Data Purge Algorithm: Deleting Unwanted Data from a DB2 LUW Database

Big data introduces data storage and system performance challenges. Keeping your growing tables small and efficient improves system performance as the smaller tables and indexes are accessed faster; all other things being equal, a small database performs better than a large one. Additionally, backups for smaller tables will complete faster and require less space. While traditional data purge techniques work well for smaller databases, they fail as the database size scales up into a few terabytes. This article will discuss an algorithm to efficiently delete terabytes of data from your DB2 database.

Background

Our DB2 Environment had a performance lab database that was 1.8TB in size with 2.5M application users. However, our client wanted performance test results from a database sized for 1.2M users. To accomplish this we decided to purge 1.3M users out of the existing 2.5M user database.

Overview

The goal was to purge data for 1.3M application users, creating a target database with data for 1.2M users at an estimated size of 0.7TB. We tried several techniques to delete such a large amount of data from the database. In the end, we chose the best technique and developed an algorithm for it. Instead of deleting unwanted data we export and reload valid data into the parent table (i.e root node of the table hierarchy). Once done, we use the **SET INTEGRITY** command to delete the corresponding unwanted data from dependent child tables. Note that the tables will remain offline as you perform the purge operation.

Algorithm

Before you proceed with this purge algorithm consider tuning the following DB2 configuration parameters. Remember that we are tuning these parameters to achieve optimal performance for purge algorithm. You may choose to reset them to original values after data purge to achieve optimal application performance.

Profile Registry Parameters

Parameter Setting	Effect
DB2_SKIPINSERTED = ON	DB2 skips uncommitted inserts on table scans which reduces locking.
DB2_USE_ALTERNATE_PAGE_CLEANING = YES	DB2 uses an alternate method for page cleaning instead of the default method.
DB2_EVALUNCOMMITTED = ON	Defers the acquisition of locks until the row is known to satisfy the predicates of the query.
DB2_SKIPDELETED = ON	DB2 skips deleted rows and index keys on table scans and reduce locking.
*DB2_PARALLEL_IO = **	DB2 prefetches extents in parallel.

Database Manager Configuration Parameters

Parameter Setting	Effect
DFT_MON_BUFPOOL = ON DFT_MON_LOCK = ON DFT_MON_SORT = ON DFT_MON_STMT = ON DFT_MON_TABLE = ON DFT_MON_UOW = ON	Instructs the database manager to collect monitor data for performance analysis.
ASLHEAPSZ = 32	Configures communication buffer between the local application and its associated agent to improve performance.

Database Configuration Parameters

Parameter Setting	Effect
STMT_CONC = LITERALS	Modifies dynamic SQLs to allow increased sharing of package cache.
DFT_DEGREE = ANY	Allows optimizer to determine the degree of intrapartition parallelism.
PCKCACHESZ = AUTOMATIC	Dynamically size the memory area of package cache for caching of sections for static and dynamic SQLs.
CATALOGCACHE_SZ = 2000	Tunes the catalog cache to reduce the overhead of accessing the system catalogs to obtain information that has previously been retrieved.
LOGBUFSZ = 2048	Buffering the log records will result in more efficient logging file I/O.
UTIL_HEAP_SZ = 524288	Tunes the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE, and LOAD utilities.
LOGFILSIZ = 16384	Tunes this parameter for a large number of update, delete and insert transactions to reduce log I/O and improve performance.
CUR_COMMIT = ON	DB2 returns only the currently committed value of data at the time when query is submitted.
SECTION_ACTUALS = BASE	Enables collection of runtime statistics that are measured during section execution.

Once configured, take row counts for all tables and note the initial database size to define a baseline. Replace **<parameter>** with the corresponding values in each of the below queries.

```
db2 -x "select 'runstats on table '||trim(tabschema)||'.'||trim(tabname)||';' from syscat.tables
where tabschema = '<schemaname>' and type='T'" >runstats.sql

db2 –tvf runstats.sql -z runstats.out

db2 "select substr(tabname,1,30) tabname,card from syscat.tables where tabschema =
'<schemaname>' and type='T'" >initialCount.out

db2 "call get_dbsize_info(?,?,?,-1)" >initialDBSize.out

db2look -d <dbname> -a -e -l -x -c > InitialDB2look.ddl
```

SET INTEGRITY will delete corresponding data in our child tables. If our child tables are large we run the risk of locking and transaction log issues. To avoid them, find tables with more than 300M*records and break their foreign key relationship with parent tables. This will ensure that large child tables are not placed in integrity pending state when we reload valid data into parent tables or in the root node.

*300M is a number we chose by trial and error method. Choose a number which best suits your database.

```
db2 "select substr(tabname,1,30) tabname,card from syscat.tables where tabschema =
'<schemaname>' and type='T' and card > 300000000" >isolatedTables.out

db2 -x "select 'alter table '||trim(a.tabschema)||'.'||trim(a.tabname)||' drop constraint ' ||
a.CONSTNAME||';' from syscat.referencesa,syscat.tables b where a.tabschema=b.tabschema
and a.tabschema='<SCHEMANAME>' and b.type='T' and a.tabname=b.tabname and b.card>
300000000" >alterTable.sql

db2 –tvf alterTable.sql -z alterTable.out
```

Each of the isolated tables is a pseudo-root node and we will recursively use the same purge algorithm, treating one isolated table as root node in each iteration of the recursive execution. After we have successfully purged data from original and pseudo-root table hierarchies we reconnect the isolated child tables to their parent table to restore the original database structure.

Instead of deleting unwanted data, export valid data from the root node. Ensure exports are on a separate filesystem from containers of the subject table's tablespace for better I/O performance. Before reloading valid data drop all indexes on the subject table and recreate them after the data is reloaded successfully (this technique will improve performance). Note that you will need to drop the primary key constraint before dropping the index being used to enforce the primary key.

```
db2 export to <export-path><root-tabname>.csv of del select * from <root-tabname> where <...>

db2look -d <dbname> -t <root-tabname-or-tablist> -e -o <output file name>

db2 -x "select 'alter table '||trim(st.tabschema)||'.'||trim(st.tabname)||' drop constraint
'||st.constname||';'  from SYSCAT.KEYCOLUSE sk inner join SYSCAT.TABCONST st on
sk.TABNAME = st.TABNAME and sk.TABSCHEMA=st.TABSCHEMA and
st.tabschema='<SCHEMANAME>' where st.type in ('P','U') and st.CONSTNAME =sk.CONSTNAME
and exists (select 1 from syscat.tables a where a.tabname=st.TABNAME and
st.TABschema=a.tabschema and a.tabname in (<root-tabname>))" >dropPK.sql

db2 –tvf dropPK.sql -z dropPK.out

db2 "select 'drop index '||trim(INDNAME)||';' from syscat.indexes where TABNAME in(<root-
tabname>) and tabschema='<schemaname>'" >dropIndexes.sql

db2 –tvf dropIndexes.sql -z dropIndexes.out
```

Once indexes are dropped you can reload the valid data using the **REPLACE** option of **LOAD** command. Using **REPLACE** with **LOAD** command will ensure that the table is truncated and then the valid data is loaded into it. Consequently, truncating the table will delete the unwanted data. Furthermore, setting integrity of the table after reloading the valid data will put all of its children in integrity pending state.

```
db2 "load from <export file>.csv of del modified by fastparse replace into <tabname> DATA
BUFFER <buffer> SORT BUFFER <buffer> CPU_PARALLELISM <value> DISK_PARALLELISM
<value>"

db2 "set integrity for <tabschema>.<tabname> immediate checked”
```

Re-create the dropped indexes and primary keys. (Refer the DB2LOOK output we took before dropping indexes). Create exception tables with two additional columns of type TIMESTAMP and CLOB for all tables in integrity pending state.

```
db2 -x "select 'CREATE TABLE '||trim(TABNAME)||'_exp '|| 'like ' || tabname ||';' from
SYSCAT.TABLES where STATUS='C' and type='T'  and TABSCHEMA='<schemaname>'"
>createExceptionTab.sql

db2 -x "select 'drop table '||trim(TABNAME)||'_exp;' from SYSCAT.TABLES where STATUS='C'
and type='T'  and TABSCHEMA='<schemaname>'"  >dropExceptionTab.sql

db2 -x "select 'Alter table '||trim(TABNAME)||'_exp '|| ' add column c1 TIMESTAMP add
column c2 CLOB; ' from SYSCAT.TABLES where STATUS='C' and type='T' and TABSCHEMA
='<schemaname>'" >>createExceptionTab.sql

db2 –tvf createExceptionTab.sql -z createExceptionTab.out
```

Set **integrity** for all tables in integrity pending state using the corresponding exception tables. This will move invalid data to the exception tables and purge improper data from the base tables. This shell script can recursively check and remove tables from integrity pending state.

Execute:**./setIntegrity.sh MYDBNAME MYSCHEMA**

```
#!/bin/ksh

before="$(date +%s)"

DBName=$1
DBSchema=$2
echo "Checking and removing tables from set integrity pending state" | tee -a SetIntegrity.log
db2 activate db $DBName>>output.out
db2 connect to $DBName>>output.out
db2 set schema $DBSchema>>output.out
db2 -x "select 'SET INTEGRITY FOR '|| TABSCHEMA ||'.'||TABNAME || ' IMMEDIATE CHECKED
FOR EXCEPTION IN ' ||TABNAME || ' USE ' ||TABNAME||'_exp ;' from SYSCAT.TABLES where
STATUS='C' and type='T' and TABSCHEMA='$DBSchema' order by card " >chkset_integrity.sql
tabcnt=$(wc -l <chkset_integrity.sql)
while [[ ${tabcnt} -gt 0 ]];
do
echo "************************************************" | tee -a SetIntegrity.log
        echo "Number of tables in set integrity pending state : $tabcnt" | tee -a SetIntegrity.log
        echo "Setting integrity of table in set integrity pending state" | tee -a SetIntegrity.log
echo "************************************************" | tee -a SetIntegrity.log
db2 connect to $DBName>>output.out #Connect to DB
db2 set schema $DBSchema>>output.out
db2 -tvfchkset_integrity.sql>>output.out
db2 "commit" >>output.out
db2 -x "select 'SET INTEGRITY FOR '|| TABSCHEMA ||'.'||TABNAME || ' IMMEDIATE CHECKED
FOR EXCEPTION IN ' ||TABNAME || ' USE ' ||TABNAME||'_exp ;' from SYSCAT.TABLES where
STATUS='C' and type='T' and TABSCHEMA='$DBSchema' order by card " >chkset_integrity.sql
tabcnt=$(wc -l <chkset_integrity.sql)
done
echo "No table in set integrity pending state" | tee -a SetIntegrity.log
after="$(date +%s)"
elapsed_seconds="$(expr $after - $before)"
timediff=`echo - | awk -v "S=$elapsed_seconds"
'{printf"%dh:%dm:%ds",S/(60*60),S%(60*60)/60,S%60}'`
echo "Time Taken to set Integrity: $timediff" | tee -a SetIntegrity.log
```

Recursively use the same process of exporting and reloading the valid data for all pseudo-nodes. Once data is purged from all tables recreate the dropped foreign key constraints to link back the pseudo-root table hierarchies to the root table to restore the original database structure. Since a significant amount of data was deleted from the tables, perform reorg operation and if possible, use temporary tablespace for reorg operation.

```
db2 -tvf dropExceptionTab.sql –z dropExceptionTab.out

db2 "select 'reorg table '||trim(tabname)||';' from syscat.tables where type='T' and
tabschema='<schemaname>'">reorg.sql
```

Perform runstats on all tables to update statistics. Then note the latest table statistics, database size, and DB2look output comparing them with the initial outputs.

```
db2 -x "select 'runstats on table '||trim(tabschema)||'.'||trim(tabname)||';' from syscat.tables
where tabschema = '<schemaname>' and type='T'" >runstats.sql

db2 –tvf runstats.sql -z runstats.out

db2 "select substr(tabname,1,30) tabname,card from syscat.tables where tabschema =
'<schemaname>' and type='T'" >finalCount.out

db2 "call get_dbsize_info(?,?,?,-1)" >finalDBSize.out

db2look –d <dbname> -a -e -l -x -c > finalDB2look.ddl
```

With the reorg operation, we have reduced the number of used pages in the tablespace, but the tablespace's high watermark may be higher than the actual number of used pages, so lower this high watermark.

```
db2 -x "select 'Alter tablespace '||trim(TBSPACE)||' lower high water mark ;' from
syscat.tablespaces where TBSPACETYPE='D'">highWatermark.sql

db2 –tvf highWatermark.sql -z highWatermark.out
```

After lowering the high watermark, if tablespace size is greater than the high watermark and you take a backup and restore the database, it will still consume more disk than that required by the used pages. To minimize this disk space requirement resize the tablespace and reduce its size.

Monitoring the Purge Job

With this purge algorithm we are deleting data from the child tables using the **SET INTEGRITY** command. In order to determine the data to be deleted, it only refers to the target table and its corresponding parent tables. This means that at any point in time, only the bufferpools of the child table and its parents are in use. Monitor with DB2TOP and tune the size of active bufferpools at each step of the purge process to achieve optimal performance.

Things to Remember

- You may consider modifying the EXPORT and LOAD commands to keep LOBS in a separate filesystem.
- Watch carefully for circular dependencies among the tables in your database. You need to break that dependency before using the **setintegrity.sh** script else the script will enter into an infinite loop. Use the query below to find circular dependencies among the tables.

```
db2 "select
substr(a.tabname,1,20)tabname,substr(a.reftabname,1,20)reftabname,substr(a.CONSTNAME
,1,20)CONSTNAME from syscat.references a where exists (select 1 from syscat.references b
where a.reftabname=b.tabname and a.tabname=b.reftabname)"
```

Comparing with Traditional Purge Techniques

Metric	Cascaded Delete	Stored Procedure with Commit Interval	Purge Algorithm
Locking Issues	High	Moderate	Low
Issues with logs	High	Moderate	Low
User Control	Low	Moderate	High
Time Required	High	High	Low
Table Unavailability	Moderate	Moderate	High

As this table shows, the data purge algorithm performs well as compared to the traditional data purge techniques with minimal or no locking and transactional logs issues. It requires minimum time and gives better user control over the purge process. The total time is composed more of the SET INTEGRITY phase than the LOAD phase for this method, so don't expect that to be short. Also, it is important to note that the tables remain online if you purge data with the cascaded delete or stored procedure methodology, whereas the purge algorithm requires that tables be offline.

Saurabh Agrawal
saurabh.ska@gmail.com