141

DB2 update

July 2004

In this issue

DB2 Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £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 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

DB2 Update on-line

Code from *DB2 Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/db2; you will need to supply a word from the printed issue.

Contributions

When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Sampling table data

This article looks at a new function available in DB2 UDB V8 FP2, which allows you to obtain sampling data from a table. Before this function became available, you had to select the entire contents of the table and then write a program to perform any sampling that you wanted. This new function makes it a lot easier! There are two types of sampling available – row-level and block-level. In basic terms, row-level sampling means that every *x* rows are sampled, whereas block-level sampling means that every *y* pages are sampled. Which method is best? Hopefully, that will become clear as you read on.

So, let's look at the mechanics. To allow you to sample data from a table, there is a new parameter associated with the SELECT command called tablesample. It is this parameter that indicates whether you will be performing row-level or block-level sampling. The help pages for tablesample say: '*The BERNOULLI keyword specifies that row-level Bernoulli sampling is performed. The SYSTEM keyword specifies that block-level Bernoulli sampling is performed unless the optimizer determines that it is more efficient to perform row-level Bernoulli sampling instead. Performance of block-level sampling is excellent because only one I/O is required for each selected page.*'

Let's look at each of these in turn.

I ran all the SQL in this article on a Windows 2000 machine running DB2 UDB 8.1 FP2.

For my example table I used a four-million row table (approximately) called EMP (based on the EMPLOYEE table in the SAMPLE database copied into itself 16 times, giving 4,194,304 rows). There are no indexes defined on the table. I have only the default buffer pool of 250 x 4KB pages defined.

Say I want to sum all the bonus payments in this EMP table for all employees in the D11 department. My traditional query would look like:

```
>db2 select sum(bonus) from EMP where workdept = 'D11'
```

This would obviously have to look at every row in the table (because we don't have an index on the table). If I want to see the sum based on a sampling of, say, 10% of all employees in the D11 department, I would issue the query (this is a row-level sampling example):

```
>db2 select sum(bonus) from EMP tablesample bernoulli (10) repeatable
(5) where workdept = 'D11'
```

Where the *(10)* is the percentage of rows that you want to sample and the *(5)* is a repeatable (seed) value.

The meaning of the percentage value is fairly obvious, but what is the repeatable value? This value gives you the ability to rerun a query several times and get the same result. Also, changing the repeatable value changes the result set, so you should experiment with this value to see which gives you the closest value to the actual result. This is shown later on.

What happens if you try to specify a percentage value of 0? You will receive this error message: 'SQL20242N The sample size specified in the TABLESAMPLE clause is not valid'.

In the discussion below, I mention certain execution time values. What is important is not the absolute execution time value, but the relationship between one time and another. While your absolute times might vary, the ratios should be in a similar proportion to the ones I show.

If I do a straight **select sum(bonus) from EMP where workdept = 'D11'** command, the sum value returned is 576,716,800.00 and the SQL execution time is 26 s. If I connect to the database once and then issue five select commands one after the other, the individual execution times are all about 26 s.

In Figure 1 below, I disconnected/reconnected to the database between every run, and executed:

```
>db2 select sum(bonus) from EMP tablesample bernoulli (x) repeatable (5)
where workdept = 'D11'
```

where *x* is the Bernoulli value in the table in Figure 1.

Bernoulli value (x)	*sum value returned*	*SQL execution time*
0.01	62,000.00	25s
0.1	572,200.00	25s
1	5,752,400.00	23s
10	57,673,400.00	23s
50	288,444,200.00	23s

Figure 1: SQL execution time for different Bernoulli values

You can see that the SQL execution time does not effectively change irrespective of the percentage value specified. This is because I didn't have an index defined on the table. The help pages state: '*If there is no index then there are no I/O savings over executing a query without sampling*'. If I didn't disconnect/ reconnect after every run, the SQL execution times obtained would still all be about the same.

So let's create an index on the EMP table called INDX1 based on the WORKDEPT column using the command below and run runstats for the table:

```
>db2 CREATE INDEX DB2ADMIN.INDX1 ON DB2ADMIN.EMP (WORKDEPT ASC) PCTFREE
10 MINPCTUSED 10
```

And let's rerun our selects – the values are shown in Figure 2.

Remember that the *full* select gave a sum value of 576,716,800. First let's look at the accuracy of the sum value returned for the

Bernoulli value	*Sum value returned*	*Sum value multiplied up*	*Sum difference (%) from full value*	*execution time (no index)*	*execution time (with index)*
0.01	62,000	620,000,000	+7.5	25s	4s
0.1	572,200	572,200,000	-0.8	25s	6s
1.0	5,752,400	575,240,000	-0.3	25s	13s
10.0	57,673,400	576,734,000	+0.03	23s	27s
50.0	288,444,200	576,888,400	+0.02	23s	30s

Figure 2: SQL execution time with an index

different sampling levels. As the sampling percentage value increases so does the accuracy of the sum returned. This can be seen by comparing the *Sum value multiplied up* column value with the actual value of 576,716,800. The trade-off is the execution time. You can see that there is a large reduction in execution time for a sampling value of 0.01 compared with that for, say, 10 and upwards. If you are interested in just an estimate, then the sampling value of 0.1 seems perfectly adequate, whereas if you need a precise value, then obviously you have to perform the *full* select. If you don't need an exact answer, then in this scenario a sampling value of 0.1 seems to give the best trade-off between execution time and result set accuracy.

We shall quickly look at the repeatable parameter. Let's run our query with differing values for the repeatable parameter. The results are shown below:

```
>db2 select sum(bonus) from EMP tablesample bernoulli (1) repeatable (y)
where workdept = 'D11'

Repeatable value (y)      Rows returned.
5                            5,752,400
10                           5,736,300
15                           5,789,100
20                           5,788,300
25                           5,680,800
```

The time taken to run the query for each repeatable value was roughly the same, but, as you can see, the number of rows returned differs for each different repeatable read value. The number of rows returned from a full select is 576,716,800, so you can see that the best value for the repeatable parameter is around the 5 mark. I also repeated the run for the same repeatable value five times, and each time the same number of rows was returned – this highlights the 'repeatable' ability of the query. As always, you need to perhaps run with more repeatable values to determine the best value for your data and query. What I hoped to show here was that changing the repeatable value does make a difference to the sum value returned.

So now let's look at block-level sampling and see how that

System value	*Sum value returned*	*Sum value multiplied up*	*Sum difference (%) from full value*	*execution time (with index)*
0.01	61,600	616,000,000	+7	0.3s
0.1	568,800	568,800,000	-1.4	0.7s
1.0	5,810,100	581,010,000	+0.7	5s
10.0	57,144,500	571,445,000	-0.9	24s
100.0	576,716,800	576,716,800	0.0	30s

Figure 3: Select sum(bonus) query resulst

works. The format of the command for block-level sampling is:

```
>db2 select sum(bonus) from EMP tablesample system(10) where workdept =
'D11'
```

where *system()* is a keyword, and *10* is the percentage of pages to sample.

You can't use a percentage value greater than 100 because a value of 100 means sample 100% of all pages (ie sample every page), which effectively is the same as not using the sampling facility.

I reran the select sum(bonus) query specifying different system percentage values and, as before, disconnecting/reconnecting to the database between runs. The results are shown in Figure 3.

You can see that the execution times of block-level sampling are lower than those for row-level sampling, but the results are generally less accurate. Also, if you didn't have an index on the *workdept* column, then the execution times wouldn't be much different from those shown.

So which method is best? The help pages indicate that this decision is dependent on your data and the distribution of values within your data. I would therefore recommend (which should come as no surprise) that you try each method and see which one fits your data best. I used only a relatively small table with a simple query, but if your table size is orders of magnitude bigger and you run the query a significant number of times in a day, then the potential execution time savings could be great. This is yet another useful tool in the DBA armoury.

C Leonard
Freelance Consultant (UK)

Stored procedures versus CICS transactions

This article examines the differences and similarities between the two most commonly used methods to process data on the mainframe side if requests are coming from smaller platforms, which today is usually from Internet or intranet applications. Those methods are stored procedures and CICS transactions.

At the first glance, the main difference between these two methods is the nature of the data they can use in their code – stored procedures use DB2 tables and CICS transactions normally use VSAM datasets (but can use DB2 tables too). One could say that since CICS transactions can use more than one source for their data, it is better to use them – however, we will see that it is not as simple as it looks. Below we will look at some topics and find out how both of these methods behave.

HOW IS IT DEFINED?

This is the point where everything begins. How easily and quickly you can have your code deployed, and how much other team members must participate in the whole process depends on how things are defined.

A stored procedure is a compiled program, stored at a DB2 local or remote server, that can execute SQL statements. A typical stored procedure contains two or more SQL statements and some manipulative or logical processing in a host language.

A stored procedure must be defined in DB2 by using a CREATE PROCEDURE SQL DDL (Data Definition Language) statement. This statement includes a description of the input and output parameters, the type of SQL processing on the DB2 tables, the number of result sets (which will be explained later), the execution environment, etc.

To be able to work with stored procedures we need to have a Package List (Collection) defined, where our programs would be bound, because stored procedures cannot run from plans.

Certain DB2 actions must be taken before we can even start to code our stored procedure:

- BIND ADD Authority must be given to you.
- Package List (Collection) must be created.
- CREATE, BIND, and EXECUTE authority must be given to you on that package.
- EXECUTE authority must be given to you on that stored procedure.
- If your stored procedure will be executed by WLM (Work Load Manager), you need to ask your DBA to create one and to give you authority to use it.

To avoid doing all these authorizations for each user, it is a good idea to define a DB2 group and then just add new users to the existing group.

Usually stored procedures are used to decrease the network traffic between a client and server by setting all the business logic in one module and, therefore, all the SQL statements that deal with DB2 tables. Typically stored procedures do not have too many SQL statements and are used to retrieve some subset of records for a client for further processing. However, with newer versions of DB2 (7 and 8) more robust stored procedures can be written.

A CICS transaction is a unit of application data processing (consisting of one or more application programs) initiated by a single request. A transaction may require the initiation of one or more tasks for its execution.

CICS transactions, and programs that perform that transaction, must be defined in a CICS System Definition (CSD) file by using the CEDA transaction. In addition, you might need to use the CEMT transaction to set some of the options for the transaction or programs. Also, you need to define your VSAM datasets to CICS in the same way. As for DB2, we also have to deal with some authorizations (to run the transaction and program). Again defining a group of users will help a lot.

We can see that both methods ask for additional work to define them and therefore additional knowledge. A DBA or CICS system programmer can do some of this work, but it is good to understand what is going on behind the scenes.

HOW IS IT INVOKED?

This is really a question about how easy it is to work with these two methods. Which one to use is only important to you and the client-side programmers, who will actually use them.

Stored procedures are invoked using standard SQL CALL statements. By giving the name of a stored procedure (together with the schema name – like SYSPROC, or one that is application specific) and the list of parameters, we will execute the stored procedure program back on mainframe. As a result of that call, we can have either output parameter values, an opened result set (one or many) on some cursor(s), or both.

Prior to using the CALL statement we need to be connected to DB2, so we need to use the SQL CONNECT statement. To make it possible we need to have a tool, like DB2 Connect. In addition, the programming language we use must have a library of database functions we can use to browse (fetch) opened result sets. For Java there is the JDBC library, for Visual Basic there is the ADO library, etc.

CICS transactions, on the other hand, can use several techniques, including EHLLAPI, APPC, or TCP/IP (just to mention those I've used so far). All of them ask for additional knowledge to that needed for the client or server programming language.

EHLLAPI also needs to have an 'open' channel to a host emulator program (like IBM PC Communications, EXTRA, Rumba, etc). APPC needs a special connection established between the client and server. TCP/IP wants to have established ports on both sides – client and server. On the CICS side, there must be a listener (a special CICS program waiting for incoming calls), and on the client side there must be Windows sockets (if Windows OS is in use).

It is not that simple to establish a connection between client code and code on our mainframe, but it looks much easier for stored procedures than for CICS transactions. EHLLAPI is a very easy way to make a connection, but it is also the one that is the most vulnerable.

WHAT IS THE OUTPUT STRUCTURE AND HOW CAN WE USE IT?

Here we will look at the structure of the output we can get by using these two methods.

Stored procedures can have both records and/or simple variables for their output. In the case of variables, the only restriction is that they must be described as valid DB2 types, while records are actual data from DB2 tables (more than one if that was the request). Records are organized in result sets and there can be more than one (with a maximum of 32,767 result sets). And while variables are used as any other variable from any other procedure call, records are processed using database functions specific to the client environment (JDBC, ODBC library). Usually you will need to use additional SQL statements like ASSOCIATE LOCATOR and ALLOCATE CURSOR, as well as some kind of FETCH statement (usually NEXT). You will know when you reach the end of a result set by catching -100 for SQLCODE (or having NEXT return FALSE).

CICS transactions are not as good as stored procedures in creating output structure. The result from a CICS transaction is usually a long string, with a structure that is already known on the client side. The process of reading and using that structure is known as screen scraping. This process is usually slow and limited by the size of the output structure, so it can take several calls to get all the data that the client requires.

This is another example where stored procedures are a better choice to use.

WHERE ELSE CAN IT BE USED?

If you can use the same code in several environments, you will get better design and less maintenance per unit of code.

Stored procedures can be used on the mainframe side with no changes at all. Therefore, if your application needs data from DB2 tables you can use stored procedures in the same way as a client application on a PC. As already said, code in a stored procedure is a compiled program, so you can use it not only as a stored procedure but also as a common module in your applications (in which case you would use a standard CALL statement from the host language). In addition, you can use stored procedures in CICS programs.

CICS transactions can also be used on the mainframe side. You can use them in CICS to process some data, but you need to be careful that all data sent to a screen is suppressed. A good example of using such transactions is an application with distributed data, where you need a tool to access data remotely (APPC or TCP/IP only). You can use such a CICS transaction in some batch processing too, by using the CICS EXCI facility.

From what I've said here, it is obvious that stored procedures are much more useful. They can be used in many address spaces (DB2, batch, CICS, on remote mainframes, and client/server environments) with almost no changes to your code.

WHAT CAN BE A SOURCE OF DATA?

It is very important to know what data sources you can use once you decide on the method you will use in your application. The more data sources the better because you are perhaps in a position to inherit data from current designs and it can be expensive to remodel everything.

Stored procedures can use only data in DB2 tables. We cannot use datasets in stored procedures. However, there are two exceptions.

The first one is when we use code from a stored procedure as an external module. If we want to do that, it is good to have an input parameter that will tell the stored procedure code what the mode of the call is (batch/CICS when the code is called by a non-SQL CALL statement, or a pure stored procedure when the code is called using an SQL CALL statement).

The other one is where we can actually print data (like messages) into a standard output dataset. We can see these messages in WLM’s SYSPRINT dataset.

CICS transactions do not have such a problem, but still there are some restrictions. All datasets we use in CICS transactions must be VSAM datasets, so we cannot use ordinary sequential ones except ESDS VSAM datasets. CICS transactions can read DB2 tables with no restrictions.

As we can see, both methods have some restrictions. CICS transactions have more freedom than stored procedures because, if we need to have basic stored procedures, we need to make additional effort in transferring datasets into DB2 tables.

IS THERE ADDITIONAL PROGRAMMING TO LEARN?

When you come to the point of deciding what new technique or tool you will start to use, you need to know much time is needed to acquire new skills and knowledge – or maybe you already have sufficient experience.

Stored procedures are very easy to code. There are no additional commands, statements or APIs to learn. All you need to know is the host language (COBOL, PL/I, etc) and SQL for DB2.

CICS transactions demand additional knowledge besides the host language and SQL (if you have data in DB2 tables). You need to learn CICS API, a set of statements that will establish a connection between the host language and CICS. As for SQL, the good thing is that there are no differences between the API for COBOL and the API for PL/I.

If you want results as soon as possible, it will be much quicker if you choose stored procedures.

HOW DOES IT MANAGE WITH LARGE AMOUNTS OF DATA IN THE RESULT SET?

The result set(s) is a set of records that fulfil the requested conditions for the current call. The number of records can be from zero to several thousand (eg a list of phone calls for some period of time for some company) so this can be a very important issue.

Stored procedures deal with DB2 tables and the way it gives the result set is to have a cursor opened and then let the calling program fetch and process the requested records. Therefore, since the number of records is not known at the beginning, cursor opening can be a time-consuming process if conditions are not set properly. Another way to make the process faster is to limit the number of records returned in the result set by using the option FETCH FIRST *n* ROWS ONLY. Be aware that this feature is available only in DB2 Versions 7 and 8.

CICS transactions on the other hand can read both DB2 tables and VSAM datasets, but there are problems if the result set is too big. It is better if we have data in VSAM datasets than in DB2 since we can control the number of records processed. If you use EHLLAPI or TCP/IP you are limited by the size of the screen (EHLLAPI) or package size (TCP/IP), so you need to have developed a technique for getting more than one screen or package on the client side.

For both methods there is a solution for large result sets. The solution is to write records from the result set into a data pool, and then read only a portion of that pool at a time. The difference is in what we will use for that pool of data. With a stored procedure, we can use DB2 tables that we will read in consecutive calls. With CICS transactions, temporary storage datasets (or other VSAM datasets) are used. With both methods, there is additional processing of this data pool once you create the result set. You need to delete it, or save it if you expect the same query to be made in the near future – in which case you need to develop a technique to retrieve previously-created result sets instead of creating a new one. Anyway, some kind of housekeeping process (deleting or keeping these records in the data pool) must be done.

While stored procedures can deal with big result sets by using new features in DB2 Version 7 (Version 7 has scrollable cursors that can be very helpful for this purpose), CICS transactions can only write to and read from only a data pool. In addition, while DB2 is working for us by returning only those records we want, with CICS transactions we need to check conditions for each record in VSAM datasets. As we can see, more reasons to use stored procedures.

An additional benefit with stored procedures is that they can return more than one result set (up to 32,767 in DB2 Version 7).

IS THERE ANY WAY TO BACK-OUT CHANGES?

By back-out, we mean the ability to undo any changes your program can make to data in either DB2 or a VSAM dataset.

Stored procedures can easily back out changes with almost no effort. By using the SQL ROLLBACK statement, any changes made during the DB2 unit of work (from the last SQL COMMIT statement) will be undone. Moreover, ROLLBACK statements can be issued both from inside and outside of stored procedures. But be aware that this will roll back the whole unit of work, including any changes in the calling program too. With DB2 Version 7 you can also COMMIT inside a stored procedure, which was not possible in previous versions, so more care must be taken. In addition, when you define your stored procedure, you can set an option for DB2 to make COMMIT ON RETURN from the stored procedure to the calling program (this is not possible if the calling program is an IMS transaction).

CICS transactions can roll back any changes, but only from inside the transaction and only for changes made on data in DB2 tables. You must use the CICS SYNCPOINT statement (with or without the ROLLBACK option), not SQL COMMIT or SQL ROLLBACK. If we have our data in VSAM datasets there is no way to undo changes just made by the use of a known statement or command. If we want to roll back changes, once we finish the transaction, we cannot do it because the transaction does not know what was done in its previous run. Keeping some kind of log of all the actions and then reversing them can replace the rollback function. However, for that purpose we need another transaction.

It is obvious that stored procedures are much better. Having the opportunity to undo changes is a very important thing, especially in client/server applications.

CONCLUSION

Considering all of the above, we can say that if you have a new project and your team has significant knowledge of DB2, it is much better to use stored procedures as your gateway to the outer world. However, if your data is already in VSAM datasets, it is worth reconsidering all the time, cost and effort involved in moving that data to DB2 tables – so you could continue to use CICS transactions. The benefits of using stored procedures are obvious – it's up to you to decide.

Predrag Jovanovic
Project Developer
Pinkerton Computer Consultants (USA)

Renaming a DB2 subsystem for datasharing – part 2

This month we conclude our detailed look at renaming a DB2 subsystem for datasharing.

MORE ALTERS

Although the statements we're going to generate are undoing the statements we've just created, they are somewhat more complicated. In changing the objects so that they do not use storage groups, we lost some information. For a user-defined object (one which doesn't use a storage group) primary quantity and secondary quantity are irrelevant, because DB2 doesn't define the datasets. When we change the objects to be DB2-defined (by adding a storage group) we need to specify the primary and secondary quantity because DB2 needs the information for reorgs, loads, etc.

The ALTER statement needed for tablespaces is:

```
ALTER TABLESPACE dbname.tsname PART part USING STOGROUP storagegroup
    PRIQTY primaryquantity SECQTY secondaryquantity
```

For indexes, the statement is:

```
ALTER INDEX creator.indexname PART part USING STOGROUP storagegroup
    PRIQTY primaryquantity SECQTY secondaryquantity
```

All the information is in SYSTABLEPART and SYSINDEXPART respectively. We need to be a bit careful with the secondary quantity. In DB2 Version 6, the old SMALLINT secondary quantity, SECQTY, was superseded by the integer field SECQTYI, but the latter field was not filled in by the Version 6 CATMAINT, and is therefore zero until an ALTER is done. We can get the correct secondary quantity using a CASE statement:

```
CASE WHEN SECQTYI = Ø
    THEN SQTY
ELSE
    SECQTYI
END
```

The other thing to remember is that the primary and secondary quantities are held in the DB2 catalog as numbers of 4KB blocks, whereas our ALTER statements need to specify the amounts in KB.

There is a problem here which is obvious from looking at the ALTER statements above – they're just too long to fit on a line – they need to go over two lines. DSNTIAUL can unload only rows, ie single lines not split lines. What we'll do is generate each statement as a single line and worry about splitting it later.

Using the approach adopted earlier, here is the SQL to generate the ALTER statements for the tablespaces:

```
SELECT CHAR(
            'ALTER TABLESPACE '                ||
            STRIP(DBNAME)||'.'||STRIP(TSNAME)  ||
            CASE WHEN PARTITION = Ø
                THEN ' '
            ELSE
                ' PART ' || STRIP(CHAR(PARTITION))
            END                                ||
            ' USING STOGROUP ' || STRIP(STORNAME)||
            ' PRIQTY ' || STRIP(CHAR(PQTY * 4))||
            ' SECQTY '                         ||
            STRIP(CHAR(
                CASE WHEN SECQTYI = Ø
                    THEN SQTY
                ELSE
                    SECQTYI
                END * 4
                ))                             ||
            ';'
           ,12Ø)
       FROM SYSIBM.SYSTABLEPART
       WHERE STORTYPE = 'I'
       ORDER BY DBNAME,TSNAME,PARTITION
  WITH UR
;
```

If you edit the file this creates, you can see the problems with the lengths of the lines.

```
EDIT       SMITHAC.TSP.UNLOAD                         Columns ØØØØ1 ØØØ72
Command ===>                                             Scroll ===> CSR
****** ************************** Top of Data *************************
ØØØØØ1 ALTER TABLESPACE DABRATE.SABRATE  USING STOGROUP DB2PØ1 PRIQTY
12912 SEC
```

```
000002 ALTER TABLESPACE DABRATE.SABWARM  USING STOGROUP SMS PRIQTY 100
SECQTY 2
000003 ALTER TABLESPACE DADMSDCH.SADCRGPD  USING STOGROUP DB2P28 PRIQTY
24 SECQ
000004 ALTER TABLESPACE DADMSDCH.SADIFCON  USING STOGROUP DB2P28 PRIQTY
32 SECQ
000005 ALTER TABLESPACE DADMSDCH.SADPJREG  USING STOGROUP SG1ADL PRIQTY
1340 SE
000006 ALTER TABLESPACE DADMSDCH.SADSYSUB  USING STOGROUP DB2P28 PRIQTY
400 SEC
000007 ALTER TABLESPACE DAIACCNT.SACCNTOT  USING STOGROUP DB2P16 PRIQTY
1800 SE
```

I'd already written a general purpose edit macro, which gets round this problem. The macro is a REXX called FOLDS. FOLDS takes one or two parameters: the first is a number specifying what column to split on – FOLDS splits the line starting at the rightmost space before the target column; if the second parameter is present (it can be set to any value), any blank lines which result from a split are suppressed.

Before executing FOLDS, you need to get rid of the single trailing X'00 at the end of each line – enter:

```
C P'.' ' ' ALL
```

which gets rid of unprintable characters.

Entering **FOLDS 72 y** while editing the dataset has this result:

```
EDIT       SMITHAC.TSP.UNLOAD                        Columns 00001 00072
Command ===>                                            Scroll ===> CSR
****** ************************** Top of Data *************************
000001 ALTER TABLESPACE DABRATE.SABRATE  USING STOGROUP DB2P01 PRIQTY
12912
000002  SECQTY 1292;
000003 ALTER TABLESPACE DABRATE.SABWARM  USING STOGROUP SMS PRIQTY 100
SECQTY
000004  20;
000005 ALTER TABLESPACE DADMSDCH.SADCRGPD  USING STOGROUP DB2P28 PRIQTY
24
000006  SECQTY 4;
000007 ALTER TABLESPACE DADMSDCH.SADIFCON  USING STOGROUP DB2P28 PRIQTY
32
000008  SECQTY 4;
000009 ALTER TABLESPACE DADMSDCH.SADPJREG  USING STOGROUP SG1ADL PRIQTY
1340
000010  SECQTY 136;
```

```
000011 ALTER TABLESPACE DADMSDCH.SADSYSUB  USING STOGROUP DB2P28 PRIQTY
400
000012  SECQTY 100;
000013 ALTER TABLESPACE DAIACCNT.SACCNTOT  USING STOGROUP DB2P16 PRIQTY
1800
000014  SECQTY 38904;
```

This is the code for FOLDS:

```
/*****************************************************************/
/*                  REXX                                         */
/* Edit Macro to split a line at nearest space below parm        */
/* Alan Smith Sep 2003                                           */
/*****************************************************************/
ADDRESS ISPEXEC
"ISREDIT MACRO (PARM)"
suppressSpace = 0
if parm = "" then
  parmind = 72
else
  do
  parse var parm parmind supp
  if supp ¬= '' then
    suppressSpace = 1
  end
halfparm = parmind / 2
lno = 1
"ISREDIT (LASTNUM) = LINENUM .ZLAST"
do while lno <= lastnum
  "ISREDIT (lStat) = XSTATUS (lno)"
  if lStat = "NX" then
    do
    ind = parmind
    "ISREDIT (theLine) = LINE (lno)"
    do while substr(theLine,ind,1) ¬= ' ' & ind > halfparm
      ind = ind -1
    end
    ind = ind - 1
    theLine2 = right(theLine,length(theLine)-ind)
    theLine = left(theLine,ind)
    "ISREDIT LINE (lno) = (theLine)"
    if (theLine2 = ' ' & suppressSpace) then
      nop
    else
      do
      "ISREDIT LINE_AFTER (lno) = (theLine2)"
      lno = lno + 1
      end
    end
  lno = lno + 1
  "ISREDIT (LASTNUM) = LINENUM .ZLAST"
end
exit
```

The SQL to generate the ALTERs for indexes looks like this:

```
SELECT CHAR(
            'ALTER INDEX '                       ||
            STRIP(IXCREATOR)||'.'||STRIP(IXNAME)  ||
            CASE WHEN PARTITION = Ø
                THEN ' '
            ELSE
                ' PART ' || STRIP(CHAR(PARTITION))
            END                                  ||
            ' USING STOGROUP ' || STRIP(STORNAME)||
            ' PRIQTY ' || STRIP(CHAR(PQTY * 4))||
            ' SECQTY '                           ||
            STRIP(CHAR(
                CASE WHEN SECQTYI = Ø
                    THEN SQTY
                ELSE
                    SECQTYI
                END * 4
                ))                               ||
            ';'
           ,12Ø)
       FROM SYSIBM.SYSINDEXPART
       WHERE STORTYPE = 'I'
       ORDER BY IXCREATOR,IXNAME,PARTITION
  WITH UR
;
```

This is some of the output after FOLDS has been applied:

```
ALTER INDEX NUAI.XAIITADJØ3  USING STOGROUP DB2P16 PRIQTY 5616 SECQTY
 564;
ALTER INDEX NUAM.XAMACA1 PART 1 USING STOGROUP DB2P1Ø PRIQTY 456ØØ
 SECQTY 456Ø;
ALTER INDEX NUAM.XAMACA1 PART 2 USING STOGROUP DB2PØ1 PRIQTY 456ØØ
 SECQTY 456Ø;
```

We've created four lots of ALTER statements. They must be copied into datasets with a RECFM of FB and LRECL of 80, so that we can run them later.

STOPS AND STARTS

The ALTERs we've generated can be performed only when the tablespaces and indexspaces are stopped. Rather than generate -STOP and -START commands for each object, it's easiest to use wildcards with the database names:

```
-STOP DB(databasename) SPACE(*)
```

Create two jobs, one with -STOP commands for all databases except DSNDB01, DSNDB06, and DSNDB07, and one with –START commands for the same databases.

STOGROUPS

In between the two lots of ALTER statements, the STOGROUPs have to be dropped and recreated with their new VCAT.

The statements for each STOGROUP will look something like this:

```
DROP STOGROUP stogroupname;
COMMIT;
CREATE STOGROUP stogroupname VOLUMES(vol1,…) VCAT DBBG;
```

How you approach this depends on the number and complexity of your STOGROUPS. If you use only one STOGROUP, and therefore rely on SMS to sort out all DASD allocation, creating the statements is trivial.

A common approach is to let SMS do nearly everything, but have a few STOGROUPs defined to take care of objects needing to be individually assigned to named disks for the sake of performance. The disks are defined as guaranteed space, which means that if they are referenced in a STOGROUP, those disks are used and no substitution takes place.

If the only volume used by a STOGROUP is '*', ie it lets SMS do the allocation, or if each STOGROUP references only one volume, then the techniques we have already seen can be used to generate the statements from SYSSTOGROUP and SYSVOLUMES. Otherwise you need to do at least some of it manually.

JOBS RUN ON THE NIGHT

The jobs we've looked at so far can be generated in advance and then run unchanged on the night. The remaining jobs can be prepared in advance but have some aspects that need to be changed on the night.

BSDS UPDATES

The BSDS needs to be informed of the new VCAT for the DB2 catalog and directory, and the old log names removed and the new ones added. All this is done with DB2 down.

The first step is to check that DB2 came down cleanly and there are no outstanding units of recovery (URs). Run a DSNJU004 job to print off the contents of the BSDS:

```
//SMAPBSDS EXEC PGM=DSNJU004
//STEPLIB  DD   DSN=SYS2.DB2P.SDSNLOAD,DISP=SHR
//SYSUT1   DD   DSN=DB2P.BSDS01,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSABEND DD   SYSOUT=*
```

In the output find the CHECKPOINT QUEUE:

```
                     CHECKPOINT QUEUE
                    18:57:59 JANUARY 02, 2004
    TIME OF CHECKPOINT       18:57:13 JANUARY 02, 2004
    BEGIN CHECKPOINT RBA            0C11FBCE8CD3
    END CHECKPOINT RBA              0C11FBD0CDA2
    TIME OF CHECKPOINT       18:42:13 JANUARY 02, 2004
    BEGIN CHECKPOINT RBA            0C11F660A191
    END CHECKPOINT RBA              0C11F663AB53
    TIME OF CHECKPOINT       18:27:13 JANUARY 02, 2004
    BEGIN CHECKPOINT RBA            0C11F2CE6970
    END CHECKPOINT RBA              0C11F2D06764
    TIME OF CHECKPOINT       18:14:42 JANUARY 02, 2004
```

Make a note of the BEGIN CHECKPOINT RBA for the most recent checkpoint (best to copy this to the clipboard to save any transcription errors).

Then run a DSN1LOGP job:

```
//STEP1   EXEC PGM=DSN1LOGP
//STEPLIB  DD DSN=SYS2.DB2P.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=X
//SYSABEND DD DUMMY
//BSDS     DD DSN=DB2P.BSDS01,DISP=SHR
//SYSSUMRY DD SYSOUT=X
```

```
//SYSIN    DD *
   RBASTART(ØC11FBCE8CD3)
   SUMMARY(ONLY)
```

The RBASTART parameter is the RBA from the BSDS print. In the output from the job, look for the RESTART SUMMARY. This should say:

```
DSN1156I ALL URS COMPLETE
DSN1159I NO DATABASE WRITES PENDING
```

This means it's safe to proceed. If you don't get these messages, you need to start up the subsystem again and shut it down cleanly.

The other information we need to get from the BSDS print is details from the active logs. The things to note are the start and end RBA for each log. The other item of interest is the status. For most it is REUSABLE, which means the log has been archived. The current active log pair will have a status of NOTREUSABLE, as will any other log pairs that have not been archived.

We're going to rename the logs, so the BSDS has to be updated with the new names. To change the active log names in the BSDS, the old log entries have to be deleted and new ones added. The statements look like this for each log:

```
 DELETE DSNAME=DB2P.LOGCOPY1.DS14
 NEWLOG DSNAME=DBBG.DBB1.LOGCOPY1.DS14,COPY1,
      STARTRBA=ØC1DC6DØ4ØØØ,ENDRBA=ØC1DE99E3FFF
```

The DELETE statement is pretty simple. The NEWLOG statement is more complicated, involving the use of RBAs. You can build these statements by hand from the BSDS print, but it takes time and is prone to error. Because of this I wrote a REXX program, RENLOGS, to produce the statements from the BSDS:

```
/* REXX program */
"EXECIO 1 DISKR inf"
more_recs = queued()
parse pull line
call process_active
call process_active
return
process_active:
active_string = substr(line,2,1Ø)
do while more_recs > Ø & active_string ¬= "ACTIVE LOG"
```

```
  "EXECIO 1 DISKR inf"
  more_recs = queued()
  parse pull line
  active_string = substr(line,2,1Ø)
end
if more_recs = Ø then
    return 12
copy_no = substr(line,18,1)
"EXECIO 2 DISKR inf ( skip"
"EXECIO 1 DISKR inf"
more_recs = queued()
parse pull line
eos = substr(line,2,1)
do while more_recs > Ø & eos ¬= "A"
  startrba = substr(line,5,12)
  endrba = substr(line,28,12)
  dsn = substr(line,67,44)
  dsn = strip(dsn,trailing)
  "EXECIO 1 DISKR inf"
  parse pull line
  status = substr(line,7Ø,11)
  push "DELETE DSNAME="||dsn
  "EXECIO 1 DISKW out"
  push "NEWLOG DSNAME="||dsn||",COPY"||copy_no||","
  "EXECIO 1 DISKW out"
  push "    STARTRBA="||startrba||",ENDRBA="||endrba
  "EXECIO 1 DISKW out"
  "EXECIO 1 DISKR inf"
  more_recs = queued()
  parse pull line
  eos = substr(line,2,1)
end
return
```

This version works with the DSNJU004 output from DB2 V7. The layout changed by a character from V6, so you need to check that the REXX works for your version.

RENLOGS produces statements like this:

```
 DELETE DSNAME=DB2P.LOGCOPY1.DS14
 NEWLOG DSNAME=DB2P.LOGCOPY1.DS14,COPY1,
     STARTRBA=ØC1DC6DØ4ØØØ,ENDRBA=ØC1DE99E3FFF
```

so you need to change the NEWLOG DSNAME using edit commands.

In addition to the DELETE and NEWLOG statements, you need a

NEWCAT statement to store the VCAT for the catalog and directory in the BSDS.

```
NEWCAT VSAMCAT=DGPG
```

The final job to prepare is the one to process the DSNDB07 tablespaces. The job just needs to run SQL like this:

```
 DROP TABLESPACE DSNDBØ7.DSN4KØ1;
 DROP TABLESPACE DSNDBØ7.DSN4KØ2;
 .
 .
 COMMIT;
 SET CURRENT SQLID = 'NUDBHSYS';
 CREATE TABLESPACE DSN4KØ1 IN DSNDBØ7
        BUFFERPOOL BP1
        CLOSE NO
        USING VCAT DGPG;
 CREATE TABLESPACE DSN4KØ2 IN DSNDBØ7
        BUFFERPOOL BP1
        CLOSE NO
        USING VCAT DGPG;
 .
 .
```

Under V6 you need to stop DSNDB06. You may be able to do without the stop under V7.

A CHECKLIST

Here's a checklist that goes through the whole process.

Preparation:

- Copy TID – copy TIDDB2P to TIDDBB1.
- Change all 'DB2P' to 'DBB1'.
- Run install CLIST – ensure values are correct on the following screens:
  - DSNTIPA1 – input member name
  - DSNTIPA2 – catalog alias – DBBG
  - DSNTIPH – HLQs – DBBG.DBB1
  - DSNTIPO – parameter module – DSNZPBB1.

- Checks:
  - check correct entries in SYS1.PARMLIB(IEFSSN*).
  - Catalog Alias DBBG exists in the correct catalog.
  - STC userid, STCs in STARTED class.
  - RACF profile for DBBG exists and STC userid has ALTER access.
  - SMS rules for tablespaces, indexes, logs, BSDSes, archive logs.
  - VTAM, TCP/IP definitions.
  - Unix System Services definitions.
  - WLM definitions.
- Libraries – create any required load libraries, etc.
- Generate tablespace/index jobs – generate stop/start tablespace/index job.
- Generate rename tablespace job.
- Generate rename index job.
- Generate rename logs job.
- Generate rename BSDSes job.
- Generate alter tablespace (no STOGROUP) job.
- Generate alter index (no STOGROUP) job.
- Generate alter tablespace (STOGROUP) job.
- Generate alter index (STOGROUP) job.
- Create other jobs:
  - Create job to drop/recreate STOGROUPs
  - Create DSN1LOGP job
  - Create DSNJU004 job

  - Create DSNJU003 job
  - Create job to drop recreate DSNDB07 TSes
  - Create new STC procs.

Implementation:

- Maint mode – stop DB2 and restart in maint mode.
- Back-up – image copy the catalog and directory and quiesce.
- Issue an -ARCHIVE LOG.
- Stop objects – run job to stop tablespaces and indexes.
- Alter objects – run jobs to alter tablespaces and indexes to not use STOGROUPs.
- Change STOGROUPs – run job to drop and recreate STOGROUPs with new VCAT.
- Stop DSNDB07.
- Stop DB2.
- Back-up – DSS dump BSDS and NOTREUSABLE active log.
- Print BSDS R – run job DSNJU004.
- Run DSN1LOGP R – run with RBASTART set to RBA from previous step.
- Check no URs outstanding.
- Rename datasets – run jobs to:
  - rename tablespace/index datasets.
  - rename logs.
  - rename BSDSes.
- ZPARM – run DSNTIJUZ to create new zparm.
- Update BSDS – run RENLOGS REXX with output from Print BSDS step.

- Add NEWCAT line.
- Run DSNJU003 job to update BSDS.
- Start subsystem with new name –start in maint mode.
- Sort works – run job to drop/recreate DSNDB07 tablespaces.
- Alter objects – run jobs to ALTER tablespaces and indexes to use STOGROUPs.
- Start objects – run jobs to start tablespaces and indexes.
- Restart normally – stop subsystem and restart normally.

PROGRAMS, JCL, ETC

Although rather out of the scope of this article, let's consider briefly what changes need to be made by other areas – the systems and application areas which use DB2.

If only the underlying dataset names are changing – the high-level qualifiers for the VSAM datasets used by tablespaces, indexspaces, logs, and the BSDS – hardly anything will need to change. No user programs or other subsystems such as CICS should be looking directly at them. You may, however, have a small number of jobs that do LISTCs against the datasets or run BSDS prints, etc.

If the subsystem name is changing, a whole lot will have to change: CICS and IMS will need to connect to the 'new' subsystem; program and utility JCL will need to use the new subsystem name. Programs using DDF are affected – where CONNECTS are used, programs will need to CONNECT to a different subsystem; three-part ALIASes will need to be changed to point to the new subsystem; plans which are bound to include in their PKLIST entries such as *subsys.collid.packagename* will need to be rebound. Programs may look at CURRENT SERVER, and then make a choice depending on the value.

If the subsystem name is not changed, but the subsystem is moved to datasharing, CICS, IMS, and JCL do not need to change, because the subsystem still exists. However, when a program does a CONNECT, it will now connect to the group location name, and CURRENT SERVER now returns the name of the group (CURRENT MEMBER returns the subsystem name).

SUMMARY

This article has described a process for renaming a DB2 subsystem. Although there are a large number of steps to go through, with sufficient preparation, the implementation can be performed in a couple of hours during the night. Automatic generation of job statements via SQL and REXX are key to the preparation.

Hopefully, I don't need to say this, but you should try renaming a test subsystem before you do a production one.

BIBLIOGRAPHY

IBM manuals:

DB2 Universal Database for OS/390 and z/OS Data Sharing: Planning and Administration SC26-9935-02 – Chapter 3: *Installing and enabling DB2 data sharing; Renaming the DB2 member*.

DB2 Universal Database for OS/390 and z/OS Administration Guide SC26-9931-02 – Chapter 7: *Altering your database design; Changing the high-level qualifier for DB2 data sets.*

Alan Smith
Norwich Union (UK)

DB2 object manager

With DB2 systems getting larger and more complex, and skilled DBAs becoming increasingly rare and expensive, it's important to have a monitor program to identify objects that need maintenance. Performing unnecessary maintenance for objects makes inefficient use of DBA time and wastes the batch window time. DB2 Version 7 provides the Real Time Statistics function as well as a stored procedure (DSNACCOR) to give recommendations for objects needing maintenance. This COBOL program will utilize DSNACCOR. Having a self-managed DB2 is the goal.

PROCEDURE DESCRIPTION

This tool provides five types (extent, image copy, reorg, restrict, and runstats) of recommendation list, based on the user input criteria. A main program processes input parameters query type, scope, and criteria, then it invokes one of the subprograms, which will in turn call DSNACCOR, retrieve the result set, and format the output. Formulas for recommending action, criteria parameter meaning, and default values can be found in the DB2 manual. The following examples will demonstrate how it works:

1 List the names of all DB2 tablespaces and indexes if their underlying VSAM dataset is over 100 extents. Sort the output in descending sequence so the objects that have the most extents will be listed first.

```
EXTENT DBNAME   % TYPE ALL EXTLIMIT 1ØØ
```

2 List any objects currently in restricted status.

```
RESTRICT DBNAME  % TYPE ALL
```

3 List all tablespaces under a database starting with TDB if any of the following conditions since the last reorg are true:

- ratio of the sum of update, insert, and delete, to the total number of rows >30%.
- ratio of unclustered inserts to the total number of rows >20%.
- ratio of imperfectly chunked LOBs to total number of rows >10% (default).
- mass delete >0 (default).

– ratio of overflow records to the total number of rows >20%.

– VSAM extents >100.

```
REORG DBNAME  TDB% TYPE  TS    REORGCRI 3Ø,2Ø,,,2Ø,1ØØ
```

CHECKLIST FOR INSTALLATION

- Change DB2LOC to your DB2 location name.
- Change SSID to your DB2 subsystem name.
- Compile, linkedit, and bind five subprograms as a package first, then the main program (objmaint) as a plan to include all the package lists.
- Use the attached JCL to run reports;1 more information can be found in the JCL.

The test environment is DB2 Version 7 for z/OS 1.3, Enjoy!

OBJMAINT PROGRAM

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    OBJMAINT.
       AUTHOR.        LIJUN  GAO;
       DATE-WRITTEN.  Ø8/Ø8/Ø3.
       DATE-COMPILED.
****************************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-37Ø.
       OBJECT-COMPUTER.  IBM-37Ø.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT MYFILE ASSIGN TO S-PARMIN
           FILE STATUS IS MASTER-CHECK-KEY
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.
/***************************************************************
       DATA DIVISION.
       FILE SECTION.
       FD  MYFILE
           RECORDING MODE IS F
           LABEL RECORDS ARE STANDARD
```

```
            BLOCK CONTAINS Ø RECORDS
            DATA RECORD IS CONTROL-PARM-REC.
        Ø1  CONTROL-PARM-REC               PIC X(8Ø).
/*****************************************************************
        WORKING-STORAGE SECTION.
        Ø1  SRC-PARM-INPUT                 PIC X(8Ø).
        Ø1  MASTER-CHECK-KEY               PIC X(2).
        Ø1  W1ØØ-MYFILE-EOF                PIC X(Ø1)  VALUE 'N'.
            88  MYFILE-EOF                            VALUE 'Y'.
        Ø1  PARM-INPUT-REC.
            Ø5  LINEA.
                Ø7  PARM-TYPE                  PIC X(Ø8).
                Ø7  FILLER                     PIC X VALUE SPACE.
                Ø7  PARM-DBNAME                PIC X(Ø8).
                Ø7  FILLER                     PIC X VALUE SPACE.
                Ø7  PARM-DBNAME-VALUE          PIC X(Ø8).
                Ø7  FILLER                     PIC X VALUE SPACE.
                Ø7  PARM-QUERY                 PIC X(4).
                Ø7  FILLER                     PIC X VALUE SPACE.
                Ø7  PARM-QUERY-TYPE            PIC X(3).
                Ø7  FILLER                     PIC X VALUE SPACE.
            Ø5  LINEB.
                Ø7  PARM-CRI               PIC X(Ø8).
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL1          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL2          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL3          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL4          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL5          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
                Ø7  PARM-CRI-VAL6          PIC 9(4) VALUE ZERO.
                Ø7  FILLER                 PIC X VALUE SPACE.
        77  PGM-NAME                       PICTURE X(8).
        Ø1  CHAR-COUNT                     PIC 9(3) USAGE BINARY.
        Ø1  CTR-1                          PIC S9(3).
        PROCEDURE DIVISION.
        ØØØØ-MAIN-LOGIC.
            PERFORM 1ØØØ-INITIALIZATION THRU 1ØØØ-EXIT.
            PERFORM 2ØØØ-READ-INPUT     THRU 2ØØØ-EXIT
                UNTIL MYFILE-EOF.
            STOP RUN.
        1ØØØ-INITIALIZATION.
            OPEN INPUT MYFILE.
            IF MASTER-CHECK-KEY NOT = "ØØ"
                DISPLAY "NONZERO FILE STATUS " MASTER-CHECK-KEY.
            MOVE SPACES TO CONTROL-PARM-REC.
```

```
     MOVE SPACES TO PARM-INPUT-REC.
     MOVE Ø TO CHAR-COUNT.
 1ØØØ-EXIT.
     EXIT.
 2ØØØ-READ-INPUT.
****************************************************************
* PROCESS INPUT PARAMETERS
****************************************************************
     READ MYFILE
        AT END
          MOVE 'Y' TO W1ØØ-MYFILE-EOF
          CLOSE MYFILE
          GO TO 2ØØØ-EXIT
     END-READ
     INSPECT CONTROL-PARM-REC
        TALLYING CHAR-COUNT FOR LEADING SPACE.
     MOVE CONTROL-PARM-REC (CHAR-COUNT + 1: ) TO SRC-PARM-INPUT.
     UNSTRING SRC-PARM-INPUT
       DELIMITED BY ALL SPACES OR "," OR X"ØØ"
       INTO PARM-TYPE           COUNT IN CTR-1
            PARM-DBNAME
            PARM-DBNAME-VALUE
            PARM-QUERY
            PARM-QUERY-TYPE
            PARM-CRI
            PARM-CRI-VAL1
            PARM-CRI-VAL2
            PARM-CRI-VAL3
            PARM-CRI-VAL4
            PARM-CRI-VAL5
            PARM-CRI-VAL6
     PERFORM 21ØØ-PROCESS-PARMS  THRU 21ØØ-EXIT.
 2ØØØ-EXIT.
     EXIT.
 21ØØ-PROCESS-PARMS.
     DISPLAY '         *****  R T S   O B J E C T  '
             ' M A N A G E R   F O R    D B 2    V1R1.ØØ  *****'.
     DISPLAY '               COPYRIGHT (C) 2ØØ2 - 2ØØ3'
             '  Author: Lijun Gao. ALL RIGHTS RESERVED.         '.
     DISPLAY ' '.
       EVALUATE PARM-TYPE
         WHEN "EXTENT"
           MOVE 'EXTIND' TO PGM-NAME
           CALL PGM-NAME USING PARM-INPUT-REC
         WHEN "REORG"
           MOVE 'REORGIND' TO PGM-NAME
           CALL PGM-NAME USING PARM-INPUT-REC
         WHEN "COPY"
           MOVE 'COPYIND' TO PGM-NAME
           CALL PGM-NAME USING PARM-INPUT-REC
```

```
                WHEN "RUNSTATS"
                  MOVE 'RUNSTIND' TO PGM-NAME
                  CALL PGM-NAME USING PARM-INPUT-REC
                WHEN "RESTRICT"
                  MOVE 'RESTRIND' TO PGM-NAME
                  CALL PGM-NAME USING PARM-INPUT-REC
                WHEN  OTHER
                  DISPLAY ".DSNGØ12E Invalid query type " PARM-TYPE
              END-EVALUATE.
        21ØØ-EXIT.
            EXIT.
```

EXTIND PROGRAM

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID.    EXTIND.
        AUTHOR.        LIJUN  GAO;
        DATE-WRITTEN.  Ø8/Ø8/Ø3.
        DATE-COMPILED.
*****************************************************************
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER.  IBM-37Ø.
        OBJECT-COMPUTER.  IBM-37Ø.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
        DATA DIVISION.
        FILE SECTION.
/******************************************************************
        WORKING-STORAGE SECTION.
       *****************************************************************
       * DISPLAY FIELDS FOR USER INPUT CRITERIA
       *****************************************************************
        Ø1  SAVE-TOTALEXTENTS          PIC 9(Ø4) VALUE ZEROES.
        Ø1  EXTENT-DIS-VAL1            PIC ZZZ9.
       *****************************************************************
       * OUTPUT TITLE FOR OBJECTS EXCEED EXTENT LIMITS
       *****************************************************************
        Ø1  LIST-EXTENT-NAMES.
           Ø2  FILLER                PIC X(12) VALUE 'DBNAME'.
           Ø2  FILLER                PIC X(12) VALUE 'NAME'.
           Ø2  FILLER                PIC X(12) VALUE 'TYPE'.
           Ø2  FILLER                PIC X(12) VALUE 'EXTENTS'.
           Ø2  FILLER                PIC X(12) VALUE 'PART'.
           Ø2  FILLER                PIC X(12) VALUE 'ASSOC-TS'.
       *****************************************************************
       * OUTPUT LIST FOR OBJECTS EXCEED EXTENT LIMITS
       *****************************************************************
        Ø1 LIST-EXTENT.
```

```
     Ø2  LIST-EXTENT-DEF  OCCURS 12ØØØ TIMES.
       Ø8  EX-DBNAME            PIC X(8).
       Ø8  FILLER               PIC X(4).
       Ø8  EX-NAME              PIC X(8) VALUE SPACES.
       Ø8  FILLER               PIC X(4).
       Ø8  EX-OBJECTTYPE        PIC X(2).
       Ø8  FILLER               PIC X(1Ø).
       Ø8  EX-TOTALEXTENTS      PIC 9(4) VALUE ZEROES.
       Ø8  FILLER               PIC X(8).
       Ø8  EX-PARTITION         PIC 9(3).
       Ø8  EX-PAR-II REDEFINES EX-PARTITION PIC X(3).
       Ø8  FILLER               PIC X(9).
       Ø8  EX-ASSOCIATEDTS      PIC X(8).
       Ø8  FILLER               PIC X(12).
****************************************************************
* COPY ALL RELATED WORKING STORAGE DEFINITION
****************************************************************
       COPY WRKINPT.
****************************************************************
*  DB2 AREA                                                    *
****************************************************************
       EXEC SQL
            INCLUDE SQLCA
       END-EXEC.
       EXEC SQL
            INCLUDE WSACCOR
       END-EXEC.
 LINKAGE SECTION.
 Ø1  EXTENT-REC.
     Ø5  LINEA.
         Ø7  EXTENT-TYPE                 PIC X(Ø8).
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-DBNAME               PIC X(Ø8).
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-DBNAME-VALUE         PIC X(Ø8).
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-OBJECT               PIC X(4).
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-OBJECT-TYPE          PIC X(3) VALUE 'ALL'.
         Ø7  FILLER                      PIC X VALUE SPACE.
     Ø5  LINEB.
         Ø7  EXTENT-CRI                  PIC X(Ø8).
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-CRI-VAL1             PIC 9(4) VALUE ZERO.
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-CRI-VAL2             PIC 9(4) VALUE ZERO.
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-CRI-VAL3             PIC 9(4) VALUE ZERO.
         Ø7  FILLER                      PIC X VALUE SPACE.
         Ø7  EXTENT-CRI-VAL4             PIC 9(4) VALUE ZERO.
```

```
        Ø7  FILLER                      PIC X VALUE SPACE.
        Ø7  EXTENT-CRI-VAL5             PIC 9(4) VALUE ZERO.
        Ø7  FILLER                      PIC X VALUE SPACE.
        Ø7  EXTENT-CRI-VAL6             PIC 9(4) VALUE ZERO.
        Ø7  FILLER                      PIC X VALUE SPACE.
 PROCEDURE DIVISION USING EXTENT-REC.
 ØØØØ-MAIN-LOGIC.
     PERFORM 1ØØØ-INIT THRU 1ØØØ-EXIT.
     PERFORM 21ØØ-PROCESS-PARMS THRU 21ØØ-EXIT.
     PERFORM 22ØØ-PROCESS-PARMS THRU 22ØØ-EXIT.
     PERFORM 3ØØØ-CONNECT-TO-SERVER THRU 3ØØØ-EXIT.
     IF OKAY THEN
        PERFORM 4ØØØ-CALL-DSNACCOR THRU 4ØØØ-EXIT
        PERFORM 5ØØØ-OUTPUT-RESULT THRU 5ØØØ-EXIT
     ELSE
        DISPLAY 'CONNECT NOT SUCCESSFUL'
        MOVE 8 TO RETURN-CODE.
     EXEC SQL
          CONNECT RESET
     END-EXEC.
     EXIT PROGRAM.
 1ØØØ-INIT.
     MOVE 'GOOD' TO RUN-STATUS.
     ACCEPT REFMOD-TIME-ITEM FROM TIME.
     ACCEPT YYYYMMDD FROM DATE.
     DISPLAY ".DSNGØØ1I Job execution starting at "
             YYYYMMDD (5:2)
                     "/"
             YYYYMMDD (7:2)
                     "/2"
             YYYYMMDD (2:3)
                     "   "
             REFMOD-TIME-ITEM (1:2)
                     ":"
             REFMOD-TIME-ITEM (3:2)
                     ":"
             REFMOD-TIME-ITEM (5:2)
                     " ..."
     DISPLAY '.DSNGØØ2I MVS=SP7.Ø.3,PID=HBB77Ø6,DFSMS=1.3.Ø'
            ',DB2=7.1.Ø'.
     DISPLAY '.DSNGØ18I Connected to Subsystem ' DB2-LOC-NAME.
 1ØØØ-EXIT.
     EXIT.
 21ØØ-PROCESS-PARMS.
       EVALUATE EXTENT-OBJECT
         WHEN "TYPE"
           MOVE EXTENT-OBJECT-TYPE TO OBJECTTYPE-DTA
           MOVE 3 TO OBJECTTYPE-LN
         WHEN OTHER
     DISPLAY ".DSNGØ13E Invalid keyword " EXTENT-OBJECT     BJECT
```

```
            STOP RUN
        END-EVALUATE
        EVALUATE EXTENT-CRI
          WHEN "EXTLIMIT"
            MOVE 'EXTENTS' TO QUERYTYPE-DTA
            MOVE 8 TO QUERYTYPE-LN
            MOVE EXTENT-CRI-VAL1 TO EXTENTLIMIT
          WHEN OTHER
            DISPLAY ".DSNGØ13E Invalid keyword " EXTENT-CRI
            STOP RUN
        END-EVALUATE
        MOVE EXTENT-DBNAME-VALUE TO CRI-VALUE.
        STRING
           CRI-NAME SPACE CRI-POINT
              DELIMITED BY SIZE
           CRI-VALUE
              DELIMITED BY SPACES
           CRI-POINT
              DELIMITED BY SIZE
           CRI-EXC
              DELIMITED BY SIZE
           INTO CRITERIA-DTA.
        MOVE 5Ø TO CRITERIA-LN.
 21ØØ-EXIT.
     EXIT.
 22ØØ-PROCESS-PARMS.
     DISPLAY ".DSNGØ15I QueryType = " EXTENT-TYPE.
     DISPLAY ".DSNGØ15I ObjectTYPE = " EXTENT-object-TYPE.
     MOVE EXTENT-CRI-VAL1 TO EXTENT-DIS-VAL1.
     DISPLAY ".DSNGØ15I ExtentLimit = " EXTENT-DIS-VAL1.
     DISPLAY ".DSNGØ15I QUERYSCOPE  = WHERE " CRITERIA-DTA.
     DISPLAY '   '.
****************************************************************
* PROCESS DSNACCOR INVOCATION PARAMETERS
****************************************************************
     MOVE 23 TO CHKLVL.
     MOVE SPACES TO LASTSTATEMENT-DTA.
     MOVE 1 TO LASTSTATEMENT-LN.
     MOVE Ø TO RETURNCODE.
     MOVE SPACES TO ERRORMSG-DTA.
     MOVE 1 TO ERRORMSG-LN.
     MOVE Ø TO IFCARETCODE.
     MOVE Ø TO IFCARESCODE.
     MOVE Ø TO XSBYTES.
*******************************************************
* SET THE INDICATOR VARIABLES TO Ø FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT    *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT      *
* PARAMETERS.                                         *
*******************************************************
```

```
           MOVE Ø TO CHKLVL-IND.
           MOVE Ø TO CRITERIA-IND.
           MOVE Ø TO EXTENTLIMIT-IND.
           MOVE Ø TO LASTSTATEMENT-IND.
           MOVE Ø TO RETURNCODE-IND.
           MOVE Ø TO ERRORMSG-IND.
           MOVE Ø TO IFCARETCODE-IND.
           MOVE Ø TO IFCARESCODE-IND.
           MOVE Ø TO XSBYTES-IND.
       22ØØ-EXIT.
           EXIT.
       3ØØØ-CONNECT-TO-SERVER.
      *****************************************************************
      * CONNECT TO THE REMOTE SERVER
      *****************************************************************
           EXEC SQL CONNECT TO :DB2-LOC-NAME END-EXEC.
           MOVE 'CONNECT' TO DB2-COMMAND.
           IF SQLCODE IS NOT EQUAL TO ZERO THEN
              PERFORM 9ØØØ-CHECK-SQLCODE.
       3ØØØ-EXIT.
           EXIT.
       4ØØØ-CALL-DSNACCOR.
      *****************
      * CALL DSNACCOR *
      *****************
           EXEC SQL CALL DSNACCOR
          (:QUERYTYPE           :QUERYTYPE-IND,
           :OBJECTTYPE          :OBJECTTYPE-IND,
           :ICTYPE              :ICTYPE-IND,
           :STATSSCHEMA         :STATSSCHEMA-IND,
           :CATLGSCHEMA         :CATLGSCHEMA-IND,
           :LOCALSCHEMA         :LOCALSCHEMA-IND,
           :CHKLVL              :CHKLVL-IND,
           :CRITERIA            :CRITERIA-IND,
           :RESTRICTED          :RESTRICTED-IND,
           :CRUPDATEDPAGESPCT   :CRUPDATEDPAGESPCT-IND,
           :CRCHANGESPCT        :CRCHANGESPCT-IND,
           :CRDAYSNCLASTCOPY    :CRDAYSNCLASTCOPY-IND,
           :ICRUPDATEDPAGESPCT  :ICRUPDATEDPAGESPCT-IND,
           :ICRCHANGESPCT       :ICRCHANGESPCT-IND,
           :CRINDEXSIZE         :CRINDEXSIZE-IND,
           :RRTINSDELUPDPCT     :RRTINSDELUPDPCT-IND,
           :RRTUNCLUSTINSPCT    :RRTUNCLUSTINSPCT-IND,
           :RRTDISORGLOBPCT     :RRTDISORGLOBPCT-IND,
           :RRTMASSDELLIMIT     :RRTMASSDELLIMIT-IND,
           :RRTINDREFLIMIT      :RRTINDREFLIMIT-IND,
           :RRIINSERTDELETEPCT  :RRIINSERTDELETEPCT-IND,
           :RRIAPPENDINSERTPCT  :RRIAPPENDINSERTPCT-IND,
           :RRIPSEUDODELETEPCT  :RRIPSEUDODELETEPCT-IND,
           :RRIMASSDELLIMIT     :RRIMASSDELLIMIT-IND,
           :RRILEAFLIMIT         :RRILEAFLIMIT-IND,
           :RRINUMLEVELSLIMIT    :RRINUMLEVELSLIMIT-IND,
           :SRTINSDELUPDPCT      :SRTINSDELUPDPCT-IND,
           :SRTINSDELUPDABS      :SRTINSDELUPDABS-IND,
           :SRTMASSDELLIMIT      :SRTMASSDELLIMIT-IND,
           :SRIINSDELUPDPCT      :SRIINSDELUPDPCT-IND,
           :SRIINSDELUPDABS      :SRIINSDELUPDABS-IND,
           :SRIMASSDELLIMIT      :SRIMASSDELLIMIT-IND,
           :EXTENTLIMIT          :EXTENTLIMIT-IND,
           :LASTSTATEMENT        :LASTSTATEMENT-IND,
           :RETURNCODE           :RETURNCODE-IND,
           :ERRORMSG             :ERRORMSG-IND,
           :IFCARETCODE          :IFCARETCODE-IND,
           :IFCARESCODE          :IFCARESCODE-IND,
           :XSBYTES              :XSBYTES-IND)
         END-EXEC.
           MOVE 'CALL' TO DB2-COMMAND.
           IF SQLCODE IS NOT EQUAL TO +466 THEN
              PERFORM 9ØØØ-CHECK-SQLCODE
           ELSE
              PERFORM 41ØØ-GET-RESULT.
     4ØØØ-EXIT.
          EXIT.
     41ØØ-GET-RESULT.
          IF RETURNCODE NOT EQUAL TO Ø THEN
          DISPLAY 'ERRORMSG ' ERRORMSG
          DISPLAY 'RETURNCODE' RETURNCODE
          DISPLAY IFCARETCODE IFCARESCODE XSBYTES
          DISPLAY 'LASTSTATEMENT' LASTSTATEMENT
          ELSE
          DISPLAY '.DSNGØ11I' ERRORMSG.
          EXEC SQL ASSOCIATE LOCATORS(:LOC1,  :LOC2)
                   WITH PROCEDURE DSNACCOR
          END-EXEC.
          EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1
          END-EXEC
          EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :LOC2
          END-EXEC
          PERFORM 42ØØ-DIS-TITLE.
          PERFORM 43ØØ-GET-RESULT2 VARYING I
           FROM 1 BY 1 UNTIL SQLCODE EQUAL TO +1ØØ.
     42ØØ-DIS-TITLE.
           DISPLAY '**************************************************'
                   '*****************************'
           DISPLAY '*  PART: Partition number for partitioned tab'
                   'le space                        *'
           DISPLAY '*  ASSO-TS: Associated tablespace name for ind'
                   'ex space                      *'
           DISPLAY '**************************************************'
                   '*****************************'
     DISPLAY ' '.
     DISPLAY LIST-EXTENT-NAMES.
     DISPLAY '-----------------------------------------------'
             '-------------------'.
 42ØØ-EXIT.
     EXIT.
 43ØØ-GET-RESULT2.
     EXEC SQL FETCH C2 INTO :RS-OUTPUT-2 :RS-OUT2-IND
     END-EXEC.
     MOVE 'FETCH' TO DB2-COMMAND.
     PERFORM 9ØØØ-CHECK-SQLCODE.
     MOVE RS-DBNAME TO EX-DBNAME(I).
     MOVE RS-NAME TO EX-NAME(I).
     MOVE RS-OBJECTTYPE TO EX-OBJECTTYPE(I).
     MOVE RS-ASSOCIATEDTS TO EX-ASSOCIATEDTS(I).
     IF RS-TOTALEXTENTS NOT EQUAL TO Ø THEN
       MOVE RS-TOTALEXTENTS TO EX-TOTALEXTENTS(I) .
     IF RS-PARTITION EQUAL TO Ø THEN
       MOVE LOW-VALUES TO EX-PAR-II(I)
     ELSE
       MOVE RS-PARTITION TO EX-PARTITION(I).
 5ØØØ-OUTPUT-RESULT.
*****************************************************************
*    SORTS THE EXTENT RECORD WITH HIGHEST NUMBER FIRST          *
*****************************************************************
      PERFORM 51ØØ-SORT-RESULT THRU 51ØØ-EXIT
          VARYING WS-IDX FROM 1 BY 1
              UNTIL EX-NAME (WS-IDX) = SPACES.
     DISPLAY '-----------------------------------------------'
             '-------------------'.
     COMPUTE I = I - 1.
     MOVE I TO DIS-I.
     DISPLAY '.DSNGØ21I TOTAL ' DIS-I ' RECORDS RETRIEVED.'.
 5ØØØ-EXIT.
     EXIT.
 51ØØ-SORT-RESULT.
*****************************************************************
*    SORTS THE EXTENT RECORD WITH HIGHEST NUMBER FIRST          *
*****************************************************************
      MOVE ZEROES TO SAVE-TOTALEXTENTS.
      PERFORM 52ØØ-HIGHEST-EXTENT THRU 52ØØ-EXIT
          VARYING WS-IDX2 FROM 1 BY 1
              UNTIL EX-NAME (WS-IDX2) = SPACES.
      DISPLAY LIST-EXTENT-DEF(WS-IDX-MAX)
      MOVE ZEROES TO EX-TOTALEXTENTS (WS-IDX-MAX).
 51ØØ-EXIT.
       EXIT.
 52ØØ-HIGHEST-EXTENT.
*****************************************************************
*    DETERMINES THE HIGHEST EXTENT NUMBER                       *
```

```
      *****************************************************************
           IF EX-TOTALEXTENTS (WS-IDX2) > SAVE-TOTALEXTENTS
               MOVE WS-IDX2 TO WS-IDX-MAX
               MOVE EX-TOTALEXTENTS(WS-IDX2) TO SAVE-TOTALEXTENTS
           ELSE
               NEXT SENTENCE.
       5200-EXIT.
             EXIT.
       9000-CHECK-SQLCODE.
      ****************************************************************
      * VERIFY THAT THE PRIOR SQL CALL COMPLETED SUCCESSFULLY
      ****************************************************************
            IF SQLCODE NOT = 0 AND SQLCODE NOT = 100 THEN
               MOVE 'BAD' TO RUN-STATUS
               DISPLAY '*    UNEXPECTED SQLCODE FROM SYSPROC.DANACCOR'
                            ' DURING ' DB2-COMMAND ' REQUEST.'
               DISPLAY '*'
               PERFORM 9100-DETAIL-SQL-ERROR.
       9100-DETAIL-SQL-ERROR.
      ****************************************************************
      * CALL DSNTIAR TO RETURN A TEXT MESSAGE FOR AN UNEXPECTED
      * SQLCODE.
      ****************************************************************
            CALL 'DSNTIAR' USING SQLCA ERROR-MESSAGE ERROR-TEXT-LEN.
            IF RETURN-CODE = ZERO
               PERFORM 9200-PRINT-SQL-ERROR-MSG VARYING ERROR-INDEX
                  FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 10.
       9200-PRINT-SQL-ERROR-MSG.
      ****************************************************************
      * PRINT MESSAGE TEXT
      ****************************************************************
            DISPLAY ERROR-TEXT (ERROR-INDEX).
```

REORGIND CODE

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    REORGIND.
       AUTHOR.        LIJUN  GAO;
       DATE-WRITTEN.  08/08/03.
       DATE-COMPILED.
*****************************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-370.
       OBJECT-COMPUTER.  IBM-370.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
/****************************************************************
       DATA DIVISION.
       FILE SECTION.
/****************************************************************
       WORKING-STORAGE SECTION.
      ****************************************************************
      * DISPLAY FIELDS FOR INPUT CRITERIA
      ****************************************************************
       Ø1  DIS-EXTENTLIMIT               PIC ZZZ9.
       Ø1  DIS-RRTINSDELUPDPCT           PIC ZZZZ9.
       Ø1  DIS-RRTUNCLUSTINSPCT          PIC ZZZZ9.
       Ø1  DIS-RRTDISORGLOBPCT           PIC ZZZZ9.
       Ø1  DIS-RRTMASSDELLIMIT           PIC ZZZZ9.
       Ø1  DIS-RRTINDREFLIMIT            PIC ZZZZ9.
      ****************************************************************
      * OUTPUT TITLE FOR OBJECTS EXCEED REORG CRITERIA LIMITS
      ****************************************************************
       Ø1  LIST-REORG-NAMES.
          Ø2  FILLER                PIC X(9) VALUE 'DBNAME'.
          Ø2  FILLER                PIC X(9) VALUE 'NAME'.
          Ø2  FILLER                PIC X(3) VALUE 'TP'.
          Ø2  FILLER                PIC X(4) VALUE 'EXT'.
          Ø2  FILLER                PIC X(6) VALUE 'IDU'.
          Ø2  FILLER                PIC X(6) VALUE 'UCI'.
          Ø2  FILLER                PIC X(6) VALUE 'DOL'.
          Ø2  FILLER                PIC X(6) VALUE 'MSD'.
          Ø2  FILLER                PIC X(6) VALUE 'IDR'.
          Ø2  FILLER                PIC X(2Ø) VALUE 'REORG-LASTTIME'.
          Ø2  FILLER                PIC X(5) VALUE 'PART'.
      ****************************************************************
      * OUTPUT LIST FOR OBJECTS EXCEED REORG LIMITS
      ****************************************************************
       Ø1 LIST-REORG.
          Ø2  LIST-REORG-DEF OCCURS 12ØØØ TIMES.
            Ø8  EX-DBNAME             PIC X(8).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-NAME               PIC X(8).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-OBJECTTYPE         PIC X(2).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-TOTALEXTENTS   PIC 9(Ø3) VALUE ZEROES.
            Ø8  FILLER                PIC X(1).
            Ø8  EX-RRTINSDELUPDPCT    PIC 9(5).
            Ø8  EX-RRIDU-II REDEFINES EX-RRTINSDELUPDPCT PIC X(5).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-RRTUNCINSPCT       PIC 9(5).
            Ø8  EX-RRUCI-II REDEFINES EX-RRTUNCINSPCT PIC X(5).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-RRTDISORGLOBPCT    PIC 9(5).
            Ø8  EX-RRDOL-II REDEFINES EX-RRTDISORGLOBPCT PIC X(5).
            Ø8  FILLER                PIC X(1).
            Ø8  EX-RRTMASSDELETE      PIC 9(5).
            Ø8  EX-RRMSD-II REDEFINES EX-RRTMASSDELETE PIC X(5).
            Ø8  FILLER               PIC X(1).
            Ø8  EX-RRTINDREF         PIC 9(5).
            Ø8  EX-RRIDR-II REDEFINES EX-RRTINDREF PIC X(5).
            Ø8  FILLER               PIC X(1).
            Ø8  EX-REORGLASTTIME     PIC X(19).
            Ø8  EX-RLT-II REDEFINES EX-REORGLASTTIME PIC X(19).
            Ø8  FILLER               PIC X(1).
            Ø8  EX-PARTITION         PIC 9(3).
            Ø8  EX-PAR-II REDEFINES EX-PARTITION PIC X(3).
            Ø8  FILLER               PIC X(1).
            Ø8  EX-ASSOCIATEDTS      PIC X(8).
            Ø8  FILLER               PIC X(1).
     *****************************************************************
      * COPY ALL RELATED WORKING STORAGE DEFINITION
      *****************************************************************
            COPY WRKINPT.
      *****************************************************************
      *  DB2 AREA                                                     *
      *****************************************************************
            EXEC SQL
                 INCLUDE SQLCA
            END-EXEC.
            EXEC SQL
                 INCLUDE WSACCOR
            END-EXEC.
       LINKAGE SECTION.
       Ø1  REORG-REC.
           Ø5  LINEA.
               Ø7  REORG-TYPE                  PIC X(Ø8).
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-DBNAME                PIC X(Ø8).
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-DBNAME-VALUE          PIC X(Ø8).
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-OBJECT                PIC X(4).
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-OBJECT-TYPE           PIC X(3).
               Ø7  FILLER                      PIC X VALUE SPACE.
           Ø5  LINEB.
               Ø7  REORG-CRI                   PIC X(Ø8).
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-CRI-VAL1              PIC 9(4) VALUE ZERO.
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-CRI-VAL2              PIC 9(4) VALUE ZERO.
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-CRI-VAL3              PIC 9(4) VALUE ZERO.
               Ø7  FILLER                      PIC X VALUE SPACE.
               Ø7  REORG-CRI-VAL4              PIC 9(4) VALUE ZERO.
               Ø7  FILLER                      PIC X VALUE SPACE.
```

```
          Ø7  REORG-CRI-VAL5             PIC 9(4) VALUE ZERO.
          Ø7  FILLER                     PIC X VALUE SPACE.
          Ø7  REORG-CRI-VAL6             PIC 9(4) VALUE ZERO.
          Ø7  FILLER                     PIC X VALUE SPACE.
 PROCEDURE DIVISION USING REORG-REC.
 ØØØØ-MAIN-LOGIC.
     PERFORM 1ØØØ-INIT          THRU 1ØØØ-EXIT.
     PERFORM 21ØØ-PROCESS-PARMS THRU 21ØØ-EXIT.
     PERFORM 22ØØ-PROCESS-PARMS THRU 22ØØ-EXIT.
     PERFORM 3ØØØ-CONNECT-TO-SERVER THRU 3ØØØ-EXIT.
     IF OKAY THEN
        PERFORM 4ØØØ-CALL-DSNACCOR THRU 4ØØØ-EXIT
     ELSE
        DISPLAY 'CONNECT NOT SUCCESSFUL'
        MOVE 8 TO RETURN-CODE.
     EXEC SQL
          CONNECT RESET
     END-EXEC.
     STOP RUN.
 1ØØØ-INIT.
     MOVE 'GOOD' TO RUN-STATUS.
     ACCEPT REFMOD-TIME-ITEM FROM TIME.
     ACCEPT YYYYMMDD FROM DATE.
     DISPLAY ".DSNGØØ1I Job execution starting at "
             YYYYMMDD (5:2)
                     "/"
             YYYYMMDD (7:2)
                     "/2"
             YYYYMMDD (2:3)
                     "   "
             REFMOD-TIME-ITEM (1:2)
                     ":"
             REFMOD-TIME-ITEM (3:2)
                     ":"
             REFMOD-TIME-ITEM (5:2)
                     " ..."
     DISPLAY '.DSNGØØ2I MVS=SP7.Ø.3,PID=HBB77Ø6,DFSMS=1.3.Ø'
            ',DB2=7.1.Ø'.
     DISPLAY '.DSNGØ18I Connected to subsystem ' DB2-LOC-NAME.
 1ØØØ-EXIT.
     EXIT.
 21ØØ-PROCESS-PARMS.
       EVALUATE REORG-OBJECT
         WHEN "TYPE"
           MOVE 'TS' TO OBJECTTYPE-DTA
           MOVE 3 TO OBJECTTYPE-LN
         WHEN OTHER
           DISPLAY ".DSNGØ13E Invalid keyword " REORG-OBJECT
           STOP RUN
       END-EVALUATE
```

```
           EVALUATE REORG-CRI
             WHEN "REORGCRI"
               MOVE 'REORG' TO QUERYTYPE-DTA
               MOVE 8 TO QUERYTYPE-LN
             WHEN OTHER
               DISPLAY ".DSNGØ13E Invalid keyword " REORG-CRI
               STOP RUN
           END-EVALUATE
           MOVE REORG-DBNAME-VALUE TO CRI-VALUE.
           STRING
              CRI-NAME SPACE CRI-POINT
                 DELIMITED BY SIZE
              CRI-VALUE
                 DELIMITED BY SPACES
              CRI-POINT
                 DELIMITED BY SIZE
              CRI-EXC
                 DELIMITED BY SIZE
              INTO CRITERIA-DTA.
           MOVE 5Ø TO CRITERIA-LN.
           IF REORG-CRI-VAL1 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL1 TO RRTINSDELUPDPCT.
           IF REORG-CRI-VAL2 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL2 TO RRTUNCLUSTINSPCT.
           IF REORG-CRI-VAL3 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL3 TO RRTDISORGLOBPCT.
           IF REORG-CRI-VAL4 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL4 TO RRTMASSDELLIMIT.
           IF REORG-CRI-VAL5 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL5 TO RRTINDREFLIMIT.
           IF REORG-CRI-VAL6 NOT EQUAL TO SPACE AND ZERO THEN
             MOVE REORG-CRI-VAL6 TO EXTENTLIMIT.
       21ØØ-EXIT.
           EXIT.
       22ØØ-PROCESS-PARMS.
      *****************************************************************
      * PROCESS DSNACCOR INVOCATION PARAMETERS
      *****************************************************************
           MOVE 59 TO CHKLVL.
           DISPLAY ".DSNGØ15I QueryType = " reorg-type
           DISPLAY ".DSNGØ15I ObjectType = " reorg-object-type
           IF REORG-OBJECT-TYPE NOT EQUAL TO "TS"
              DISPLAY '.DSNGØ16I Query type REORG will be'
                      ' limited to tablespace only'
           END-IF
           MOVE RRTINSDELUPDPCT TO DIS-RRTINSDELUPDPCT.
           MOVE RRTUNCLUSTINSPCT TO DIS-RRTUNCLUSTINSPCT
           MOVE RRTDISORGLOBPCT TO DIS-RRTDISORGLOBPCT
           MOVE RRTMASSDELLIMIT TO DIS-RRTMASSDELLIMIT
           MOVE RRTINDREFLIMIT TO DIS-RRTINDREFLIMIT
```

```
     MOVE EXTENTLIMIT    TO DIS-EXTENTLIMIT.
     DISPLAY ".DSNGØ15I RRTINSDELUPDPCT = " DIS-RRTINSDELUPDPCT
     DISPLAY ".DSNGØ15I RRTUNCLUSTINSPCT = " DIS-RRTUNCLUSTINSPCT
     DISPLAY ".DSNGØ15I RRTDISORGLOBPCT = " DIS-RRTDISORGLOBPCT
     DISPLAY ".DSNGØ15I RRTMASSDELLIMIT = " DIS-RRTMASSDELLIMIT
     DISPLAY ".DSNGØ15I RRTINDREFLIMIT = " DIS-RRTINDREFLIMIT
     DISPLAY ".DSNGØ15I EXTENTLIMIT = " DIS-EXTENTLIMIT
     DISPLAY ".DSNGØ15I QueryScope  = WHERE " CRITERIA-DTA.
     DISPLAY '  '.
********************************
* INITIALIZE OUTPUT PARAMETERS *
********************************
     MOVE SPACES TO LASTSTATEMENT-DTA.
     MOVE 1 TO LASTSTATEMENT-LN.
     MOVE Ø TO RETURNCODE.
     MOVE SPACES TO ERRORMSG-DTA.
     MOVE 1 TO ERRORMSG-LN.
     MOVE Ø TO IFCARETCODE.
     MOVE Ø TO IFCARESCODE.
     MOVE Ø TO XSBYTES.
*****************************************************
* SET THE INDICATOR VARIABLES TO Ø FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT    *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT      *
* PARAMETERS.                                         *
*****************************************************
      MOVE Ø TO CHKLVL-IND.
      MOVE Ø TO CRITERIA-IND.
      MOVE Ø TO RRTINSDELUPDPCT-IND.
      MOVE Ø TO RRTUNCLUSTINSPCT-IND.
      MOVE Ø TO RRTDISORGLOBPCT-IND.
      MOVE Ø TO RRTMASSDELLIMIT-IND.
      MOVE Ø TO RRTINDREFLIMIT-IND.
      MOVE Ø TO EXTENTLIMIT-IND.
      MOVE Ø TO LASTSTATEMENT-IND.
      MOVE Ø TO RETURNCODE-IND.
      MOVE Ø TO ERRORMSG-IND.
      MOVE Ø TO IFCARETCODE-IND.
      MOVE Ø TO IFCARESCODE-IND.
      MOVE Ø TO XSBYTES-IND.
 22ØØ-EXIT.
     EXIT.
 3ØØØ-CONNECT-TO-SERVER.
***************************************************************
* CONNECT TO THE REMOTE SERVER
***************************************************************
      EXEC SQL CONNECT TO :DB2-LOC-NAME END-EXEC.
      MOVE 'CONNECT' TO DB2-COMMAND.
      IF SQLCODE IS NOT EQUAL TO ZERO THEN
         PERFORM 9ØØØ-CHECK-SQLCODE.
 3ØØØ-EXIT.
     EXIT.
 4ØØØ-CALL-DSNACCOR.
*****************
* CALL DSNACCOR *
*****************
      EXEC SQL CALL DSNACCOR
     (:QUERYTYPE           :QUERYTYPE-IND,
      :OBJECTTYPE          :OBJECTTYPE-IND,
      :ICTYPE              :ICTYPE-IND,
      :STATSSCHEMA         :STATSSCHEMA-IND,
      :CATLGSCHEMA         :CATLGSCHEMA-IND,
      :LOCALSCHEMA         :LOCALSCHEMA-IND,
      :CHKLVL              :CHKLVL-IND,
      :CRITERIA            :CRITERIA-IND,
      :RESTRICTED          :RESTRICTED-IND,
      :CRUPDATEDPAGESPCT   :CRUPDATEDPAGESPCT-IND,
      :CRCHANGESPCT        :CRCHANGESPCT-IND,
      :CRDAYSNCLASTCOPY    :CRDAYSNCLASTCOPY-IND,
      :ICRUPDATEDPAGESPCT  :ICRUPDATEDPAGESPCT-IND,
      :ICRCHANGESPCT       :ICRCHANGESPCT-IND,
      :CRINDEXSIZE         :CRINDEXSIZE-IND,
      :RRTINSDELUPDPCT     :RRTINSDELUPDPCT-IND,
      :RRTUNCLUSTINSPCT    :RRTUNCLUSTINSPCT-IND,
      :RRTDISORGLOBPCT     :RRTDISORGLOBPCT-IND,
      :RRTMASSDELLIMIT     :RRTMASSDELLIMIT-IND,
      :RRTINDREFLIMIT      :RRTINDREFLIMIT-IND,
      :RRIINSERTDELETEPCT  :RRIINSERTDELETEPCT-IND,
      :RRIAPPENDINSERTPCT  :RRIAPPENDINSERTPCT-IND,
      :RRIPSEUDODELETEPCT  :RRIPSEUDODELETEPCT-IND,
      :RRIMASSDELLIMIT     :RRIMASSDELLIMIT-IND,
      :RRILEAFLIMIT        :RRILEAFLIMIT-IND,
      :RRINUMLEVELSLIMIT   :RRINUMLEVELSLIMIT-IND,
      :SRTINSDELUPDPCT     :SRTINSDELUPDPCT-IND,
      :SRTINSDELUPDABS     :SRTINSDELUPDABS-IND,
      :SRTMASSDELLIMIT     :SRTMASSDELLIMIT-IND,
      :SRIINSDELUPDPCT     :SRIINSDELUPDPCT-IND,
      :SRIINSDELUPDABS     :SRIINSDELUPDABS-IND,
      :SRIMASSDELLIMIT     :SRIMASSDELLIMIT-IND,
      :EXTENTLIMIT         :EXTENTLIMIT-IND,
      :LASTSTATEMENT       :LASTSTATEMENT-IND,
      :RETURNCODE          :RETURNCODE-IND,
      :ERRORMSG            :ERRORMSG-IND,
      :IFCARETCODE         :IFCARETCODE-IND,
      :IFCARESCODE         :IFCARESCODE-IND,
      :XSBYTES             :XSBYTES-IND)
    END-EXEC.
      MOVE 'CALL' TO DB2-COMMAND.
      IF SQLCODE IS NOT EQUAL TO +466 THEN
         PERFORM 9ØØØ-CHECK-SQLCODE
     ELSE
        PERFORM 41ØØ-GET-RESULT.
4ØØØ-EXIT.
    EXIT.
41ØØ-GET-RESULT.
    IF RETURNCODE NOT EQUAL TO Ø THEN
    DISPLAY 'ERRORMSG' ERRORMSG
    DISPLAY 'RETURNCODE' RETURNCODE
    DISPLAY 'LASTSTATEMENT' LASTSTATEMENT
    DISPLAY IFCARETCODE IFCARESCODE XSBYTES
    ELSE
    DISPLAY '.DSNGØ11I ' ERRORMSG.
    EXEC SQL ASSOCIATE LOCATORS(:LOC1,  :LOC2)
             WITH PROCEDURE DSNACCOR
    END-EXEC.
    EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1
    END-EXEC
    EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :LOC2
    END-EXEC
    PERFORM 4Ø5Ø-DIS-TITLE
    PERFORM 43ØØ-GET-RESULT2 THRU 43ØØ-EXIT VARYING I
     FROM 1 BY 1 UNTIL SQLCODE EQUAL TO +1ØØ.
    DISPLAY '-----------------------------------------------'
            '-----------------------------'.
    COMPUTE I = I - 1.
    COMPUTE J = J - 1.
    MOVE I TO DIS-I.
    MOVE J TO DIS-J.
    DISPLAY '.DSNGØ21I TOTAL ' DIS-I ' RECORDS RETRIEVED AND '
            DIS-J ' RECORDS DISPLAYED.'.
4Ø5Ø-DIS-TITLE.
     DISPLAY '*************************************************'
             '*****************************'
     DISPLAY '*  iDU: The ratio of the sum of INS, UPD, DEL'
             ' to the total number of rows.   *'
     DISPLAY '*  UCI: The ratio of number of unclustered INS'
             ' to the total number of rows.  *'
     DISPLAY '*  DOL: The ratio of number of chunked LOBs'
             ' to the total number of rows.     *'
     DISPLAY '*  MSD: The number of mass DEL.           '
             '                                  *'
     DISPLAY '*  IDR: The number of overflow records created'
             ' to the total number of rows.  *'
     DISPLAY '*************************************************'
             '*****************************'
    DISPLAY ' '.
    DISPLAY LIST-REORG-NAMES.
    DISPLAY '-----------------------------------------------'
            '-----------------------------'.
```

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

Lijun Gao (legend_gao@yahoo.com)
Senior DB2 System Programmer (USA)

DB2 news

Embarcadero Technologies has announced that its DBArtisan Workbench has been extended to support DB2 UDB Versions 7.2 and 8.1.

DBArtisan Workbench is a database administration solution. It includes all three of the Embarcadero Analyst Series components: DBArtisan Space Analyst, DBArtisan Capacity Analyst, and DBArtisan Performance Analyst. Through a user interface, users can detect and correct problems early without coding long and complicated scripts. Users can also prevent downtime by anticipating future database growth and adding capacity before problems occur, and save time by automating the space maintenance of DB2 objects.

For further information contact:
Embarcadero Technologies, 100 California St, 12th Floor, San Francisco, CA 94111, USA.
Tel: (415) 834 3131.
URL: http://www.embarcadero.com/products/dbartisan/index.html.

* * *

Quest Software has announced Quest Central for Databases, which offers support for DB2 SYSPLEX. The product can perform real-time diagnostics of performance bottlenecks in data sharing environments, which helps to identify and resolve performance issues. This in turn reduces the resource costs related to managing and maintaining complex DB2 SYSPLEX environments.

The Quest Spotlight graphical interface displays all of the members of a data-sharing group concurrently. It performs diagnostics and provides a summarized view of the activity that is taking place across the SYSPLEX.

For further information contact:
Quest Software, 800 Irvine Center Drive, Irvine, CA 92618, USA.
Tel: (949) 9754 8000.
URL: http://www.quest.com/quest_central/db2.

* * *

TeamQuest Software has announced TeamQuest Performance 9.1 with scalability, statistical analysis, and multi-system modelling advancements. The new version of the performance management and capacity planning software provides additional capabilities, making it easier to predict performance when servers are added to horizontally scale a tiered network of servers. The product now includes agents for DB2 UDB, WebSphere, and EMC.

For further information contact:
Teamquest, One TeamQuest Way, Clear Lake, IA 50428, USA.
Tel: (641) 357 2700.
URL: http://www.teamquest.com/newsletter/2004/1q/highlights.shtml.

* * *

DataDirect Technologies has announced Version 2.1 of DataDirect Connect for .Net, which offers enhanced connection, failover, and client-side load balancing capabilities. The software, for environments that run Microsoft's .Net, also offers password and user ID encryption for DB2.

For further information contact:
DataDirect Technologies, 3202 Tower Oaks Blvd, Suite 300, Rockville, MD 20852, USA.
Tel: (301) 468 8501.
URL: http://www.datadirect.com/products/dotnet/index.ssp.