An Oracle White Paper
May 2010

Guide for Developing High-Performance Database Applications

ORACLE

Introduction

The Oracle database has been engineered to provide very high performance and scale to thousands of users. However, a database application should be properly designed to take advantage of the database server capabilities. This white paper discusses fundamental rules and best practices for developing high-performance database applications

Each database feature has its pros and cons and only thorough understanding of how features work allows developers to make right choices. The purpose of this document is to help application developers focus on the most important performance optimization techniques available with the Oracle database server. This guide is in no way complete and not intended to be used without consulting Oracle documentation and experienced application developers.

Many of the tips listed in this document not only improve application performance, but also have impact on application security, availability and reliability; code readability and maintainability; memory and disk space usage. For simplicity, we did not describe these aspects of the tips; however, a developer should consider all of them to balance different requirements for the application.

The tips do not include performance optimization techniques that are usually used by DBAs and can be applied to any application, such as setting initialization parameters, optimizer tuning, space management, parallel operations, partitioning, RAC, etc. However, it is beneficial for an application developer to be familiar with these techniques as well and use them when working with DBAs on performance tuning.

This guide was created for Oracle Database 11g Release 2. Some of the tips might not be applicable or produce a different effect if used with earlier releases.

APPLICATION DESIGN PRINCIPLES

This section lists some basic rules developers should adhere to when designing database applications. A failure to follow the rules will result in a poorly performing application.

Reuse database connections

Establishing a database connection is an expensive operation. Applications should avoid continually creating and releasing database connections. In a simple client-server application, a connection should be created at application initialization and used for all transactions. In a Web-based or multi-tiered application where application servers multiplex user connections, connection pooling should be used to ensure that database connections are not reestablished for each request.

When using connection pooling always set a connection wait timeout to prevent frequent attempts to reconnect to the database if there are no connections available in the pool. Consider using Transparent Application Failover (TAF) for retrying connections because it can also provide your application with connect-time failover for high-availability and client load balancing.

Minimize SQL statement parsing

SQL statement parsing is a CPU-intensive operation. Application code should be written to reduce the amount of parsing required. If a SQL statement needs to be executed multiple times, provide a persistent handle (cursor) to the parsed statements from call to call. This will ensure that a repeatedly executed statement is parsed only once. Do not use literals in SQL statements, instead use bind variables. This will minimize parsing by reusing parsed statements cached in the shared pool. For existing applications where rewriting the code to use bind variables is impractical, use the CURSOR_SHARING initialization parameter to avoid some of the parsing overhead. Also consider using stored procedures because they are stored in a parsed form, which reduces runtime parsing.

There are some exceptions to using bind variables. On a system (e.g. data warehouse) where performance of a long individual query is more important than number of queries per second, using bind variables might hurt performance; also bind variables will prevent the query optimizer from using histograms on columns with highly skewed data.

Process multiple rows at a time whenever possible

Fetching, processing, and writing rows in bulk is much faster than doing it row by row. A single SQL statement that processes all rows and performs all operations offers the optimal performance. This is because it requires just one network round trip and uses set-oriented processing that is highly optimized and parallelized in the database server. If you need to use a procedural approach, then design your application to process multiple rows at a time. For example, in PL/SQL you can use the FORALL statement and BULK COLLECT clause together with PL/SQL collection types to implement bulk processing. The Oracle JDBC driver provides support for statement batching and

mapping Oracle collections to Java arrays. The OCI array interface provides support for bulk processing for C/OCI applications.

Make sure that OLTP workloads use indexes

Using an index to select, update or delete a few rows in a big table is orders of magnitude faster than using a table scan. Make sure that all big tables used for OLTP have indexes, and that the indexes are actually used in query plans generated by the optimizer. Design your application to constrain the user so that OLTP transactions never perform full scans of big tables.

However, if your application needs to create report or perform batch data processing off-line, full table scans that use multi-block I/O may be the optimal solution. If your knowledge of the application suggests that a full table scan is the fastest way to execute a particular query, verify that the optimizer indeed plans to perform the scan and, if not, specify the ALL_ROWS hint in the select statement.

Reduce data contention

Data contention can substantially hurt application performance. To reduce contention in your application distribute data in multiple tablespaces/tables/partitions and avoid constant updates of the same row (e.g. to calculate balance), and run periodic reports instead. For more information on partitioning, go to this Oracle OTN page:

http://www.oracle.com/technology/products/bi/db/11g/dbbi_tech_info_part.html

Use object-orientation with care

The use of object-oriented techniques and languages in application development might hide the data access layer from the business logic. As a result, programmers might invoke methods without knowledge of the efficiency of the data access method being used. This tends to result in suboptimal database access: performing row filtering in the application, and doing row-at-a-time instead of array processing. To achieve high performance when using object-oriented languages try to place data access calls next to the business logic code. If you are storing your data for long-term or anticipate application development on the same schema, always use the relational storage method for the best performance and flexibility. Object-orientation at the programming level should not lead to de-normalization of the relational schema.

BEST PRACTICES

This section describes best practices that can allow developers to enhance performance and scalability of database applications.

Include performance testing during application development and maintenance

Build a test environment. Thoroughly test your application before going production. Periodically benchmark application performance, especially after database upgrades, major version releases and applying any software patches. Test against representative data and with multiple concurrent users. Only move to multi-user testing after you are satisfied with single-user performance.

Use Real Application Testing (RAT) to perform real-world testing. Enable database auditing. Monitor SQL statement execution using the V$SQL_MONITOR and V$SQL_PLAN_MONITOR views. Maintain and analyze performance statistics using AWR and ADDM. Use profiling tools to identify performance hot spots in your application. Eliminate identified hot spots so that optimal performance can be achieved.

Optimize most frequent operations

Before implementing an application, analyze how people will use it and identify the most critical operations. Always design your data model, database schema and application code to answer the most common queries as efficiently as possible. For non-critical queries, shortcuts in design can be adopted to enable a more rapid application development.

Do not re-implement database features

Use the features provided by the database instead of trying to re-implement them. For example, application or middleware should not try to re-implement primary and foreign keys, database integrity enforcement, joins and other costly operations, as well as auditing, replication, locking, message queuing, maintenance of history of changes, or sequences.

Choose indexing techniques that are best for your application

The Oracle Database supports a wide variety of indexing techniques. The B*Tree index is sufficient for the majority of applications. If they do not meet your requirements, then look at alternatives such as reverse-key, descending, bitmap, function-based, domain indexes and index-organized tables. Each has advantages and disadvantages that are described in the Concepts Guide. The SQL Access Advisor can be helpful in determining which indexes are required for your application.

Use external tables for loading and merging data

External tables allow the database to load data from flat files. An external table can be joined to another table during a load. External tables make it possible to filter the data by using any SQL

predicate. When loading from an external table Oracle will automatically determine the degree of parallelism, split up the input files and start parallel query slaves. Using an external table and the MERGE command allows you to efficiently update an existing database table with data from a flat file.

Understand how joins work

Oracle supports many types of joins including nested loop, hash, sort-merge, Cartesian, full outer and anti-join. It is important to understand how these joins are performed to decide if your application can take advantage of a join and choose an appropriate technique.

Use correct datatypes

Using incorrect datatypes might decrease the efficiency of the optimizer and hurt performance. It might also cause your application to perform unnecessary data conversions. Don't use a string to store dates, times or numbers. Ensure that conditional expressions compare the same data types. Do not use type conversion functions (such as a TO_DATE or TO_CHAR) on indexed columns. Instead use the functions against the values being compared to a column.

Define column constraints

Column constraints are often considered only from the data integrity point of view. However, the query optimizer also uses constraint definitions to build high-performance execution plans. Define NOT-NULL constraints when applicable; it does not impose noticeable overhead during query processing.

Implementing other constraints in the database might negatively affect performance. In this case, if a constraint is enforced by some other means, define an appropriate RELY constraint to help the optimizer.

Direct all branches of an XA transaction to a single RAC instance

In Oracle 11g, by default, branches of an XA transaction can be executed on multiple RAC instances and 2PC requests can be sent to any instance. This allows any application that uses XA to transparently take advantage of RAC. However, spanning an XA transaction across RAC nodes might cause frequent use of the RAC distributed lock manager and result in poor performance. In such cases, direct all branches of a distributed transaction to a single RAC instance. This can be done by using the Oracle Distributed Transaction Processing (DTP) service. To load balance across the cluster, it is better to have several groups of smaller application servers with each group directing its transactions to a single service, or set of services, than to have one or two larger application servers.

If you need to use a procedural language, consider PL/SQL or Java

PL/SQL is the most efficient language for massively data bound applications that do not require complex computation. Its data types are the same as in SQL, therefore, no type conversion is required

in application code. It provides implicit cursor caching that helps to minimize hard parsing. Generally, the server side is the preferred place to store the PL/SQL.

For very complex compute-bound applications with light data access consider using Java in the middle tier. Consider other procedural languages only if PL/SQL or Java cannot be used. Keep in mind that for any procedural language the most important approach to achieving high application performance is to choose the right algorithm and data structures.

Use static SQL whenever possible

There are many advantages of using static over dynamic SQL in PL/SQL. Specifically, SQL statements that are fixed at compile time are usually executed faster and the principle "parse once, execute many" is automatically applied to them. Also, the PL/SQL compiler automatically 'bulkifies' static SQL to improve performance. Dynamic SQL should be used only when static SQL is no longer practical, e.g. code becomes too complex with static SQL; or when only dynamic SQL will do the job.

If you need to use dynamic SQL, first try native dynamic SQL

Native dynamic SQL was introduced as an improvement on the DBMS_SQL API because it is easier to write and executes faster. It is the preferred way of implementing dynamic SQL code. DBMS_SQL is still maintained because of the inability of native dynamic SQL to perform a so-called "Method 4 Dynamic SQL" where the name/number of SELECT columns or the name/number of bind variables is dynamic. Also native dynamic SQL cannot be used for operations performed on a remote database.

Use pipelined table functions for multiple transformations of data

Pipelined table functions let you use PL/SQL to program a row source. A table function can take a collection of rows as input. Execution of a table function can be parallelized, and returned rows can be streamed directly to the next process without intermediate staging. With pipelined functions rows are iteratively returned as they are produced, instead of in a batch after all processing is completed. Streaming, pipelining, and parallel execution of table functions can dramatically improve performance of complex queries.

Use analytic functions

Analytics provides spreadsheet-like functionality in SQL language. It allows users to compute aggregates without grouping rows and display them with query results. Analytics apply to systems that generate reports, perform batch jobs, cleanse/validate data, and so on. Analytics can also be used to efficiently find a row in a partition and top-N in a group of queries. It can help to partition a table into non-overlapping chunks for parallel processing by multiple copies of a PL/SQL routine. Such application-managed parallelism makes sense when a parallel query cannot be used because of the inherently single-threaded nature of the PL/SQL routine.

Consider other techniques to improve SQL performance

Materialized views can store aggregated and pre-joined data from multiple tables and provide fast access to the data when frequent queries of the actual tables are extremely expensive. The ROWNUM pseudo column can also be used to materialize a query result set. It can also reduce the number of PL/SQL function calls, and improve performance of multiple joins and top-N query processing. Materialized views, pseudo columns, client and server result cache, hierarchical queries, merge statements and other techniques described in the Oracle documentation may significantly improve performance for certain query types.

Conclusion

Application performance directly impacts the end user experience. Oracle Database provides a broad range of features and capabilities to achieve high application performance. It is a responsibility of application developers to correctly choose and utilize database features that can satisfy performance and scalability goals. This paper describes rules, tips and techniques that can be used to ensure that an application is designed for performance.

REFERENCES

1. Oracle Database Concepts 11g Release 2 (Oracle Corporation, 2009)
2. Oracle Database Advanced Application Developer's Guide 11g Release 2 (Oracle Corporation, 2009)
3. Oracle Database Performance Tuning Guide 11g Release 2 (Oracle Corporation, 2009)
4. Oracle Database PL/SQL Language Reference 11g Release 2 (Oracle Corporation, 2009)

Guide for Developing High-Performance Database Applications
May 2010
Author: Mark Dilman
Contributing Authors: Wei Hu, Juan Loaiza, Kevin Jernigan

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment