OCTOBER 21, 2016 BY ROY BOXWELL

2016-10 Discovering hidden recovery problems in the SYSLGRNX

A query to read the SYSLGRNX

This month I want to have a look inside the SYSLGRNX – Not just for fun but actually to see if I, and you, have a serious RECOVER problem.

The Problem began with too many TP Image Copy steps

The problems all began when it was noticed by one of our customers that our DB2 Utility generation software (RTDX) was generating Image Copy jobs for "old" PBG partitions that were no longer being updated and they asked us why we were recommending an Image Copy.

RTDX and Image Copy

RTS COPY counters have not been updated then there has not been an update and we do not need a Copy”. This is normally extended with a special copy rule “But generate a Full Image Copy every xx days even if no updates”. This is done to guarantee that you can perform a recovery even if your non-DB2 migration/back-up software automatically deletes datasets older than xx days.

Everything green?

So you think you are covered? Not so fast! When the RTS were first introduced back in DB2 V7 they were not that stable and, sometimes, the RTS DB was actually stopped. This meant that the numbers were sometimes “not that reliable” – Now for REORG and RUNSTATS it does not really matter because, at some point, the counters start to count again and all is OK. But for IMAGE COPY it would be a disaster to “miss” an update! RTDX handles this with an extra “belt and braces” or, for the Americans amongst us, “belt and suspenders” check: “CHECK SYSLGRNX? Y/N” setting. If set to “Y” then RTDX generates an extra job that extracts the SYSLGRNX data and loads it into a DB2 table. This can then be queried by RTDX to see when objects were opened for update (Just like DB2 does in RECOVER processing to get the required log ranges.)

Something Old Something New

So now you know how the system works I can explain what happened at the customer site. They use Partition By Growth tablespaces (PBGs) a lot as “containers” to just insert records into. They had PBGs with hundreds of partitions and they wondered why RTDX generated Image Copy steps for partitions 1 – 136 when they only held “old non-changed data”.

Check the code

worked with the customer directly using log prints and tracing until we finally saw what was happening.

PBG doing the ripple!

When a new partition was added, all of the previous partitions header pages got an update “max no. of partitions”. Now this update counts as a system page and so is *not* in the RTS Counters as they only count data pages! RECOVER of course does not care! It would demand to look for the log ranges and what would happen if your logs have been deleted? Yep – Unrecoverable data!

APAR PI60104 fixes the problem

So you can apply the APAR and all is good? Not really. You still have to go and check if you have any potentially unrecoverable PBG datasets out there. How do you do that? Well – you ask me!

Reading SYSLGRNX is not so easy

The problem with SYSIBM.SYSLGRNX is that the data is nearly unusable when you want to join to the DB2 Catalog. E.g. Two byte character fields that actually contain SMALLINT values and, sometimes, with the high bit set…

Just creating a table and then a dumb INSERT INTO SELECT FROM dies because of the DSNDB01 Cursor rules. So first you’ll need to create your new version of SYSLGRNX:

```
  -- CREATE NEW VERSION OF SYSLGRNX WITH SMALLINTS --
```

```
   ,LGRSRBA          CHAR(10)          FOR BIT DATA NOT NULL
   ,LGRSPBA          CHAR(10)          FOR BIT DATA NOT NULL
   ,LGRPART          SMALLINT          NOT NULL
   ,LGRSLRSN         CHAR(10)          FOR BIT DATA NOT NULL
   ,LGRELRSN         CHAR(10)          FOR BIT DATA NOT NULL
   ,LGRMEMB          SMALLINT          NOT NULL
   ,LGRNEGPSID       CHAR(1)           NOT NULL)
;
COMMIT ;
```

Cross loader to the rescue!

After trying various tricks to get at the data I hit on the idea of abusing the cross loader to do it for me. Here is my JCL that you must tailor for your site to do the dirty deed:

```
SELECT COALESCE(
       CASE WHEN ASCII(SUBSTR(A.LGRDBID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRDBID, 2, 1)) > 32767 THEN
                (ASCII(SUBSTR(A.LGRDBID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRDBID, 2, 1))) - 32768
            ELSE ASCII(SUBSTR(A.LGRDBID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRDBID, 2, 1))
       END , 0)         AS LGRDBID
      ,COALESCE(
       CASE WHEN ASCII(SUBSTR(A.LGRPSID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRPSID, 2, 1)) > 32767 THEN
                (ASCII(SUBSTR(A.LGRPSID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRPSID, 2, 1))) - 32768
            ELSE ASCII(SUBSTR(A.LGRPSID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRPSID, 2, 1))
       END , 0)         AS LGRPSID
      ,TIMESTAMP(
           SUBSTR(A.LGRUCDT, 5 , 2 ) CONCAT '-' CONCAT
           SUBSTR(A.LGRUCDT, 1 , 2 ) CONCAT '-' CONCAT
           SUBSTR(A.LGRUCDT, 3 , 2 ) CONCAT '-' CONCAT
           SUBSTR(A.LGRUCTM, 1 , 2 ) CONCAT '.' CONCAT
           SUBSTR(A.LGRUCTM, 3 , 2 ) CONCAT '.' CONCAT
           SUBSTR(A.LGRUCTM, 5 , 2 ) CONCAT '.' CONCAT
           SUBSTR(A.LGRUCTM, 6 , 2 ) CONCAT '0000'
                 )      AS LGRUCTS
      ,A.LGRSRBA
      ,A.LGRSPBA
      ,A.LGRPART
      ,A.LGRSLRSN
      ,A.LGRELRSN
      ,ASCII(SUBSTR(A.LGRMEMB, 2, 1)) AS LGRMEMB
      ,CASE WHEN ASCII(SUBSTR(A.LGRPSID, 1, 1)) * 256
               + ASCII(SUBSTR(A.LGRPSID, 2, 1)) > 32767 THEN ‘Y’
            ELSE ‘N’
       END              AS LGRNEGPSID
FROM SYSIBM.SYSLGRNX A
ENDEXEC
LOAD DATA INCURSOR(C1) RESUME NO SHRLEVEL NONE REPLACE
  INTO TABLE BOXWELL.SYSLGRNX2
/*
```

The use of ASCII is to “translate” the character data into smallint and also to detect if we have negative numbers. This happens if you run out of DBIDs and also if you use CLONE objects. The requirement for the two COALESCEs is down to the cross loader syntax check on the CASE statements by the way…
This could load a lot of data of course!

Do not forget the INDEX

```
-- CREATE INDEX ON THE COLUMNS WE NEED --
CREATE INDEX BOXWELL.SYSLGRNX_IX
          ON BOXWELL.SYSLGRNX2
       (LGRDBID,LGRPSID,LGRPART,LGRMEMB)
        USING STOGROUP SYSDEFLT
             PRIQTY -1
             SECQTY -1
        CLUSTER
        CLOSE YES
;
COMMIT ;
```

And the RUNSTATS

```
RUNSTATS  TABLESPACE <yourdatabase>.<yourtablespace>
          TABLE (ALL)
          INDEX(ALL)
          FREQVAL NUMCOLS 1 COUNT 100 BOTH
          FREQVAL NUMCOLS 2 COUNT 100 BOTH
          FREQVAL NUMCOLS 3 COUNT 100 BOTH
          SHRLEVEL CHANGE REPORT NO UPDATE ALL HISTORY NONE
```

Now we can SQL So finally we have the SYSLGRNX data in a format that we can actually use! Here is my SQL that lists out any PBG spaces that have, according to the RTS, no updates but the last Image Copy timestamp is earlier than the SYSLGRNX record:

```
-- FINALLY THE QUERY TO SEE IF A PROBLEM EXISTS OR NOT --
--
-- SELECT ALL PBGS IN SYSTABLESPACE THAT HAVE DATA IN THE
-- SYSLGRNX WHICH ALSO HAVE AN ENTRY IN THE RTS WHERE THE
-- COUNTERS ARE NULL OR ZERO BUT THE COPYLASTTIME IS EARLIER
```

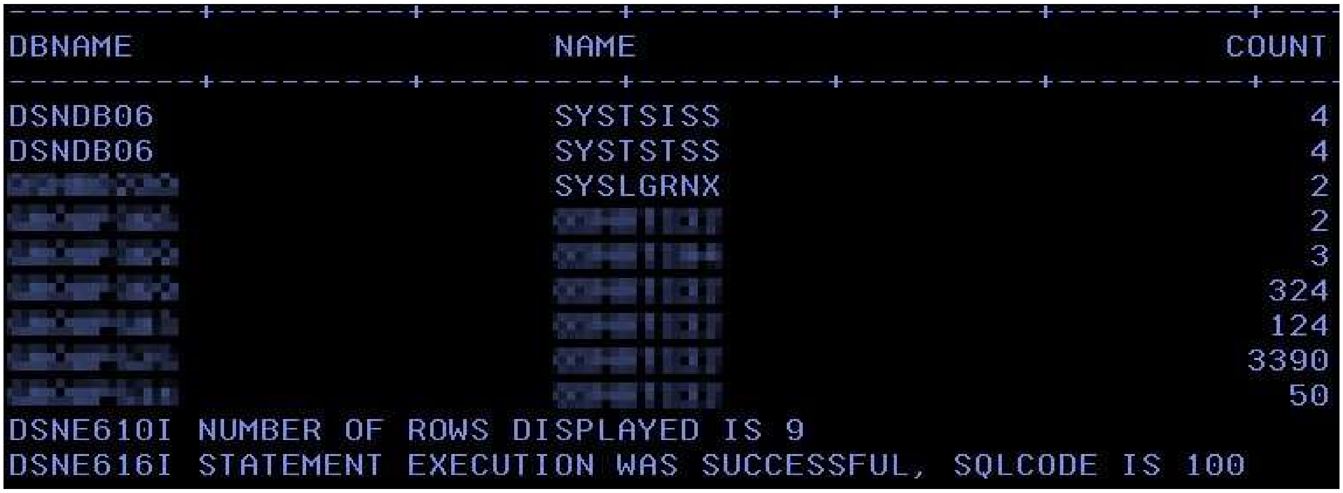
```
SELECT A.DBNAME, A.NAME, B.LGRPART, A.DBID, A.OBID, A.PSID
      ,B.LGRUCTS, B.LGRSRBA, B.LGRSPBA, B.LGRSLRSN
      ,B.LGRELRSN, HEX(B.LGRMEMB) AS MEMBER
      ,C.COPYLASTTIME
      ,MAX(D.START_RBA) AS HIGHEST_IC_RBA
      ,MAX(D.TIMESTAMP) AS HIGHEST_IC_TS
FROM SYSIBM.SYSTABLESPACE A
INNER JOIN
    BOXWELL.SYSLGRNX2    B
 ON  A.DBID     = B.LGRDBID
 AND A.PSID     = B.LGRPSID
 AND A.INSTANCE = B.LGRINST
INNER JOIN
     SYSIBM.SYSTABLESPACESTATS C
 ON  C.DBID     = A.DBID
 AND C.PSID     = A.PSID
 AND C.PARTITION= B.LGRPART
 AND C.INSTANCE = A.INSTANCE
LEFT OUTER JOIN
     SYSIBM.SYSCOPY       D
  ON D.DBNAME    = A.DBNAME
 AND D.TSNAME    = A.NAME
 AND D.DSNUM   IN ( 0 , B.LGRPART)
WHERE A.TYPE   = 'G'
 AND (C.COPYUPDATEDPAGES = 0
  OR C.COPYUPDATEDPAGES IS NULL)
 AND (C.COPYCHANGES = 0
  OR C.COPYCHANGES IS NULL)
 AND (C.COPYLASTTIME < B.LGRUCTS
  OR C.COPYLASTTIME IS NULL)
 AND (D.ICTYPE = 'F'
  OR D.ICTYPE IS NULL)
GROUP BY A.DBNAME, A.NAME, B.LGRPART, A.DBID, A.OBID, A.PSID
     ,B.LGRUCTS, B.LGRSRBA, B.LGRSPBA, B.LGRSLRSN
     ,B.LGRELRSN, HEX(B.LGRMEMB)
```

And the winner is?

I hope you have zero records returned? Even if you do have some records, it is relatively easy to quickly get them image copied so that you can sleep at night. If zero, you still have a great copy of the SYSLGRNX that you can use for other purposes!

A customer replies

Here’s some real data back from a customer:



```
---------+---------+---------+---------+---------+---------+---
DBNAME                   NAME                         COUNT
---------+---------+---------+---------+---------+---------+---
DSNDB06                  SYSTSISS                         4
DSNDB06                  SYSTSTSS                         4
[redacted]               SYSLGRNX                         2
[redacted]               [redacted]                       2
[redacted]               [redacted]                       3
[redacted]               [redacted]                     324
[redacted]               [redacted]                     124
[redacted]               [redacted]                    3390
[redacted]               [redacted]                      50
DSNE610I NUMBER OF ROWS DISPLAYED IS 9
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

The COUNT is the number of SYSLGRNX entries they have – So they have nine DB & TS groups and from a further analysis (removing the data sharing member duplicates) they have 1926 distinct DB, TS and Partitions!

They scheduled image copy processing for all of these and were then done.

I hope you liked this month’s topic.

Roy Boxwell

DB2 SPACE ISSUE, NEWS FROM THE LABS

FULL IMAGE COPY, RECOVER, RECOVERY PROBLEM, RTS, SPACE MONITORING, SYSLGRNX