types, and their REXX conversions, are shown in Figure 3. Note that the store clock value is the MVS TOD clock, which is stored as 64 unsigned integers. This value contains the time and date in microseconds since 1 January 1900. Normally, a mapping macro will have sufficient documentation to indicate that a field of data type CL8 is a store clock value. The 51st bit is equal to 1 microsecond, hence you need to ignore the last 1.5 bytes of the 8 bytes field. The microsecond value should be converted as below.

| *Assembler data type* | *Comments* | *REXX conversion* |
|---|---|---|
| F | This is float data type | C2D(SUBSTR(record,offset,4)) |
| H | This is halfword data type | C2D(SUBSTR(record,offset,2)) |
| X | This is hexadecimal value | C2D(SUBSTR(record,offset,1)) |
| XL4 | This is hexadecimal value | C2D(SUBSTR(record,offset,1)) |
| XL8 | Hexadecimal value of 8 bytes. It could be store clock value | NUMERIC DIGITS 50 C2D(SUBSTR(record,offset,8)) |
| A | This variable contains the Address value with length 4 | C2X(SUBSTR(record,offset,4)) |
| C | This is character data type | SUBSTR(record,offset,1)) |
| CL4 | This is character data type with length 4 | SUBSTR(record,offset,4)) |
| CL8 | This variable sometimes represents character of length 8 and sometimes contains store clock value | SUBSTR(record,offset,8) See below for store clock conversion |
| 0F, 0H, 0CL20, etc. | Any data type start with zero indicates duplication factor. This is a kind of group level field | No conversion required |

*Figure 3: Common data types and REXX conversions*

Please note that REXX uses nine significant digits. To convert 52 bits of store clock into decimal, you need to increase the significant digits to at least 16.

```
/*  REXX  */
hex_value = SUBSTR(C2X(stck,1,8)) /*convert char leng 8 to hexadecimal*/
Hex_mic_sec = SUBSTR(hex_value,1,13)       /* truncate last 1.5 bytes */
NUMERIC DIGITS 20                          /* Increase the precision  */
Micro_sec = X2D(Hex_mic_sec)               /* value in micro seconds  */
```

For formatting DB2 trace records, it is not usually necessary to convert the store clock value to date and time. However, if you do need to convert store clock value from binary form to timestamp format, you can use the procedure given at the end of this article.

## FORMATTING TRACE RECORDS

The following steps will help you to format trace records using REXX.

### Formatting the trace header

SMF header is mapped by DSNDQWAS.

```
* SMF COMMON HEADER MAPPING MACRO FOR ACCOUNTING  (SMF 1Ø1)
SM1Ø1LEN DS    XL2   SM1Ø1 TOTAL LENGTH
SM1Ø1SGD DS    XL2   ZZ BYTES
SM1Ø1FLG DS    XL1   SYSTEM INDICATOR
SM1Ø1RTY DS    XL1   RECORD TYPE X'65'
SM1Ø1TME DS    XL4   TIME SMF MOVED RECORD
SM1Ø1DTE DS    XL4   DATE SMF MOVED RECORD
SM1Ø1SID DS    CL4   SYSTEM ID (SID)
SM1Ø1SSI DS    CL4   SUBSYSTEM ID
SM1Ø1STF DS    XL1   RESERVED
SM1Ø1RI  DS    CL1   RESERVED
SM1Ø1SQ  DS    ØXL4  RESERVED
SM1Ø1BUF DS    XL4   TEMPORARY POINTER TO BUFFER AREA
```

The four bytes long RDW, which contains SM101LEN and SM101SGD, will not be available to REXX if you use DFSORT to extract trace records. Therefore, the SMF header, in essence, starts from field SMF101FLG. Some useful fields are shown in Figure 4.

If you use DSN1SDMP to extract your trace records from OP buffers into an output dataset, you won’t see the IFI header because this header is only 4 bytes (fullword field) and contains the total length of the complete trace record. This means there is no formatting required for the header part.

### Formatting the self-defining section

The self-defining section is placed immediately after the header. If you use DFSORT to extract SMF records, then the self-defining section would start on column position 25 and, in the case of

| *Column position* | *Field name* | *Assembler data type* | *Possible REXX conversion* | *Comments* |
|---|---|---|---|---|
| 2 | SM101RTY | XL1 | C2D(SUBSTR(rec,2,1)) | Record types could be 100-102 |
| 3 | SM101TME | XL4 | C2D(SUBSTR(rec,3,4)) | Needs further conversion into hh:mm:ss.nn format Field contains time in msecs since midnight |
| 7 | SM101DTE | XL4 | Var =C2X(SUBSTR(rec,7,4)) Var1 = SUBSTR(Var,1,7) Date=X2D(Var1) (julian date) | This field contains Julian date in packed decimal format |
| 11 | SM101SID | CL4 | SUBSTR(rec,11,4) | MVS system ID |
| 15 | SM101SSI | CL4 | SUBSTR(rec,15,4) | DB2 subsystem ID |

*Figure 4: Some useful fields*

DSN1SDMP, you would find the very first field in the output dataset is the start of the self-defining section.

The self-defining section consists of numerous pointer triplets. A pointer triplet consists of three fields – offset (full word), length (half word), and number of times repeated (half word). The first triplet field in the self-defining section contains the offset to the product section. Note that four bytes (to compensate for RDW) have to be subtracted from the offset to arrive at the actual offset. One is also added to get the actual start position because the start position is always one more than the actual offset value.

**Formatting the product section**

The product section can contain various headers, depending on the TDATA parameter. The product section contains a standard header, followed by any other headers. The standard header contains Resource

Manager ID (RMID), IFCID, number of self-defining sections, etc. The standard header is mapped by macro DSNDQWHS.

```
QWHSLEN  DS    XL2  LENGTH OF THE STANDARD HEADER
QWHSTYP  DS    XL1  HEADER TYPE
QWHSRMID DS    XL1  RESOURCE MANAGER ID
QWHSIID  DS    XL2  IFCID
QWHSRELN DS    ØXL2 RELEASE NUMBER AREA
QWHSNSDA DS    XL1  NUMBER OF SELF-DEFINING AREAS
.
.
QWHSLUCC DS    FL2  COMMIT COUNT
QWHSEND  DS    ØF   END OF STANDARD HEADER
```

The table in Figure 5 shows some useful fields. There are many other fields that are useful, and this is documented in the macro itself. You could verify various fields in the product section (eg validate FCID) before moving on to format the rest of the record.

| *Start position* | *Field name* | *Assembler data type* | *Possible REXX conversion* | *Comments* |
|---|---|---|---|---|
| 3 | QWHSTYP | XL1 | C2D(SUBSTR(rec,3,1)) | The following values indicate header type:<br>1 - standard<br>2 - correlation<br>4 - trace<br>8 - cpu<br>16 - distributed<br>32 - data sharing |
| 5 | QWHSIID | XL2 | C2D(SUBSTR(rec,5,2) | IFCID |
| 7 | QWHSNSDA | XL1 | C2D(SUBSTR(rec,7,1) | This field contains number of self-defining sections the trace record contains |

*Figure 5: Useful fields*

### Formatting remaining self-defining sections

The accounting trace record (IFCID 3) contains multiple data sections and the actual number of data sections can vary. Use macro DSNDQWA0 to format the self-defining section of accounting trace records (IFCID 3). Collect the pointers to all required data sections and calculate the actual start position in the record (subtract 4 bytes for RDW and add one to position the cursor). Similarly, if you find that you have multiple data sections, then you need to search for the appropriate mapping macro. Normally, most of the performance trace records have only one data section, but there are exceptions to this.

### Formatting the data sections

Once you hook on to the correct start position of the data section, you need to get help from the documentation in member DSNWMSGS and mapping macro DSNDQ* provided by IBM. With a little practice, you will find it easy to format the data section using REXX or COBOL.

### Generating customized reports

After formatting all the desired fields, you can create customized reports. The REXX code given below is only a sample and does not contain any error-checking logic.

```
/* REXX */
"ALLOC DD(INP) DS('input file (sortout from dfsort)') SHR REU"
"ALLOC DD(OUP) DS('output report') SHR REU"
 CNT = Ø
 /*   WRITE HEADER       */
 CALL WRITE_HEADER
 /*  START PROCESSING    */
DO FOREVER
   "EXECIO 1 DISKR INP"
   IF RC > Ø THEN LEAVE
   PARSE PULL INPUT_REC
   I = 1
   OFFSET = 1
   CALL DSNDQWAS                        /* MAP SMF HEADER          */
   IF SM1Ø1RTY = 1Ø1 & SM1Ø1SSI = 'DB2A' THEN  /* CHECK SMF REC TYPE */
   DO
     CALL DSNDQWAØ                      /* MAP SELF-DEFINING SECT  */
     OFFSET = QWAØ1PSO - 4 + 1
     CALL DSNDQWHS                       /* MAP STANDARD HEADER     */
     CALL DSNDQWHC                       /* MAP CORRELATED HEADER   */
     OFFSET = QWAØ1R1O - 4 + 1
     IF QWHSIID = 3  THEN
     DO
       CALL DSNDQWAC                     /* MAP ACCOUNTING SECTION  */
       CALL WRITE_REPORT
     END
   END
END
"EXECIO Ø DISKW OUP (STEM INL. FINIS"
"EXECIO Ø DISKR INP (STEM INL. FINIS"
"FREE DD(INP)"
"FREE DD(OUP)"
EXIT

DSNDQWAS:                                 /* MAP SMF HEADER         */
OFFSET = OFFSET + 1
/* SM1Ø1RTY DS    XL1            RECORD TYPE X'65' OR 1Ø1           */
SM1Ø1RTY = C2D(SUBSTR(INPUT_REC,OFFSET,1))
OFFSET = OFFSET + 1
/* SM1Ø1TME DS    XL4            TIME SMF MOVED RECORD              */
SM1Ø1TME = C2D(SUBSTR(INPUT_REC,OFFSET,4))
CALL GET_FMT_TIME
OFFSET = OFFSET + 12
/* SM1Ø1SSI DS    CL4            SUBSYSTEM ID                       */
SM1Ø1SSI = SUBSTR(INPUT_REC,OFFSET,4)
OFFSET = OFFSET + 1Ø
/*   TOTAL LENGTH = 28  */
RETURN

DSNDQWAØ:                      /* MAP SELF-DEFINING SECT  */
/* QWAØ1PSO DS    AL4           OFFSET TO THE PRODUCT SECTION       */
QWAØ1PSO = C2D(SUBSTR(INPUT_REC,OFFSET,4))
OFFSET = OFFSET + 4
OFFSET = OFFSET + 4
/* QWAØ1R1O DS    AL4          OFFSET TO THE ACCOUNTING SECTION     */
QWAØ1R1O = C2D(SUBSTR(INPUT_REC,OFFSET,4))
OFFSET = OFFSET + 78
RETURN
DSNDQWHS:                      /* MAP STANDARD HEADER               */
OFFSET = OFFSET + 4
/* QWHSIID  DS    XL2               IFCID                           */
QWHSIID = C2D(SUBSTR(INPUT_REC,OFFSET,2))
OFFSET = OFFSET + 8
/* QWHSSSID DS    CL4               SUBSYSTEM NAME                  */
QWHSSSID = SUBSTR(INPUT_REC,OFFSET,4)
OFFSET = OFFSET + 64
/*   TOTAL LENGTH = 76  */
RETURN
```

```
DSNDQWHC:                        /* MAP CORRELATED HEADER   */
OFFSET = OFFSET + 24
/* QWHCCN   DS     CL8    CONNECTION NAME                          */
QWHCCN = SUBSTR(INPUT_REC,OFFSET,8)
OFFSET = OFFSET + 8
/* QWHCPLAN DS     CL8               PLAN NAME                     */
QWHCPLAN = SUBSTR(INPUT_REC,OFFSET,8)
OFFSET = OFFSET + 42
/*   TOTAL LENGTH = 74  */
RETURN

DSNDQWAC:                             /* MAP ACCOUNTING DATA SECTION */
/* QWACBSC  DS     XL8        CLASS 1     BEGINNING STORE CLOCK VALU */
NUMERIC DIGITS 2Ø
QWACBSC = C2X(SUBSTR(INPUT_REC,OFFSET,8))    /*CONVERT INTO HEX VALUE*/
QWACBSC = X2D(SUBSTR(QWACBSC,1,13))          /*ELIMINATE 1.5 BYTES   */
OFFSET = OFFSET + 8
/* QWACESC  DS     XL8        CLASS 1     ENDING    STORE CLOCK VALU */
QWACESC = C2X(SUBSTR(INPUT_REC,OFFSET,8))   /*CONVERT INTO HEX VALUE */
QWACESC = X2D(SUBSTR(QWACESC,1,13))         /*ELIMINATE 1.5 BYTES    */
OFFSET = OFFSET + 8
ELAPSED_TIME = ( QWACESC - QWACBSC ) /1ØØØØØØ
/* QWACBJST DS     XL8      BEGINNING TCB CPU TIME FROM MVS (CLASS 1)*/
QWACBJST = C2X(SUBSTR(INPUT_REC,OFFSET,8))   /*CONVERT INTO HEX VALUE*/
QWACBJST = X2D(SUBSTR(QWACBJST,1,13))        /*ELIMINATE 1.5 BYTES   */
OFFSET = OFFSET + 8
/* QWACEJST DS     XL8       ENDING TCB CPU TIME IN ALL ENVIRONMENTS */
QWACEJST = C2X(SUBSTR(INPUT_REC,OFFSET,8))   /*CONVERT INTO HEX VALUE*/
QWACEJST = X2D(SUBSTR(QWACEJST,1,13))        /*ELIMINATE 1.5 BYTES   */
OFFSET = OFFSET + 8
TCB_TIME =  (QWACEJST - QWACBJST)/1ØØØØØØ
OFFSET = OFFSET + 26Ø
/*   TOTAL LENGTH = 292  */
RETURN

GET_FMT_TIME:
  RUN_HH = SM1Ø1TME % 36ØØØØ
  RUN_HH = RIGHT(RUN_HH,2,'Ø')
  RUN_MIN = SM1Ø1TME % 6ØØØ  - RUN_HH*6Ø
  RUN_MIN = RIGHT(RUN_MIN,2,'Ø')
  RUN_SEC = SM1Ø1TME % 1ØØ  - RUN_HH *36ØØ - RUN_MIN*6Ø
  RUN_SEC = RIGHT(RUN_SEC,2,'Ø')
  RUN_FMT_TIME = RUN_HH||':'||RUN_MIN||':'||RUN_SEC
RETURN

WRITE_HEADER:
 VAR =  'RUN TIME CONECTION   PLAN     ELAPSED TIME    TCB TIME'
 PUSH VAR
 "EXECIO 1 DISKW OUP"
 VAR =  ' '
 PUSH VAR
 "EXECIO 1 DISKW OUP"
RETURN

WRITE_REPORT:
       CNT = CNT + 1
       VAR = RUN_FMT_TIME||' '||LEFT(QWHCCN,12,' ')||QWHCPLAN||' '||,
                    RIGHT(ELAPSED_TIME,13,' ')||RIGHT(TCB_TIME,13,' ')
       PUSH VAR
       "EXECIO 1 DISKW OUP "              /* WRITE THE REPORT        */
RETURN
```

## CONVERTING STORE CLOCK VALUE INTO TIMESTAMP

You can plug the following procedure into your REXX code:

```
     Time_stamp = #STCKCONV(store_clock)
```

and use it as follows as follows:

```
#STCKCONV: Procedure
NUMERIC DIGITS 20              /* increase significant digits to 20 */
stck = arg(1)
secs = x2d(substr(stck,1,13))     /* Ignore last 1.5 bytes     */
micsec = ((secs / 1000000) - (secs % 1000000))* 1000000
micsec = trunc(micsec,0)
secs = (secs - micsec)%1000000
year = secs / (3600*24*365.25)
addon = trunc(year,0) / 4 - trunc(year,0) % 4 /* take care of leap yr*/
jd   = (year - trunc(year,0)) * 365.25 + addon
if addon = 0 then jd = jd + 1    /* leap year */
hours = (jd  - trunc(jd,0) ) * 24
min   = (hours - trunc(hours,0)) * 60
sec   = (min - trunc(min,0))*60
prec = sec - trunc(sec,0)
if prec > 0.98 then sec = sec + 1
time = right('00'||trunc(hours,0),2) || '.' || right('00'||,
 trunc(min,0),2) || '.' || right('00'||trunc(sec,0),2) || '.' ||micsec
NUMERIC DIGITS 9               /* restore default significant digits */
year = trunc(year,0) + 1900      /*  add 1900-01-01  */
jd   = trunc((jd+1),0)                 /* to current date     */
jdcal          = '031059090120151181212243273304334365'
jdcal_leap = '031060091121152182213244274305335366'
mon = 1
prev_mon_days = 0
do forever
  start_pos = (mon-1)*3 + 1
  if addon = Ø then days = substr(jdcal_leap,start_pos,3)
  else days = substr(jdcal,start_pos,3)
  if jd < days | jd = days then
    do
      days = jd - prev_mon_days
      leave
    end
    mon = mon + 1
    prev_mon_days = days
end
date = year|| '-' || Right('ØØ'||mon,2) || '-' || Right('ØØ'||days,2)
return (date || "-" || time)
```

*Venkat Pillay*
*DBA (USA)*

## Timestamp checking program – part 2

*This month we conclude the program that performs a timestamp ‘health check’ on a given group of load modules.*

```
CheckDBRM:
  /*  Check whether the DBRM matches the package (ie catalog) T/S  */
  ADDRESS ISPEXEC "LMMFIND DATAID("dbrmdsid") MEMBER("prev_package")",
       "STATS(YES)"
  if rc > 8 then do
    say "Error executing LMMFIND for DBRM member" name.i
    zispfrc = 8
    signal exit
  end
  if rc = 8 then return
  if rc = Ø then found_dbrm = "YES"
  ADDRESS ISPEXEC "LMGET DATAID("dbrmdsid") MODE(INVAR)",
      "DATALOC(DBRMLINE) DATALEN(LENVAR) MAXLEN(3276Ø)"
  get_rc = rc
  if get_rc > 8 then do
    say "Error during LMGET for load module" prev_package
    say "Return code = " rc
    zispfrc = 8
    signal exit
  end
  dbrm_timestamp = substr(dbrmline,25,8)
  do j = 1 to pkg_count - 1
    if pkg_timestamp.j = dbrm_timestamp then do
      match_dbrm = "YES"
      return
    end
  end
return

DisplayResults:
  /*  Print out results if necessary based on errlvl indicator   */
  /*  This logic is a bit messy due to the permutations and      */
  /*  combinations of parameters and events possible - apologies */
  if (errlvl = "E" & match_load = "NO" & found_load = "YES") |,
     ((found_load = "NO" | match_load = "NO" |,
     found_dbrm = "NO" | match_dbrm = "NO") & errlvl = "W"),
     then say prev_package
  if found_load = "NO" then do
    if errlvl = "W" then,
      say "   Load module containing program not found"
  end
  else if match_load = "NO" then,
     say "   No package timestamp matched in load module" module
  if (found_load = "YES" & match_load = "NO") | errlvl = "W" then do
    if found_dbrm = "NO" then say "   DBRM not found"
    else if match_dbrm = "NO" then,
       say "   DBRM timestamp does not match any package"
    if match_load_dbrm = "NO" &,
       found_load = "YES" &,
       found_dbrm = "YES" then,
      say "   Load module and DBRM timestamps do not match"
    else if found_load = "YES" & found_DBRM = "YES" &,
          match_load = "NO" then,
      say "   Load module and DBRM timestamps match"
  end
return

BuildXREF:
  /*  Create module cross reference table                      */
  ADDRESS ISPEXEC "TBCREATE MODXREF KEYS(CSECT MODULE)",
     "NOWRITE REPLACE"
  if rc > 4 then do
    say "Error creating MODXREF ISPF table. Return code =" rc
    zispfrc = 8
    signal exit
  end
  /*  Read AMBLIST listing  */
  /* "ALLOC F(AMBLIST) DA(USER.PRINT) SHR REUSE" */
  "EXECIO * DISKR AMBLIST (STEM line. FINIS"
  if rc > Ø then do
    say "Error reading from AMB listing. Return code =" rc
    zispfrc = 8
    signal exit
  end
  start = "NO"
```

```
db2_module = "NO"
csect_num = 1
do i = 1 to line.Ø
  if pos("ALPHABETICAL CROSS-REFERENCE",line.i) > Ø then do
    parse var line.i . . . . . . . module .
    if datatype(module,"ALPHA") <> "1" then exit
    if start <> "YES" then do
      csect.csect_num = module
      csect_num = csect_num + 1
    end
    start = "YES"
  end
  if pos("**    END OF MAP",line.i) > Ø then do
    start = "NO"
    /*  if load module contains DB2 interface module then     */
    /*  insert into CSECT list in cross reference table for   */
    /*  this load module                                      */
    if db2_module = "YES" | appl = "PE" then do
      do j = 1 to csect_num - 1
        /* say "   " csect.j */
        csect = csect.j
        ADDRESS ISPEXEC "TBADD MODXREF"
        tbadd_rc = rc
        if tbadd_rc <> Ø then do
          say "Error adding row with module="module,
            "csect="csect "to ISPF table"
          say "Return code =" tbadd_rc
          zispfrc = 8
          signal exit
        end
      end
    end
    csect. = ""
    csect_num = 1
    db2_module = "NO"
  end
  if start = "YES" then do
    parse var line.i csect_name .
    /*  Skip over unwanted CSECTs and flag whether module  */
    /*  contains DB2.                                      */
    if csect_name = "1" |,
       csect_name = "Ø" |,
       csect_name = "-" then parse var line.i . csect_name .
    if csect_name = "DSNHLI" then db2_module = "YES"
    if csect_name = "SYMBOL" |,
       csect_name = "" |,
       csect_name = "ALPHABETICAL" |,
       substr(csect_name,1,3) = "DSN" |,
       substr(csect_name,1,3) = "DFH" |,
       substr(csect_name,1,3) = "IGZ" |,
       substr(csect_name,1,3) = "ILB" |,
         substr(csect_name,1,3) = "PLI" then iterate
      found_csect = "NO"
      /*  Check whether we already know about this csect      */
      /*  If not then add it to the list for this module      */
      do j = 1 to csect_num
        if csect.j = csect_name then do
          found_csect = "YES"
          leave
        end
      end
      if found_csect = "NO" then do
        csect.csect_num = csect_name
        csect_num = csect_num + 1
      end
    end
  end
return

AllocLibs:

  /*  A lot of the naming here is site-specific. Be sure to alter  */
  /*  this to match your site's naming standards                   */

  if appl = "EL" then env = 'CPSR'
  else env = 'COMM'
  loadlib.Ø = 4
  loadlib.1 = "TLM."env".ACPT.CICSLOAD"
  loadlib.2 = "TLM."env".PROD.CICSLOAD"
  loadlib.3 = "TLM."env".ACPT.LOADLIB"
  loadlib.4 = "TLM."env".PROD.LOADLIB"
  if appl = "PE" then do
    loadlib.Ø = 2
    loadlib.1 = "APE.ACPT.LOADLIB"
    loadlib.2 = "PPE.PROD.LOADLIB"
  end
  dsname = ""
  do i = 1 to loadlib.Ø
    dsname = dsname||"'"||loadlib.i||"' "
  end
  "ALLOC F(LOADSCAN) DA("dsname") SHR REUSE"
  dbrmlib.Ø = 2
  dbrmlib.1 = "TLM."env".ACPT.DBRM"
  dbrmlib.2 = "TLM."env".PROD.DBRM"
  if appl = "PE" then do
    dbrmlib.Ø = 2
    dbrmlib.1 = "APE.M3Ø13J.DBRMLIB"
    dbrmlib.2 = "PPE.M3Ø13J.DBRMLIB"
  end
  dsname = ""
  do i = 1 to dbrmlib.Ø
    dsname = dsname||"'"||dbrmlib.i||"' "
  end
  "ALLOC F(DBRMSCAN) DA("dsname") SHR REUSE"
  ADDRESS ISPEXEC "LMINIT DATAID(LOADDSID)  DDNAME(LOADSCAN) ENQ(SHR)"
  if rc <> Ø then do
    say "Error executing load library LMINIT.  Return code =" rc
    zispfrc = 8
    signal exit
  end
  ADDRESS ISPEXEC "LMINIT DATAID(DBRMDSID)  DDNAME(DBRMSCAN) ENQ(SHR)"
  if rc <> Ø then do
    say "Error executing DBRM library LMINIT.  Return code =" rc
    zispfrc = 8
    signal exit
  end
  ADDRESS ISPEXEC "LMOPEN DATAID("loaddsid") OPTION(INPUT)"
  if rc <> Ø then do
    say "Error executing load library LMOPEN.  Return code =" rc
    zispfrc = 8
    signal exit
  end
  ADDRESS ISPEXEC "LMOPEN DATAID("dbrmdsid") OPTION(INPUT)"
  if rc <> Ø then do
    say "Error executing DBRM library LMOPEN.  Return code =" rc
    zispfrc = 8
    signal exit
  end
return

RunChecks:
  found_load = "NO"
  found_dbrm = "NO"
  match_load = "NO"
  match_dbrm = "NO"
  match_load_dbrm = "NO"
  call CheckDBRM
  if static_flag = "Y" then do
    /*  If we are assuming static linking, then find each load */
    /*  module that contains the current package and check it  */
    ADDRESS ISPEXEC "TBVCLEAR MODXREF"
    csect = prev_package
    module = ""
    ADDRESS ISPEXEC "TBSARG MODXREF NEXT NAMECOND(CSECT,EQ)"
    ADDRESS ISPEXEC "TBSCAN MODXREF"
    scan_rc = rc
    do until (scan_rc > Ø)
      found_load = "NO"
      match_load = "NO"
      match_load_dbrm = "NO"
      if scan_rc = Ø then call CheckLoad
      call DisplayResults
      ADDRESS ISPEXEC "TBSCAN MODXREF"
      scan_rc = rc
    end
    if scan_rc > 8 then do
      say "Error executing TBSCAN. Return code =" scan_rc
      zispfrc = 8
      ADDRESS ISPEXEC (VPUT ZISPFRC)
      exit 8
    end
  end
  else do
    module = prev_package
    call CheckLoad
    call DisplayResults
  end
return
```

## JCL

```
//T5MKISPF JOB ,'ISPF BAT',NOTIFY=&SYSUID,MSGCLASS=X,CLASS=B,REGION=4M
//*
//*  Generate AMBLIST commands for selected members
//*
//GENAMB   EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
//ISPPROF  DD DSN=&&PROFILE,DISP=(NEW,PASS),UNIT=SYSDA,
//         DCB=(LRECL=8Ø,BLKSIZE=2344Ø,RECFM=FB),
//         SPACE=(TRK,(1,1,1))
//ISPPLIB  DD DSN=SYS1.PLIB,DISP=SHR
//ISPSLIB  DD DSN=SYS1.SLIB,DISP=SHR
//         DD DSN=T5MK.USER.ISPSLIB,DISP=SHR
//         DD DSN=P1DP.PROD.ISPSLIB,DISP=SHR
//ISPMLIB  DD DSN=SYS1.MLIB,DISP=SHR
//ISPLLIB  DD DSN=SYS1.LLIB,DISP=SHR
//         DD DSN=SYS2.PUBLIC.ISPLLIB,DISP=SHR
//ISPTLIB  DD DSN=SYS1.TLIB,DISP=SHR
//SYSPROC  DD DSN=SYS1.CLIB,DISP=SHR
//         DD DSN=T5MK.USER.CLIST,DISP=SHR
//         DD DSN=P1DP.PROD.CLIST,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//AMBCMD   DD DSN=&&AMBCMD,DISP=(NEW,PASS),SPACE=(TRK,(1Ø,1Ø)),
//         DCB=(RECFM=FB,LRECL=8Ø,BLKSIZE=Ø),UNIT=SYSDA
//ISPLOG   DD DUMMY
//SYSIN    DD DUMMY
//SYSTSIN  DD *
 ISPSTART CMD(GENAMB AM)
//*
//*  Run AMBLIST for selected members. Make sure that the LOADLIBx
//*  datasets match those defined in the GENAMB REXX EXEC.
//*
//         IF (GENAMB.RC EQ Ø) THEN
```

```
//AMBLIST  EXEC PGM=AMBLIST,COND=(Ø,NE)
//SYSPRINT DD DSN=&&AMBLIST,DISP=(NEW,PASS),
//         SPACE=(CYL,(5Ø,5Ø),RLSE),UNIT=SYSDA,
//         DCB=(RECFM=FBA,LRECL=133,BLKSIZE=Ø)
//LOADLIB1 DD DSN=TLM.COMM.ACPT.CICSLOAD,DISP=SHR
//LOADLIB2 DD DSN=TLM.COMM.PROD.CICSLOAD,DISP=SHR
//LOADLIB3 DD DSN=TLM.COMM.ACPT.LOADLIB,DISP=SHR
//LOADLIB4 DD DSN=TLM.COMM.PROD.LOADLIB,DISP=SHR
//SYSIN    DD DSN=&&AMBCMD,DISP=(OLD,DELETE)
//*
//*    Run timestamp checks.  This is the only step required if
//*    you are not using static linking.
//*
//         IF (AMBLIST.RC EQ Ø) THEN
//TSCHECK2 EXEC PGM=IKJEFTØ1,DYNAMNBR=2Ø
//ISPPROF  DD DSN=&&PROFILE,DISP=(NEW,PASS),UNIT=SYSDA,
//         DCB=(LRECL=8Ø,BLKSIZE=2344Ø,RECFM=FB),
//         SPACE=(TRK,(1,1,1))
//ISPPLIB  DD DSN=SYS1.PLIB,DISP=SHR
//ISPSLIB  DD DSN=SYS1.SLIB,DISP=SHR
//         DD DSN=T5MK.USER.ISPSLIB,DISP=SHR
//         DD DSN=P1DP.PROD.ISPSLIB,DISP=SHR
//ISPMLIB  DD DSN=SYS1.MLIB,DISP=SHR
//ISPLLIB  DD DSN=SYS1.LLIB,DISP=SHR
//         DD DSN=SYS2.PUBLIC.ISPLLIB,DISP=SHR
//ISPTLIB  DD DSN=SYS1.TLIB,DISP=SHR
//SYSPROC  DD DSN=SYS1.CLIB,DISP=SHR
//         DD DSN=T5MK.USER.CLIST,DISP=SHR
//         DD DSN=P1DP.PROD.CLIST,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//AMBCMD   DD DSN=&&AMBCMD,DISP=(NEW,PASS),SPACE=(TRK,(1Ø,1Ø)),
//         DCB=(RECFM=FB,LRECL=8Ø,BLKSIZE=Ø),UNIT=SYSDA
//ISPLOG   DD DUMMY
//SYSIN    DD DUMMY
//SYSTSIN  DD *
 ISPSTART CMD(TSCHECK2 DB2A AM E Y)
//*
//         ENDIF
//*
//*   Print out AMBLIST output if anything goes wrong.
//*
//         IF (AMBLIST.RC NE Ø) THEN
//IEBGENR1 EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DSN=&&AMBLIST,DISP=(OLD,DELETE)
//SYSUT2   DD  SYSOUT=*
//SYSIN    DD  DUMMY
//         ENDIF
//         ENDIF
```

*Matthew Keene (Australia)* © Xephon 1999

## Quick table information

The REXX procedure TIN gives quick DB2 table information, column information, index information, and referential integrity information, and generates a report. Figure 1 shows the TIN Entry panel, where ‘DB2’ is the subsystem identifier and ‘Creator’ and ‘Table’ are DB2 Catalog search conditions.

```
Table Information

DB2     : DSNN
Creator : NADI
Table   : TL00%

Enter:Continue            PF3:End
```

*Figure 1: Entry panel*

The Selection Result panel (shown in Figure 2) appears when you press ‘Enter’ on the Entry panel. The panel shows the Selection result for the input search conditions.

```
----------------------------  Selection Result  ----------  Row 1 to 9 of 9
Command ===>                                                 Scroll ===> PAGE
------------------------------------------------------------------------------
Valid cmd: S More Information C Column I Index R Ref.Integrity D Document
Enter Valid cmd and press Enter                                    PF3 Return
------------------------------------------------------------------------------
cmd   Creator    Table                Dbname     Tsname               Card
 -    NADI       TL001                DBTTEMP    TS001           1.067.495
 -    NADI       TL002                DBTTEMP    TS002             853.222
 -    NADI       TL003                DBTTEMP    TS003             196.116
 -    NADI       TL004                DBTTEMP    TS004               4.490
 -    NADI       TL005                DBTTEMP    TS005               1.639
 -    NADI       TL006                DBTTEMP    TS006               6.278
 -    NADI       TL007                DBTTEMP    TS007               1.381
 -    NADI       TL008                DBTTEMP    TS008              10.710
 -    NADI       TL009                DBTTEMP    TS009                   8
```

*Figure 2: Selection Result panel*

The valid commands in the ‘cmd’ field are:

- S – additional table information (detail information).
- C – detail column information.
- I – index information.
- R – referential integrity information.
- D – document report.

The components of TIN are as follows:

- TIN is the driver procedure.
- TINM0 is the main menu.
- TINM1 is the table selection result panel.
- TINM2 is the detail table information.
- TINM3 is the column selection result panel.
- TINM4 is the detail column information.
- TINM5 is the index selection result panel.
- TINM6 is the additional index information.
- TINM7 is the referential integrity information.
- TIN00 is the TIN message.
- PTINF01 is PL/I source code (table detail).
- PTINF02 is PL/I source code (column detail).
- PTINF03 is PL/I source code (index detail).
- PTINF04 is PL/I source code (referential integrity).
- PTINF05 is PL/I source code (table document report).

TIN

```
/* REXX */
/* Table information                                    */
/* trace r */
 zpfctl = 'OFF'
```

```
Y=MSG("OFF")
address ispexec 'vput (zpfctl) profile'
Call Aloc
cur='cre'
TOP:
address ispexec "display panel(TINMØ) cursor("CUR")"
if rc=8 then do
   Call Free_proc
   ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.TABLE'"
   exit
end
/* Check input parameters                              */
if DB2=' ' then do
   message = 'Enter DB2 ssid. !'
   Call Error 'DB2'
end
if cre=' ' & tab=' ' then do
   message='At least one Catalog search field must be entered.'
   Call Error 'cre'
end
parm=substr(cre,1,8)||substr(tab,1,18)
ADDRESS TSO
QUEUE "RUN PROGRAM(PTINFO1) PLAN(PTINFO1),
       LIBRARY ('SKUPNI.BATCH.LOADLIB'),
       PARMS ('/"parm"')"
QUEUE "END "
"DSN SYSTEM("DB2")"
if rc=12 then do
   "delstack"
   Call Free_proc
   Call Aloc
   message = 'Error.   'DB2||' ssid is not valid. !'
   Call Error 'DB2'
end
"EXECIO * DISKR SYSPRINT (STEM ROW."
if row.2 = 'NO CATALOG ENTRIES FOUND' then do
   Call Free_proc
   Call Aloc
   message = 'No catalog entries found, check Search Fields.'
   Call Error 'cre'
end
else do
   address ispexec 'tbcreate "tlist" names(cre1 tab1 dbn1 tsn1 card)'
   do i=2 to row.Ø BY 3
      cre1= word(row.i,5)
      tab1= word(row.i,4)
      dbn1= word(row.i,2)
      tsn1= word(row.i,3)
      card= right(word(row.i,6),14)
      address ispexec 'tbadd "tlist"'
   end
   address ispexec 'tbtop "tlist"';
   Call Display_table
end
DIS:
Select
  when(cmd='S') THEN DO
   Call Table_detail
   Call Display_table
   Signal DIS
  end
  when(cmd='C') THEN DO
   Call Columns
   Call Display_table
   Signal DIS
  end
  when(cmd='I') THEN DO
   Call Indexes
   Call Display_table
   Signal DIS
  end
  when(cmd='R') THEN DO
   Call RI
   Call Display_table
   Signal DIS
  end
  when(cmd='D') THEN DO
   Call Document
   Call Display_table
   Signal DIS
  end
  otherwise rc=Ø
End
Call Free_proc
address ispexec 'tbend "tlist"'
ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.TABLE'"
Call Aloc
Signal TOP
Aloc:
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.TABLE'"
  "ALLOC DD(SYSPRINT) DSN('"SYSVAR(SYSUID)".DB2.TABLE') SPACE(24 8),
  TRACK MOD UNIT(339Ø) RECFM(F,B) LRECL(254) BLKSIZE(254) ,
  F(SYSPRINT) CATALOG REUSE "
Return
Error:
  ARG cur_par
  cur=cur_par
  address ispexec "setmsg msg(tinØØ1)"
  signal top
Return
Free_proc:
  "execio Ø diskr sysprint (finis"
  address tso "free f(sysprint)"
```

```
Return
Display_table:
  address ispexec 'tbdispl "tlist" panel(TINM1)'
  if rc=8 then do
     Call Free_proc
     address ispexec 'tbend "tlist"'
     Call Aloc
     signal top
  end
Return
Check_dsn:
 if rc>Ø then do
    message=file||' not found.'
    address ispexec "setmsg msg(tinØØ1)"
    Call Free_proc
    Call Aloc
    address ispexec 'tbend "tlist"'
    signal top
 end
Return
Table_detail:
  ADDRESS ISPEXEC 'ADDPOP ROW(2) COLUMN(7)'
  crtb=cre1||tab1
  do j=1 to row.Ø while (crtb<>word(row.j,5)||word(row.j,4))
  end
  j=j+1
  tv.1='-'
  tv.2=cre1
  tv.3=tab1
  jj=j+1
  tv.4=subword(row.jj,1)
  tv.5=tsn1
  tv.6=dbn1
  do v=7 to 29
     tv.v=word(row.j,v-6)
  end
  do j=1 to 29
     dti = tv||j
     INTERPRET dti '= strip(tv.j)'
  end
  address ispexec "display panel(TINM2)"
  ADDRESS ISPEXEC "REMPOP"
Return
Columns:
  Call Free_proc
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.COLUMNS'"
  "ALLOC DD(SYSPRINT) DSN('"SYSVAR(SYSUID)".DB2.COLUMNS'),
  SPACE(24 8) TRACK MOD UNIT(339Ø) RECFM(F,B) LRECL(254),
  BLKSIZE(254) F(SYSPRINT) CATALOG REUSE "
  parm=substr(cre1,1,8)||substr(tab1,1,18)
  ADDRESS TSO
  QUEUE "RUN PROGRAM(PTINFO2) PLAN(PTINFO2),
         LIBRARY ('SKUPNI.BATCH.LOADLIB'),
         PARMS ('/"parm"')"
  QUEUE "END "
  "DSN SYSTEM("DB2")"
  "EXECIO * DISKR SYSPRINT (STEM col."
  tabinfo=cre1||'.'||tab1
  address ispexec 'tbcreate "clist",
          names(cname ctyp clen csca cnul crem)'
  ccmd=''
  do c=1 to col.Ø by 3
     cname= word(col.c,2)
     ctyp = word(col.c,3)
     clen = right(word(col.c,4),6)
     csca = right(word(col.c,5),5)
     cnul = word(col.c,6)
     crem = subword(col.c,7)
     address ispexec 'tbadd "clist"'
  end
  address ispexec 'tbtop "clist"';
  DISC:
  address ispexec 'tbdispl "clist" panel(TINM3)'
  if ccmd='S' then do
   Call Column_detail
   Signal DISC
  end
  Call Free_proc
  address ispexec 'tbend "clist"'
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.COLUMNS'"
Return
Column_detail:
  ADDRESS ISPEXEC 'ADDPOP ROW(1) COLUMN(13)'
  do k=1 to col.Ø while (cname<>word(col.k,2))
  end
  k=k+1
  col1 = cname
  col2 = crem
  col4 = cre1
  col5 = tab1
  col6 = strip(word(col.k,3))
  col7 = ctyp
  col8 = strip(clen)
  col9 = strip(csca)
  col1Ø= cnul
  col11= strip(word(col.k,2))
  col12= strip(word(col.k,4))
  col13= strip(word(col.k,5))
  col14= strip(subword(col.k,12))
  col16= strip(word(col.k,6))
  col17= strip(word(col.k,7))
  col18= strip(word(col.k,8))
  col19= strip(word(col.k,9))
  col2Ø= strip(word(col.k,1Ø))
  col21= strip(word(col.k,11))
  k=k+1
  col3 = strip(subword(col.k,2))
  address ispexec "display panel(TINM4)"
  ccmd=''
  ADDRESS ISPEXEC "REMPOP"
Return
Indexes:
  Call Free_proc
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.INDEXES'"
  "ALLOC DD(SYSPRINT) DSN('"SYSVAR(SYSUID)".DB2.INDEXES'),
  SPACE(24 8) TRACK MOD UNIT(339Ø) RECFM(F,B) LRECL(254),
  BLKSIZE(254) F(SYSPRINT) CATALOG REUSE "
  parm=substr(cre1,1,8)||substr(tab1,1,18)
  ADDRESS TSO
  QUEUE "RUN PROGRAM(PTINFO3) PLAN(PTINFO3),
         LIBRARY ('SKUPNI.BATCH.LOADLIB'),
         PARMS ('/"parm"')"
  QUEUE "END "
  "DSN SYSTEM("DB2")"
  "EXECIO * DISKR SYSPRINT (STEM inx."
  tabinfo=cre1||'.'||tab1
  address ispexec 'tbcreate "ilist",
          names(iname icre iuni iing ired itio bp ityp)'
  icmd=''
  do i=2 to inx.Ø by 1
     if word(inx.i,1)='I' then do
        iname= word(inx.i,2)
        icre = word(inx.i,3)
        iuni = word(inx.i,4)
        iing = word(inx.i,5)
        ired = word(inx.i,6)
        itio = left(word(inx.i,7),6)
        bp   = word(inx.i,8)
        ityp = word(inx.i,9)
        address ispexec 'tbadd "ilist"'
     end
  end
  address ispexec 'tbtop "ilist"';
  DISI:
  address ispexec 'tbdispl "ilist" panel(TINM5)'
  if icmd='S' then do
   Call Index_detail
   Signal DISI
  end
  Call Free_proc
  address ispexec 'tbend "ilist"'
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.INDEXES'"
Return
Index_detail:
  icmd=''
  ixinfo=icre||'.'||iname
  inic=iname||icre
  address ispexec 'tbcreate "klist",
          names(colname ordering colcard)'
  do i=1 to inx.Ø while (inic<>word(inx.i,2)||word(inx.i,3))
  end
  i=i+1
  do j=i to inx.Ø while (word(inx.j,1)='K')
     colname =word(inx.j,2)
     if word(inx.j,3)='D'
     then ordering='Desc'
     else ordering='Asc'
     colcard =right(word(inx.j,4),1Ø)
     address ispexec 'tbadd "klist"'
  end
  address ispexec 'tbtop "klist"';
  address ispexec 'addpop row(3) column(15)'
  address ispexec 'tbdispl "klist" panel(TINM6)'
  address ispexec 'rempop'
  address ispexec 'tbend "klist"';
Return
RI:
  Call Free_proc
  ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.RI'"
  "ALLOC DD(SYSPRINT) DSN('"SYSVAR(SYSUID)".DB2.RI'),
  SPACE(24 8) TRACK MOD UNIT(339Ø) RECFM(F,B) LRECL(8Ø),
  BLKSIZE(8Ø) F(SYSPRINT) CATALOG REUSE "
  parm=substr(cre1,1,8)||substr(tab1,1,18)
  ADDRESS TSO
  QUEUE "RUN PROGRAM(PTINFO4) PLAN(PTINFO4),
         LIBRARY ('SKUPNI.BATCH.LOADLIB'),
         PARMS ('/"parm"')"
  QUEUE "END "
  "DSN SYSTEM("DB2")"
  "EXECIO * DISKR SYSPRINT (STEM ri."
  address ispexec 'tbcreate "rlist",
          names(rship rcre rtab reln drule)'
  tabr=cre1||'.'||tab1
  rip=Ø
  ric=Ø
  do i=2 to ri.Ø
     rship= word(ri.i,1)
     if rship='P'
     then do
          if rip=Ø
          then rship='Parent table'
          else rship=' '
          rip=1
     end
     if rship='C'
     then do
          if ric=Ø
          then rship='Child  table'
```

```
           else rship=' '
           ric=1
      end
      rcre = word(ri.i,2)
      rtab = word(ri.i,3)
      reln = word(ri.i,4)
      drule= word(ri.i,5)
      if drule='R' then drule='Restrict'
      if drule='C' then drule='Cascade'
      if drule='N' then drule='Set Null'
      address ispexec 'tbadd "rlist"'
   end
   address ispexec 'tbtop "rlist"';
   address ispexec 'tbdispl "rlist" panel(TINM7)'
   address ispexec 'tbend "rlist"'
   Call Free_proc
   ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.RI'"
 Return
 Document:
   Call Free_proc
   ADDRESS TSO "DELETE '"SYSVAR(SYSUID)".DB2.DOC'"
   "ALLOC DD(SYSPRINT) DSN('"SYSVAR(SYSUID)".DB2.DOC'),
   SPACE(24 8) TRACK MOD UNIT(339Ø) RECFM(F,B) LRECL(13Ø),
   BLKSIZE(13ØØ) F(SYSPRINT) CATALOG REUSE "
   parm=substr(cre1,1,8)||substr(tab1,1,18)
   ADDRESS TSO
   QUEUE "RUN PROGRAM(PTINFO5) PLAN(PTINFO5),
          LIBRARY ('SKUPNI.BATCH.LOADLIB'),
          PARMS ('/"parm"')"
   QUEUE "END "
   "DSN SYSTEM("DB2")"
   "Execio * diskr Sysprint (stem doc. finis"
   address ispexec "browse dataset('"sysvar(sysuid)".DB2.doc')"
 Return
```

## TINM0

```
)ATTR
   $ type(text)   color(white) caps (off) hilite(reverse) intens(high)
   | type(text)   color(white) hilite(reverse) intens(high)
   ( type(text)   color(yellow)    hilite(reverse) intens(high)
   ) type(text)   color(green)                     intens(high)
   _ type(input)  color(red)       intens(high) pad(_)
)BODY WINDOW(36,12)
+
$          Table Information
| +                                 |
| )  DB2     :_DB2 +                |
| )  Creator :_cre     +            |
| )  Table   :_tab               +|
| +                                 |
( Enter:Continue            PF3:End
)INIT
  if (&DB2 ¬= ' ')
      .attr (DB2) = 'pad(nulls)'
  if (&cre ¬= ' ')
      .attr (cre) = 'pad(nulls)'
  if (&tab ¬= ' ')
      .attr (tab) = 'pad(nulls)'
)PROC
  IF (.PFKEY = PFØ3) &PF3 = EXIT
  VPUT (TAB CRE DB2) PROFILE
)END
```

## TINM1

```
)Attr Default(%+_)
   | type(text)   intens(high) caps(on ) color(yellow)
   $ type(output) intens(high) caps(off) color(yellow)
   ? type(text)   intens(high) caps(on ) color(green) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   } type(text)   intens(high) caps(off) color(white)
   [ type( input) intens(high) caps(on ) just(left )
   ] type( input) intens(high) caps(on ) just(left ) pad('-')
   ¬ type(output) intens(low ) caps(off) just(asis ) color(green)
)Body  Expand(//)
%-/-/- ? Selection Result +%-/-/-
%Command ===>_zcmd                                      / /%Scroll
===>_amt +
+________________________________________
+Valid cmd:|S+More
Information|C+Column|I+Index|R+Ref.Integrity|D+Document
+Enter Valid cmd and press|Enter+
|PF3+Return
+________________________________________
#cmd+ #Creator + #Table            + #Dbname  + #Tsname  + #
Card+
)Model
 ]z+  ¬z       + ¬z                + ¬z       + ¬z       + ¬z
+
)Init
  .ZVARS = '(cmd cre1 tab1 dbn1 tsn1 card)'
  &amt = PAGE
  &cmd = ''
)Reinit
)Proc
)End
```

*Editor's note: this article will be continued next month.*

*Bernard Zver*
*Database Administrator*
*Informatika Maribor (Slovenia)*

# January 1995 – October 1999 index

Items below are references to articles that have appeared in *DB2 Update* since January 1995. References show the issue number followed by the page number(s). Back-issues of *DB2 Update* are available back to Issue 15 (January 1994). See page 2 for details.

StarBase has announced Version 4.1 of its StarTeam and StarTeam Enterprise integrated technical collaboration and software configuration management tool, with improved support and enhanced scalability and new integration with DB2 and Informix Dynamic Server.

Version 4.1 includes an improved repository management tool, providing the ability to verify repository integrity, upgrade the repository to new releases, migrate the repository to a different database, and export the StarTeam database catalogue to comma-delimited files. It also includes several general performance enhancements like a smaller memory footprint (server and client), faster check in/out, faster client connections, faster query executions, and improved response for binary files over 100MB.

For further information contact:
Starbase, 18872 MacArthur Boulevard, Irvine, CA 92715, USA.
Tel: (949) 442 4400.
URL: http://www.starbase.com.

* * *

DB2 users can benefit from Wall Data's Cyberprise BI Studio, a suite of information access and analysis tools for the company's Cyberprise Portal Server.

The suite includes Report Query and Cube Designer, DBApp Developer, and Cyberprise InfoPublisher, allowing users to create reports, OLAP cubes, queries, and forms from DB2, Oracle, SQL Server, and other databases.

For further information contact:
Wall Data, 11332 North East 122nd Way, Kirkland, WA 98034-6931, USA.
Tel: (425) 814 9255.
Wall Data (UK), Wall Data House, 418 Bath Road, Longford, West Drayton, Middlesex, UB7 0EA, UK.
Tel: (0181) 476 5000.
URL: http://www.walldata.com.

* * *

Princeton Softech has announced its SyncPoint software, which supports DB2 for OS/390 and DB2 UDB.

SyncPoint transforms corporate database information so that it can be distributed to and synchronized at multiple sites. It maintains data accuracy while automatically segmenting, distributing, securing, and synchronizing data in a multi-vendor database and application independent environment. It accommodates complex applications that have a large number of users that require simultaneous synchronization, via scalable parallel processing and dynamic load balancing.

For further information contact:
Princeton Softech, 1060 State Road, Princeton, NJ 08540-1423, USA.
Tel: (609) 497 0205.
URL: http://www.princetonsoftech.com.

* * *