

Data Partitioning in Sybase Adaptive Server[®] Enterprise 15 for Lower Costs, Higher Performance

TABLE OF CONTENTS

- 1 The Management and Performance Challenges of VLDBs
- 1 What is Data Partitioning in ASE 15?
- 2 Why Use Data Partitioning?
- 4 Data Availability
- 4 Improved Query Performance When Using Data Partitions
- 7 Efficient Parallel Query Processing
- 7 How and When to Use Data Partitioning
- 9 Conclusion

THE MANAGEMENT AND PERFORMANCE CHALLENGES OF VLDBS

The size of data sets is already growing exponentially, and we can expect that growth rate to accelerate. At the same time businesses are using a higher and higher percentage of data for decision support and analysis. The result of these two trends is the development of VLDBs (very large databases),

What exactly is a VLDB? Since one business's VLDB can be another one's average size database, VLDBs are best defined not in terms of a fixed size but by the challenges they present.

First, they make it difficult to fit maintenance and management tasks into an allotted time to avoid negatively impacting application performance and data availability.

Second, they can impede query performance by consuming a great deal of time and resources in order to return the required data. A query that has to search through an entire large table or index can slow overall performance to unacceptable levels. This is particularly true in decision support systems (DSS) such as a data warehouses and especially in ODSS (operational DSS) where both DSS and on-line-transaction-processing (OLTP) occur simultaneously.

Third, they affect the cost of operating applications on ASE.

To address these VLDB challenges, ASE 15's data partitioning capability reduces the time needed for management and increases the performance of applications. ASE 15's data partitioning will in fact help in the management, maintenance and performance of ASE databases of any size.

WHAT IS DATA PARTITIONING IN ASE 15?

Data partitioning breaks up large tables and indexes into smaller pieces which can reside on separate partitions, allowing DBAs to have a very fine level of control over data placement, maintenance and management.

A 'segment' is a portion of a device that is defined within ASE. It is used for the storage of specific types of data such as system data, log data and the data itself. Partitions can be placed on individual segments or multiple partitions can be placed on a single segment. In turn, a segment or segments can be placed on any logical or physical device thus isolating I/O and aiding performance and data availability.

Data in a table or index on one partition can be managed and processed separately from data in other partitions. Queries only need to access partitions that contain the required data.

DBAs can perform management and maintenance tasks much more quickly on individual smaller partitions rather than on huge tables and indexes. To save even more time, some common tasks can be run in parallel on multiple partitions. Plus a DBA can automate tasks on partitions. As the size of the data grows, more partitions can be added.

There are four methods of data partitioning offered in ASE 15 which we will introduce here and discuss in more detail later. The first is called round-robin partitioning, the only method used in pre-ASE 15 versions. This method places data sequentially on partitions. There is no way to specify which data goes on which partition, and all partitions will be involved in query processing.

The next three methods are referred to as 'semantic data partitioning' methods because the placement of data on partitions can be specified.

The partition method expected to be widely used in ASE 15 is Range Partitioning. With this method, DBAs can specify the ranges of values to be placed on each partition.

The next is List Partitioning, where individual values can be placed on separate partitions.

The third type of semantic partitioning is Hash Partitioning. Here the data is placed on partitions based on the columns specified and an internal hashing algorithm.

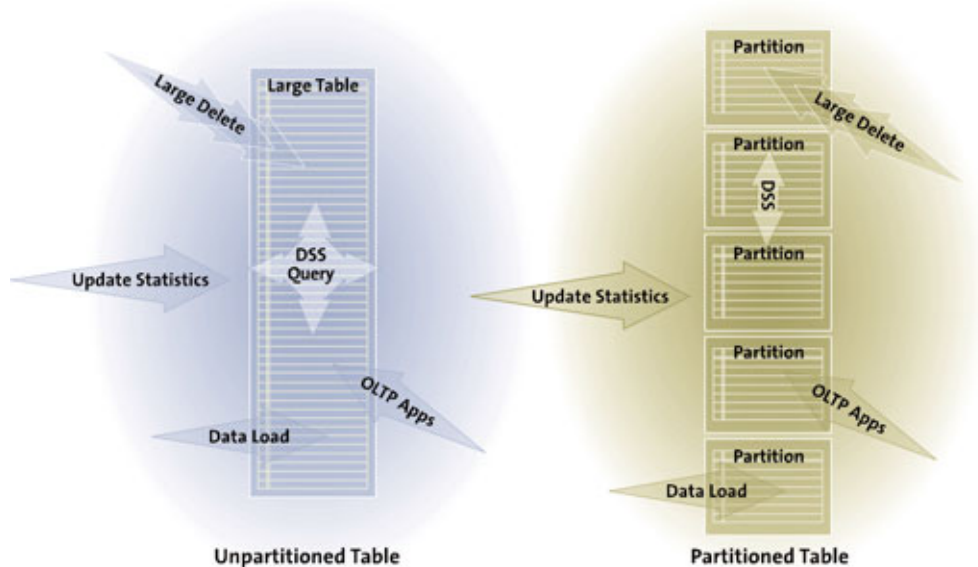
WHY USE DATA PARTITIONING?

By using data partitioning in ASE 15, businesses lower the cost of keeping and using databases of any size on ASE, while improving the performance of their applications—and making their data more available.

Reducing the Cost of Managing and Maintaining Databases using Data Partitioning

The most costly aspect of operating a database of any size is the time and effort required by DBAs to manage and maintain them, Databases can become so large that there simply isn't enough time to perform all of the necessary maintenance tasks. This not only has a direct impact on application performance but also puts data in danger by making backups less practical due to the time required.

By making the use of various utilities much more efficient, data partitioning in ASE 15 has a positive impact on the time and resources spent maintaining and managing large databases



The diagram above shows a number of activities occurring on a table. A large complex DSS style query is running, OLTP processing is occurring, BCP is being used to load data and update statistics is running to freshen statistics. In the un-partitioned table on the left, the activities under way are enough to slow operations to an unacceptable level if not bring many of them to a halt. In the table on the right, however, data is partitioned, and each of the operations are running on separate partitions, without affecting the others.

By partitioning a table or index into smaller pieces, DBAs can run utilities on a per partition basis. This results in the utilities running faster, allowing other operations to work efficiently on data in other partitions and insuring that the bulk of the data in the table is available for applications. In this way the use of data partitioning in ASE 15 increases the operational scalability of applications using VLDBs.

With partitioned tables and indexes, a DBA can schedule maintenance tasks and cycle through one partition of a table at a time, or perform tasks simultaneously on multiple partitions if preferred. If maintenance on a given partition is not practical at a certain time, that partition can be skipped and returned to later. The maintenance schedule can also be tightly integrated with ASE's Job Scheduler, thus freeing up the DBA for other important activities.

Two commonly run utilities are also two of the most resource and time consuming—`update statistics` and `reorg`. Both need to be run on tables and indexes on a regular basis in order to assure good performance from queries.

The `update statistics` utility gathers statistics about the table, its indexes and the distribution of data in the columns of the table. This information is needed by the query processing engine's optimizer to determine the most efficient way to access the required data. The `reorg` utility is used to reclaim empty space in tables and indexes. This operation is important for space management and for efficient query performance. Some tables have grown so large that it is virtually impossible to run either of these two important utilities without exceeding the allotted time and impeding regular business operations.

As an example, many databases contain a date/time column used when recording a transaction. It's important to keep the distribution statistics for such columns up to date so that queries run as efficiently as possible. But to update statistics for an un-partitioned table and index, the entire column has to be read. With huge tables, this can have a very major impact on performance and data availability.

The same operations on a partitioned table and index will take considerably less time and have minimal impact on overall performance and data availability. This is because the distribution statistics only have to be gathered from the partition that contains the most recent date/time values. The same applies to the `reorg` utility. When run on a table and index that are partitioned, only the target partition will be affected, keeping the rest of the table and index free for use.

Another time-consuming but necessary task is the running of common table and index diagnostic utilities such as the DBCCs that check for data consistency. Running these diagnostics on a per-partition basis makes this critical task more efficient.

Still another time and resource intensive operation is the archiving off or deletion of large amounts of data that are no longer needed in the table. The BCP (bulk copy program) can be run on individual partitions allowing data to be put into or taken out of a table without interfering with other operations that are using that same table. Only the partition whose data is being imported or exported via BCP will be affected. By partitioning tables, data that 'ages out' can be managed more easily. BCP can also run in parallel on multiple partitions simultaneously.

The `truncate table . .partition` command is used to perform mass deletions of data from a table. Often this is data copied off of the table via BCP or other backup methods. In the past truncating the table deleted all data from it and could take a long time. Now, however, data can be deleted on a per-partition basis.

DATA AVAILABILITY

The availability of data to applications is vital for the functioning of any business. If applications can't get to the data they require, work doesn't get done.

As we have seen, the purpose of partitioning tables and indexes is to divide them into separate and smaller 'pieces' so that operations on each piece can be isolated from other operations on other pieces, thus insuring that more data can be accessed by applications more of the time. The illustration of a partitioned index below demonstrates this.



The reorg utility is running on partition 2 while statistics on partition 3 are being updated. Either of these utilities would make the index less available if this were an un-partitioned index. In this case both are running simultaneously while data on partition 1 is still available to an application.

Similarly, if queries are changing the data, indexes can be updated on one partition without affecting operations on others.

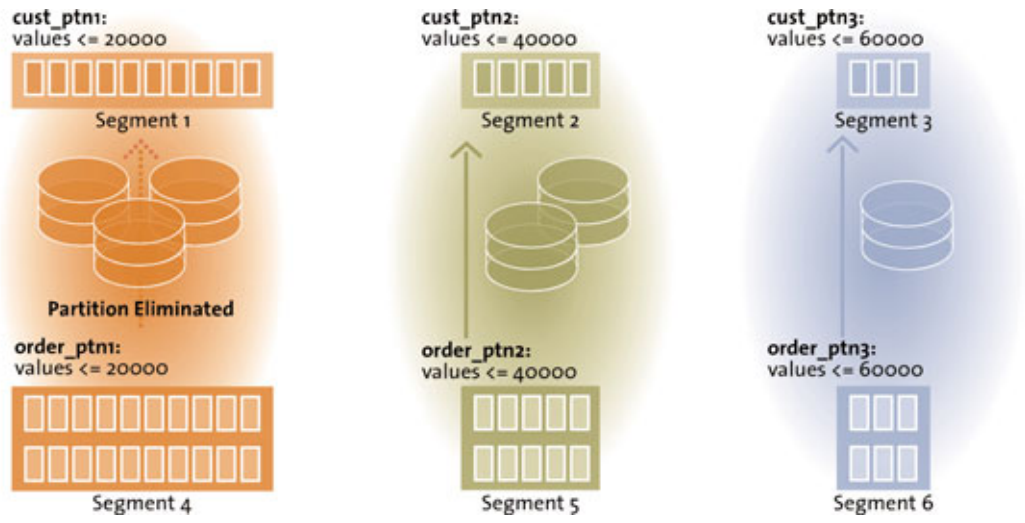
In an ODSS environment DBAs can design partitions so that transactional operations occur on one partition while DSS queries run on others. The data availability that partitioning offers is critical to the performance of applications running in an ODSS environment.

Partitions also enhance availability by being placed on various physical devices. If one device fails, the remaining partitions will still be available for use.

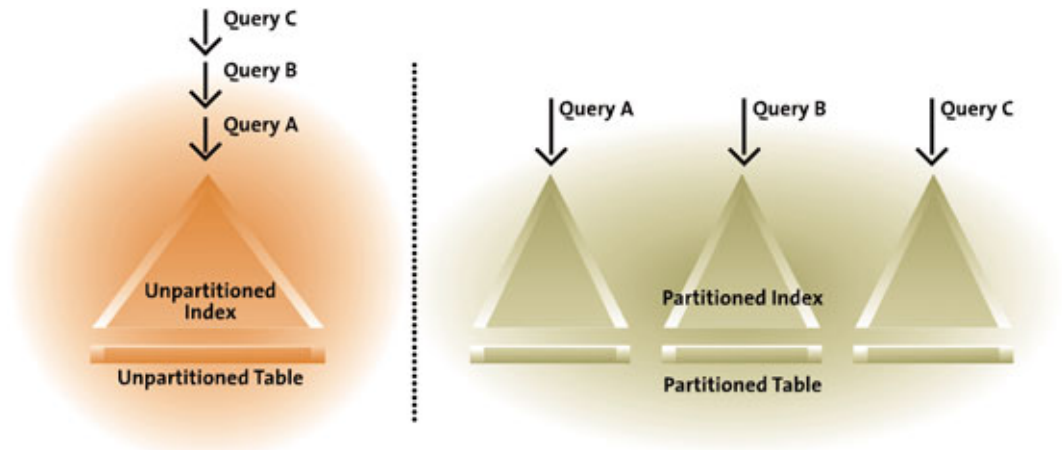
IMPROVED QUERY PERFORMANCE WHEN USING DATA PARTITIONS

Not only do data partitions reduce the time it takes to perform database maintenance and management tasks, but as mentioned earlier they also have a positive affect on the performance of queries and therefore applications. As we have 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. Partitioned tables and indexes require far less processing of joined data.

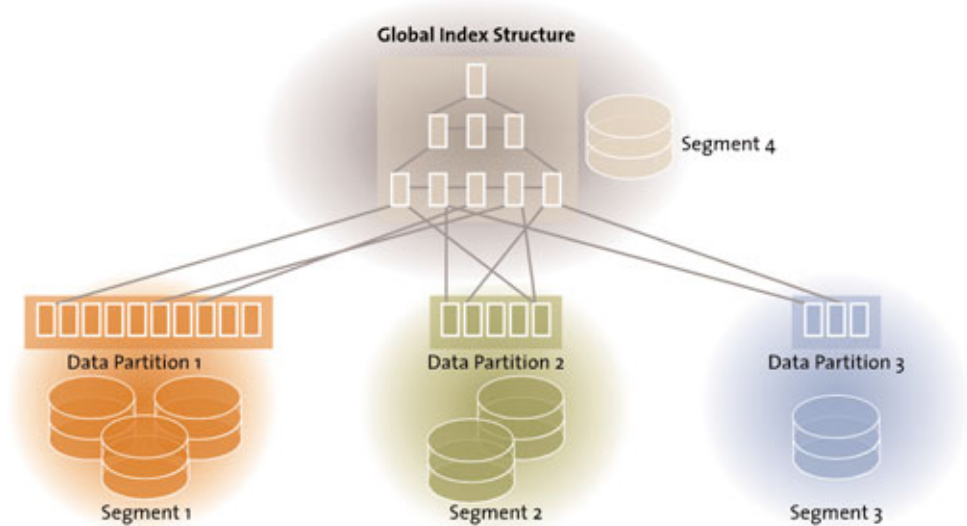
ASE 15's query processing engine will ignore partitions that do not contain relevant the data. In the illustration below, a query is looking for all rows that have a cust_ptn that is greater than or equal to 30000. Since the table has been partitioned by range based on the cust_ptn column, the only partitions that have to be read are partitions 2 and 3. Known as 'partition elimination', this process is a major performance enhancement in ASE 15. The less that needs to be read, the faster the query runs.



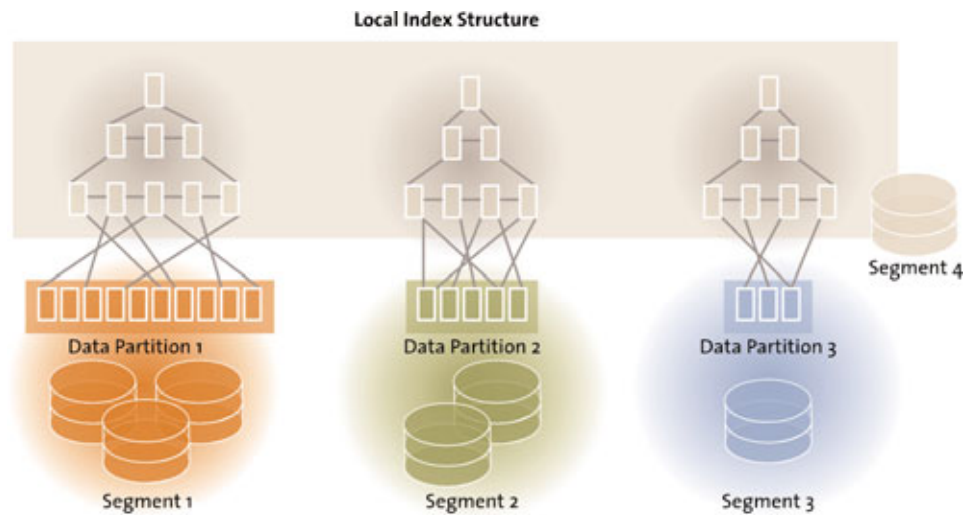
As mentioned earlier, indexes, like tables, can be partitioned. The advantages of partitioning an index are numerous. First, multiple queries that need to access the index may be able to use the index simultaneously to access data on different partitions. Secondly, while a utility is running on one partition of the index, queries can still access those parts of the index that are on other partitions. Finally, operations such as update, insert and delete cause the rows of the index to be changed. This writing activity only needs to be done on the portion of the partitioned index that is affected by the change in the data. The illustration below gives you an idea of how partitioned indexes can be advantageous.



There are two methods of index partitioning—global and local. A global index has its entire structure on one partition. When queries on partitioned tables use a global index, they will follow pointers from the index to the data on the table's data partition(s). 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 similar to having smaller individual indexes pointing to data in corresponding data partitions. 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 increases 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. The result is less blocking, less I/O, more data availability and higher performance.



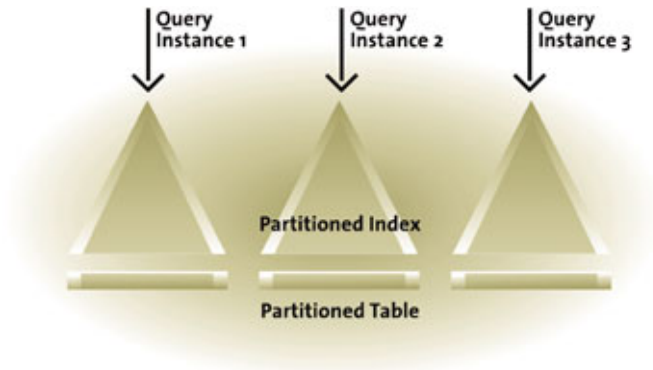
The illustration above shows a partitioned table with a partitioned non-clustered index on it. Obviously less I/O is required to find the necessary data. This is especially true when conditions in the query further limit what data is required. A clustered index on a partitioned table (not using 'round-robin' partitioning) will always be a local index.

Another advantage to using partitioned indexes is that the size of each is based on the number of rows in its corresponding table partition. Smaller indexes take less time to scan.

EFFICIENT PARALLEL QUERY PROCESSING

ASE 15 introduces new parallel query processing functionality that dramatically improves query performance. The combination of parallelism and partitions further improves the overall performance of ASE 15.

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 or disk devices. 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 be run simultaneously on different partitions to gather the data needed for the query as a whole. The illustration below shows a query taking advantage of horizontal parallelism.



Parallel query processing and partitioning are particularly efficient in a 'mixed-workload' or ODSS environment where very large columns are commonly read. When applications are running in such an environment, high performance is critical, and one process should not be blocking another.

HOW AND WHEN TO USE DATA PARTITIONING

We have seen how partitioning in ASE 15 can substantially lower the cost of operating ASE on VLDBs by enabling DBAs to effectively perform necessary maintenance and management tasks. We have also seen how partitioning vastly improves the overall performance of applications, particularly in an ODSS environment.

Partitioning tables and indexes is relatively simple. A table can be partitioned at its creation or later. Once partitioned, data put into the table will follow the rules set by the partition, whether it's done with an insert, update or using BCP to import data. An index can also be partitioned when it's created, but an existing index cannot be partitioned. It will need to be dropped and recreated with partitioning.

Here are some situations where DBAs might consider using the three semantic methods of partitioning described earlier:

Range Partitioning – This method allows DBAs to specify what data will be contained in each partition based on a range of values in the key column. For example the values 1,2,3,4 could be on partition 1 while 5,6,7,8 are on partition 2 and so on. Below is an example of the syntax used to create a table that uses range partitioning on a date/time column:

```
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 <= (3/31/05) on segment1,  
ord_date2 values <= (6/30/05) on segment2,  
ord_date3 values <= (9/30/05) on segment3  
ord_date4 values <= (12/31/05) on segment4)
```

Range partitioning is especially useful for tables with constant updates, inserts and deletes that contain a column or columns with sequential data in them such as a customer ID or an order/transaction date. Such tables require the most attention from DBAs for maintenance and management. These tables are also commonly used in decision support style queries making them excellent candidates for range partitioning. While transactions are occurring on a single partition, DBA tasks and DSS queries can access other partitions. At the same time it is critical that the DBA keep the distribution statistics up to date on the active transactional partition so that new data can be described to the query processing engine. As we have seen, the time to update the statistics is considerably shorter when it only has to be run on a single partition.

List Partitioning – List partitioning is similar to Range partitioning, but here the actual value(s) to be placed on a partition are specified. The example given earlier was of list partitioning done on a 'region' key column that contained regions of the globe. List partitioning is useful to control where specific values are stored, even if the column itself is not sorted, and where the order of values in the partition is not important. Below is an example of the syntax used to create a table with list partitioning:

```
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 – This method distributes values to partitions based on the column(s) specified and an internal hashing mechanism. There is no need to specify a list or range of values. If the column key contains unique values, or values that have little duplication, hash partitioning will balance the data across all the partitions. However, if there is extensive duplication of values, the partitions can become skewed, with some containing more rows than others.

Hash partitions are useful when many partitions are desired for a large table and the values in the key column are not in a sorted order. They also help equality searches done by the query processing engine run more efficiently.

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

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

CONCLUSION

A central feature among many new features in ASE 15, data partitioning offers significantly easier maintenance and management of VLDBs and considerably higher performance of ASE in general. Easier, more efficient maintenance and management drive down the cost of running ASE on VLDBs, and higher performance insures that applications perform their best in a mixed-workload environment.

