Performance Tuning – Delete Processing

Performance considerations for Optim delete processing

Introduction

There are a number of factors that affect the overall runtime of a delete process. To understand the effects of these factors, you should first have an understanding of how Optim performs delete processing.

First we'll take a look at Optim delete theory, and build a basic understanding of the processing involved. Next we'll discuss some tuning tips to help you get the best performance from your delete run.

Basic delete processing theory

Optim delete processes are driven by archive files. When you run a delete process, you always associate it with an archive file. Only the data contained within the archive file will be deleted from the database, and only for the tables that were marked DAA (delete after archive) when the archive process was run and the archive file created. Delete processing does not support "selective" processing. All rows within an archive file, for tables marked DAA, will be deleted from the database.

At the start of a delete process, Optim evaluates the relationship paths between the tables residing in the archive file as they currently exist in the database. It does this to determine the proper order to process the list of tables involved. Conditions where tables have multiple children and/or multiple parents are taken into account when evaluating the process model path. Tables must be processed child-to-parent to successfully delete data across the complete process model.

When Optim chooses a table to process, it deletes rows in the database based on the primary key values contained in the row images within the archive file. Each row is deleted individually by its primary key value. When the process begins for a given table, a sorted memory list is built that contains all of the primary key values found in the archive file for the table and the RBA (relative byte address) pointers to the corresponding row images in the archive file. Once the list is complete, Optim runs through the list and processes each key value.

For each key value in the list, the database table is queried to retrieve the current row. The RBA recorded in the list for the key value is used to directly access the row image in the archive file.

The current row is compared to the archive file row image to ensure that the row has not changed since it was archived. If the current row matches its image in the archive file, the delete is issued for its primary key value. Each delete execution is counted. When the delete counter exceeds the commit frequency specified for the process, a commit is executed.

The delete status of each row is recorded in the control file. Success and failure counts are included in the process report. If the process report shows delete failures, the control file can be browsed to determine the nature of the failure of any given row.

Tuning tips

Now that we've discussed basic Optim delete processing, let's take a look at some conditions that cause variations in the process.

Primary Keys

Of all the conditions that can affect a delete process, the status of the table's primary key may be the most important. For the purpose of this discussion, there are 2 types of primary keys:

- Unique primary key -- A unique primary key is defined to the database, or it is a PST primary key with a supporting unique index.
- Non-unique primary key -- A non-unique primary key is a PST primary key that does not have a supporting unique index. It may be defined over part of a unique index, over a non-unique index, or it may not have an index at all.

When the table you're processing has a unique primary key, many parts of the delete process become more efficient and other performance related options become available. Since Optim deletes rows by primary key, the type of primary key on the table greatly affects how efficiently the database can find and remove rows. The database will process a delete request against a unique index faster than a non-unique index, and it will process a non-unique index faster than no index at all.

Optim also changes its processing based on the existence and type of index under the table's primary key. When a unique primary key exists, Optim doesn't need to build the memory list of primary keys and RBA pointers. In this case, it can read blocks of data from the archive file, identify the primary key values in each row image, and issue the delete requests. Bypassing the building of the key list saves time and memory.

When a non-unique key exists, Optim must perform a significant amount of processing to ensure that the correct rows are deleted from the database because any given primary key value could affect multiple rows in the table. Optim passes through the row images in the archive file before beginning the actual delete process and builds the key list in memory. The list will contain only one copy of each key value, but any given key value could have a list of RBAs that point to the various row images in the archive. Once the list is built, each key is used to query the table to retrieve all rows that match the key. The list of row images is then retrieved from the archive file and is used to search through the cursor of table rows to find exact row matches before deleting. Optim uses the WHERE CURRENT OF CURSOR option of the delete statement when a match

is found to delete the specific row that represents the image from the archive. This process continues as Optim passes through the key list until all primary key values are processed for the table.

When the non-unique primary key has no index at all, the table queries to retrieve the row sets for the key values may cost a lot of time. This is because the database has no choice but to scan the entire table on each query. Depending on the number of keys in the list (and thus the number of queries that will hit the table) and the number of rows in the table, Optim may decide to simply retrieve the entire table and manually search for the matching rows.

Optim delete options

Optim provides several options that modify the delete process and can be used to help trim down the overall runtime. Although these options may speed up the execution of the process, they come with their own costs.

The "compare row contents" option provides a level of safety by ensuring that each and every row being deleted exactly matches the current corresponding row in the database before removing it. A significant amount of overhead comes along with this option, and deselecting it usually reduces the overall runtime of the delete process. When this option is deselected, Optim doesn't need to retrieve rows from the database and compare them to the images on the archive. When Optim compares row contents, it locks the database row when it retrieves it from the live table to prevent another process from modifying it before the comparison and delete can take place. When deselecting "compare row contents", Optim doesn't query the table with locking. The only locks that occur are those that the database naturally applies when deletes are issued.

Pros:

- Ensures that rows have not changed since they were archived. Using this feature ensures that you have a correct copy of your data before deleting.

Cons:

- Uses a significant amount of resources (CPU, I/O, memory, database services, time).

Requirements:

- There are no requirements for selecting this option.
- The table must have a unique primary key to deselect this option.

Notes:

- This is a process level option. Although you can only set this option at the process level, if you choose to deselect it, Optim will always activate compare for any tables that don't have unique primary keys.

"Multiple key" delete processing is a feature of Optim where multiple primary key values are stacked on a single delete call to the database. UDB and Oracle databases have built-in mechanisms for this type of processing, and Optim will make use of those database features when using multiple key delete. For all other supported databases, Optim accomplishes multiple key delete processing by constructing a multiple key WHERE clause on the delete statement. When conditions lend themselves to using multiple key delete, Optim will automatically use the built-in database features of UDB and Oracle when processing those DB platforms. For the remaining DB platforms where the database doesn't have a built-in "bulk" delete feature, you must instruct Optim to employ its multiple key delete feature by increasing the "key lookup

limit" setting on the desired table. This setting is found on the "table strategy" dialog within the delete request editor.

Pros:

- Database processes multiple primary keys (and thus deletes multiple rows) for each delete statement issued. This is generally a more efficient method of deleting multiple rows.

Cons:

- Optim can't report on individual row statuses when using multiple key delete. The response from the database on a multiple key delete is either success or failure. Success simply indicates that the delete statement was processed. Failure simply means that the statement couldn't be processed. The database provides no detailed status information on the delete of the individual keys, and therefore Optim cannot accurately report row level failures.

Requirements:

- The table must have a unique primary key.
- You must deselect "compare row contents" on the process.
- You cannot have any row level delete actions on the table ("Before Delete of Row" or "After Delete of Row")

Notes:

- This is a table level option. You can control the use of multiple key delete processing at the table level when designing the delete request. Be aware that Optim will override your choice to use multiple key delete processing on a table at runtime if the conditions required no longer exist (for example: The primary key of the table changed, or "compare row contents" was selected at runtime to override the setting in the delete request).

"Multiple database connections" may provide some improvement in overall runtime by processing multiple tables concurrently. Be careful when experimenting with this setting. It may be tempting to set this option to "max", expecting to get the maximum benefit. Depending on your system environment, using the "max" setting may actually degrade the process. If you're unfamiliar with this option, you can safely experiment with it by specifying a low number in the range of 2 to 8. You should refer to additional documentation from Princeton Softech on using multiple database connections for details on how and when Optim uses this feature.

Pros:

- Allows processing of multiple tables concurrently.

Cons:

- An excessive number of database connections can overload the database server.

Requirements:

- Table must have a unique primary key.
- You must deselect "compare row contents" on the process.

Notes:

- This is a table level option. When you specify multiple database connection on a process, Optim will decide on a table by table basis whether to use them, and how many can be used.

The "lock tables" option causes Optim to obtain an exclusive lock on the table currently being processed. An exclusive lock prevents other processes from accessing the table while the lock is in effect. Selecting this option could improve performance of the delete because it prevents lower level row and page locking which can be expensive within the database server.
Pros:

- Prevents low level database locking and multiple commits while processing a table.
- Ensures that live data can't be modified by other processes while Optim is deleting from the table.

Cons:

- Prevents concurrent access to the table by other processes.
- Optim performs a commit after all deletes against the table are completed. For tables which a large number of rows in the archive file (and/or rows that are very long), the recovery logs in the database could become stressed.

Requirements:

- This option has no other requirements. It can be selected on any delete request.

Notes:

- This is a process level option. It affects all tables in the delete process.

Tuning checklist summary

Refer to the following checklist when tuning your Optim delete process.

I) Primary keys and indexes:

1. Whenever possible, define a unique primary key on each table to be processed. Define either a DBMS primary key or define a PST primary key over a unique index.
2. Define your primary key as small as possible. If you must define a PST primary key and the table has multiple unique indexes, choose the smallest one for your primary key. If your process does build the primary key list in memory, using the smallest available key size will reduce memory usage.
3. Any index is better than no index. When defining a PST primary key, if only a non-unique index is available, use it! Any index under your primary key will help the DB server search for keys faster than having no index at all.

II) Optim delete performance options:

1. Bypass "Compare Row Contents". If you are confident that the data you have archived has not changed in the database, consider deselecting the "Compare Row Contents" option on the delete request or override it at process time (batch operations).
2. Bypass "Include LOB columns in row comparison". If one of more of the tables being processed contain LOB type columns, and you are confident that the data you have archived has not changed in the database, consider deselecting the "Include LOB columns in row comparison" option on the delete request. Depending on the size of your LOB type columns, including them in the row comparison process can be resource intensive.

3. On large tables (many rows in the archive file to delete), consider using "Force Key Lookup" access method and setting the key lookup limit to a value greater than 1. You can only adjust the key limit value if:
   - The table has a unique primary key defined.
   - You are not using the "compare row contents" option.
   - The delete process has no row level delete actions defined.

   Note that adjusting the key limit value is only available if the table resides in a Sybase, Informix, or SQL Server database. Optim will always use multiple key style delete operations for UDB and Oracle databases when the other required conditions are met.
4. Use multiple database connections. When your delete process model contains multiple leaf tables, Optim can process them concurrently. If you've allowed parallel database connections (Optim product options), then you can specify multiple connections for your delete process. Select a value greater than 1 in the "Database Connections" drop-down control on "General" tab of the delete request.
5. Use "Lock Tables" options. If you're running your process at a time when concurrent access to the tables involved isn't an issue, you may choose to lock the tables during the delete process. Check this option on the "General" tab of the delete request. Be aware that, because the commit occurs at the end of processing each table, processing a large amount of data (large row lengths and/or many rows per table) can cause issues with the redo or rollback areas in your database server.

III) Operational and environmental:

1. Run your delete process when the database is quite. If at all possible run a delete process when activity against the tables involved is low. Delete processing generates locks. The delete process may be delayed if other processes are holding locks on needed rows, pages, or tables. It can also cause other processes to wait when it acquires locks.
2. Give your delete process as much memory as you're allowed. Don't specifically limit the memory available to an Optim delete process. The memory a delete process will consume is unpredictable and dependent on the options you choose, the state and size of all primary keys involved, and the size and number of rows being processed.