Oracle 8

VLDB Design & Migration Considerations Under Oracle8™

An Oracle Technical White Paper

June 1997

ORACLE®

Enabling the Information Age™

VLDB Design & Migration Considerations Under Oracle8

INTRODUCTION

As databases grow larger, the task of managing and maintaining them increases at least proportionally, if not exponentially. Activities such as backing up, reorganizing, loading, and purging become more complex and time consuming as the amount of data increases. Often it becomes impossible to perform these maintenance operations within the window of time provided by the customer.

Very Large Databases (VLDBs) are approaching terabyte size and soon will exceed into the tens of hundreds of terabytes and maybe even to one petabyte. Not only does the size increase, so too does the up-time requirements of the databases. Maintenance tasks must be completed quickly with the impact of scheduled outages minimized.

Many operational and management issues must be considered in designing a very large database under Oracle8 or migrating from an Oracle7 database. If the database is not designed properly, the customer will not be able to take full advantage of Oracle8's new features. This white paper discusses issues related to designing a VLDB under Oracle8 or migrating from an Oracle7 database.

PARTITIONING STRATEGIES – DIVIDE AND CONQUER

One of the core features of Oracle8 is the ability to physically partition a table and its associated indexes. By partitioning tables and indexes into smaller components while maintaining the table as a single database entity, the management and maintenance of data in the table becomes more flexible. The data management now can be accomplished at the finer-grained, partition level, while queries still can be performed at the table level. For example, applications do not need to be modified to run against the newly partitioned tables.

The divide-and-conquer principle allows data to be manipulated at the partition level, reducing the amount of data per operation. In most cases, this standard also allows partition-level operations to be performed in parallel with the same operations on other partitions of the same table, speeding up the entire operation.

BENEFITS FROM TABLE PARTITIONING

The greatest benefits from Oracle8 partitioning is the ability to *maintain* and *administer* very large databases. These gains far outweigh any *performance* benefits. The following dimensions of scalability[1] can be found in Oracle8:

- Higher Availability
- Greater Manageability
- Enhanced Performance

Higher Availability

Using intelligent partitioning strategies, Oracle8 can help meet the increasing availability demands of VLDBs. Oracle8 reduces the amount and duration of scheduled downtime by providing the ability to perform downtime operations when the database is still open and in use. Refer to the paper, *Optimal Use of Oracle8 Partitions,* for further information about this topic. The key to higher availability is partition autonomy, the ability to design partitions to be independent entities within the table. For example, any operation performed on partition X should not impact operations on a partition Y in the same table.

Greater Manageability

Managing a database consists of moving data in and out, backing up, restoring and rearranging data due to fragmentation or performance bottlenecks. As data volumes increase, the job of data management becomes more difficult. Oracle8 supports table growth while placing data management at a finer-grain, partition level. In Oracle7, the unit of data management was the table, while in Oracle8 it is the partition. As the table grows in Oracle8, the partition need not also grow; instead the *number* of partitions increases. All of the data management functions in Oracle7 still exist in Oracle8. But with the ability to act against a partition, a new medium for parallel processing within a table now is available. Similar rules apply to designing a database for high availability. The key is data-segment size and to a lesser degree, autonomy.

Enhanced Performance

The strategy for enhanced performance is divide-and-conquer through parallelism. This paradigm inherently results in performance improvements because most operations performed at the table level in Oracle7, now can be achieved at the partition level in Oracle8. However, if a database is not designed correctly under Oracle8, the level of achievable parallelism and resulting performance gains will be limited. The table partitioning strategy used in Oracle7 may not necessarily be adequate to make optimal use of the Oracle8 features and functionality.

TO PARTITION OR NOT TO PARTITION

When should a table be partitioned?

The following guidelines can be used to determine when a table should be partitioned:

- If migrating from Oracle7 to Oracle8, entities represented as UNION VIEWS should become partitioned tables under Oracle8. The number of partitions may be different but in most cases, the reason a UNION VIEW was created in Oracle7 still applies in Oracle8.
- For manageability, the author recommends that tables greater than 2GB should be partitioned. This is not a strict maximum, but a number to use as a guide.

1 Refer to the Scalability Workshop by Enterprise Scalable Solutions, Center of Excellence.

- Tables used to perform parallel Data Manipulation Language (DML) operations MUST be partitioned. These tables normally are the subject of large-batch jobs.
- Large tables with a definable proportion of read-only records should be partitioned. A typical example is historical tables where only the current month of data is updateable and the other 11 months are read-only.
- Tables within a well-partitioned, parallel server environment that are modified by multiple instances, such as an intersecting set of tables,[2] should be partitioned.
- Tables that need to be accessed with parallel-index scans should be partitioned.
- Tables with data that may need to be off-line in the future, such as historical tables, should be partitioned.
- Tables where data-to-disk affinity is crucial should be partitioned. Data-to-disk affinity is typical in Massively Parallel Processing (MPP) environments that have a table striped across multiple nodes. For example, better performance yields are achieved when records A-E reside on a disk local to node 1, F-P on a disk local to node 2, etc..

HOW MANY PARTITIONS SHOULD I USE?

The number of partitions used to define a table is not an arbitrary determination. Various external influences must be analyzed before making this important design decision, and often these considerations may conflict with one another. Sometimes a tradeoff must be made between database availability and performance. So, the designer first must determine what is most important to the customer and build the database to best satisfy the higher-priority problems.

Manageability

Data volume

The easiest way to determine the number of table partitions is to consider the amount of data in the table. A partition should be small enough to manage, backup, and rebuild if necessary. The management of many partitions in Oracle8 is negligible compared to Oracle 7.3 union views.

Tip 1: You may want to let the partition size dictate the number of partitions in the table.

Backups

If using "export" as part of a complete backup strategy, then partitions should be small enough to export to a single filesystem. Also a table can be exported in parallel by running a separate export against each of the partitions in the table.

Tip 2: If using "export" as part of the backup strategy, then a partition must be small enough to fit on the filesystem.

Read-only Data

Often the amount of data in a table that regularly changes is relatively small, and the remaining data is static. Under Oracle7, the entire table had to be backed up, even though only a small portion of the table changed. By partitioning the read-only data from the read-write data, and placing each partition into its own tablespace, only the read-write data needs to be backed up regularly under Oracle8. The tablespaces containing read-only data would, of course, need to be marked READ-ONLY, as in Oracle7, and backed up only once in Oracle8. If

2 Refer to the Scalability Workshop by Enterprise Scalable Solutions, Center of Excellence.

backing up the database is a high-priority issue, then partitioning under Oracle8 can be used to reduce the amount of data needed to back up regularly.

Tip 3: Partition a table to separate read-only from read-write data to reduce the amount of data to back up.

Parallel Index Create

The maximum degree of parallelism obtainable during a global index create is dependent upon the number of partitions in the table. When issuing a "CREATE INDEX" statement, one parallel slave will be assigned to each table partition. For example, if the table has 10 partitions, then a maximum of 10 slaves will perform the index build, one per partition. If nine of these 10 table partitions contain no rows, then 10 slaves still will be allocated, but nine of them will complete quickly and sit in "dequeue wait" until the tenth slave has finished. This data skew causes nine slaves to be unavailable until the tenth is complete.

When deciding the number of partitions in the table, consider how the indexes will be built and the degree of parallelism required to build them in a reasonable time. Instead of 10 average-size partitions that would allow a maximum parallelism of degree 10, use 100 smaller partitions allowing up to a parallel degree of 100.

A workaround exists to achieve a degree of parallelism in the index build greater than the number of partitions in the table. First, create the empty, partitioned table and create the global index on the table. Set the index as UNUSABLE, then load the data into the table. After the load is completed the global index can be *rebuilt* using the ALTER INDEX REBUILD PARTITION syntax. This statement rebuilds a single partition of the index, allowing a degree of parallelism to be applied to that partition build.

Tip 4: The maximum degree of parallelism in a standard, global index build is equal to the number of partitions in the underlying table unless you use the ALTER INDEX REBUILD PARTITION syntax.

Parallel Analyze

The time to analyze a table can be significant in a data warehouse running the cost-based optimizer. Because partitions in a table can be analyzed independently, a parallel analysis of a table can run by executing multiple partition analyzes in parallel. The ability to parallel analyze will become automatic in future releases of Oracle8.

Reducing Fragmentation

If an application periodically deletes batches of records from a table and causes fragmented indexes, consider partitioning around this requirement. For example, if the deletes occur "half daily," then partition by half day to allow an entire partition to be dropped along with its associated local indexes.

Tip 5: In applications that have large purge requirements, consider partitioning your table around the purge cycles.

Managing Data Skew

The data skew needs to be considered when determining the number of partitions and their boundaries. The data skew can cause partition size to vary greatly across the table, creating a few, very large partitions and leaving the remaining partitions with very little data. Partitions of near-equal size ease table management overhead. If nine of 10 partitions are empty, the partition ranges should be better defined to spread the data more evenly across all partitions. Or if several partitions contain more data than the rest, break larger partitions into smaller partitions so that the partition sizes are more balanced across the entire table. For example, if the months of November, December and January contain more data than the rest of the year because of increased retail trading, break these monthly partitions into weekly partitions.

> *Tip 6: Consider the data skew when determining the number of table partitions and their associated upper bounds.*

Availability

Smaller partitions/local indexes

If an application's most important requirement is availability, the designer must attempt to have tables with truly *autonomous* partitions. This can be achieved by minimizing or completely removing the use of global indexes on a table. Autonomous partitions are those that are not impacted by operations on other partitions in the same table.

If a global index exists on a table, then dropping or invalidating one table partition will invalidate the entire global index. At this point, the global index no longer can be used and must be rebuilt. Any data access that relies on that global index no longer can be used. The data in the other partitions still is available but not via that global index. Instead, the data is accessible through a full-table scan or via an alternate index if one exists.

The above example is a case where table partitions can greatly impact access to other table partitions, as shown in the following diagram. For example, dropping table partition 3 causes its local index to be dropped and the global index to become unavailable. All data access, even if the data is in partitions 1 or 2, will be unable to use that global index. By using only local indexes, invalidating any one table partition will not impact any other partition. However, data access via a global index may be faster than a local index, and a tradeoff may need to be made between availability and performance.

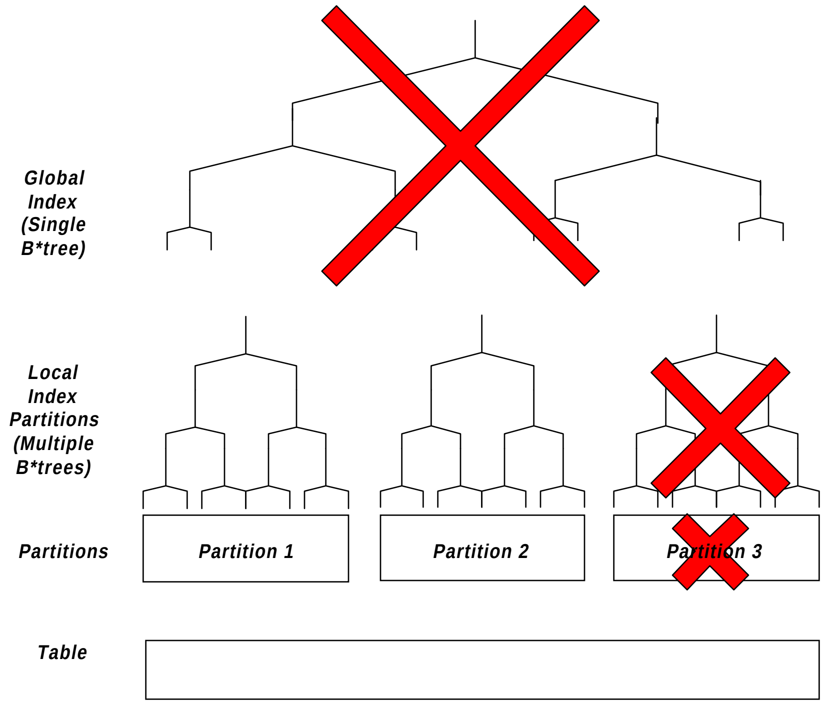
Global Index (Single B*tree)
Local Index Partitions (Multiple B*trees)
Partitions
Partition 1
Partition 2
Partition 3
Table

Figure 1: Impact of global indexes on availability.

Tip 7: Minimize the number of global indexes in applications where table availability is a key requirement.

Performance

Degree of parallelism during DML operations.

If the application is utilizing parallel DML functionality, the table running against the DML needs to be partitioned. The degree of parallelism achievable is equal to the number of partitions in the partitioned table of the DML operation. When the DML statement is initiated, one parallel slave is assigned to each partition. For example, a table partitioned 10-way will use a maximum degree of parallelism of 10, even if a higher degree is specified.

If a DML statement only requires data from one partition, then it will not be parallelized because only one slave will be allocated. When designing a database, this degree of parallelism is a very important consideration. Often, batch jobs perform the majority of DML operations in a database. The degree of parallelism achievable by a batch job greatly influences the execution time of the job. Therefore, it is important to partition tables to support parallel data manipulation language (PDML).

Batch jobs need to be analyzed to determine which data they process. For example, a table partitioned into 12 monthly partitions may not be sufficient to parallelize batch jobs if the jobs only run against a single month of data. In this case, partition weekly so the degree of parallelism achievable would be four to five—depending upon how the month overlap is handled. An alternative solution is to add a second column to the partition key to create more partitions.

Tip 8: Determine the degree of parallelism required from PDML operations. The maximum degree of parallelism achievable equals the number of partitions in the table.

Transactional Skew

Another consideration when partitioning a table is the transactional skew, which is the distribution of workload across the table partitions. If 80 percent of the workload is on 20 percent of the data, then the volume of access to that data must be examined. Transactional skew can cause unbalanced I/Os, where a small number of disks support the majority of the workload. When determining how to partition data, ensure that the most frequently accessed data is partitioned adequately to support the load. To better distribute I/O, break partitions into smaller ones on separate disks and in different tablespaces or stripe the existing partitions across more disks.

When designing PDML operations, remember the slaves are assigned two partitions assuming a uniform transaction distribution. If most of the data required by the operation is located in a small number of partitions, then the operation will not execute as quickly as if the data-access requirements were evenly spread across all partitions. In this case, dividing the most-active partitions into multiple, smaller ones will balance the workload more evenly across the parallel slaves, resulting in faster execution time.

Tip 9: Watch for transactional skew that causes the majority of the workload to fall against a small number of partitions. Break these partitions into multiple ones to gain the maximum benefits from parallel processing, and put them on different disks if I/O is a potential bottleneck.

Oracle® Parallel Server partitioning

Table partitioning provides greater control over the physical location of data because partitions can be defined on a specific datafile or disk. For example, take an application partitioning strategy that balances the workload across the instances by customer surname. An Oracle Parallel Server configuration with three instances could have surnames between A-E serviced on instance 1, surnames from F-P on instance 2, and surnames from Q-Z on instance 3.

When considering a table-partitioning strategy try to at least match the instance-partitioning strategy. In this case, the customer table could be partitioned at least three ways by customer surname: A-E, F-P, Q-Z. Parallel Cache Management (PCM) lock allocation is simplified by putting each of these partitions in their own tablespace and assigning minimal locks to those associated datafiles.

The partition layer provides data dependent routing to ensure that datafile access always is local to a particular Oracle Parallel Server instance (node) because the application partitioning strategy matches the data partition-ing strategy. This is relevant particularly to PDML operations. The number of partitions need not be limited to three in the example explained above. The important point is to ensure that one datafile corresponds with one instance, producing a tidy, easily managed parallel-server configuration.

Tip 10: Under Oracle Parallel Server, attempt to at least match the application partitioning strategy when considering a table-partitioning strategy.

On Line Transaction Processing (OLTP) Index Lookups

Tradeoffs often must be made between availability/manageability and performance when making database-design requirements. If performance is the primary success factor, then global indexes provide an edge over local ones and global indexes may need to be used.

Tip 11: Global indexes can provide faster lookups to non-unique data than local, non-prefixed indexes.

Parallel Index Scans

Under Oracle7, table accesses would be parallelized only if the operation was a full-table scan; a single b*tree could not be scanned in parallel. With Oracle8, local indexes are represented as multiple b*trees, and even though a single scan cannot be parallelized, parallelism can be achieved because more than one b*trees constitutes the index.

UNION VIEWS

Application requirements often drive the initial database design. The reasons for building UNION VIEWS under Oracle7 are the same when partitioning a table in Oracle8. Time-based partitioning strategies also remain the same in Oracle8, except for the number of partitions. In Oracle8, the number of partitions is based on maintenance, availability and the performance requirements of the application. Also, since Oracle8 partitions are part of a single table, the restricted number of tables found in an Oracle7 UNION VIEW (due to the maximum length of a SQL statement) is not an issue.

Tip 12 : The maximum number of tables in a UNION VIEW due to the SQL statement length does not exist for Oracle8 partitions.

GLOBAL INDEXES AND PDML

Maximize the use of local indexes for maintainability and global indexes for performance. If global indexes are needed on a table, beware of the following:

1. Global indexes will need to be set UNUSABLE prior to a parallel insert, and rebuilt following it. A parallel insert cannot be performed into a global index.
2. Global unique indexes will need to be set UNUSABLE prior to parallel update operations, and rebuilt following them. A parallel update cannot be performed on a global unique index.
3. If a table partition needs to be recovered and a global index exists on the table, the global index will need to be rebuilt after the recovery because the index must be consistent with all table partitions in the table.

PARTITIONING FOR ORACLE PARALLEL SERVER PERFORMANCE

Application Partitioning and Table Partitioning

When designing a non-READ-ONLY application to run on Oracle Parallel Server, the design should attempt to achieve some level of partitioning—application, departmental/user, or transactional partitioning. The objective is to distribute the workload across the instances in the configuration while minimizing the amount of pinging that occurs. In 90 percent of the cases, pinging will occur due to processes on different instances wanting access to the same tables. Standard methods are available to reduce the amount of pinging that occurs in these scenarios.

The pinging can be removed by partitioning intersecting tables using the same method that distributed the application across the Oracle Parallel Server configuration. For example, an application that handles insurance claims can process personal liability claims on instance 1 and worker's compensation claims on instance 2. Both of these areas insert/update records in the CLAIMS tables, and can be a potential area for pinging. By partitioning the CLAIMS table into 20 partitions—10 for personal liability and 10 for worker's compensation—the potential for pinging is removed. The table partitioning strategy separates the worker's compensation data from the personal liability data corresponding to the load partitioning strategy.

Partition to Instance Mapping

When a PDML statement is executed across an Oracle Parallel Server configuration in an MPP environment, the parallel slaves are allocated to the instances based on the locality of the table's datafiles. Designing a database in an MPP environment that needs to support PDML must take the distribution of parallel slaves into account. A large table partitioned 100-way that will be the subject of a large PDML operation should be distributed across the MPP nodes to utilize all CPUs. For example, on a 10 node MPP, localize 10 partitions per node. The datafiles that make up this partition also should be local to that node.

Read-Only Data

Most tables are a mixture of read-only records and read-write records. In an Oracle Parallel Server environment, read-only tables require a small number of PCM locks allocated to their underlying datafiles. Separate the read-only data from the read-write data into different partitions with its own tablespace. The read-only tablespaces then can be set to READ-ONLY and be allocated a small number of PCM locks, while the read/write data can be assigned the bulk of the PCM locks. This method will reduce the total number of PCM locks allocated in the system and therefore provide memory savings.

SUMMARY

Designing a very large database for availability and manageability differs slightly from designing it for performance. The number, size, type, and autonomy of the table and index partitions need to be considered based on the customer requirements. Compromises may need to be made in situations where availability, performance and manageability are of equally high priority. Correctly designing a VLDB will ensure that the Oracle8 functionality is used optimally.

ORACLE®

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.415.506.7000
Fax +1.415.506.7200
http://www.oracle.com/