Data purge algorithm: Efficiently delete terabytes of data from DB2 for Linux, UNIX, and Windows

Deleting unwanted data

Saurabh Agrawal (saurabh.ska@gmail.com)
Database Administrator
ACI Payment Systems

29 January 2015

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

Background

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

Overview

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 (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 we 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. These are the parameter settings that gave best performance on our environment. You can consider tuning them to best suit your environment.

Configuration

Table 1. Tuning profile registry parameters for optimal performance

Parameter setting	Effect
`DB2_SKIPINSERTED = ON`	Skips uncommitted inserts on table scans which reduces locking
`DB2_USE_ALTERNATE_PAGE_CLEANING = YES`	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`	Skips deleted rows and index keys on table scans and reduce locking
`DB2_PARALLEL_IO = *`	Pre-fetches extents in parallel

Table 2. Tuning DBM CFG parameters for optimal performance

Parameter setting	Effect
`DFT_MON_BUFPOOL = ON`	Instructs the database manager to collect bufferpool monitor data for performance analysis
`DFT_MON_LOCK = ON`	Instructs the database manager to collect locking information for performance analysis
`DFT_MON_SORT = ON`	Instructs the database manager to collect sorting information for performance analysis
`DFT_MON_STMT = ON`	Instructs the database manager to collect statement level information for performance analysis
`DFT_MON_TABLE = ON`	Instructs the database manager to collect table level information for performance analysis
`DFT_MON_UOW = ON`	Instructs the database manager to collect UOW information for performance analysis
`ASLHEAPSZ = 32`	Configures communication buffer between the local application and its associated agent to improve performance

Table 3. Tuning DB CFG parameters for optimal performance

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 intra-partition parallelism
`PCKCACHESZ = AUTOMATIC`	Dynamically size the memory area of package cache for caching of sections for static and dynamic SQLs
`DFT_MON_STMT = ON`	Instructs the database manager to collect statement level information for performance analysis
`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

`SECTION_ACTUALS = BASE`	Enables collection of runtime statistics that are measured during section execution
`CUR_COMMIT = ON`	Returns only the currently committed value of data at the time when query is submitted

Row counts

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

Listing 1. Database metrics (baseline)

```
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
```

Large child tables

`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 300 million 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. 300 million was a number we chose by trial and error, choose a number which best suits your database.

Listing 2. Isolating the large child tables

```
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.references a,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 isolated table 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.

Exporting data

Instead of deleting unwanted data, export valid data from the root node. Ensure that exports are on a separate file system from containers of the subject table's tablespace for better I/O performance.

Before reloading valid data, drop all indices 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.

Listing 3. Exporting valid data and dropping indices

```
db2 export to EXPORT-PATH ROOT-TABNAME.csv of del select * from ROOT-TABNAME where WHERE-CLAUSE-CONDITIONS

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
```

Loading data

Once indices are dropped, you can reload the valid data using the `REPLACE` option of the `LOAD` command. Using `REPLACE` with `LOAD` will ensure that the table is truncated, 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.

Listing 4. Reloading the valid data

```
db2 "load from export file.csv of del modified by fastparse replace into TABNAME data buffer VALUE sort
 buffer VALUE cpu_parallelism VALUE disk_parallelism VALUE"

db2 "set integrity for TABSCHEMA.TABNAME immediate checked”
```

Creating exception tables

Create exception tables with two additional columns of type `TIMESTAMP` and `CLOB` for all tables in integrity pending state. Also, recreate the dropped indices and primary keys by referring to the DB2LOOK output we took before dropping indices.

Listing 5. Creating the exception tables

```
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
```

Recursive set integrity for tables

Set integrity for all tables in integrity pending state using the corresponding exception tables. This will purge invalid data from the base tables by moving it to the exception tables. Save the below shell commands as a shell script (setIntegrity.sh) and use it to recursively remove tables from integrity pending state.

Listing 6. Recursively setting integrity of child tables

```
#!/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
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
```

Execute the command `./setIntegrity.sh DBNAME SCHEMANAME`.

Re-creating Foreign Keys

Recursively use the same process of exporting and reloading 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 and restore the original database structure. Also, drop the exception tables as we no longer need them.

Listing 7. Dropping exception tables

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

Final metrics

Since a significant amount of data was deleted from the tables, perform reorg operation and if possible, use temporary tablespace for reorg operation. Perform runstats on all tables to update

statistics. Then note the latest row counts, database size, and DB2look output for comparing them with the initial outputs (baseline).

Listing 8. Finding the final table and database metrics

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

db2 –tvf reorg.sql -z reorg.out

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 reduced the number of used pages in the tablespace, but the tablespace's high watermark may still be higher than the actual number of used pages, so lower this high watermark.

Listing 9. Lowering high watermark for DMS tablespaces

```
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, consider resizing the tablespace and reduce its size.

Monitoring the purge process

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 target 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 them before using the setintegrity.sh script or 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)"
```

Conclusion

As shown below, 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.

Figure 1. Comparing 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

Resources

Learn

- "Database Shrinking: Purge a Large Amount of Data from Database" explains the initial approaches that failed to purge such humongous data.
- The DB2 Night Show Episode 131: DB2 Data Purge Algorithm provides a recipe for success with command examples.
- Visit the developerWorks Information Management zone to find more resources for DB2 developers and administrators.
- Follow developerWorks on Twitter.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, or use a product in a cloud environment.

Discuss

- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Saurabh Agrawal

Saurabh Agrawal is a database administrator with IBM Certification in DB2 LUW v10.1. He featured in season 5 of the DB2Night Show and enjoys sharing knowledge through his blog, Memoir.