Boosting Database Performance & Easing Maintenance Tasks with Data Partitioning in Sybase Adaptive Server® Enterprise 15

BY ERIC MINER

EXECUTIVE OVERVIEW

ASE 15’s Data Partitioning option is a powerful addition to your database maintenance and performance toolbox. It can greatly reduce the time it takes to complete regular maintenance tasks while eliminating some all together. Data availability is vastly improved and your life as a DBA is less complicated. ASE 15’s Data Partitioning option will also provide an overall performance boost. Queries will run against smaller ‘pieces’ of data and the optimizer and query processor know what data values are on an individual partition and will avoid reading any that don’t contain the required data.

WHAT CAN DATA PARTITIONS IN ASE 15 DO FOR YOU?

There are a couple of very good reasons to implement the ASE 15 Data Partitions option—Making your data management easier and boosting your overall performance.

Let’s be honest, you don’t like wasting your time waiting around for the right moment to run simple and boring database maintenance jobs. You don’t like laying in bed hoping the phone doesn’t ring with news that an update statistics job won’t finish until lunch time the next day. You have better things to do with your work and free time. So you’re naturally interested in anything that might make your database maintenance tasks more efficient and reliable.

ASE 15’s Data Partitioning option can put you in control of ever shrinking database maintenance windows. As the saying goes—‘But wait! There’s more!’

SO, WHAT IS ASE 15’S DATA PARTITIONING?

Partitioning is a way to break-up your data into smaller ‘pieces’ and place it on individual data partitions that in turn may be put onto separate logical or physical devices.

You may see the term ‘semantic’ used in reference to data partitioning, it applies to the fact that you can specify exactly what data goes on what partition by using ASE command syntax.

In earlier versions of ASE you had very limited control over how data was placed on partitions and there was no way for the optimizer and query processing engine to take advantage of smaller ‘pieces’ of tables and indexes. Once set up, new data was put into the various partitions in a ‘round-robin’ fashion. In ASE 15 the Data Partitioning option not only allows you to break up your data and put it on various partitions it gives you very granular control over how data is divided up among Data Partitions. On top of this it may also give you a performance boost because the optimizer is very aware of not only the existence of data partitions but how you’ve chosen to place your data on them.

Data partitions are ‘objects’ within the database just as tables and indexes are. In fact they are subsets of tables and indexes, and individual pieces of a single table or index can reside on separate data partitions.

PARTITIONED TABLES

The main types of database objects that can be partitioned are tables and indexes. ASE 15’s Data Partitions option offers four methods for specifying how data is placed and stored on partitions:

(The following three partitioning methods are call “semantic” because they allow you to specify how data is placed on the data partitions you set up.)
Range Partitioning – This method allows you to place data on partitions based on a specified range of values. For example you can place customers whose name begins with A-M on partition one and N-Z on partition two. Of course you can be as granular as you wish. You can see how nicely this method fits with queries that search for both ranges of values and a single value. The optimizer knows that it will only have to read from the partition(s) that contain the single search value or that cover the range being searched on. Range Partitioning is particularly useful for OLTP and DSS environments.

Tip: You’ll want to be careful when choosing ranges of values so that all the partitions are well balanced with close to an equal number of rows on each. You’ll need to have a good idea of how values on the partition key (column) are distributed in the table. Why balance the partitions? So that accessing each will have very close to the same cost in terms of I/O.

Below is an example of the syntax used to create a table that uses Range Partitioning on a date/time column spanning four data partitions:

create table customer (ord_date datetime not null,
name varchar(20) not null,
address varchar(40) not null, other columns ........)

partition by range (ord_date)
(ord_date1 values <= (3/31/08) on segment1,
ord_date2 values <= (6/30/08) on segment2,
ord_date3 values <= (9/30/08) on segment3
ord_date4 values <= (12/31/08) on segment4)

As another example, imagine a table containing customer data that could be partitioned by name, customers with names beginning with A-M could be placed on one partition and N-Z on another. The possibilities are almost endless.

As of this writing most ASE 15 Data Partitions users have chosen to use the Range Partitioning method for their databases.

Tip: A table can also be partitioned using the alter table command.

create table nation (nationkey integer not null,name char(25) not null,regionkey varchar(30) not null,comment varchar(152) not null)on segment 1

partition by list (n_regionkey)
(region1 values ('Americas'),
region2 values ('Asia'),
region3 values ('Europe'),
region4 values ('Australia', 'Other') )

Hash Partitioning – The third type of semantic data partitioning is Hash Partitioning. Here the data is placed on Data Partitions based on the columns (key) specified and an internal hashing algorithm. You determine the partition key and ASE determines what values go on which partition. Hash Partitions are useful when there are large tables with many partitions in a DSS environment. They are also beneficial for queries with equality searches (=) on the partition key column or for tables with no particular order such as an alpha-numeric product code.

create table lineitem ( l_orderkey integer not null, l_partkey integer not null, l_suppkey integer not null, l_linenumber integer not null, l_quantity double not null, l_extendedprice double not null, l_extendedprice double not null, other columns ....)

partition by hash (l_orderkey, l_linenumber)
(litem_hash1 on segment1, litem_hash2 on segment2, litem_hash3 on segment3, litem_hash4 on segment4 )

These new choices of partition type make ASE 15’s Data Partitions very flexible.
Round Robin – This method will place data sequentially on the partitions you set up. ASE will insert rows so that each partition has close to the same number of rows in order to load balance. You can’t specify which data goes on a particular partition and queries have to read from more partitions to find the data they need. Maintenance will also require that all partitions be accessed. This is the same method used in earlier versions of ASE.

PARTITIONED INDEXES

There are two kinds of indexes you can use with Data Partitions—global and local. A global index has its entire structure on one partition, so it’s not partitioned itself. When queries on partitioned tables use a global index they will follow pointers from the index to the data in a particular data partition of the table. The index structure covers all Data Partitions of the table. The graphic below helps illustrate how a global index is structured.

The second type of index is a local index. Here the index’s structure is broken up into pieces which are stored together on a single partition. Since the partitioned index has to be based on the same column order of values that the Data Partitions are, a local index is akin to having smaller individual indexes for each data partition. If a query is searching for only a fraction of the data in the table it will only have to read one piece of the index. This improves concurrency by allowing different queries to simultaneously access different pieces of the same index. A query that is retrieving data from one partition will not interfere with a query that is using another partition resulting in less blocking, less I/O, more data availability and higher performance.

The illustration above shows a partitioned table with a partitioned non-clustered index on it. You can see why less I/O will be used to find the necessary data. A clustered index on a partitioned table (other than ‘round-robin’ partitioning) will always be a local index.

Just as with partitioned tables, partitioned indexes mean smaller pieces to read through.

Note: It’s important to make sure that a Local Index includes the partition key (column that the partition is based on) otherwise the index cannot be unique. Review the ASE product manuals for detailed information on designing and setting up your data partitioning strategy.
As you know all too well there are data maintenance and management tasks that you need to perform regularly, sometimes on a daily basis. Some of these, such as updating the statistics take time, on some tables a lot of time! When they're running they interfere with users and cause performance bottlenecks. You also know all too well that your maintenance window is usually very small. To get these jobs done you might be running them after hours, if there are after hours, or maybe over the weekend. You might prefer that they are run more often but let's face it, systems running 24/7 have become far more common then they were just a few short years ago.

Imagine your largest table, there are the usual OLTP transactions happening, on top of that complex DSS queries are also running. A large delete is working and data needs to be BCP'd in followed by a run of update statistics that needs to be done ASAP. Actually this may be a bit tame compared to some of your tables.

When you implement data partitions in ASE 15 you can eliminate the need to run 'reorg' and update statistics, in many cases, on the majority of your data. You can also reduce the time it takes to run them when you have to.

In most cases your maintenance window is way too small to regularly run these jobs on large tables and indexes. Data partitions allow you to spread this work over more but smaller windows.

So, how do ASE 15 Data Partitions eliminate the need for updating statistics or reorging a table or index? Simple, if your table is atomic at one end and spread across multiple partitions most of your data will be static. You only need to update statistics once since the values don’t change. Same with running reorg, it only needs to run once since there will be no chance for the tables or indexes to become fragmented. Keep in mind that the statistics are stored for the portion of a table that is stored on a partition. So, if you have a huge table spread over four partitions there will be four sets of statistics corresponding to the rows of the table stored on each partition.

Since each data partition in ASE 15 is a separate object in a database each can be accessed without interfering with other partitions. What this means is that you can run a maintenance job, such as update statistics, on one without interfering with any processes running on other partitions.

Here’s an example of an atomic ‘orders’ table and its data partitions –

Users are inserting into it all day long. Let's say that the table grows by 1 million rows per month. You want each partition to contain one quarter's data. Your business also requires that you keep five years of data in the table for DSS reports.

```
cREATE TABLE orders
(id INT, order_date DATETIME, ...) partition by range (order_date)
(ord_date1 values <= ('01-Jan-2007'), 5 years = 63 million rows
ord_date2 values <= ('01-Apr-2007'), 3 million rows
ord_date3 values <= ('01-Jul-2007'), 3 million rows
ord_date4 values <= ('01-Oct-2007'), 3 million rows
ord_date5 values <= ('01-Jan-2008'), 3 million rows
ord_date6 values <= ('01-Apr-2008')) 3 million rows
```

Here we’ve created the orders table with six partitions on the order_date column (the partition key) using the Range method.

The first partition contains data from the last five years. Update statistics and any needed reorg jobs only need to be run once, same for partitions 2 through 5 since this data is static the statistics will not be changed by users and no fragmentation will occur. On partition 6 you’ll have to run the maintenance commands regularly until the last business day of June 2008. Along with making your maintenance jobs easier this setup also makes it easy to archive off old data. Just BCP out the rows for the oldest year from partition 1 and rerun update statistics on it.

Note: Remember to rerun update statistics on the partition containing the archived data whenever you remove and/or add some. This applies to any partition whose data changes.

By the way—For the orders table above with 78 million rows, update statistics on the whole table takes about 1 hour and it only takes about 45 seconds on an individual partition such as partition ord_date4.

What maintenance commands can be run at the partition level? There are more than a few –

- `UPDATE [INDEX] STATISTICS table_name partition index_ptn_name`
- `REORG COMPACT table_name partition ptn_name` also
- `REORG FORWARDED_ROWS, REORG RECLAIM_SPACE REORG REBUILD table_nameix_name partition ptn_name`
- `DBCC CHECKTABLE(table_name, null, ptn_name)`
- `DBCC CHECKINDEX(table_name, index_id, null, ix_ptn_name)`
- `DBCC REBUILD_TEXT(table_name, col_name, o, ptn_name)`

[Note: double check this syntax]
A quick tip: Use the datachange() function to determine when to run update statistics on the active partition. Below is a fragment of a script to run update statistics on partition ord_date6 from above when 5% of the rows in the partition have changed in any way. You can also use the derived_stat() function to determine when reorg needs to be run.

```sql
declare @dc float
select @dc = datachange('orders', 'p3', null)
if @dc > 5
begin
  update statistics orders partition ord_date6
end
```

It all comes down to this—dealing with smaller individual pieces of data rather than whole tables and indexes allows your maintenance tasks to run faster and more efficiently.

BOOSTING PERFORMANCE

So, as we’ve seen ASE 15’s Data Partitions option can make your database maintenance chores easier, and in some cases eliminating them all together. While data partitions are designed as a data management tool they are also a performance booster. How? In the simplest terms since data partitions are much smaller pieces of data a worst case scenario table scan will have much fewer rows to scan through. At the query level the optimizer is aware of what values are on what partition and can decided which partition(s) need to be read while avoiding reads of the others. At a higher level data partitions are the building blocks for parallel processing and can scientifically improve performance when used in combination. Let’s take a look at the partitioning methods and how they relate to queries.

Data Partitions and Queries

When a query contains a SARG (search argument) value on the column used to create the partition (partition key) the optimizer can use the search values to eliminate reading data partitions that don’t contain the SARG value. This is called ‘partition elimination’, and you’ll see a message that it was used in the output of some optimizer analysis tools [get names]. One thing to keep in mind here is that partition elimination does have a restriction – On Range, List and Hash partitions the optimizer can’t use partition elimination for queries containing ‘not equal to’ (!=) SARGs or SARGs that have expressions on the partitioning column.

Partition elimination will occur most often when your queries are searching for a value or range of values in the partition key column. As we’ve already seen, breaking a table or index into smaller pieces makes all operations on individual pieces much faster. A query that only needs to scan a portion of a table or index will run considerably faster than one that has to read the entire object. This is especially true when a query joins large tables, with partitioned tables and indexes there is far less processing of joined data to be done.

Highly Efficient Parallel Query Processing

ASE 15 introduces new parallel query processing functionality that dramatically improves query performance. ‘Parallelism’ and Data Partitions combined go even further to improve ASE 15’s performance.

ASE 15 supports both vertical and horizontal parallelism. Vertical parallelism provides the ability to use multiple CPUs at the same time for one or more operations within a query. Horizontal parallelism allows multiple instances of the same piece of the query to be run on different data located on different partitions. This is where partitioning helps improve performance. With partitioned tables and indexes parallelism will allow the same query to be broken down into pieces (instances) that can then be run simultaneously on different partitions. The illustration below shows a query taking advantage of horizontal parallelism.
Parallel query processing and partitioning are particularly efficient in a ‘mixed-workload’ or DSS environment where very large columns are commonly read. High performance and no blocking of one process by another are critical when you have an application running in such an environment.

EASIER LOADING AND ARCHIVING OF DATA

Partitioning can be used to speed up the loading of large amounts of data to a table. You can use this method even if you plan to have an un-partitioned table.

The steps to do this are:
1. Create an empty table, and partition it n ways.

   ```
   create table currentpublishers
   (pub_id char(4) not null,
    pub_name varchar(40) null,
    city varchar(20) null,
    state char(2) null)
   partition by roundrobin 3 on (seg1)
   ```

2. Then bcp in the data using the partition_id option. Copy presorted data into each partition. For example, to copy datafile1.dat into the first partition of currentpublishers:

   ```
   bcp pubs2..currentpublishers:1 in datafile1.dat
   ```

3. Unpartition the table using alter table unpartition.

4. Create a clustered index with the with sorted_data clause.

When Data Partitions are created, ASE 15 places an entry for each in the syspartitions table. When you bcp in with the partition_id option it loads data into each partition in the order listed in syspartitions.

CONCLUSION

ASE 15’s Data Partitioning option is a powerful addition to your database maintenance and performance toolbox. It can greatly reduce the time it takes to complete regular maintenance tasks while eliminating some all together. With data partitions in place it’s easy to set up maintenance tasks not only so that they run faster, but they can also be set up to not interfere with other processes and operations running on the same table. Data availability is vastly improved and your life as a DBA is less complicated.

On top of improving your database maintenance, ASE 15’s Data Partitioning option will also provide an overall performance boost. Queries will run against smaller ‘pieces’ of data and the optimizer and query processor are fully aware of what data values are on an individual partition and will avoid reading any that don’t contain the required data.

All in all data partitioning is well worth implementing on your ASE 15.

---

Eric Miner is an independent consultant who has worked on database management systems for over 15 years. He was with Sybase from 1992 until 2004 working in the ASE Support and Engineering organizations. While in Engineering he was a member of the ASE Optimizer team. Eric served as the ISUG (International Sybase User Group) ASE Enhancements Director and received the organization’s ‘Outstanding Achievement’ award in 2002. He has written several articles for the ISUG Technical Journal, co-authored the book “ASE 12.5 Performance and Tuning”, delivered sessions at TechWave conferences and has participated in a number of ISUG international tours and user education sessions. Eric is a longtime amateur photographer and has won awards in international competitions. He lives in Charlotte, North Carolina with his wife Laurie and son Liam.