



# ***Translating Procedural Statements Between Oracle and SQL Server***

---

White Paper

**© Copyright Quest® Software, Inc. 2005. All rights reserved.**

The information in this publication is furnished for information use only, does not constitute a commitment from Quest Software Inc. of any features or functions discussed and is subject to change without notice. Quest Software, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication.

Last revised: June 2005

# TABLE OF CONTENTS

<b>OVERVIEW</b> .....	<b>4</b>
<b>HISTORY</b> .....	<b>5</b>
<b>THE RULES</b> .....	<b>6</b>
IDENTIFIERS .....	6
DATATYPE DIFFERENCES .....	7
VARIABLES AND PARAMETERS .....	8
BEGIN...END DIFFERENCES .....	9
CONTROL STRUCTURES .....	9
LOOP STRUCTURES .....	10
CURSORS .....	11
EXCEPTION HANDLING AND ERROR CONTROL .....	13
PROCEDURES, FUNCTIONS, AND TRIGGERS .....	14
PACKAGES .....	15
BUILT-IN'S .....	16
SOME TRANSLATIONS OF COMMONLY USED COMMANDS .....	17
<b>SUMMARY</b> .....	<b>18</b>
<b>ABOUT QUEST SOFTWARE, INC.</b> .....	<b>19</b>
CONTACTING QUEST SOFTWARE .....	19
TRADEMARKS .....	19

# OVERVIEW

This white paper is intended to give you a **QUICK** but thorough overview of the issues you may encounter when translating procedural code, such as stored procedures or user-defined functions, between Oracle's PL/SQL and Microsoft SQL Server's Transact-SQL procedural extensions to the ANSI SQL standard. The document should be equally useful for SQL Server DBAs and developers who need to become familiar with PL/SQL, as for Oracle DBAs and developers who want to get up to speed quickly on Transact-SQL.

# HISTORY

Microsoft does not issue separate releases Transact-SQL from the database. When thinking about the version number of a given set of Transact-SQL code, you must really think about the version number of the database (and service pack) that the code was written for.

Oracle, on the other hand, issues releases of PL/SQL separately from the database. Why? Oracle has long supported PL/SQL not only with the relational database, but also in other development products like Oracle Forms and Oracle Reports. Consequently, certain releases of the procedural language were necessary to provide features for the separate client and server applications. Oracle is now in its 8.x release of PL/SQL, but other prominent releases include 1.x and 2.x.

Another interesting aspect of PL/SQL is that it has shared roots with the ADA language, most prevalent in the Department of Defense of the U.S. federal government. It shares many of ADA's capabilities including extensive exception handling features and distinctive block structure.

Oracle also supports Java as a procedural language within the relational database. Although there doesn't seem to be a high rate of adoption in the user community, it is possible to write all of your stored procedures and user-defined functions on an Oracle database using Java. Microsoft SQL Server will offer similar capabilities, such as writing stored procedures in C# or VB.Net, in the next release of SQL Server.

# THE RULES

Translating procedural code is pretty simple when you follow some basic rules. First, learn what the differences are in identifiers, datatypes, variables, and parameters. After that, it's simply a matter of applying those differences and learning analogous function calls and extensions.

## Identifiers

We'll make this go a lot faster by breaking the identifier rules out in a table.

	ORACLE	SQL SERVER
Identifier size	30 bytes (number of characters depends on the character set); database names are limited to 8 bytes	128 characters (temp tables are limited to 116 characters)
Identifier may contain?	any number, character, and the underscore ( <code>_</code> ), pound ( <code>#</code> ), and dollar ( <code>\$</code> ) symbols	any number, character, and the underscore ( <code>_</code> ), at sign ( <code>@</code> ), pound ( <code>#</code> ), and dollar ( <code>\$</code> ) symbols
Identifier must begin with?	a letter	a letter, underscore ( <code>_</code> ), at sign ( <code>@</code> ), or pound ( <code>#</code> ). Note that <code>#</code> and <code>##</code> are used to denote local temporary and global temporary tables, respectively.
Identifier cannot contain?	Double-quote ( <code>"</code> )	Spaces or special characters
Both allow quoted identifiers? (Identifier may be a reserved word or key word only as long as they are a quoted identifier.)	double-quotes ( <code>""</code> )	double-quotes ( <code>""</code> ) or brackets ( <code>[]</code> ), which are preferred
Schema addressing?	<code>database.schema.object@servername</code>	<code>server.database.schema.object</code>

A couple side notes about identifiers:

- Identifiers must be unique within their scope on both platforms
- Oracle database links are limited to 128 characters with *NO* quoted identifiers allowed

# Datatype Differences

The two platforms support the common ANSI standard datatypes such as *CHARACTER* or *INTEGER*. However, they both offer a number of extensions that need a little explaining.

Here's a quick list of tips when translating datatypes between the two platforms:

- Use Oracle's *BFILE* or *BLOB* instead of MSSQL's *IMAGE*. *BFILE* is used to store images on a file on the file system, while *BLOB* stores it within the database. MSSQL's *IMAGE* is most like Oracle's *BLOB*. Use Oracle's *CLOB* or *NLOB* large text object datatypes instead of MSSQL's *TEXT* large text object datatype. Oracle's *CLOB* is for straight text while *NLOB* is for national language text support.
- Use Oracle's *DATE* instead of MSSQL's *DATETIME* or *SMALLDATETIME*.
- Use Oracle's *DLOB* to hold documents (get it? – *DLOB* = 'document large object'?) but use MSSQL's *IMAGE* datatype.
- Oracle supports *INTERVAL* datatypes while MSSQL does not.
- Oracle supports, but is phasing out *LONG* (character) and *LONG RAW* (binary) datatypes. *RAW* (also binary) is also on the way out. When converting, remember that *LONG* is functionally similar to *CLOB* and *LONG RAW* is similar to *BLOB*. Use Oracle's *NVARCHAR2* and *VARCHAR2* instead of *VARCHAR*. Beware *VARCHAR* on Oracle because it has very different capabilities than it does on MSSQL.
- Oracle's beloved *ROWID* (and *UROWID*) pseudocolumn has a very different meaning in SQL Server. If you are used to using *ROWID* on Oracle, you're out of luck on SQL Server. You'll have to build a similar functionality by hand.
- The *TABLE* and *CURSOR* datatypes is only supported as a PL/SQL variable datatype, not as a column datatype. Oracle also supports a *VARRAY* variable datatype. This differs from MSSQL's handling of the *TABLE* and *CURSOR* datatypes. MSSQL does not support *VARRAY*.
- Oracle's *TIMESTAMP* is an actual date and time value more like SQL Server's *GETDATE()*. From the Oracle perspective, SQL Server does not support the following Oracle datatypes: *BFILE*, *BLOB*, *DATE*, *DLOB*, *INTERVAL*, *LONG*, *LONG RAW*, *NCLOB*, *NVARCHAR2*, *RAW*, *ROWID*, *UROWID*, and *VARCHAR2*.
- From the SQL Server perspective, Oracle does not support the following SQL Server datatypes: *BIGINT*, *BINARY*, *DATETIME*, *IMAGE*, *MONEY*, *NTEXT*, *ROWVERSION*, *SMALLDATETIME*, *SMALLMONEY*, *SQL\_VARIANT*, or *TEXT*.

- From the Oracle perspective:
  - a) SQL Server doesn't support the *VARRAY* variable datatype and handles *TABLE* differently.
  - b) Oracle offers the *%TYPE* and *%ROWTYPE* shortcut and anchored datatype, e.g.:
    - `my_company company.name%TYPE;`
- From the SQL Server perspective, Oracle does not support the following datatypes: *BIGINT*, *BINARY*, *CURSOR*, *DATETIME*, *IMAGE*, *MONEY*, *NTEXT*, *ROWVERSION*, *SMALLDATETIME*, *SMALLMONEY*, *SQL\_VARIANT*, or *TEXT*

## Variables and Parameters

Variables and parameters are very important constructs for use in procedural code like stored procedures or user-defined functions. Variables are used to temporarily store values while processing procedural code. Parameters, which are very similar to variables, are used to pass temporary values into or out of procedural code.

- In Oracle, variables are created with *DECLARE* block similar SQL Server. For example, Oracle allows *DEFAULT* and *[NOT] NULL* constraints on variables just as SQL Server does.
- In SQL Server, variables are prefixed by the 'at' symbol (@). (Oracle doesn't require a special prefix.) Certain special global variables are available that use two 'at' symbols (@@). However, global variables are hard-coded by the system and are not available to end-users.
- Parameters in SQL Server are simply variables placed after the *CREATE PROC* keywords and before the *AS* keyword. On Oracle, parameters are further defined as *IN*, *OUT* or *IN OUT* mode.
  - *IN* parameters can only pass a value into the stored procedure.
  - *OUT* parameters can only pass a value out of the stored procedure.
  - *IN OUT* parameters can pass a value both into and out of the stored procedure.
- In Oracle, variable and parameter values are assigned with *:=* (e.g. `my_int_var := 100;`). SQL Server uses the syntax `SET @my_int_var = 100` or `SELECT @my_int_var = 100`.
- Oracle allows *SUBTYPES* – a permutation on a variable that specifies the same rules as the original datatype, but allows only a subset of values:

```
-SUBTYPE subtype_name IS base_type;
```

For example,

```
SUBTYPE desk_phone IS phone_nbr;
```

## BEGIN...END Differences

In a nutshell, SQL Server uses *BEGIN...END* blocks mostly for encapsulation. For example, in an *IF...THEN* condition, SQL Server will process the next single Transact-SQL statement when executing the *THEN* condition. If you have many Transact-SQL statements after a *THEN*, you must put them inside of a *BEGIN...END* block.

Oracle is more robust in this respect, allowing named *BEGIN...END* blocks for greater control when nesting blocks of PL/SQL code. Each block may have its own subordinant *DECLARE* block and *EXCEPTION* block. If you include a *DECLARE* and *EXCEPTION* block, you also gain the feature that all variables and exceptions are then scoped only to that block. For example:

```
BEGIN
  BEGIN
    DELETE FROM child1 WHERE ...;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
  END;
  DECLARE
    my_var CHAR(5);
  BEGIN
    DELETE FROM child2 WHERE ...;
  END;
END;
```

## Control Structures

There is a big difference in the branching control structures of these two platforms. SQL Server allows only *IF...THEN*. Oracle supports *IF...THEN...ELSIF...ELSE* structure and requires an *END IF* delimiter. In addition, you must include *THEN* on Oracle, but not SQL Server. Nested *IF* statements are OK on both:

```
IF <condition>
THEN
  ...statements...
[ELSIF <more conditions>
  THEN
    ...more statements...][...]
[ELSE
  ...yet more statements...]
END IF;
```

Both Oracle and SQL Server supports *GOTO*. But the syntax have some variation.

On Oracle, you leap to the block via:

```
GOTO goto_block_name;
```

And label the block like this:

```
<<goto_block_name>>
```

In SQL Server, you leap to the block via:

```
GOTO goto_block_name
```

And label the block like this:

```
:goto_block_name
```

Finally, Oracle offers an additional control structure - the *NULL* statement. Oracle allows you to enter a “*NULL;*” command in situation where you are required to have at least one PL/SQL statement but you want nothing to occur. This might be important to nullify the effect of a raise exception or the *ELSE* of an *IF* statement, e.g:

```
IF :report.selection = 'DETAIL'  
THEN  
    exec_detail_report;  
ELSE  
    NULL;  
END IF;
```

## LOOP Structures

SQL Server only has *WHILE* loops compared to Oracle, with three kinds of loops.

First, Oracle offers a simple loop structure. *LOOP* cycles infinitely and must be manually exited, either externally with *IF* or internally with *EXIT WHEN*. For example, on Oracle:

```
BEGIN  
    IF max_rank_in >= 1  
    THEN  
        LOOP  
            set_rank (ranking_level);  
            ranking_level := ranking_level + 1;  
            EXIT WHEN ranking_level > max_rank_in;  
        END LOOP;  
    END IF;  
END;
```

Oracle's *FOR* loop has nice control features built-in, since you automatically control the number of times it will loop through the code, and usually requires the least amount of code. For example, on Oracle:

```
BEGIN
  FOR ranking_level IN 1 .. max_rank_in LOOP
    set_rank (ranking_level);
  END LOOP;
END;
```



Oracle loops also allow an optional loop label (i.e. a name for the loop) and require an *END LOOP* clause.

Oracle's *WHILE* loop is similar to SQL Server's, though you need the *END LOOP* loop boundary. For example, on Oracle:

```
BEGIN
  WHILE ranking_level IN 1 <= max_rank_in LOOP
    set_rank (ranking_level);
    ranking_level := ranking_level + 1;
  END LOOP;
END;
```

Oracle supports *EXIT*, *EXIT...WHEN*, and *RETURN* keywords to exit a simple loop. Do not use these in a *FOR* or *WHILE* loop since you explicitly control the execution of the loop in its definition.

## Cursors

Oracle cursors are very similar, syntactically, to SQL Server. In Oracle, you use the *DECLARE* block, *OPEN*, *FETCH*, and *CLOSE* just like SQL Server. Behind the scenes, it is a different story, because Oracle uses implicit forward-only cursors in every PL/SQL DML statement that retrieves a result set! SQL Server, on the other hand, uses cursor structure only when you need to circumvent the standard set processing behavior of the query engine.

In coding there are only two considerations to keep in mind. In Oracle, remember that a *COMMIT* statement on each pass through the loop frees locks held during the cursor sweep. And while Oracle supports the *WHERE CURRENT OF* syntax, it does not support any of the special cursor types supported in SQL Server including *FORWARD\_ONLY*, *SCROLL*, *STATIC*, *DYNAMIC*, *FAST\_FORWARD*, etc.

In Oracle, the *FETCH* statement is similarly limited. You can *FETCH cursor\_name INTO variable*, but you cannot use any of the special fetch characteristics available in SQL Server like *FIRST*, *NEXT*, *PRIOR*, *LAST*, etc.

Oracle's syntax varies a bit from SQL Server in the *DECLARE CURSOR* statement. It requires an *IS* keyword and has the optional *RETURN* clause. These features were designed to allow black box programming, like in Oracle packages. For example:

```
CURSOR current_emp RETURN employee%ROWTYPE
IS
    SELECT * FROM employee WHERE dept_no = 10;
```

You can verify a cursor is open with *%ISOPEN*:

```
IF NOT current_emp%ISOPEN
THEN
    OPEN current_emp
END IF;
```

*FETCH* and *CLOSE* are essentially the same between Oracle and SQL Server:

```
FETCH cursor_name INTO variables;
```

```
CLOSE cursor_name;
```



There is no *DEALLOCATE* statement in Oracle as there is in SQL Server.

The maximum number of cursors on an Oracle server is controlled by the *OPEN\_CURSORS* initialization parameter. Oracle also provides a few built-in functions to check the four main attributes of a cursor:

- *%FOUND* – tells if the cursor finds the next record (TRUE) or failed to (FALSE)
- *%NOTFOUND* – tells if the cursor didn't find another record (TRUE) or did (FALSE)  
E.G., EXIT WHEN current\_emp%NOTFOUND
- *%ROWCOUNT* – tells how many records the cursor has retrieved  
E.G., EXIT WHEN current\_emp%ROWCOUNT > 10
- *%OPEN* – tells if the cursor is open (TRUE) or not open (FALSE)

# Exception Handling and Error Control

Oracle's exception handling system is styled after the ADA programming language. And it is much more powerful than anything that exists in Transact-SQL. You control exception handling by defining exception blocks that appear at the end of your *BEGIN...END* block. For example:

```
EXCEPTION
  WHEN exception_name [OR exception_name ... ]
  THEN executable_statements
END;
```

Oracle ships with lots of predefined exceptions:

- CURSOR\_ALREADY\_OPEN
- DUP\_VAL\_ON\_INDEX
- ZERO\_DIVIDE
- OTHERS
- And many more...

You declare your own exceptions in the *DECLARE* block by listing its name with the *EXCEPTION* keyword:

```
exception_name EXCEPTION;
```

These can be referenced in two ways in your PL/SQL code via *RAISE* and *WHEN*:

```
IF security_rating <= 10 THEN
RAISE unauthorized_reviewer;

WHEN unauthorized_reviewer THEN kick_them_out(login_id);
```

Oracle also allows *pragmas*, special instructions to the compiler that are processed at compile time instead of runtime.

# PROCEDURES, FUNCTIONS, and TRIGGERS

The biggest difference to get used to when converting between Oracle and SQL Server is that Oracle stored procedures do not return result sets, though they do return *OUT* or *IN OUT* parameters. Another minor difference when creating procedures, functions or triggers is that SQL Server supports the syntax *CREATE object\_type object\_name AS...* while Oracle supports the syntax *CREATE OR REPLACE object\_type object\_name AS...* where *object\_type* is *PROCEDURE*, *FUNCTION*, or *TRIGGER*. (Oracle supports an additional type of trigger – *BEFORE* triggers that execute their code before the DML operation is begun.)

In Oracle, stored procedures and UDF syntax is based upon a more rigid block structure than in SQL Server. The block structure is always the same:

```
CREATE OR REPLACE object_type object_name
IS
    [Declare_block]
BEGIN [name]
    executable_commands
[EXCEPTION
    executable_commands]
END [name];
```



A Warning about procedures - Oracle does not support the use of data stream result sets!

You can get around this by:

- Define Oracle procedures with a PL/SQL *TABLE* datatype as an output parameter (i.e. an *OUT* or *IN OUT* parameter).
- Define Oracle procedures with a PL/SQL *CURSOR* datatype as an output parameter (i.e. an *OUT* or *IN OUT* parameter).
- Populate a temporary table with the result set, adding a column to identify which SPID / SESSION\_ID generated the values, and then retrieve the values stored in the temporary table.

# PACKAGES

The Oracle construct called a *package* has no corollary in SQL Server. Packages are “black box” groupings of stored procedures and functions that share all variables, input, and output. They provide a number of benefits:

- Hide internals, a black box
- Object-oriented and top-down design
- Performance improvement via caching
- Object persistence and globalization of package components

When converting from Oracle PL/SQL to SQL Server Transact-SQL code:

- break packages into many procs, UDFs, etc...
- consider using global objects (temp tables, cursors) for easier migration

When converting from SQL Server to Oracle:

- review object design to bundle logically related procs, UDFs, etc...
- review shared variables in procedural code for commonality

In Oracle, there are two units of code that go into a package, the *package specification* and the *package body*. First, the package specification declares all of the component procedures and user-defined functions and in parameters they may possess:

```
PACKAGE calc_wages
IS
    PROCEDURE capture_hours (hrs IN DEC(9,2));
    PROCEDURE check_sal (emp_ID IN CHAR(5));
END calc_wages;
```

The package body contains all the stored procedures, user-defined functions, and general blocks of PL/SQL code that make up its component parts:

```
PACKAGE BODY calc_wages
IS
    PROCEDURE capture_hours (hrs IN DEC(9,2)) IS
    BEGIN...
    END;
    ...
END calc_wages;
```

## Built-In's

Built-ins are Oracle's equivalent to system stored procedures and internal functions. Most built-ins are usually a package, but they are sometimes user-defined functions or simple stored procedures. Here are some of the most commonly used built-ins on the Oracle platform:

- *DBMS\_ALERT*: Controls alerts
- *DBMS\_JOB*: Controls jobs and their schedule
- *DBMS\_LOB*: Manipulates LOBs (PL/SQL 8 only)
- *DBMS\_LOCK*: Controls locking for your user context
- *DBMS\_OUTPUT*: Displays output from PL/SQL programs to the terminal
- *DBMS\_PIPE*: Communicates between different Oracle sessions through a pipe in the RDBMS shared memory.
- *DBMS\_SESSION*: Controls user sessions
- *DBMS\_SQL*: Controls dynamic SQL in PL/SQL procedural code from v2.1
- *DBMS\_TRANSACTION*: Controls transactions
- *DBMS\_UTILITY*: Contains miscellaneous utilities package somewhat like SQL Server's DBCC does. For example, *FORMAT\_CALL\_STACK* returns the current stack of called modules
- *UTL\_FILE*: Controls read and write operations to system files from v2.3

## Some Translations of Commonly Used Commands

SQL SERVER	ORACLE
CASE	CASE , DECODE
CONVERT, CAST	TO_DATE, TO_CHAR, TO_NUMBER
DATEDIFF, DATEADD	Date arithmetic
EXEC	EXEC
GETDATE( )	SYSDATE
GO	;
ISNULL	NVL
PRINT	DBMS_PIPE
RAISERROR	RAISE_APPLICATION_ERROR
RETURN [numeric]	RETURN [any literal]
SUBSTRING	SUBSTR
@@ERROR	SQLCODE
@@ROWCOUNT	SQL%ROWCOUNT

## SUMMARY

This material is intended to help you quickly see some of the most prominent differences between SQL Server Transact-SQL procedural code and Oracle PL/SQL procedural code. By reviewing this document, you should be able to read procedural code from either database platform and have a good understanding of what it is trying to accomplish. You should also be able to tell what areas of the code might be challenging for you to migrate and, thereby, know what areas might require further research.

There are a multitude of many other small differences between Oracle and SQL Server, for example in the kind of functions you can call. In addition, there are many differences in actual SQL statements such as *CREATE*, *DROP*, *GRANT*, *SELECT*, *INSERT*, *UPDATE*, and *DELETE*. This document is not intended to teach you everything you should know when doing a full-blown conversion from one platform to the other, but hopefully, it will give you a quick idea of some of the major challenges you face in project of this type.

## ABOUT QUEST SOFTWARE, INC.

Quest Software, Inc. delivers innovative products that help organizations get more performance and productivity from their applications, databases and infrastructure. Through a deep expertise in IT operations and a continued focus on what works best, Quest helps more than 18,000 customers worldwide meet higher expectations for enterprise IT. Quest Software, headquartered in Irvine, Calif., can be found in offices around the globe and at [www.quest.com](http://www.quest.com).

## Contacting Quest Software

Mail:	Quest Software, Inc. World Headquarters 8001 Irvine Center Drive Irvine, CA 92618 USA
Web site	<a href="http://www.quest.com">www.quest.com</a>
Email:	<a href="mailto:info@quest.com">info@quest.com</a>
Phones:	1.800.306.9329 (Inside U.S.) 1.949.754.8000 (Outside U.S.)

Please refer to our Web site for regional and international office information. For more information on other Quest Software solutions, visit [www.quest.com](http://www.quest.com).

## Trademarks

All trademarks and registered trademarks used in this guide are property of their respective owners.