Oracle Development - Part II: Coding Guidelines for Customizations

By Cheetah Solutions

Editor's Note: *In this second of a three white paper series on Oracle® Custom Development, Cheetah Solutions tackles the issue of coding guidelines; a "rulebook", if you will, for ensuring Oracle customizations are developed to your organization's and Oracle's recommended guidelines for maximum efficiency, performance, and reusability of code. Cheetah Solutions also discusses best practices for log files and, like any good development shop, reviews the need for and types of documentation that should accompany any custom effort. In their last white paper coming to the ORAtips July issue, Cheetah will round out the topic with a look at development standards including Explain Plans, variable usage conventions, and quality control.*

Introduction

Oracle customizations, too many approaches, and too many points of failure can impact both database performance and your ability to debug code. This document consists of recommended guidelines for coding in SQL and PL/SQL. It lists many of the Oracle industry-accepted general rules that should be followed by Oracle developers when developing, testing, and submitting code.

Objectives

This document draws guidelines necessary to code SQL and PL/SQL programs in a best possible efficient way. It is aimed at improving applications performance as a whole and minimizing Oracle database traffic. The guideline objectives are simple:

- To reduce load on Oracle database processes, thereby improving performance.
- To achieve better maintainability of custom code, better support, and faster debugging.
- To achieve higher levels of quality in the maintenance of Oracle Applications.
- To facilitate the process of code migration to a Production environment.
- To facilitate the process of patch applications and version maintenance.
- To achieve higher levels of accuracy in Oracle data processing operations.
- To achieve the goal of code reusability and thereby reduce Oracle development time.

Coding Standards

ERD

Any program design or database design that alters the standard design should not be considered. New objects should be created and defined under custom schemas.

DML

Any new or existing event-driven PL/SQL object should not be allowed to perform DML operations on any standard Oracle Applications database object.

IUD

With the exception of Oracle Applications Interface tables (OIT), any new or existing PL/SQL block should not perform INSERT or DELETE on standard Oracle Applications database objects unless absolutely necessary.

Minor UPDATEs can be performed; for example, changing the value of an attribute column.

SQL*Plus

Never perform a calculation on an indexed column. Ideally, the column to be indexed should assume the following characteristics:

- More distinct values
- NUMBER data type

Avoid the use of NOT or HAVING in the WHERE clause. Instead use the NOT EXIST clause.

Avoid the LIKE parameter. Instead, use "=" when ever feasible.

Using any Oracle function will invalidate an index, causing the use of a full table scan.

Use Oracle Cost Based Optimizer hints for all coding purposes.

Avoid using sub-queries. Instead use a JOIN in the WHERE clause. This will do the job with much better performance results.

Use the Oracle DECODE function to minimize the number of times a table has to be selected.

Whenever feasible, use the UNION statement instead of an OR condition. If you must specify multiple OR conditions, specify them in brackets. This will cause the Oracle kernel to change the precedence of executing conditions inside a WHERE Clause

Avoid using the COUNT (*) clause. Instead use the COUNT (INDEXED NOT NULL COLUMN NAME) clause.

Try to use optimizer hints if the WHERE clause is not using an indexed column.

INSERT statements should contain table columns especially when Oracle standard tables are involved.

The WHERE clause should be configured so that the maximum number of records involved in a query are eliminated at the early stage of the query itself. Filter conditions should appear towards the concluding part of the "WHERE" clause, since the execution order is from bottom to top.

Don't use the DUAL table to assign a value to a variable. Use direct assignment; for example, use a "select" statement to query the database to get the value.

Use untransformed column values. For example,

use	WHERE a.invoice_no = b.invoice_no
rather than	WHERE TO_NUMBER (TRIMa.invoice_no, 1,8)) = TO_NUMBER (TRIM(a.invoice_no,1,8))

Try to minimize the number of tables to be selected in the "FROM" clause in a single query. Oracle Support recommends a maximum of five tables for best performance. All custom indexes should be created on separate tablespace.

To properly maintain custom tables and indexes, it is always desirable to include storage parameters in the definition of Oracle database objects.

Custom tables and indexes should be sized, ideally, for one year of growth.

> To properly size a table or index, calculate the prospective transaction volume first. Using transactional information from the user community, the Oracle Developer can compute the necessary storage parameters.
>
> If storage parameters cannot be computed, then the storage parameters should be listed as follows:
>
> initial 1M, next 1M, maxextent 40 and pctincrease 0
>
> The calculations used for the storage parameters or why the default parameters were used should be included and detailed in the Oracle Developer's documentation for the custom code.

During the design phase for new development, it is best practice to consider some or all of Oracle's delivered features for performance. Partitioning, materialized views, index organized tables, and nested tables can all provide significant contribution towards the performance of the system when used properly.

Always try to avoid a full tablescan. Instead, write a PL/SQL Block with %ROWTYPE.

Before finalizing any part of custom code in any Oracle module, perform an Execution plan to review its tuning.

In SQL*Plus, the COMPUTE clause with BREAK statement on a database column works faster than a GROUP BY clause in a SELECT statement.

A query can be designed in numerous ways. Avoid using too many sub-queries. Instead use co-related sub-queries.

PL/SQL

Unless exclusively required, usage of hardcode values in a PL/SQL block should not be practiced.

If possible, the PL/SQL code should be upward compatible. It should not use commands and features that may become obsolete in future versions.

Similar to a cursor, a cursor variable points to the current row in the active set of a multi-row query. But, unlike a cursor, a cursor variable can be opened for any type-compatible query. It is not tied to a specific query. Cursor variables are true PL/SQL variables, to which new values can be assigned and passed to subprograms stored in an Oracle database. This allows more flexibility and a convenient way to centralize data retrieval. Caution – cursor variables can lead to readability and transactional issues, so proceed with care.

Avoid implicit data conversion. It is considered poor programming practice to rely on implicit data type conversions, which could impact performance and not remain consistent from one software release to the next. Implicit conversions are context sensitive and not always predictable. It is best to use data type conversion functions to ensure the reliability of your applications and for ease of maintenance.

Always use DECODE, rather than an “if-then-else” loop. DECODE is a command with many faces. The code will be faster and flexible because the Oracle database will not have to perform repetitive table reads.

Avoid database triggers. Instead design your coding logic in a stored procedure. Extensive usage of database triggers can jeopardize an Oracle Applications database. This is because processing in Oracle is primarily driven by concurrent processing and lock mechanisms, and database objects are tightly integrated. If you must use a trigger, save it for highly specialized purposes such as audit train or enforcing complex security rules.

Custom PL/SQL code should contribute to improving the database server performance. The code should be designed to not waste database resources. For example, if a PL/SQL block is accessing high traffic tables through a cursor, the cursor should be closed immediately after performing the task. The code should not contribute to a deadlock situation. A deadlock is a situation when two or more transaction takes place on the same database object at the same time. If the application logic does not permit the situation to be rectified, passing frequent commit or rollbacks should solve it.

To improve the efficiency of a PL/SQL block, the SET TRANSACTION statement should be used to begin a read-only or read-write transaction, or to assign current transactions to a specified rollback segment. Read-only transactions are useful for running multiple queries against one or more tables while other users update the same tables.

During a read-only transaction, all queries refer to the same snapshot of the database, providing a multi-table, multi-query, read-consistent view. Other users can continue to query or update data as usual. A commit or rollback ends the transaction.

The SET TRANSACTION statement must be the first SQL statement in a read-only transaction and can only appear once in a transaction. Setting a transaction to READ ONLY means subsequent queries see only changes committed before the transaction began. The use of READ ONLY does not affect other users or transactions.

For complex solutions, it is best practice to consider some or all of Oracle’s delivered features such as procedure overloading, recursion, packaged cursors and variables, and forward declarations.

Do not declare variables datatypes arbitrarily. Some Oracle programmers are habitual, declaring variables in advance with CHAR data type more than is required. This consumes space in the

Program Global Area (PGA) of the Oracle kernel, which ultimately results in adverse performance. For large ranges of alphanumerical data, use VARCHAR2.

Do not design complex dependencies on Oracle database and PL/SQL objects. Use the KISS principal: the code should be simple and easily understandable.

Make rich use of Oracle Applications AOL user exits and built ins. They can significantly contribute to the reliability, robustness, and performance of Oracle Applications.

For large-scale DML operations, use dynamic SQL with DBMS packages instead of solely writing DML statements in a PL/SQL block.

Performance is always a critical factor in PL/SQL. Therefore, take care to do enough performance related consideration during the design phase. For example, if a table is indexed, it is advisable to use joins rather than sub-queries, since joins are faster. Otherwise a sub-query is faster because it produces output for the main query unlike a join, which is a comparison of each row between two tables. Nullify the impact of indexes on individual SELECT statements by using hints and placing +0 in the WHERE clause.

Consider mutating of tables when designing the code. A mutating table problem is always a developer's nightmare. The PL/SQL code that is executed from a trigger can execute additional SQL SELECT, INSERT, UPDATE, and DELETE statements – not only against other tables but also against the table firing the trigger. In order to avoid runtime Oracle mutating table errors, do not program SQL statements in a database trigger to execute concurrently.

Logically relevant PL/SQL objects should always be packaged for better control over code, as well as reusability, modularity, and better integration.

Design for reusability, develop for reusability. If possible, avoid using variables from external modules (such as OUT variables). Instead create packages of global variables.

Try to isolate Application specific components from native PL/SQL components.

Avoid boilerplate graphics and hard coded text on report and form headings. Minimize the use of titles and help texts. Use generic terminology; use database objects instead of application version.

Design a custom module in layers and subsystems that minimize connection between users. The more functionality hidden, the easier it is for Oracle developers to understand what the module does. In other words, instead of incorporating a complex business requirement within one module, split it into multiple small modules. Splitting makes it easier to maintain.

Avoid using PL/SQL for networking and operating system related operations such as emailing the error log file, renaming the file, copying the file, etc. Instead write a caller program (preferably an operating system host script) to perform these functions. The caller program can also take care of native PL/SQL code.

Application Log Standards

A PL/SQL code is executed in the form of concurrent programs in Oracle Applications. When a concurrent program is executed, it generates a log file. This log file contains runtime details and it

is unique to each execution. By default, the log file shows only the completion status and timestamp. However, the log file can be filled with considerably more information through messages in the same PL/SQL code. Quite often, this information is useful for debugging and troubleshooting, and it is the only way of tracing source code behavior during its execution in the Oracle Applications.

An Application log file should contain the following:

- The process begin status
- The runtime value of variables and database columns
- The runtime value of cursor variables
- Status messages when appropriate milestones are reached within the code
- Exception messages
- The completion status

Documentation

And finally, no Oracle custom enhancement is complete without detailed documentation at every stage that chronicles the development for debugging, trouble-shooting, and re-engineering purposes following an upgrade or patch application.

Every Oracle Developer should document their programming effort. Examples of document include:

Development Specification Document (MD-50 or DSD)

This document sets the stage for the design effort and should be started at the same time as the commencement of the software life cycle development process. Typically this document is reviewed and approved by the Business Analysts before development begins.

Functional Module design Document (MD-70)

This document describes the functions performed by the custom code, be it a report, form, or interface for example. It should include all prerequisites to perform the function, its effects, points of integration, entities involved, validation issues, and topical essay. Typically this document too is reviewed by the Business Analysts as development proceeds.

Application Setup Document (SR-100)

This document should provide a complete step-by-step setup path for the Oracle System Administrator to configure the Oracle Applications.

Test Plan (UAT-100)

This document should describe testing cycle, scenarios considered and test plan under various instances, and also results of testing.

User Training Document (TR-100)
Once the development and implementation are completed, end-user training to use the new customization is recommended and required.

Conclusion

Designing and developing custom code for Oracle is easy. Designing and developing efficient code that meets the business requirements without impacting system performance or creating a hostage situation where only the original developer can maintain it is achievable. But only if you follow the recommended guidelines and industry accepted standards for programming.

This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips.
For training, consulting, and articles on JD Edwards or SAP, please visit our websites: www.JDEtips.com & www.SAPtips.com.