

Sybase to Oracle SQL Code Comparison

September 1, 2004

These two SQL scripts do the same thing. The green script is for Sybase. The red script is for Oracle. The functional purpose of these scripts is to delete thousands of rows from many types of sales statistics tables for those managers and salesmen who have left the company. The scripts use cursors to keep the transactions small.

The purpose in showing you these scripts is to demonstrate the differences between how Sybase and Oracle handle table management, cursors, user interface and errors. If you are Microsoft SQL Server user, you can equate yourself to a Sybase user. While there are minor differences in cursor and print syntax, the overall concepts and structure of Microsoft SQL Server and Sybase T-SQL are the same.

Opening Comments

- First difference is this. Note the opening comments. They explain the necessity of BTPURGELOG table and BTPURGELOG_SEQ sequence for Oracle.

Oracle's Set SQL*Plus Characteristics

- While this section contains quite a bit of functionality from the Oracle perspective, it is not needed in Sybase.
- set autocommit on: In Oracle by default, transactions are not committed until you explicitly say "commit". You can change the default by turning on this SQL*Plus setting. Nonetheless, Oracle ignores the setting within embedded PL/SQL blocks. PL/SQL has its own independent commit syntax.
- set serveroutput . . . is part of Oracle's substitute for the Sybase "print" statement. If you want to print informational messages from a PL/SQL block, you must set serveroutput first. Set serveroutput establishes a buffer for dbms_output.put_line() functions to write their information. Without the buffer, you will not see messages. Also note that Oracle cannot write informational messages in real-time. dbms_output.put_line only builds a buffer which Oracle prints *after* the PL/SQL block completes. The largest buffer is 1 MB.

Exit Upon Error

- This is a useful Oracle feature. Sybase's isql could use the ability to exit a script if it detects an error. While isql does not do this, psSQL, DBPowerSuite's Sybase user-session program, has this feature. DBPowerSuite's psOSQL program (an ODBC client program which attaches to any database) also has this feature. All DBPowerSuite programs exit a checkable error code. If you are a Sybase customer, or you use ODBC in any form, you can write UNIX scripts with embedded psSQL sessions and just check the error code coming out of psSQL. You don't have to grep for "Msg" or "Error".

Drop Objects Commands

The goal is to drop certain objects if they exist. But . . .

- Oracle has no if / then syntax in SQL*Plus, so one cannot test for the existence of objects before dropping them. Most Oracle programmers practice the "try-and-clean-up-afterwards" way of writing code. They will code a drop table statement and let SQL*Plus log an error. Oracle DBAs expect scripts to behave like this. DBAs have become so used to it, they expect errors from all scripts. When they don't see errors, they get suspicious that something went wrong. The try-and-clean-up-approach is second-nature in the Oracle world, but it still defies the first axiom of programming: write code that prevents errors. Any messages in a log are suspect. and require examination. No messages are desirable.
- Most times, the programmer wants to *create and replace* objects. Oracle programmers use the *create and replace* syntax which does not exist in Sybase. Nonetheless, simply dropping objects, doing house-cleaning type operations, poses a problem in Oracle.
- The spool-select-start commands in the Oracle code is one way to prevent DDL errors. The spool-select-start commands create a temporary file, generate a command in the temporary file, then execute the temporary file. If the object does not exist in the first place, then the file is empty. Executing the empty file does nothing.
- The second way to conditionally drop an object in Oracle is to use a PL/SQL block that conditionally executes an embedded *dynamic* SQL command. This second method is a better way because the operation doesn't create a file in the current directory, and thus, does not require write permission into the directory. There are many programs in DBPowerSuite which write Oracle scripts. All these programs use the embedded dynamic SQL approach.
- Sybase, SQL Server and Oracle support dynamic SQL. The difference is, is that in Sybase and SQL Server, most DDL does not have to be run dynamically; while in Oracle, all DDL has to be executed dynamically. In Sybase, the definition of "dynamic" SQL is SQL which is built and assembled at run-time. The definition in Oracle is SQL that is executed as a string inside a PL/SQL block. Because PL/SQL does not support any straight DDL execution, PL/SQL forces you to execute any DDL statement dynamically, whether it was assembled at run-time or was fixed when the application was first written. Consequent DML used on the dynamically built object, must employ special dynamic syntax, because latter SQL code in the PL/SQL block is not aware that you created an object beforehand. You will come to the realization that there is yet a fourth language in Oracle: the dynamic SQL language that resides within PL/SQL.
- Note that the Oracle script executes one SQL statement at a time. That is because Oracle does not support the *batch* concept. Oracle cannot execute any more than one SQL statement at a time. Back-to-back Server SQL statements are not possible with Oracle.

Create Tables and Sequence Commands

- The Oracle script contains an extra BTPURGELOG table, BTPURGELOG_SEQ sequence and BT_INSERT_LOG stored procedure. The code to create and manage these tables is a consequence of Oracle's lack of a "print" statement.
- Compare the datatypes. In Oracle, you see a number(10). That is equivalent of a Sybase numeric(10), not an int. int is more efficient in many ways. One cannot use an 'int' in the Oracle script because Oracle does not support its equivalent, nor does Oracle store any numbers in the binary native to the computer. Engineering and scientific programmers need to know this. An operating system's native floating datatypes inherently support scientific notation. But Oracle does not support the native operating system data types. Hence, Oracle does not support scientific notation, and thus, numbers one stores in Oracle is limited to the range Oracle supports.

Create Procedure Statements in General

- The arguments to stored procedures in Oracle take the form of function arguments. In other words, you use parenthesis. In Sybase, you list them out without the parenthesis.
- In Sybase, you can list the procedure arguments after the comment. In Oracle, you must list the arguments in juxtaposition with the procedure name. Oracle blows up if you don't.
- Oracle stored procedures cannot exit with a return code. They just return. Sybase procedures, by nature, always return with a status code. The Sybase language provides the mechanism to bubble up errors from below. The Sybase programmer checks the status to see the procedure worked. Oracle's behavior is "let it blow up and clean up afterwards".
- In Oracle, you can also create SQL functions, not just stored procedures. You can create functions in Sybase too, it is just that they have to be written in Java.

Create BT_INSERT_LOG Procedure

This procedure exists only for Oracle's sake.

- The dual table is an Oracle dynamically-built table. The dual table is not a physical table. This table just contains various system global variables which the programmer can access. For example, the Sybase equivalent of Oracle's "select to_char(sysdate, 'YYYY-MON-DD HH24:MI:SS' is select getdate()).
- commit work : Even if this procedure block was an embedded PL/SQL block within SQL*Plus, PL/SQL still will ignore SQL*Plus's autocommit setting.

Create PURGE_SAMPLES Procedure

- Note the *type . . . is table* command in Oracle's declaration section. Oracle requires all these special commands because Oracle is not capable of creating a normal temporary table, or any normal table for that matter in PL/SQL. This syntax is a work-around. Any you cannot do any standard table operation like joining with this kind of table.
- Oracle's select into: In Oracle PL/SQL, *select into* means to set a variable equal to a value. In Sybase, one uses select into to create and fill tables.
- Oracle's ltrim(to_char(vvarchar(12), '999999999999')): Even though the '999999999999' is a useful print format style in Oracle, it is not necessary in Sybase.
- Oracle's select distinct . . . bulk collect into: This syntax is a work-around to Oracle's inability to create and process normal tables within PL/SQL. In Sybase, a single simple normal *insert into #temp* does the job.
- Note that every time a message prints to the log in Oracle, one calls the procedure BT_INSERT_LOG. Oracle has no way to print messages to standard output in real-time from within a stored procedure. Hence, the code insert rows into a real table, and then from a different process, the user can select rows from the log table to see the progress.

Common Differences In all the Other Stored Procedures

- There are differences in the way which Sybase and Oracle name and set variables. For example, Sybase begins all variable names with a '@' sign. This is pretty useful because you can always discern between column names and variables in Sybase.
- Oracle sets a variable using the ':=' while Sybase uses a select = statement.

- The biggest difference between the Sybase and Oracle procedures is the fact that Sybase can create and drop database objects within a stored procedure without using dynamic syntax while Oracle must employ dynamic syntax. The dynamic creation has consequences.
- The dynamic "execute immediate", "open cursor for 'SQL'", and special "fetch/select into/using" statements only exist in the Oracle scripts.
- Everything from create and select statements to analyze commands must be expressed in Oracle dynamic syntax because Oracle cannot do any standard operations on a table that was initially created dynamically.
- The dynamic syntax also requires two additional things:
- The user which creates tables with dynamic SQL needs CREATE ANY TABLE privilege.
- When writing such procedures, Oracle does not check the syntax of the dynamic SQL. So your testing time increases by a factor of at two. You must run your scripts for Oracle to see and test the dynamic SQL.

Sybase does have dynamic SQL ability. However, it is not needed in this example. The SQL here is *static*. It doesn't change from run to run.