Copying DB2 Runstats from One Subsystem to Another

Nancy Perkinson

Rarely will you find two organizations that have the same types of DBAs. Some shops have a solid DBA group that is involved in applications and performance as well as system-level activities. Some shops split up the DBAs with emphasis on applications or performance or system-level activities. Other companies have DBAs that simply handle the facts and figures given them by application areas and just do the maintenance work of setting up tables, indexes, and necessary utilities. Still other shops let their application folks set up their own tables and indexes with very little DBA education or direction (gasp!). Many DBA groups are overworked and/or understaffed, and generally they appreciate any help that a Performance Analyst can give them in an effort to make the DB2 subsystems perform better.

Your DBAs may already have a utility to copy DB2 runstats from one subsystem to another, and if so, thanks for reading, you can now move on to another article. But if you do not yet have access to such a utility this article will give you the step by step process that you'll need to follow to create one. The finished product may be something you hand off to the DBAs, or give to application programmers, or keep for the performance staff – its final home is entirely based on your unique situation and roles of your DBAs.

There is a never-ending argument concerning what should be reflected by the runstats in a test DB2 subsystem. Should the test runstats reflect the actual data in the test tables, or should the test runstats reflect the equivalent production tables?

Arguments for test runstats reflecting actual data in test DB2 tables:

- Things will run optimally for testing situations. If your test table if filled with skewed data, then the test runstats will show this and the optimizer can make his decisions using the best information for the reality of the situation.
- You can have different indexes defined for test tables, and you don't have to worry about table/index discrepancies between production and test.

Arguments for test runstats reflecting their equivalent production tables:

- Production runstats copied to test provide the optimizer with the information to best emulate the performance you'll see in production.
- Great big tables/indexes in test subsystems take great big amounts of CPU to gather runstats on them. It uses very little CPU to copy production runstats to a test environment.
- Typically production runstats are more up-to-date, and test runstats can languish without being updated, thus the test runstats can really reflect nothing that looks like reality.

I'm sure every shop can come up with more arguments to enhance each side of the debate. The beauty of having a Prod-to-Test runstat copy is that you can pick and choose which DB2 tables will follow which side of the argument.

Step-by-Step SQL calls for copying runstat information:

You can code some simple JCL to pass in a DB2 table name, or fashion some other way to get a table name (or list of table names) into the program. For our example, this information will be stored in WS-INPUT-TBNAME and WS-INPUT-TBCREATOR. You can put the following SQL calls in one small program, padded with very basic code to hold temporary information in several storage arrays.

First, connect to the source DB2 subsystem for the stats (probably your PROD subsystem).

Then, select from SYSTABLES and SYSTABLESPACE (QUERY #1):

QUERY #1:

```
EXEC SQL
    SELECT A.CARD
          ,A.CARDF
          ,A.NPAGES
          ,A.NPAGESF
          ,B.NACTIVE
          ,B.NACTIVEF
          ,A.PCTROWCOMP
          ,B.NAME
          ,B.DBNAME
    INTO
          :WS-SYSTABLES-CARD
         ,:WS-SYSTABLES-CARDF
         ,:WS-SYSTABLES-NPAGES
         ,:WS-SYSTABLES-NPAGESF
         ,:WS-SYSTABLESPACE-NACTIVE
         ,:WS-SYSTABLESPACE-NACTIVEF
         ,:WS-SYSTABLES-PCTROWCOMP
         ,:WS-SYSTABLESPACE-V-NAME
         ,:WS-SYSTABLESPACE-V-DBNAME
    FROM SYSIBM.SYSTABLES A, SYSIBM.SYSTABLESPACE B
     WHERE A.NAME    = :WS-INPUT-TBNAME
       AND A.CREATOR = :WS-INPUT-TBCREATOR
       AND A.TSNAME  = B.NAME
       AND A.DBNAME  = B.DBNAME
 END-EXEC
```

Store this information in storage variables because you'll need it in just a little bit.

Then, open a cursor on SYSTABSTATS for each DBNAME and TSNAME:

<u>QUERY #2:</u>

```
EXEC SQL
   DECLARE SYTT_C CURSOR FOR SELECT
            A.CARD
           ,A.NPAGES
           ,A.PCTPAGES
           ,A.NACTIVE
           ,A.PCTROWCOMP
           ,A.STATSTIME
           ,A.IBMREQD
           ,A.DBNAME
           ,A.TSNAME
           ,A.PARTITION
           ,A.OWNER
           ,A.NAME
           ,A.CARDF
      FROM SYSIBM.SYSTABSTATS A
      WHERE A.TSNAME  = :WS-SYSTABLESPACE-V-NAME
        AND A.DBNAME  = :WS-SYSTABLESPACE-V-DBNAME
      ORDER BY A.PARTITION
END-EXEC.
```

Then fetch each row from this cursor and store the info in a storage array (ARRAY #1).

When you are done fetching, close that cursor and open a new one on SYSCOLUMNS for each TBNAME and TBCREATOR:

<u>QUERY #3:</u>

```
EXEC SQL
   DECLARE SYCL_C CURSOR FOR SELECT
            A.NAME
           ,A.COLCARD
           ,A.COLCARDF
           ,A.HIGH2KEY
           ,A.LOW2KEY
      FROM SYSIBM.SYSCOLUMNS A
      WHERE A.TBNAME    = :WS-INPUT-TBNAME
        AND A.TBCREATOR = :WS-INPUT-TBCREATOR
      ORDER BY A.NAME
END-EXEC.
```

Fetch each row for this cursor and store the info in another storage array (ARRAY #2).

When you are done fetching, close that cursor and open a new one on SYSINDEXES for each TBNAME and TBCREATOR:

QUERY #4:

```
EXEC SQL
   DECLARE SYIX_C CURSOR FOR SELECT
            A.CLUSTERRATIO
           ,A.CLUSTERRATIOF
           ,A.FULLKEYCARD
           ,A.FULLKEYCARDF
           ,A.FIRSTKEYCARD
           ,A.FIRSTKEYCARDF
           ,A.NAME
           ,A.CREATOR
           ,A.NLEAF
           ,A.NLEVELS
      FROM SYSIBM.SYSINDEXES A
      WHERE A.TBNAME    = :WS-INPUT-TBNAME
        AND A.TBCREATOR = :WS-INPUT-TBCREATOR
      ORDER BY A.NAME
END-EXEC
```

Fetch each row for this cursor and store the info in another storage array (ARRAY #3).

When you are done fetching, close that cursor and open a new cursor on SYSCOLDIST using TBNAME and TBCREATOR:

QUERY #5:

```
EXEC SQL
   DECLARE SYCD_C CURSOR FOR SELECT
            A.NAME
           ,A.COLGROUPCOLNO
           ,A.COLVALUE
           ,A.FREQUENCY
           ,A.FREQUENCYF
           ,A.NUMCOLUMNS
           ,A.STATSTIME
           ,A.TBNAME
           ,A.TBOWNER
           ,A.IBMREQD
           ,A.TYPE
           ,A.CARDF
      FROM SYSIBM.SYSCOLDIST A
      WHERE A.TBNAME    = :WS-INPUT-TBNAME
        AND A.TBOWNER   = :WS-INPUT-TBCREATOR
      ORDER BY A.NAME
END-EXEC.
```

Fetch each row for this cursor and store the info in another storage array (ARRAY #4).

When done fetching all rows, close this cursor.

Now, connect to your target DB2 subsystem (probably your test subsystem).

Once you are connected to your target subsystem, start updating the information. First, update SYSTABLES with the information you got back from that first select you did on the source subsystem (ref QUERY #1):

```
EXEC SQL UPDATE SYSIBM.SYSTABLES
    SET
        CARD            = :WS-SYSTABLES-CARD
       ,CARDF           = :WS-SYSTABLES-CARDF
       ,NPAGES          = :WS-SYSTABLES-NPAGES
       ,NPAGESF         = :WS-SYSTABLES-NPAGESF
       ,PCTROWCOMP      = :WS-SYSTABLES-PCTROWCOMP
    WHERE NAME          = :WS-INPUT-TBNAME
      AND CREATOR       = :WS-INPUT-TBCREATOR
END-EXEC
```

Then update SYSTABLESPACE with those same of stats retrieved from QUERY #1:

```
EXEC SQL UPDATE SYSIBM.SYSTABLESPACE
    SET
        NACTIVE         = :WS-SYSTABLESPACE-NACTIVE
       ,NACTIVEF        = :WS-SYSTABLESPACE-NACTIVEF
    WHERE NAME          = :WS-SYSTABLESPACE-V-NAME
      AND DBNAME        = :WS-SYSTABLESPACE-V-DBNAME
END-EXEC
```

Next, delete the stats on SYSTABSTATS in your target subsystem:

```
EXEC SQL
    DELETE
    FROM SYSIBM.SYSTABSTATS
    WHERE TSNAME        = :WS-SYSTABLESPACE-V-NAME
      AND DBNAME        = :WS-SYSTABLESPACE-V-DBNAME
END-EXEC
```

Then, insert the new stats on SYSTABSTATS in your target subsystem – go through each item in your array that you saved earlier for this (ARRAY #1) referenced in QUERY #2:

```
EXEC SQL INSERT
    INTO SYSIBM.SYSTABSTATS
    (
        CARD,
        NPAGES,
        PCTPAGES,
        NACTIVE,
        PCTROWCOMP,
        STATSTIME,
        IBMREQD,
        DBNAME,
        TSNAME,
        PARTITION,
        OWNER,
        NAME,
        CARDF
    )
    VALUES
    (
        :WS-SYSTABSTATS-CARD,
        :WS-SYSTABSTATS-NPAGES,
         :WS-SYSTABSTATS-PCTPAGES,
         :WS-SYSTABSTATS-NACTIVE,
         :WS-SYSTABSTATS-PCTROWCOMP,
         :WS-SYSTABSTATS-STATSTIME,
         :WS-SYSTABSTATS-IBMREQD,
         :WS-SYSTABSTATS-V-DBNAME,
         :WS-SYSTABSTATS-V-TSNAME,
         :WS-SYSTABSTATS-PARTITION,
         :WS-SYSTABSTATS-V-OWNER,
         :WS-SYSTABSTATS-V-NAME,
         :WS-SYSTABSTATS-CARDF
     )
 END-EXEC
```

Then, update the stats on SYSCOLUMNS in your target subsystem – go through each item in your array that you saved earlier for this (ARRAY #2) referenced in QUERY #3. You saved the corresponding COL-NAME with each item in the array, so use this in the WHERE clause each time:

```
 EXEC SQL UPDATE SYSIBM.SYSCOLUMNS
    SET
```

```
        COLCARD          = :WS-SYSCOLUMNS-COLCARD
       ,COLCARDF         = :WS-SYSCOLUMNS-COLCARDF
       ,HIGH2KEY         = :WS-SYSCOLUMNS-V-HIGH2KEY
       ,LOW2KEY          = :WS-SYSCOLUMNS-V-LOW2KEY
    WHERE TBNAME         = :WS-INPUT-TBNAME
      AND TBCREATOR      = :WS-INPUT-TBCREATOR
      AND NAME           = :WS-SYSCOLUMNS-V-NAME
END-EXEC
```

Next, update the stats on SYSINDEXES in your target subsystem – go through each item in your array that you saved earlier for this (ARRAY #3) referenced in QUERY #4. You saved the corresponding IX-NAME and IX-CREATOR with each item in the array, so use this in the where clause each time:

```
 EXEC SQL UPDATE SYSIBM.SYSINDEXES
    SET
        FIRSTKEYCARD     = :WS-SYSINDEXES-FIRSTKEYCARD
       ,FIRSTKEYCARDF    = :WS-SYSINDEXES-FIRSTKEYCARDF
       ,FULLKEYCARD      = :WS-SYSINDEXES-FULLKEYCARD
       ,FULLKEYCARDF     = :WS-SYSINDEXES-FULLKEYCARDF
       ,CLUSTERRATIO     = :WS-SYSINDEXES-CLUSTERRATIO
       ,CLUSTERRATIOF    = :WS-SYSINDEXES-CLUSTERRATIOF
       ,NLEAF            = :WS-SYSINDEXES-NLEAF
       ,NLEVELS          = :WS-SYSINDEXES-NLEVELS
    WHERE NAME           = :WS-SYSINDEXES-V-NAME
      AND CREATOR        = :WS-SYSINDEXES-V-CREATOR
END-EXEC
```

Next, delete the stats on SYSCOLDIST in your target subsystem for the TBNAME and TBOWNER (generally the same as TBCREATOR in other tables):

```
EXEC SQL
    DELETE
    FROM SYSIBM.SYSCOLDIST
    WHERE TBOWNER     = :WS-INPUT-TBCREATOR
      AND TBNAME      = :WS-INPUT-TBNAME
END-EXEC
```

Then, insert the new stats on SYSCOLDIST in your target subsystem – go through each item in your array that you saved earlier for this (ARRAY #4) referenced in QUERY #5:

```
EXEC SQL INSERT
    INTO SYSIBM.SYSCOLDIST
    (
         NAME
        ,COLGROUPCOLNO
        ,COLVALUE
```

```
            ,FREQUENCY
            ,FREQUENCYF
            ,NUMCOLUMNS
            ,STATSTIME
            ,TBNAME
            ,TBOWNER
            ,IBMREQD
            ,TYPE
            ,CARDF
        )
        VALUES
        (
             :WS-SYSCOLDIST-V-NAME
            ,:WS-SYSCOLDIST-V-COLGROUPCOLNO
            ,:WS-SYSCOLDIST-V-COLVALUE
            ,:WS-SYSCOLDIST-FREQUENCY
            ,:WS-SYSCOLDIST-FREQUENCYF
            ,:WS-SYSCOLDIST-NUMCOLUMNS
            ,:WS-SYSCOLDIST-STATSTIME
            ,:WS-SYSCOLDIST-V-TBNAME
            ,:WS-SYSCOLDIST-V-TBOWNER
            ,:WS-SYSCOLDIST-IBMREQD
            ,:WS-SYSCOLDIST-TYPE
            ,:WS-SYSCOLDIST-CARDF
        )
    END-EXEC
```

This is the end of the process for one table, moving stats from a source DB2 subsystem to a target DB2 subsystem. Simple changes are all that is required to make this loop through multiple tables, but make certain you issue a commit at the end of each UOW in the loop.

Ok, here's my disclaimers:

- Not every shop will allow PUBLIC SELECT authority on catalog tables (even though that is how DB2 is predefined) and/or on Plan Tables. As a Performance Analyst you can gain invaluable information from the DB2 catalog tables and Plan Tables, and you really should do your best to work the argument of at least getting SELECT authority on these tables for YOU.
- The SQL statements included in this article are specific to DB2 V8. As DB2 versions change, there is occasionally the need to change queries hitting catalog tables because their definitions sometimes are slightly altered.
- All the SQL calls should return good SQLCODES (+000), but if you encounter an EOT SQLCODE (+100) or other error SQLCODE you'll want to gracefully exit with a message saying what specific data in the catalog tables is not in sync

between production and test. This will most often occur if you have different index definitions between the subsystems for the table.

- There are some 3rd party tools out there that can do this prod-to-test runstat copy for you, or you can use IBM's Optimization Service Center (OSC) (free download, although out of service after V9) to do it also. We simply are a shop that likes to control and customize everything, so we wrote our own.
- If you are in a strong DB2 shop, don't forget to be nice to the DBAs. They can provide invaluable assistance to Performance Analysts.