Improving Database Performance with Oracle8™

An Oracle Technical White Paper

June 1997

ORACLE®

Enabling the Information Age™

Improving Database Performance with Oracle8

INTRODUCTION

Oracle8 should improve performance in all types of applications. Oracle8 offers many new features and optimizations to improve database and application performance. Applications will benefit from many of these improvements simply by upgrading to Oracle8, with no application code or database structure changes. In other cases, a change to a data structure or to an application may be required. Much of the focus of the performance optimizations has been for support of very large databases, mission-critical Online Transaction Processing (OLTP) systems, replication, and internal optimizations.

Oracle8 provides many performance benefits to Oracle users through new optimizations, new database storage structures, and more parallel operations. This document covers performance improvements in the following application and functional areas.

Data Warehouse Applications

- Improved star-query processing
- Parallel Data Manipulation Language (DML) operations for faster bulk data operations (insert, update, and delete)
- Parallel index scans
- Improved performance and functionality of constraints

OLTP Applications

- Advanced Queuing for application scalability and performance
- Better utilization of resources

Very Large Database (VLDB) Support

- Partitioning of tables and indexes
- Index-organized tables

Oracle® Parallel Server (OPS) Configurations

- Controlled pinging
- Parallel generation of system change numbers (SCN) numbers
- Performance gains with improved disk affinity
- Global dynamic views for easier performance tuning
- Reverse key indexes to minimize pinging

Oracle8 Advanced Replication

- Internalized triggers
- Minimization of data propagation
- Parallel propagation of changes

Other Performance Improvements

- LOB support
- NCHAR datatype
- Internal optimizations
- Oracle Call Interface™ (OCI)
- PL/SQL™ performance improvements

DATA WAREHOUSE APPLICATIONS

Oracle8 has many enhancements that improve performance in a data warehouse or decision support system (DSS). These improvements include improvements to the optimizer, new functionality to simplify the management and performance of large databases, and more operations that can run in parallel.

Improved Star Query Optimization

Oracle8 offers a new algorithm that dramatically improves star-query performance. Oracle8 introduces significant performance improvements to the processing of star queries, which are common in data warehouse applications. A star query, or star schema, occurs when there is one or more very large tables, often called a fact table, with relationships to multiple smaller, or dimension, tables. Oracle7™ introduced the functionality of star-query optimization, which provides excellent performance for these types of queries. In Oracle8, star- query processing is further improved.

In Oracle8, an innovative new method for executing star queries has been introduced. Using both new features inherent in the Oracle8 server and also a more efficient algorithm, Oracle8 star-query processing provides a significant performance boost to data warehouse applications.

Several specific types of star queries have been optimized in Oracle8, including star schemas with “sparse” fact tables, one in which the criteria eliminates a great number of the fact table rows. Also, when a schema has multiple fact tables, the optimizer efficiently processes the query. Finally, Oracle8 can efficiently process star queries with large or many dimension tables, unconstrained dimension tables, and dimension tables that have a “snowflake” design schema.

Star queries do not require the use of a Cartesian product to optimize the star joins. The new algorithm, unlike the algorithm utilized in Oracle7 star-query optimization, does not perform a Cartesian product. The new algorithm actually simplifies the task of the optimizer, since it reduces the need for considering join orderings with Cartesian products. This allows for better optimizations of more complex star queries, such as one with multiple fact tables.

Oracle8 processes a star query in two basic phases. First, it retrieves exactly the necessary rows from the fact table. This retrieval is done via bitmapped indexes on the foreign key columns and is very efficient. The second phase joins this result set from the fact table to the relevant dimension tables.

The Oracle8 star-query optimization also leverages new features available with Oracle8. The new algorithm is completely parallelized, including parallel index scans on both partitioned and non-partitioned tables. The new algorithm also uses bitmapped indexes for efficient data access.

It is easy and straightforward to implement this new feature. Instead of using B-tree indexes, bitmapped indexes are created on the foreign key columns of the fact tables. By using single column bitmapped indexes, significant storage savings is achieved over the Oracle7 star-query optimization, where concatenated column B-tree indexes were required. Bitmapped indexes also provide the needed functionality to support efficient processing of star queries with sparse fact tables, because rows matching the query criteria can be quickly located and processed.

The Oracle7 star-query optimization method remains unchanged and is available in Oracle8. This allows for an easy migration without changing application code or index structures. Although Oracle8 star-query optimization is superior to the Oracle7 method, the Oracle7 method is still an appropriate technique and many users will choose to continue to use this technique. In cases where there are very dense fact tables and small dimension tables, the Oracle7 star-query optimization may perform better.

Parallel DML

Oracle8 supports the execution of insert, update, and delete operations in parallel. This operation, known as parallel DML, is executed in parallel across multiple processes and is completed much more quickly than if the same operation was executed in a serial fashion. Parallel DML complements parallel query by providing parallel execution of both queries and updates.

Parallel DML is useful in a decision support system (DSS) or data warehouse environment where DML operations are common. However, parallel DML operations can also speed up batch jobs running in an OLTP database.

Oracle8 supports parallel insert, updates, and deletes into partitioned tables. Oracle8 also supports parallel inserts into non-partitioned tables.

In partitioned tables, Oracle8 supports parallelism of DML operations across partitions. For example, if a table has four data partitions, the maximum degree of parallelism will be four, and the DML operation will be parallelized across each of the four partitions.

A parallel slave process can update or delete from multiple partitions, but each partition can only be updated or deleted by one slave process. If the degree of parallelism is less than the number of partitions, then the first slave process to finish work on one partition continues working on another partition, and so on, until the work is finished on all partitions. If the degree of parallelism is greater than the number of partitions involved in the operation, then the excess slave processes have no work to do.

The parallel insert operation is similar to the direct path load operation that is available in Oracle. The parallel insert into a table improves performance by formatting and writing disk blocks directly into the datafiles, bypassing the buffer cache and space management bottlenecks. In this case, each scan process of the insert query inserts data into a segment above the high watermark of the table. When all of the insert processes are complete, the transaction commits and the high watermark is moved beyond the new segments.

The degree of parallelism of an insert into a non-partitioned table is equal to the parallelism of the query scan. The degree of parallelism of an insert into a partitioned table is equal to or less than the number of partitions into which it is inserted.

Oracle8 provides several variations of serial or parallel INSERT with respect to syntax, logging, and implementation. A major benefit of parallel INSERT is that you can load data without logging redo or undo entries, which improves the insert performance significantly.

To use this functionality, parallel DML must be enabled prior to the execution of the insert, update, and delete operation. Normally, parallel DML operations are done in batch programs or within an application that executes a bulk insert, update, or delete.

Parallel Index Scans

Parallel index scans improve query performance in both partitioned and non-partitioned tables. Oracle8 can dramatically improve query performance because parallel index scans can be done in both partitioned and non-partitioned indexes. In a non-partitioned index, a parallel index scan is done when the query plan indicates a fast full-index scan is needed. A fast full-index scan is when a full table scan is required, but can be satisfied with data in the index. In this case, the index scan is performed in parallel.

In a partitioned index, an index scan is done in parallel, parallelized across partitions. In this case, one query slave is assigned to scan each partition of an index, so range scans, complete scans, and individual index probes are all performed in parallel on partitioned indexes.

Improved Performance and Availability when Enabling Constraints

Constraints can be enabled with a new option, NOVALIDATE. Oracle8 introduces a new option for the ENABLE CONSTRAINT command, which is useful in a data warehouse environment where large amounts of data are loaded periodically. This command, used to enable a table constraint, can now be used with an optional parameter NOVALIDATE. When the ENABLE command is used with this parameter, the existing data in the table is not checked against the constraint, but all subsequent inserts and deletes, and some updates, are checked against the constraint.

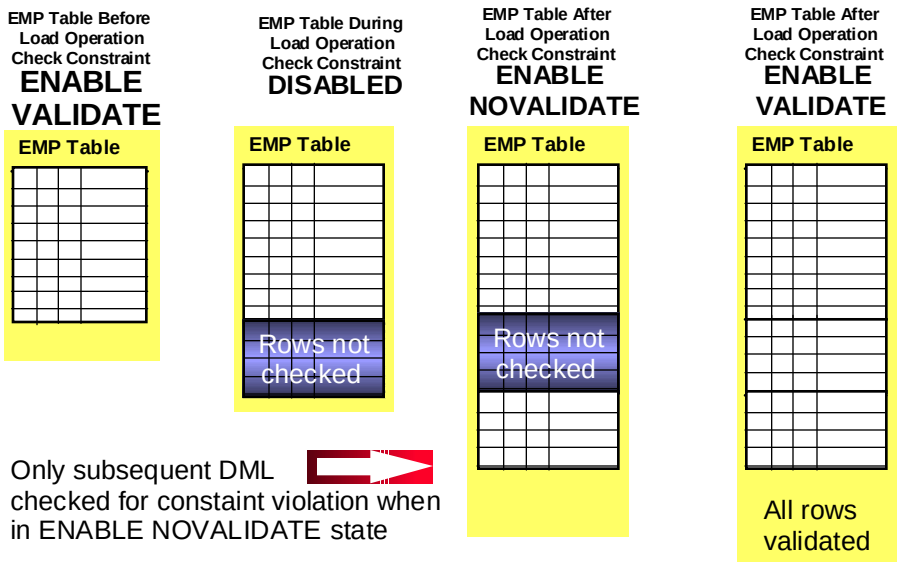

Figure 1: Enabling constraints in Oracle8

The benefits of using this new command are two-fold. One benefit is that the constraint can be turned on nearly instantaneously, without the server having to check all the existing records for constraint violations. The second benefit is that the table is not locked at anytime during the ENABLE NOVALIDATE operation or when a table constraint is fully enabled after being in the ENABLE NOVALIDATE state.

Table constraints can now be fully enabled using a two-step process. The first step is when the NOVALIDATE parameter is used. After enabling a constraint with the NOVALIDATE option, the server only checks constraints on subsequent inserts and deletes. Updates are checked when the update is to the column that is germane to the constraint. The second step is when a constraint is enabled without the NOVALIDATE parameter. This operation will check all rows for constraint integrity.

Enabling a constraint in two steps, as in Oracle8, offers two additional benefits as compared to using the disabled-to-fully-enabled process, as in Oracle7. If the additional step of ENABLE NOVALIDATE is used, the constraint can be enabled very quickly, while providing integrity checks during subsequent table DML. Also, the performance of enabling multiple constraints can be significantly faster when they are enabled concurrently, which is possible with this two-step process.

The new option of NOVALIDATE on the enable constraint command allows for higher availability, less disruption, and better performance when table data is checked for constraint violations.

OLTP APPLICATIONS

Oracle8 improves scalability of applications, connections, and hardware. Oracle8 provides dramatic improvements for OLTP systems in the area of scalability—scalability of applications, scalability of connections, and scalability of hardware and transactions.

- Advanced Queuing provides the foundation for scalability of applications, even under high transaction workloads. Advanced Queuing is a mechanism to defer work to a later time, such as at night.
- To address scalability of connections and users, Oracle8 and Net8™ have been improved such that network resources are used much more efficiently, thus allowing a larger number of users to connect to a single Oracle8 instance.

- In order to scale on all hardware systems, Oracle8 has major improvements when applications are running in an Oracle Parallel Server or massively parallel processing (MPP) environment. These improvements are discussed later in this document. Also, Oracle8 offers tighter integration with transaction processing (TP) monitors which provide high performance in high-transaction OLTP applications.

Oracle8 provides many internal optimizations that improve performance in high-transaction OLTP applications. Many of these features are transparent to both DBAs and applications, so the performance benefits are gained immediately when Oracle8 is implemented. These internal optimizations are explained later in this document.

Advanced Queuing

Queuing defers work until a later time. Advanced queuing adds support in the database for deferring work related to a transaction to a later time and in a particular order. The use of queues can significantly improve the performance of an application by deferring work, such that the work is not performed within the foreground transaction.

Oracle8 provides functionality for application developers to insert information or a message into a queue, and to have another application take this message out of the queue by any number of priorities.

Oracle8 provides the queue structure, known as queue tables, enqueuing and dequeuing functions, and administrative tools to manage and install queues.

An example of the use of Advanced Queuing is in order-entry and shipping applications. Instead of an application doing both the booking and shipping of an order, Advanced Queuing can be used such that the order-entry application books an order, but only adds an entry into the queue of the shipping application. Since less work is done at order-entry time, the order-entry application runs faster. The remaining work is deferred until a later time, when orders are taken out of the shipping queue for further processing. The shipping application, can dequeue orders from the queue by a given priority, such as first-in/first-out, by customer, or by order amount.

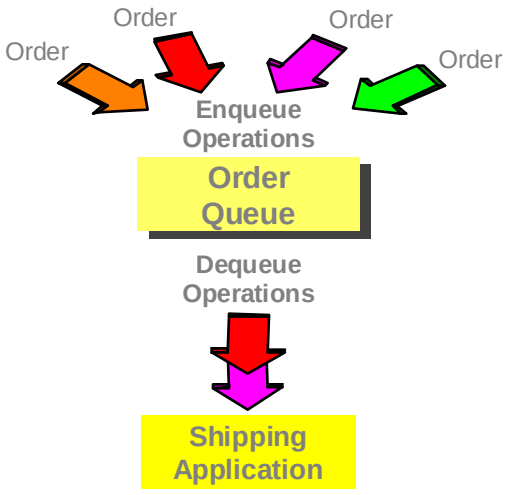

Figure 2: Queuing decouples distributed transactions

To take advantage of Advanced Queuing, developers must design or modify applications so that information is put into and taken out of queue tables and processed. Advanced Queuing provides the foundation and functionality to implement a solution, but application code must exist to initiate the enqueue and dequeue operations and any subsequent processing.

VERY LARGE DATABASE SUPPORT

Very large databases are supported with data partitioning and index-organized tables. Oracle8 introduces significant new features that give DBAs choices on how to physically store data and indexes to improve performance. One major new feature is the introduction of partitioning of data with the Oracle8 Partitioning Option. Partitioning improves availability, management, and the performance of both OLTP and data warehouse applications.

In addition to partitioning, Oracle8 introduces a new type of table, called an index-organized table. This new type of table offers savings in storage and improves query processing performance.

Partitioning Tables and Indexes

Partitions improve performance, availability, and manageability. As tables have grown in size, the challenges facing DBAs to administer and maintain good performance have also grown. A table or index can be divided into several smaller pieces, called partitions, where each partition contains a subset of the data. Oracle8 introduces partitioning to meet these challenges by providing better availability, better performance, and easier administration of these large tables. By dividing a large table or index into multiple partitions, both OLTP and data warehouse applications gain significant benefits.

When creating a partitioned object, the DBA selects a partition key and a key range for the table or index. This key and key range determines in which partition an inserted record is physically stored. Once the partitioning strategy is implemented via the create table command, the server manages the partition strategy. Partitioning is transparent to both applications and users, so a table can be converted from a non-partitioned structure to a partitioned structure without changing application code or re-training users.

A typical example of a partition strategy would be partitioning sales data by quarter of the year, with partitions being Q1, Q2, Q3, and Q4. In this case, the partition key is the sales date column, and the range of the Q1 partition is January 1st to March 31st of a given year. By partitioning the sales data by quarter, each quarter can be managed independently of the others. Another example would be partitioning data by region of the world. In this case, the administrator can load the Asia partition with data without impacting the performance of the Europe partition.

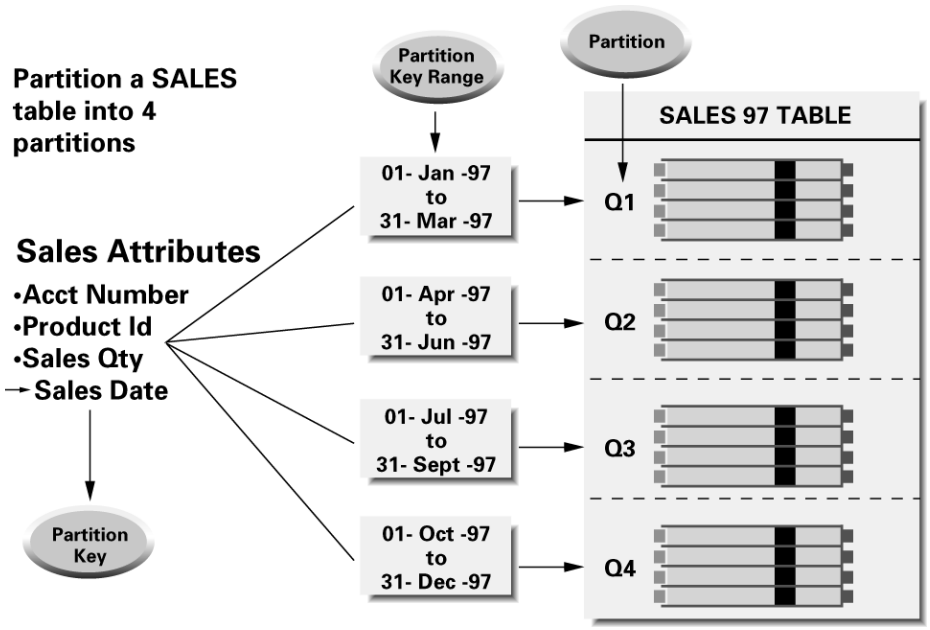

Figure 3: Oracle8 Partitions Tables by Key Values

Choosing a partitioning strategy is another database design decision to maximize the benefits of availability, manageability, and performance of the particular application.

This paper addresses the performance benefits of partitioning. Another white paper, "Taking Advantage of Oracle8 Partitioning," discusses partitioning in detail, including the benefits of availability and easier management of large tables and indexes.

Partitioning for Performance

Partitioning provides load balancing and better query optimization. One performance benefit of partitioning is that the administrator has control over where a specific subset of records can be placed. This improves performance by allowing the administrator to distribute the data across several devices, also known as striping data.

In previous versions of Oracle's universal data server, the administrator could strip a table, but had no control over what records were placed on which device. The advantage of partitioning is that the administrator knows what data is located in a particular partition. Because the administrator can determine the device on which a specific partition resides on, frequently accessed or critical data can be located on faster disk drives. To assist in managing and moving partitions, the SQL command MOVE PARTITION is available.

Partitioning provides efficient query optimization through partition elimination. The Oracle8 optimizer is partition-aware and uses this information to create efficient query plans when accessing data in a partitioned table or index. Since the Oracle universal data server knows the partition strategy of an object, it knows what partition to "look" into if the query criteria contains the partition key or a subset of it. If a partition can not possibly contain any of the rows a query requires, that partition is eliminated from the search. This is known as partition elimination.

The query optimizer is able to eliminate unnecessary partitions from the query execution plan. For example, if a query is issued to retrieve the total number of orders from January 1 to February 5, the optimizer eliminates all partitions, except Q1.

Partitions Improve Join Performance

Partitioning can also help the performance of certain join operations. If the application contains joins of several tables on a common key, and it selects predictable subsets of the data, partitioning can speed up the operation. For example, an application may have the personnel schema illustrated in Figure 4.

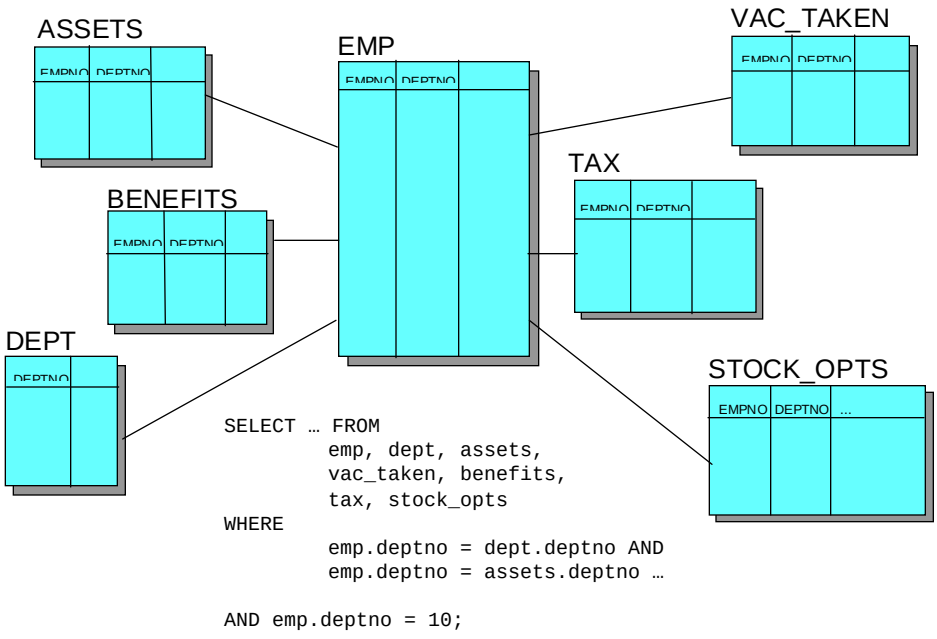

Figure 4: A Join Operation without Partitioning

The query in the figure above requires the information from seven relatively small tables to be joined together. The execution plan for the query may require a full table scan for all of the tables prior to the join operations, depending on the size of the tables. Even if the tables are small, the query can be sped up by partitioning the data on the join condition (in our example, DEPTNO).

Choosing the foreign key as the partition key can improve join performance. In the case of queries that always search several tables for a common set of column values, partitioning can improve performance. In Figure 4, a human resources application always selects employee information for an entire department. In Figure 5, it illustrates how partitioning can speed up the execution of the query by eliminating the unneeded partitions and performing a scan of only the requested partition in each table.

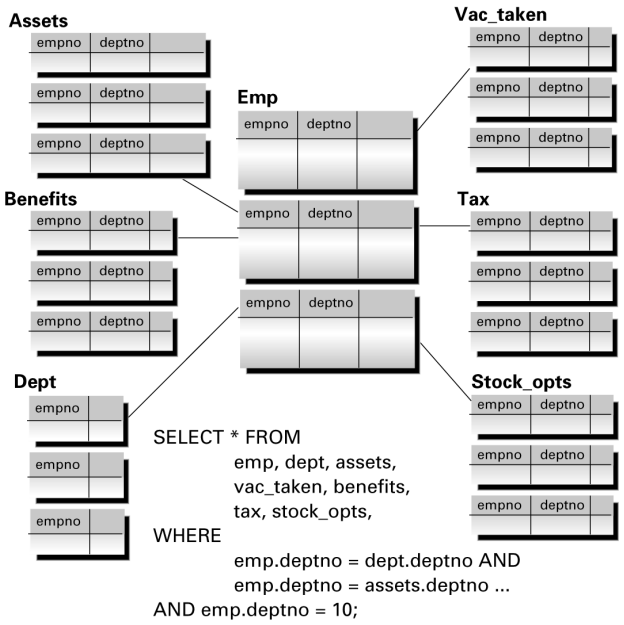

Figure 5: Partitioning on the Same Key Improves Join Performance

The statement in Figure 5 requires a scan of only one partition for each of the tables in the seven-way join. This eliminates most of the data in each of the tables and dramatically improves the performance of the statement.

Index-Organized Tables

Index-organized tables are useful in data warehouse and information retrieval applications. Oracle8 introduces a new type of table to improve performance and reduce storage in certain data design implementations, where data is only accessed in one way. This new type of table, the index-organized table, is useful in information retrieval applications (e.g., ConText® Cartridge) and in some data warehouse applications. An index-organized table is structurally similar to a B-tree index, but all table data is stored in the B-tree structure.

In a B-tree index, the table data is stored in a database object, and the index data, the B-tree, is stored in another database object. The index data object includes the indexed column values and a ROWID pointing into the data table. In other words, the indexed values are stored twice, once in the table, and once in index.

In an index-organized table, only one database object is used to store both the data and the index. In this case, everything is stored in a B-tree index structure—there is no database object used to only store the data. The B-tree index structure holds both the indexed column values and the non-indexed column values in the index entries. No ROWIDs are needed or stored since ROWIDs are used to point from an index to a data table row, which is unnecessary in this type of structure.

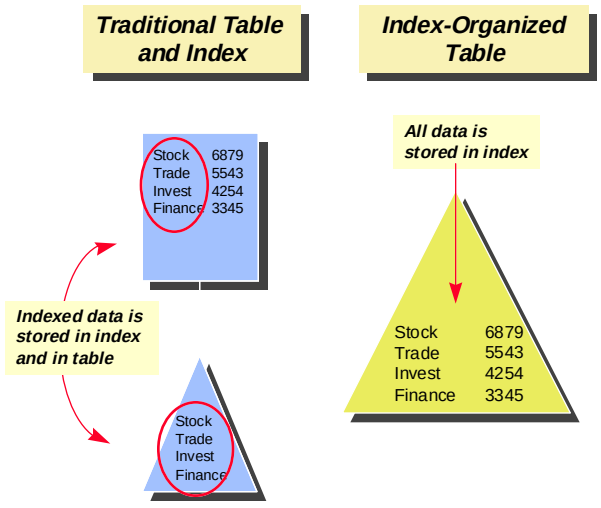

Figure 6: Index Organized Tables Illuminate Redundant Storage

To take advantage of this feature, a table must be created as a index-organized table. Or an existing table must be re-created. However, when a table is created as an index-organized table, it is transparent to applications. Applications manipulate the index-organized table just like a regular table, using standard SQL statements for query, insert, update, or delete operations. Some disadvantages exist when implementing index-organized tables, such as no secondary indexes can be created on the table. Also, in some cases, an updated row may force the entire row to be moved. Two advantages of an index-organized are better performance and less storage.

Because all data, key columns, and the remaining columns are stored in the B-tree structure, index-organized tables provide a faster, key-based access to table data for queries involving exact match and/or range search. Once the search has located the key values, the remaining data is present at that location, so there is no need to follow a ROWID back to the table data, as would be the case with a table and B-tree index structure. This eliminates one I/O, namely the read of the table.

The storage requirements are reduced as key columns are not duplicated in the table and the index. Also, since ROWIDs are not necessary, this storage amount is saved for each row.

ORACLE PARALLEL SERVER CONFIGURATIONS

Oracle Parallel Server performance is improved dramatically in high transaction applications. Oracle8 provides additional capabilities and optimizations that dramatically improve the performance of *Oracle Parallel Server*. Performance is particularly improved by using partitioned tables and indexes and through the introduction of reverse key indexes, which reduces block contention, and pinging.

Controlled Pinging

Oracle8 reduces pinging by improving the process in which an owning node gives a data block over to a node that has requested it. The owning node is given priority over the requesting node to allow it to possibly complete the transaction involving the block. This avoids a situation where control of a data block is passed back and forth between nodes, or pinged back and forth. By giving the owning node priority, it has the opportunity to complete its entire transaction involving the block, thus relinquishing it to the requesting node without having to request it again.

Parallel Generation of SCN Numbers

In Oracle8, *Oracle Parallel Server* introduces the functionality of parallel generation of SCN. All transactions in Oracle require an SCN to be generated. In an Oracle Parallel Server environment, SCNs are generated in parallel on all instances, which avoids bottlenecks. This significantly and transparently improves Oracle Parallel Server performance in high commit-rate applications.

Performance Gains with Improved Disk Affinity

Partition-to-node mapping reduces pinging, which improves performance. Oracle Parallel Server can leverage the new partitioning functionality of the Oracle8 Partitioning Option. Disk affinity, and therefore performance, is further enhanced when a partition-to-node affinity is configured. Partition-to-node affinity information determines slave allocation and work assignments for each node in the clustered environment. This results in a fairly deterministic and very efficient assignment of partitions to Oracle instances.

A typical example of using disk affinity in conjunction with Oracle Parallel Server and partitioning is when the data in a table is partitioned by department and each department's node is assigned its corresponding partition. This reduces pinging between nodes and improved buffer cache hit ratios, resulting in better overall performance.

Global Dynamic Performance Views

Global dynamic views make it easier to tune a clustered or MPP configuration. Global dynamic performance views are introduced in Oracle8 to make it easier to tune the performance of an Oracle Parallel Server environment.

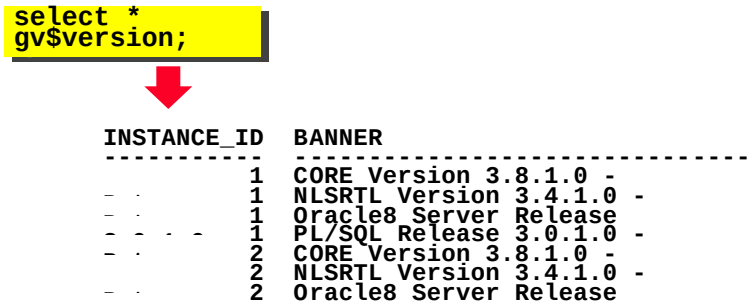

Figure 7: Improved Management and Monitoring of Oracle Parallel Server with Global Dynamic Performance Views

With global dynamic views, the DBA can look at all statistics in the Oracle Parallel Server system environment from a single instance session. These views extend the current administrative and tuning mechanism provided with the V$ views. With the addition of global dynamic views, known as GV$ views, it is unnecessary for the DBA to log on and query each node for status information separately. This feature is particularly useful in tuning and administering cluster and massively parallel hardware systems.

Reverse Key Indexes Minimize Pinging

Reverse key indexes reduce hot spots in a clustered environment. Reverse key indexes are new a type of index available in Oracle8 that dramatically improves performance in Oracle Parallel Server configurations that have high transactions rates. This type of index also has advantages in non-Oracle Parallel Server environments, such as in a data warehouse application, where data is loaded and deleted periodically.

A reverse key index is architecturally the same as the traditional B-tree index, except that when the indexed columns are indexed, the bytes of column data values are reversed. By reversing the indexed values, sequential index entries are evenly distributed in the B-tree structure. This results in performance gains in high insert rate applications because it avoids hot spots when inserting index values that are sequential.

A reverse key index is especially useful in an Oracle Parallel Server environment, when multiple nodes may compete for an Oracle block that contains the index entries of a sequentially generated identifier. In this case, performance can be adversely affected by block pinging as two instances encounter contention while inserting sequential keys into the same index block repeatedly. By reversing the bytes of the index keys, the sequential index values are not put into the same Oracle block, thus minimizing pinging when index maintenance is done.

Reverse key indexes avoid sliding indexes. This type of index also has advantages in a non-Oracle Parallel Server environment. A reverse key index can be implemented to avoid a 'sliding' index, which occurs when data is inserted or deleted by a range of an indexed value—for example, if an application deletes last year's January sales data while loading this year's January sales data, where the index is on sales date. In a normal B-tree index, the index would essentially be sliding due to the B-tree structure and the indexed values. A reverse key index would maintain an evenly balanced and level B-tree index, because the entries are evenly spread out over B-tree since the index column values are reversed.

To take advantage of a reverse key index, the index must be rebuilt as such. Applications need not change to get the performance improvements.

ORACLE8 ADVANCED REPLICATION

Oracle8 Advanced Replication includes performance improvements related to the way changes are captured and propagated. These improvements include:

- Internal triggers to capture the changes
- Minimizing the amount of data that must be sent when propagating a transaction
- Using multiple sessions to propagate deferred transactions in parallel while preserving transactional integrity

Internalized Triggers

Internal triggers are easy to maintain and execute efficiently. Performance in replication is improved through tighter integration between the replication code and Oracle8 server. In Oracle8, the triggers used to capture replicated changes are written in C and compiled directly into the kernel. Previously, PL/SQL triggers were generated for each replicated table. The Oracle8 internal triggers run faster, have more efficient access to data, and do not have the size limitations of PL/SQL triggers. Internal triggers have the advantage of being both fast and efficient, while being virtually maintenance free. There is no need to install and maintain any external triggers. The internal trigger for a table is automatically activated when you configure a replicated table, and never needs generation.

Minimizing Data Propagation

Oracle8 Advanced Replication can minimize the amount of data that is propagated to remote sites. Once a transaction is captured and stored in the local queue, it must be propagated and applied at each remote location. Considering network transmission time and the time to apply the change, the less data you send and the faster you send it, the better your overall replication performance.

With Oracle8 Advanced Replication, the administrator can choose to send only the minimum amount of data necessary to successfully apply a change. Only the following data items are needed to be propagated to each remote site:

- Values that uniquely identify a row being updated
- The new column values for each row
- The column values needed to detect and resolve conflicts

By minimizing the amount of data being propagated and applied, significant performance gains may be achieved, while also minimizing network traffic to each remote site.

Parallel Propagation of Changes

Multiple streams can be used to propagate changes to remote sites. After capturing and queuing the data to replicate a transaction, this data must be propagated to each remote location. Oracle8 Advanced Replication maximizes throughput by using multiple sessions to parallelize the transaction stream to each remote site. By having multiple parallel streams transmit data, instead of just one serial stream, dramatic performance gains are achieved.

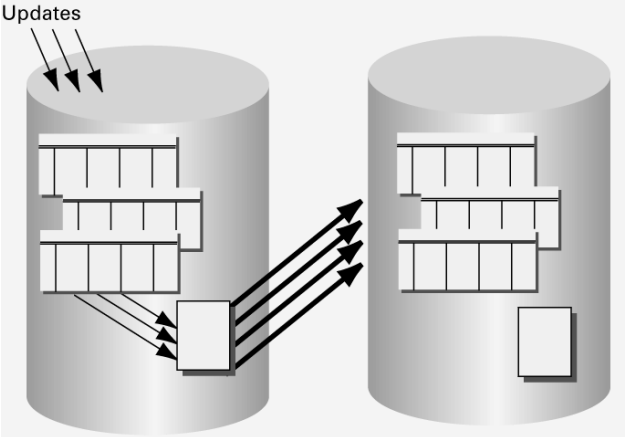

Figure 8: Parallel Propagation Dramatically Improves Performance by Maximizing Throughput

Oracle8 Advanced Replication automatically parallelizes transaction streams to each site, while preserving transaction integrity by identifying transaction dependencies. Oracle detects and tracks dependencies between transactions to ensure proper ordering when necessary. Because Oracle proactively manages the dependencies *before* propagating the transactions, you consistently receive superior performance without sacrificing data integrity.

Other improvements to Oracle8 Advanced Replication also result in performance gains. The number of round-trips needed to propagate transactions is greatly reduced by improving the underlying data propagation mechanism and by using a more efficient protocol for resolving errors. Snapshot refresh performance is also improved by reducing roundtrips when pulling down changes, as well as by taking advantage of parallel propagation when pushing changes back to the master.

Many Oracle8 Advanced Replication enhancements are transparent and need very little or no setup. Oracle8 Advanced Replication includes numerous enhancements designed to improve the performance of a replication environment. Most of the enhancements, such as internal triggers and reduced data propagation, are transparent to the end user and administrator; nothing needs to be done to the application, it just performs better. Others, such as parallel propagation, merely require the administrator to set appropriate initialization parameters to realize significant gains in performance.

OTHER PERFORMANCE IMPROVEMENTS

Oracle8 has internal improvements and new datatypes that improvement performance. Oracle8 has many internal optimizations and several new datatypes that improve performance and resource consumption within the Oracle database environment. Many of these optimizations are transparent to applications and users, so performance benefits are gained immediately once Oracle8 is implemented. The new datatypes can be used when existing tables are created or altered under Oracle8.

Large Object (LOB) Support

Large objects can be stored inside or outside the database. Oracle8 introduces significant new functionality and performance enhancements when applications store and manage large objects. In previous versions of Oracle's universal data server and also in Oracle8, the LONG datatype was used to store large binary objects. Now, in addition to the LONG datatype, Oracle8 introduces four new datatypes used to store large objects. They are:

- BLOB for binary large objects
- CLOB for character large objects
- NCLOB for character large objects stored in the national character; analogous to NCHAR
- BFILE for binary files stored outside of the Oracle's universal data server

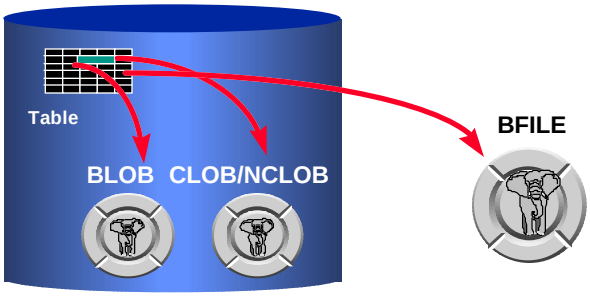

Figure 9: Type of LOBs

Oracle8 offers the functionality to store a LOB either inside or outside the database. The datatype BFILE stores a locator within the Oracle8 table that points to an operating system file. By storing a LOB outside of Oracle8, performance is enhanced since it is not subject to the backup and/or recovery processing as Oracle8 managed data. Yet, the file is still accessible and can be queried by Oracle users through new functions.

One of the key features of LOBs is that random, piecewise access to data is available through the database. This allows an application to access fixed-size pieces of a LOB without accessing the entire LOB. Also, a LOB can be accessed by offset, which allows an application to quickly access a piece without having to scan through from the beginning.

Another key feature of LOBs is that they can be stored in a tablespace that is different than the tablespace used to store the other table data. This improves performance by allowing the data administrator to place large objects on a device best suited for that type of data.

To take advantage of the performance improvements and new functionality that Oracle8 offers for large objects, one of the four new datatypes must be used. LONG datatypes are still available in Oracle8, but they do not offer the benefits or functionality described above.

NCHAR Datatype

A new NCHAR datatype allows dual character sets in one database. This improves performance and storage predictability for some Asian language multibyte character-set databases.

Internal Optimizations

Oracle8 requires less memory. A number of critical areas were addressed in Oracle8 to reduce memory consumption. The memory consumption improvements are focused on per-user memory usage. A number of memory savings enhancements are implemented in Oracle8, and many of the most significant savings require no changes to applications.

Oracle8 optimizes CPU resources. Most of the improvements in Oracle8 were to improve CPU usage areas transparent to applications. A more consolidated dictionary cache reduces latch contention. This is especially useful for applications written using third-party tools that do not use bind-variables and have poor shared-cursor utilization.

Performance of array inserts is optimized. In previous versions of Oracle's universal data server, a batch of rows could be sent to the server in one round-trip, and the server would do an array insert into the table, which is very efficient. Oracle8 improves the performance of such a transaction by also executing an array insert into the index. Array index inserts in Oracle8 make an array insert into a table and a corresponding index up to 20 percent faster.

Full table scans performance improved. Optimizations to the code that reads table blocks also results in up to 10 percent better performance for a full table scan.

Oracle Call Interface™

Oracle8 includes significant performance and functionality enhancements to Oracle Call Interface. One of the most encompassing and major enhancements available in Oracle8 is in the Oracle Call Interface. One change is a shift from the cursor-based, connection-oriented model to a message-based connection-less one. Every Oracle Call Interface call operates on a handle instead of a cursor.

With Oracle Call Interface, more work is bundled into a single call, reducing required round-trips. Oracle Call Interface also allows for transparent pre-fetch of selected data. This results in better performance since round-trips are reduced. Round-trips are also reduced because Oracle Call Interface enables message "piggy-backing." In this case, no network round-trips are necessary for commit, open, close, cancel, and other operations.

Initially, developers must code using the new Oracle Call Interface to gain the benefits described above. Oracle products that use Oracle Call Interface are being upgraded to leverage the new functionality and performance gains in the new Oracle Call Interface, but no release dates are available.

PL/SQL™ Performance Improvements

Many applications will realize better PL/SQL performance transparently. Significant portions of the PL/SQL runtime engine have been rewritten in Oracle8 to improve PL/SQL performance. Most of the optimizations are available to applications just by re-compiling the PL/SQL under Oracle8.

The performance of calling PL/SQL from SQL, such as when a PL/SQL function is called within a SQL statement, has been dramatically improved. Also, execution of dynamic SQL from PL/SQL using DBMS_SQL has been optimized. Using the DBMS_SQL interface for executing SQL from PL/SQL is nearly as fast as executing pre-compiled static SQL. This also allows for an array interface, such as array inserts, from PL/SQL.

SUMMARY

Through many internal optimizations, a variety of new database storage structures, and new server functionality, Oracle8 provides a solid foundation for improved database performance. Although many of the features are targeted for large database support or high transaction applications, all database systems will see performance benefits when Oracle8 is implemented. Many significant improvements can be implemented with very little or no database or application changes, making it very beneficial and easy to migrate to the Oracle8 server.

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/

Copyright © Oracle Corporation 1997
All Rights Reserved

This document is provided for informational purposes only, and the information herein is subject to change without notice. Please report any errors herein to Oracle Corporation. Oracle Corporation does not provide any warranties covering and specifically disclaims any liability in connection with this document.

Oracle and ConText are registered trademarks, and Enabling the Information Age, Oracle7, Oracle8, Oracle Call Interface, PL/SQL, and Net8 are trademarks of Oracle Corporation.

All other company and product names mentioned are used for identification purposes only and may be trademarks of their respective owners.