

## **Oracle® Database**

SQL Developer Supplementary Information for Microsoft SQL  
Server Migrations

Release 1.2

**E10379-01**

June 2007

This document contains information for migrating from  
Microsoft SQL Server to Oracle. It supplements the  
information about migration in *Oracle Database SQL Developer  
User's Guide*.

Oracle Database SQL Developer Supplementary Information for Microsoft SQL Server Migrations, Release 1.2

E10379-01

Copyright © 1998, 2007, Oracle. All rights reserved.

Primary Author: Chuck Murray

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

---

---

# Contents

<b>Preface</b> .....	vii
Intended Audience.....	vii
Documentation Accessibility .....	vii
What You Should Already Know .....	viii
Related Documents .....	viii
Conventions .....	viii
Third-Party License Information.....	viii
<b>1 Overview</b>	
<b>Introduction</b> .....	1-1
<b>Product Description</b> .....	1-1
<b>Features</b> .....	1-2
<b>Glossary</b> .....	1-2
<b>2 Microsoft SQL Server and Oracle Compared</b>	
<b>Schema Migration</b> .....	2-1
Schema Object Similarities.....	2-1
Schema Object Names .....	2-3
Table Design Considerations.....	2-3
Data Types .....	2-3
DATETIME Data Types .....	2-3
IMAGE and TEXT Data Types (Binary Large Objects) .....	2-5
Microsoft SQL Server User-Defined Data Types .....	2-5
Entity Integrity Constraints .....	2-6
Referential Integrity Constraints .....	2-6
Unique Key Constraints.....	2-6
Check Constraints.....	2-7
SQL Server Rule: .....	2-7
<b>Data Types</b> .....	2-8
Data Types Table.....	2-8
<b>Data Storage Concepts</b> .....	2-18
Data Storage Concepts Table.....	2-19
<b>Data Manipulation Language</b> .....	2-23
Connecting to the Database .....	2-24
SELECT Statement .....	2-25

SELECT Statements without FROM Clauses:.....	2-28
SELECT INTO Statement:.....	2-28
Column Aliases: .....	2-28
Table Aliases: .....	2-29
Compute:.....	2-29
SELECT with GROUP BY Statement.....	2-29
INSERT Statement.....	2-29
UPDATE Statement .....	2-30
Method 1 - Convert UPDATE statements with FROM clauses:.....	2-31
Method 2 - Convert UPDATE statements with FROM clauses:.....	2-32
DELETE Statement.....	2-32
Remove Second FROM Clause: .....	2-33
Operators.....	2-34
Comparison Operators.....	2-34
Arithmetic Operators .....	2-37
String Operators.....	2-38
Set Operators .....	2-38
Bit Operators.....	2-39
Built-In Functions.....	2-39
Character Functions.....	2-39
Miscellaneous Functions.....	2-42
Defining Functions in Oracle: .....	2-42
Date Functions.....	2-42
Mathematical Functions.....	2-44
Locking Concepts and Data Concurrency Issues.....	2-45
Locking.....	2-45
Row-Level Versus Page-Level Locking.....	2-48
Read Consistency.....	2-49
Logical Transaction Handling.....	2-50

### 3 Triggers and Stored Procedures

<b>Triggers</b> .....	3-1
<b>Stored Procedures</b> .....	3-3
Individual SQL Statements.....	3-3
Microsoft SQL Server: .....	3-4
Oracle: .....	3-4
Microsoft SQL Server: .....	3-4
Oracle: .....	3-4
Logical Transaction Handling.....	3-5
Transaction-Handling Statements.....	3-5
Error Handling Within the Stored Procedure.....	3-5
RAISERROR Statement.....	3-6
Customized Error Messages.....	3-6
<b>Data Types</b> .....	3-7
Local Variable.....	3-7
Server Data Types.....	3-7
Composite Data Types.....	3-7

<b>Schema Objects</b> .....	3-8
Procedure .....	3-8
Create.....	3-9
Drop .....	3-10
Execute.....	3-11
Alter .....	3-13
Function.....	3-14
Create.....	3-15
Drop .....	3-16
Execute.....	3-17
Alter .....	3-17
Package .....	3-18
Create.....	3-19
Drop .....	3-20
Alter .....	3-21
Package Body.....	3-22
Create.....	3-22
Drop .....	3-24
Alter .....	3-24
<b>T/SQL Versus PL/SQL Constructs</b> .....	3-25
CREATE PROCEDURE Statement .....	3-27
Parameter Passing.....	3-27
DECLARE Statement.....	3-28
IF Statement .....	3-29
RETURN Statement .....	3-31
RAISERROR Statement .....	3-32
EXECUTE Statement.....	3-32
WHILE Statement .....	3-33
GOTO Statement .....	3-37
@@Rowcount and @@Error Variables.....	3-38
ASSIGNMENT Statement.....	3-39
SELECT Statement .....	3-39
SELECT Statement with GROUP BY Clause.....	3-40
Column Aliases .....	3-41
UPDATE with FROM Statement .....	3-42
DELETE with FROM Statement.....	3-42
Temporary Tables .....	3-43
Cursor Handling .....	3-44
<b>T/SQL and PL/SQL Language Elements</b> .....	3-44
Transaction Handling Semantics .....	3-45
Conversion Preparation Recommendations .....	3-47
Exception-Handling and Error-Handling Semantics.....	3-49
Special Global Variables.....	3-50
Operators.....	3-51
Built-in Functions .....	3-51
DDL Constructs within Microsoft SQL Server Stored Procedures.....	3-51

## 4 Distributed Environments

<b>Distributed Environments</b> .....	4-1
Accessing Remote Databases in a Distributed Environment .....	4-1
Oracle and Remote Objects.....	4-2
Microsoft SQL Server and Remote Objects .....	4-2
Replication.....	4-3
<b>Application Development Tools</b> .....	4-3

## 5 Disconnected Source Model Loading

<b>Generating Database Metadata Flat Files</b> .....	5-1
Flat File Generation Scripts.....	5-1
Running the Scripts .....	5-2

## Index

---

---

# Preface

*Oracle SQL Developer Supplementary Information for Microsoft SQL Server Migrations* provides detailed information about migrating a database from Microsoft SQL Server 6.5, 7.0, or 2000 to Oracle. It is a useful guide regardless of the conversion tool you are using to perform the migration, but the recommended tool for such migrations is Oracle Migration Workbench (Migration Workbench). This reference guide describes several differences between Microsoft SQL Server and Oracle and outlines how those differences are handled by the Migration Workbench during the conversion process.

## Intended Audience

This guide is intended for anyone who is involved in converting a Microsoft SQL Server database to Oracle using the Migration Workbench.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

### **Accessibility of Code Examples in Documentation**

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### **Accessibility of Links to External Web Sites in Documentation**

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

## TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

## What You Should Already Know

You should be familiar with relational database concepts and with the operating system environments under which you are running Oracle and Microsoft SQL Server.

## Related Documents

For more information, see these Oracle SQL Developer resources:

- *Oracle SQL Developer Frequently Asked Questions (FAQ)*
- *Oracle Migration Workbench Release Notes*
- Oracle SQL Developer Online Help

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and you can do it at:

<http://otn.oracle.com/membership/index.htm>

If you already have a user name and password for OTN, then you can go directly to the SQL Developer documentation section of the OTN Web site at:

<http://otn.oracle.com/tech/migration/workbench>

## Conventions

The following text conventions are used in this document:

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

## Third-Party License Information

Oracle SQL Developer contains third-party code. Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the third-party software, and the terms contained in the following notices do not change those rights.

### Apache Regular Expression Package 2.0

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at: <http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

### **Antlr v 2.7.3**

<http://www.antlr.org/rights.html>

OracleAS TopLink uses Antlr for EJB QL parsing. Antlr (ANother Tool for Language Recognition), is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions. The ANTLR parser and translator generator is fully in the public domain.

### **JGoodies Looks and Forms**

Copyright © 2003 JGoodies Karsten Lentzsch. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of JGoodies Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



---

---

# Overview

This chapter introduces Oracle SQL Developer (SQL Developer). It contains the following sections:

- [Introduction](#)
- [Product Description](#)
- [Features](#)
- [Glossary](#)

## Introduction

SQL Developer is a tool that simplifies the process of migrating data and applications from a Microsoft SQL Server environment to an Oracle database. SQL Developer allows you to quickly and easily migrate an entire application system, that is the database schema including triggers and stored procedures, in an integrated, visual environment.

---

---

**Note:** Microsoft SQL Server is used in this document to refer to Microsoft SQL Server 6.5, Microsoft SQL Server 7.0, and Microsoft SQL Server 2000 unless otherwise stated.

---

---

## Product Description

SQL Developer allows you to migrate a Microsoft SQL Server database to an Oracle database. SQL Developer employs an intuitive and informative user interface to simplify the migration process.

SQL Developer uses a repository to store migration information. This allows you to query the initial state of the application before migration. By initially loading the components of the application system that can be migrated into a repository, you can work independently of the production application.

Furthermore, SQL Developer saves useful dependency information about the components you are converting. For example, SQL Developer keeps a record of all the tables accessed by a stored procedure. You can then use this information to understand the impact of modifying a given table.

## Features

SQL Developer Release 1.2 includes core features and Microsoft SQL Server migration specific features. SQL Developer allows you to:

- Migrate a complete Microsoft SQL Server database to an Oracle database.
- Migrate groups, users, tables, primary keys, foreign keys, unique constraints, indexes, rules, check constraints, views, triggers, stored procedures, user-defined types, and privileges to Oracle.
- Migrate multiple Microsoft SQL Server source databases to a single Oracle database.
- Customize the parser for stored procedures, triggers, or views.
- Generate the Oracle SQL\*Loader and SQL Server BCP scripts for offline data loading.
- Display a representation of the source database and its Oracle equivalent.
- Generate and view a summary report of the migration.
- Customize users, tables, indexes, and tablespaces.
- Customize the default data type mapping rules.
- Create ANSI-compliant names.
- Automatically resolve conflicts such as Oracle reserved words.
- Remove and rename objects in the Oracle Model.

## Glossary

The following terms are used to describe SQL Developer:

*Dependency* is used to define a relationship between two migration entities. For example, a database view is dependent upon the table it references.

*Destination Database* is the Oracle database to which SQL Developer migrates the data dictionary of the source database.

*Migration Component* is part of an application system that can be migrated to an Oracle database. Examples of migration components are tables and stored procedures.

*Migration Entity* is an instance of a migration component. The table EMP would be a migration entity belonging to the table MIGRATION COMPONENT.

*Migration Repository* is the area in an Oracle database used to store the persistent information necessary for SQL Developer to migrate a source database.

*Navigator Pane* is the part of the SQL Developer User Interface that contains the tree views representing the Source Model and the Oracle Model.

*Oracle Model* is a series of Oracle tables that is created from the information in the Source Model. It is a visual representation of how the source database looks when generated in an Oracle environment.

*Properties Pane* is the part of the SQL Developer User Interface that displays the properties of a migration entity that has been selected in one of the tree views in the Navigator Pane.

*Progress Window* is the part of the SQL Developer User Interface that contains informational, error, or warning messages describing the progress of the migration process.

*Software Development Kit (SDK)* is a set of well-defined application programming interfaces (APIs) that provide services that a software developer can use.

*Source Database* is the database containing the data dictionary of the application system being migrated by SQL Developer. The source database is a database other than Oracle, for example, Microsoft SQL Server.

*Source Model* is a replica of the data dictionary of the source database. It is stored in the Oracle SQL Developer Repository and is loaded by SQL Developer with the contents of the data dictionary of the source database.



---

---

# Microsoft SQL Server and Oracle Compared

This chapter contains information comparing the Microsoft SQL Server database and the Oracle database. It contains the following sections:

- [Schema Migration](#)
- [Data Types](#)
- [Data Storage Concepts](#)
- [Data Manipulation Language](#)

## Schema Migration

The schema contains the definitions of the tables, views, indexes, users, constraints, stored procedures, triggers, and other database-specific objects. Most relational databases work with similar objects.

The schema migration topics discussed here include the following:

- [Schema Object Similarities](#)
- [Schema Object Names](#)
- [Table Design Considerations](#)

## Schema Object Similarities

There are many similarities between schema objects in Oracle and schema objects in Microsoft SQL Server. However, some schema objects differ between these databases, as shown in the following table:

**Table 2-1 Schema Objects in Oracle and Microsoft SQL Server**

<b>Oracle</b>	<b>Microsoft SQL Server</b>
Database	Database
Schema	Database and database owner (DBO)
Tablespace	Database
User	User
Role	Group/Role
Table	Table
Temporary tables	Temporary tables
Cluster	N/A
Column-level check constraint	Column-level check constraint
Column default	Column default
Unique key	Unique key or identity property for a column
Primary key	Primary key
Foreign key	Foreign key
Index	Non-unique index
PL/SQL Procedure	Transact-SQL (T-SQL) stored procedure
PL/SQL Function	T-SQL stored procedure
Packages	N/A
AFTER triggers	Triggers
BEFORE triggers	Complex rules
Triggers for each row	N/A
Synonyms	N/A
Sequences	Identity property for a column
Snapshot	N/A
View	View

## Schema Object Names

Reserved words differ between Oracle and Microsoft SQL Server. Many Oracle reserved words are valid object or column names in Microsoft SQL Server. For example, DATE is a reserved word in Oracle, but it is not a reserved word in Microsoft SQL Server. Therefore, no column is allowed to have the name DATE in Oracle, but a column can be named DATE in Microsoft SQL Server. Use of reserved words as schema object names makes it impossible to use the same names across databases.

You should choose a schema object name that is unique by case and by at least one other characteristic, and ensure that the object name is not a reserved word from either database.

For a list of reserved words in Oracle, see *Oracle Database SQL Language Reference*.

## Table Design Considerations

This section discusses the many table design issues that you need to consider when converting Microsoft SQL Server databases to Oracle. These issues are discussed under the following headings:

- [Data Types](#)
- [Entity Integrity Constraints](#)
- [Referential Integrity Constraints](#)
- [Unique Key Constraints](#)
- [Check Constraints](#)

### Data Types

This section describes conversion considerations for the following data types:

- [DATETIME Data Types](#)
- [IMAGE and TEXT Data Types \(Binary Large Objects\)](#)
- [Microsoft SQL Server User-Defined Data Types](#)

**DATETIME Data Types** The date/time precision in Microsoft SQL Server is 1/300th of a second. Oracle has the data type `TIMESTAMP` which has a precision of 1/100000000th of a second. Oracle also has a `DATE` data type that stores date and time values accurate to one second. SQL Developer has a default mapping to the `DATE` data type.

For applications that require finer date/time precision than seconds, the `TIMESTAMP` data type should be selected for the data type mapping of date data types in Microsoft SQL Server. The databases store point-in-time values for `DATE` and `TIME` data types.

As an alternative, if a Microsoft SQL Server application uses the `DATETIME` column to provide unique IDs instead of point-in-time values, replace the `DATETIME` column with a `SEQUENCE` in the Oracle schema definition.

In the following examples, the original design does not allow the `DATETIME` precision to exceed seconds in the Oracle table. This example assumes that the `DATETIME` column is used to provide unique IDs. If millisecond precision is not required, the table design outlined in the following example is sufficient:

### Original Table Design

#### Microsoft SQL Server:

```
CREATE TABLE example_table
(datetime_column    datetime        not null,
text_column        text                null,
varchar_column     varchar(10)         null)
```

#### Oracle:

```
CREATE TABLE example_table
(datetime_column    date                not null,
text_column        long                null,
varchar_column     varchar2(10)        null)
```

The following design allows the value of the sequence to be inserted into the `integer_column`. This allows you to order the rows in the table beyond the allowed precision of one second for `DATE` data type fields in Oracle. If you include this column in the Microsoft SQL Server table, you can keep the same table design for the Oracle database.

### Revised Table Design

#### Microsoft SQL Server:

```
CREATE TABLE example_table
(datetime_column    datetime        not null,
integer_column     int              null,
text_column        text                null,
varchar_column     varchar(10)         null)
```

**Oracle:**

```
CREATE TABLE example_table
(datetime_column  date           not null,
integer_column   number         null,
text_column      long           null,
varchar_column   varchar2(10)   null)
```

For the Microsoft SQL Server database, the value in the `integer_column` is always NULL. For Oracle, the value for the field `integer_column` is updated with the next value of the sequence.

Create the sequence by issuing the following command:

```
CREATE SEQUENCE datetime_seq
```

Values generated for this sequence start at 1 and are incremented by 1.

Many applications do not use DATETIME values as UNIQUE IDs, but still require the date/time precision to be higher than seconds. For example, the timestamp of a scientific application may have to be expressed in milliseconds, microseconds, and nanoseconds. The precision of the Microsoft SQL Server DATETIME data type is 1/300th of a second; the precision of the Oracle DATE data type is one second. The Oracle TIMESTAMP data type has a precision to 1/100000000th of a second. However, the precision recorded is dependent on the operating system.

**IMAGE and TEXT Data Types (Binary Large Objects)**

The physical and logical storage methods for IMAGE and TEXT data differ from Oracle to Microsoft SQL Server. In Microsoft SQL Server, a pointer to the IMAGE or TEXT data is stored with the rows in the table while the IMAGE or TEXT data is stored separately. This arrangement allows multiple columns of IMAGE or TEXT data per table. In Oracle, IMAGE data may be stored in a BLOB type field and TEXT data may be stored in a CLOB type field. Oracle allows multiple BLOB and CLOB columns per table. BLOBS and CLOBS may or may not be stored in the row depending on their size.

If the Microsoft SQL Server TEXT column is such that the data never exceeds 4000 bytes, convert the column to an Oracle VARCHAR2 data type column instead of a CLOB column. An Oracle table can define multiple VARCHAR2 columns. This size of TEXT data is suitable for most applications.

**Microsoft SQL Server User-Defined Data Types**

This Microsoft SQL Server T-SQL-specific enhancement to SQL allows users to define and name their own data types to supplement the system data types. A user-defined data type can be used as the data type for any column in the database. Defaults and rules (check constraints) can be bound to these user-defined data types, which are applied automatically to the individual columns of these user-defined data types.

When migrating to Oracle PL/SQL, the Migration Workbench determines the base data type for each user-defined data type, and it finds the equivalent PL/SQL data type.

---

---

**Note:** User-defined data types make the data definition language code and procedural SQL code less portable across different database servers.

---

---

### Entity Integrity Constraints

You can define a primary key for a table in Microsoft SQL Server. Primary keys can be defined in a `CREATE TABLE` statement or an `ALTER TABLE` statement.

Oracle provides declarative referential integrity. A primary key can be defined as part of a `CREATE TABLE` or an `ALTER TABLE` statement. Oracle internally creates a unique index to enforce the integrity.

### Referential Integrity Constraints

You can define a foreign key for a table in Microsoft SQL Server. Foreign keys can be defined in a `CREATE TABLE` statement or an `ALTER TABLE` statement.

Oracle provides declarative referential integrity. A `CREATE TABLE` or `ALTER TABLE` statement can add foreign keys to the table definition. For information about referential integrity constraints, see *Oracle Database Concepts*.

### Unique Key Constraints

You can define a unique key for a table in Microsoft SQL Server. Unique keys can be defined in a `CREATE TABLE` statement or an `ALTER TABLE` statement.

Oracle defines unique keys as part of `CREATE TABLE` or `ALTER TABLE` statements. Oracle internally creates unique indexes to enforce these constraints.

Unique keys map one-to-one from Microsoft SQL Server to Oracle.

## Check Constraints

Check constraints can be defined in a CREATE TABLE statement or an ALTER TABLE statement in Microsoft SQL Server. Multiple check constraints can be defined on a table. A table-level check constraint can reference any column in the constrained table. A column can have only one check constraint. A column-level check constraint can reference only the constrained column. These check constraints support complex regular expressions.

Oracle defines check constraints as part of the CREATE TABLE or ALTER TABLE statements. A check constraint is defined at the TABLE level and not at the COLUMN level. Therefore, it can reference any column in the table. Oracle, however, does not support complex regular expressions.

### SQL Server Rule:

```
create rule phone_rule
as
@phone_number like
"([0-9][0-9][0-9])[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]"
```

This rule passes all the phone numbers that resemble the following:  
(650)506-7000

This rule fails all the phone numbers that resemble the following:

650-506-7000  
650-GET-HELP

There are a few ways to implement this INTEGRITY constraint in Oracle:

- Simulate the behavior of phone-rule in a check constraint using a combination of SUBSTR, TRANSLATE, and LIKE clauses
- Write a trigger and use PL/SQL

Table-level check constraints from Microsoft SQL Server databases map one-to-one with Oracle check constraints. You can implement the column-level check constraints from the Microsoft SQL Server database to Oracle table-level check constraints. While converting the regular expressions, convert all simple regular expressions to check constraints in Oracle. Microsoft SQL Server check constraints with complex regular expressions can be either reworked as check constraints including a combination of simple regular expressions, or you can write Oracle database triggers to achieve the same functionality.

## Data Types

This chapter provides detailed descriptions of the differences in data types used by Microsoft SQL Server and Oracle databases. Specifically, this chapter contains the following information:

- A table showing the base Microsoft SQL Server data types available and how they are mapped to Oracle data types
- Recommendations based on the information listed in the table

### Data Types Table

**Table 2–2 Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
INTEGER	Four-byte integer, 31 bits, and a sign. May be abbreviated as "INT" (this abbreviation was required prior to version 5).	NUMBER(10)	It is possible to place a table constraint on columns of this type (as an option) to force values between $-2^{31}$ and $2^{31}$ . Or, place appropriate constraints such as: STATE_NO between 1 and 50
SMALLINT	Two-byte integer, 15 bits, and a sign.	NUMBER(6)	It is possible to place a table constraint on columns of this type (optionally) to force values between $-2^{15}$ and $2^{15}$ . Or, place appropriate constraints such as: STATE_NO between 1 and 50
TINYINT	One byte integer, 8 bits and no sign. Holds whole numbers between 0 and 255.	NUMBER(3)	You may add a check constraint of (x between 0 and 255) where x is column name.

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
REAL	<p>Floating point number. Storage is four bytes and has a binary precision of 24 bits, a 7-digit precision.</p> <p>Data can range from <math>-3.40E+38</math> to <math>3.40E+38</math>.</p>	FLOAT	<p>The ANSI data type conversion to Oracle for REAL is FLOAT(63). By default, the Oracle Migration Workbench maps REAL to FLOAT(24) that stores up to 8 significant decimal digits in Oracle.</p> <p>The Oracle NUMBER data type is used to store both fixed and floating-point numbers in a format that is compatible with decimal arithmetic. You may want to add a check constraint to constrain range of values. Also, you get different answers when performing operations on this data type as the Oracle NUMBER type is more precise and portable than REAL. Floating-point numbers can be specified in Oracle in the following format: FLOAT[(b)]. Where [(b)] is the binary precision b and can range from 1 to 126. [(b)] defaults to 126. To check what a particular binary precision is in terms of decimal precision, multiply [(b)] by 0.30103 and round up to the next whole number.</p>

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
FLOAT	A floating point number. This column has 15-digit precision.	FLOAT	<p>The ANSI data type conversion to Oracle for FLOAT(p) is FLOAT(p). The ANSI data type conversion to Oracle for DOUBLE PRECISION is FLOAT(126). By default, the Oracle Migration Workbench maps FLOAT to FLOAT(53), that stores up to 16 significant decimal digits in Oracle.</p> <p>The Oracle NUMBER data type is used to store both fixed and floating-point numbers in a format compatible with decimal arithmetic. You get different answers when performing operations on this type due to the fact that the Oracle NUMBER type is much more precise and portable than FLOAT, but it does not have the same range. The NUMBER data type data can range from -9.99.99E+125 to 9.99.99E+125 (38 nines followed by 88 zeros).</p> <p>NOTE: If you try to migrate floating point data greater than or equal to 1.0E+126 then Migration Workbench will fail to insert this data in the Oracle database and will return an error. This also applies to negative values less than or equal to -1.0E+126.</p>

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
			<p>Floating-point numbers can be specified in Oracle using <code>FLOAT[(b)]</code>, where [(b)] is the binary precision [(b)] and can range from 1 to 126. [(b)] defaults to 126. To check what a particular binary precision is in terms of decimal precision multiply [(b)] by 0.30103 and round up to the next whole number.</p> <p>If they are outside of the range, large floating-point numbers will overflow, and small floating-point numbers will underflow.</p>
BIT	A Boolean 0 or 1 stored as one bit of a byte. Up to 8-bit columns from a table may be stored in a single byte, even if not contiguous. Bit data cannot be NULL, except for Microsoft SQL Server 7.0, where null is allowed by the BIT data type.	NUMBER(1)	In Oracle, a bit is stored in a number(1) (or char). In Oracle, it is possible to store bits in a char or varchar field (packed) and supply PL/SQL functions to set / unset / retrieve / query on them.
CHAR(n)	Fixed-length string of exactly n 8-bit characters, blank padded. Synonym for CHARACTER. 0 < n < 256 for Microsoft SQL Server. 0 < n < 8000 for Microsoft SQL Server 7.0.	CHAR(n)	<p>Pro*C client programs must use <code>mode=ansi</code> to have characters interpreted correctly for string comparison, <code>mode=oracle</code> otherwise.</p> <p>A CHAR data type with a range of 2001 to 4000 is invalid. SQL Developer automatically converts a CHAR datatype with this range to VARCHAR2.</p>

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Description</b>	<b>Oracle</b>	<b>Comments</b>
VARCHAR(n)	Varying-length character string. 0 < n < 256 for Microsoft SQL Server. 0 < n < 8000 for Microsoft SQL Server 7.0.	VARCHAR2(n)	
TEXT	Character string of 8-bit bytes allocated in increments of 2k pages. "TEXT" is stored as a linked-list of 2024-byte pages, blank padded. TEXT columns can hold up to (231-1) characters.	CLOB	The CLOB field can hold up to 4GB.
IMAGE	Binary string of 8-bit bytes. Holds up to (231-1) bytes of binary data.	BLOB	The BLOB field can hold up to 4GB.
BINARY(n)	Fixed length binary string of exactly n 8-bit bytes. 0 < n < 256 for Microsoft SQL Server. 0 < n < 8000 for Microsoft SQL Server 7.0.	RAW(n)/BLOB	
VARBINARY(n)	Varying length binary string of up to n 8-bit bytes. 0 < n < 256 for Microsoft SQL Server. 0 < n < 8000 for Microsoft SQL Server 7.0.	RAW(n)/BLOB	

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
DATETIME	Date and time are stored as two 4-byte integers. The date portion is represented as a count of the number of days offset from a baseline date (1/1/1900) and is stored in the first integer. Permitted values are legal dates between 1st January, 1753 AD and 31st December, 9999 AD. Permitted values in the time portion are legal times in the range 0 to 25920000. Accuracy is to the nearest 3.33 milliseconds with rounding downward. Columns of type DATETIME have a default value of 1/1/1900.	DATE	<p>The precision of DATE in Oracle and DATETIME in Microsoft SQL Server is different. The DATETIME data type has higher precision than the DATE data type. This may have some implications if the DATETIME column is supposed to be UNIQUE. In Microsoft SQL Server, the column of type DATETIME can contain UNIQUE values because the DATETIME precision in Microsoft SQL Server is to the hundredth of a second. In Oracle, however, these values may not be UNIQUE as the date precision is to the second. You can replace a DATETIME column with two columns, one with data type DATE and another with a sequence, in order to get the UNIQUE combination. It is preferable to store hundredths of seconds in the second column.</p> <p>The Oracle TIMESTAMP data type can also be used. It has a precision of 1/10000000th of a second.</p>
SMALL-DATETIME	Date and time stored as two 2-byte integers. Date ranges from 1/1/1900 to 6/6/2079. Time is the count of the number of minutes since midnight.	DATE	With optional check constraint to validate the smaller range.

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Description</b>	<b>Oracle</b>	<b>Comments</b>
MONEY	<p>A monetary value represented as an integer portion and a decimal fraction, and stored as two 4-byte integers. Accuracy is to the nearest 1/10,000. When inputting Data of this type it should be preceded by a dollar sign (\$). In the absence of the "\$" sign, Microsoft SQL Server creates the value as a float.</p> <p>Monetary data values can range from -922,337,203,685,477.5808 to 922,337,203,685,477.5807, with accuracy to a ten-thousandth of a monetary unit. Storage size is 8 bytes.</p>	NUMBER(19,4)	<p>Microsoft SQL Server inputs MONEY data types as a numeric data type with a preceding dollar sign (\$) as in the following example, select * from table_x where y &gt; \$5.00 You must remove the "\$" sign from queries. Oracle is more general and works in international environments where the use of the "\$" sign cannot be assumed. Support for other currency symbols and ISO standards through NLS is available in Oracle.</p>

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
NCHAR(n)	<p>Fixed-length character data type which uses the UNICODE UCS-2 character set. n must be a value in the range 1 to 4000. SQL Server storage size is two times n.</p> <p>Note: Microsoft SQL Server storage size is two times n. The Oracle Migration Workbench maps columns sizes using byte semantics, and the size of Microsoft SQL Server NCHAR data types appear in the Oracle Migration Workbench Source Model with "Size" specifying the number of bytes, as opposed to the number of Unicode characters. Thus, a SQL Server column NCHAR(1000) will appear in the Source Model as NCHAR(2000).</p>	CHAR(n*2)	

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Description</b>	<b>Oracle</b>	<b>Comments</b>
NVARCHAR(n)	Fixed-length character data type which uses the UNICODE UCS-2 character set. n must be a value in the range 1 to 4000. SQL Server storage size is two times n.  Note: Microsoft SQL Server storage size is two times n. The Oracle Migration Workbench maps column sizes using byte semantics, and the size of Microsoft SQL Server NVARCHAR data types appear in the Oracle Migration Workbench Source Model with "Size" specifying the number of bytes, as opposed to the number of Unicode characters. Thus, a SQL Server column NVARCHAR(1000) will appear in the Source Model as NVARCHAR(2000).	VARCHAR(n*2)	
SMALLMONEY	Same as MONEY except monetary data values from -214,748.3648 to +214,748.3647, with accuracy to one ten-thousandth of a monetary unit. Storage size is 4 bytes.	NUMBER(10,4)	Since the range is -214,748.3648 to 214,748.364, NUMBER(10,4) suffices for this field.

**Table 2–2 (Cont.) Data Types in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Description	Oracle	Comments
TIMESTAMP	TIMESTAMP is defined as VARBINARY(8) with NULL allowed. Every time a row containing a TIMESTAMP column is updated or inserted, the TIMESTAMP column is automatically increment by the system. A TIMESTAMP column may not be updated by users.	NUMBER	You must place triggers on columns of this type to maintain them. In Oracle you can have multiple triggers of the same type without having to integrate them all into one big trigger. You may want to supply triggers to prevent updates of this column to enforce full compatibility.
SYSNAME	VARCHAR(30) in Microsoft SQL Server. NVARCHAR(128) in Microsoft SQL Server 7.0.	VARCHAR2(30) and VARCHAR2(128) respectively.	

TEXT and IMAGE data types in Microsoft SQL Server follow these rules:

- The column of these data types cannot be indexed.
- The column cannot be a primary key.
- The column cannot be used in the GROUP BY, ORDER BY, HAVING, and DISTINCT clauses.
- IMAGE and TEXT data types can be referred to in the WHERE clause with the LIKE construct.
- IMAGE and TEXT data types can also be used with the SUBSTR and LENGTH functions.

In Microsoft SQL Server, only columns with variable-length data types can store NULL values. When you create a column that allows NULLs with a fixed-length data type, the column is automatically converted to a system variable-length data type, as illustrated in [Table 2–3](#). These variable-length data types are reserved system data types, and users cannot use them to create columns

**Table 2–3 Data Type Conversion for NULL Values**

Fixed-Length Data Type	Variable-Length Data Type
CHAR	VARCHAR
NCHAR	NVARCHAR
BINARY	VARBINARY
DATETIME, SMALLDATETIME	DATETIME
FLOAT	FLOATN
INT, SMALLINT, TINYINT	INTN
DECIMAL	DECIMALN
NUMERIC	NUMERICN
MONEY, SMALLMONEY	MONEYN

---

**Note:** The Oracle Migration Workbench Source Model will display table system data types for each column.

---

### Recommendations

In addition to the data types listed in Table 2-2, users can define their own data types in Microsoft SQL Server databases. These user-defined data types translate to the base data types that are provided by the server. They do not allow users to store additional types of data, but can be useful in implementing standard data types for an entire application.

You can map data types from Microsoft SQL Server to Oracle with the equivalent data types listed in Table 2–3. SQL Developer converts user-defined data types to their base type. You can define how the base type is mapped to an Oracle type in the Data Type Mappings page in the Options dialog.

## Data Storage Concepts

This section provides a detailed description of the conceptual differences in data storage for the Microsoft SQL Server and Oracle databases.

Specifically, it contains the following information:

- A table (Table 2–4) comparing the data storage concepts of Microsoft SQL Server, and Oracle databases
- Recommendations based on the information listed in the table

## Data Storage Concepts Table

**Table 2–4 Data Storage Concepts in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Database Devices:</b></p> <p>A database device is mapped to the specified physical disk files.</p>	<p><b>Data Files:</b></p> <p>One or more data files are created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the data files in a tablespace is the total storage capacity of the tablespace. The combined storage capacity of a the tablespaces in a database is the total storage capacity of the database. Once created, a data file cannot change in size. This limitation does not exist in Oracle.</p>
<p><b>Page:</b></p> <p>Many pages constitute a database device. Each page contains a certain number of bytes.</p>	<p><b>Data Block:</b></p> <p>One data block corresponds to a specific number of bytes, of physical database space, on the disk. The size of the data block can be specified when creating the database. A database uses and allocates free database space in Oracle data blocks.</p>
<p><b>Extent:</b></p> <p>Eight pages make one extent. Space is allocated to all the databases in increments of one extent at a time.</p>	<p><b>Extent:</b></p> <p>An extent is a specific number of contiguous data blocks, obtained in a single allocation.</p>
<p>N/A</p>	<p><b>Segments:</b></p> <p>A segment is a set of extents allocated for a certain logical structure. The extents of a segment may or may not be contiguous on disk, and may or may not span the data files.</p>

**Table 2–4 (Cont.) Data Storage Concepts in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>
<p><b>Segments (corresponds to Oracle Tablespace):</b></p> <p>A segment is the name given to one or more database devices. Segment names are used in CREATE TABLE and CREATE INDEX constructs to place these objects on specific database devices. Segments can be extended to include additional devices as and when needed by using the SP_EXTENDSEGMENT system procedure.</p> <p>The following segments are created along with the database:</p> <ul style="list-style-type: none"> <li>■ System segment Stores the system tables.</li> <li>■ Log segment Stores the transaction log.</li> <li>■ Default segment All other database objects are stored on this segment unless specified otherwise.</li> </ul> <p>Segments are subsets of database devices.</p>	<p><b>Tablespace (corresponds to Microsoft SQL Server Segments):</b></p> <p>A database is divided into logical storage units called tablespaces. A tablespace is used to group related logical structures together. A database typically has one system tablespace and one or more user tablespaces.</p> <p><b>Tablespace Extent:</b></p> <p>An extent is a specific number of contiguous data blocks within the same tablespace.</p> <p><b>Tablespace Segments:</b></p> <p>A segment is a set of extents allocated for a certain logical database object. All the segments assigned to one object must be in the same tablespace. The segments get the extents allocated to them as and when needed.</p> <p>There are four different types of segments as follows:</p> <ul style="list-style-type: none"> <li>■ Data segment Each table has a data segment. All of the table's data is stored in the extents of its data segments. The tables in Oracle can be stored as clusters as well. A cluster is a group of two or more tables that are stored together. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.</li> </ul>

**Table 2–4 (Cont.) Data Storage Concepts in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
	<p data-bbox="857 296 1170 323"><b>Tablespace Segments (Cont):</b></p> <ul style="list-style-type: none"> <li data-bbox="857 338 1305 415">■ Index segment Each index has an index segment that stores all of its data.</li> <li data-bbox="857 430 1305 716">■ Rollback segment One or more rollback segments are created by the DBA for a database to temporarily store "undo" information. This is the information about all the transactions that are not yet committed. This information is used to generate read-consistent database information during database recovery to rollback uncommitted transactions for users.</li> <li data-bbox="857 779 1305 989">■ Temporary segment Temporary segments are created by Oracle when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the extents in the temporary segment are returned to the system for future use.</li> </ul>
<p data-bbox="342 1010 488 1037"><b>Log Devices:</b></p> <p data-bbox="342 1052 829 1129">These are logical devices assigned to store the log. The database device to store the logs can be specified while creating the database.</p>	<p data-bbox="857 1010 1024 1037"><b>Redo Log Files:</b></p> <p data-bbox="857 1052 1305 1287">Each database has a set of two or more redo log files. All changes made to the database are recorded in the redo log. Redo log files are critical in protecting a database against failures. Oracle allows mirrored redo log files so that two or more copies of these files can be maintained. This protects the redo log files against failure of the hardware the log file reside on.</p>

**Table 2–4 (Cont.) Data Storage Concepts in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>
<p><b>Database Devices:</b></p> <p>A database device contains the database objects. A logical device does not necessarily refer to any particular physical disk or file in the file system.</p> <p>The database and logs are stored on database devices. Each database device must be initialized before being used for database storage. Initialization of the database device initializes the device for storage and registers the device with the server. After initialization, the device can be:</p> <ul style="list-style-type: none"> <li>■ Allocated to the free space available to a database</li> <li>■ Allocated to store specific user objects</li> <li>■ Used to store the transaction log of a database</li> <li>■ Labeled as default device to create and alter database objects</li> </ul> <p>The SP_HELPDEVICES system procedure displays all the devices that are registered with the server. Use the DROP DEVICE DEVICE_NAME command to drop the device. The system administrator (SA) should restart the server after dropping the device.</p> <p>A device can be labeled as a default device so that the new databases need not specify the device at the time of creation. Use the SP_DISKDEFAULT system procedure to label the device as a default device.</p>	<p>N/A</p>
<p><b>Dump Devices</b></p> <p>These are logical devices. A database dump is stored on these devices. The DUMP DATABASE command uses the dump device to dump the database.</p>	<p>N/A</p>

**Table 2–4 (Cont.) Data Storage Concepts in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p data-bbox="857 296 1003 323"><b>Control Files:</b></p> <p data-bbox="857 338 1284 443">Each database has a control file. This file records the physical structure of the database. It contains the following information:</p> <ul data-bbox="857 457 1279 590" style="list-style-type: none"> <li data-bbox="857 457 1062 485">■ database name</li> <li data-bbox="857 499 1279 548">■ names and locations of a database's data files and redo log files</li> <li data-bbox="857 562 1240 590">■ time stamp of database creation</li> </ul> <p data-bbox="857 604 1317 863">It is possible to have mirrored control files. Each time an instance of an Oracle database is started, its control file is used to identify the database, the physical structure of the data, and the redo log files that must be opened for the database operation to proceed. The control file is also used for recovery if necessary. The control files hold information similar to the master database in Microsoft SQL Server.</p>

**Recommendations:**

The conceptual differences in the storage structures do not affect the conversion process directly. However, the physical storage structures need to be in place before conversion of the database begins.

Oracle and Microsoft SQL Server have a way to control the physical placement of a database object. In Microsoft SQL Server, you use the ON SEGMENT clause and in Oracle you use the TABLESPACE clause.

An attempt should be made to preserve as much of the storage information as possible when converting from Microsoft SQL Server to Oracle. The decisions that were made when defining the storage of the database objects for Microsoft SQL Server should also apply for Oracle. Especially important are initial object sizes and physical object placement.

## Data Manipulation Language

This section uses tables to compare the syntax and description of Data Manipulation Language (DML) elements in the Microsoft SQL Server and Oracle databases. Each

table is followed by a recommendations section based on the information in the tables. The following topics are presented in this section:

- [Connecting to the Database](#)
- [SELECT Statement](#)
- [SELECT with GROUP BY Statement](#)
- [INSERT Statement](#)
- [UPDATE Statement](#)
- [DELETE Statement](#)
- [Operators](#)
  - [Comparison Operators](#)
  - [Arithmetic Operators](#)
  - [String Operators](#)
  - [Set Operators](#)
  - [Bit Operators](#)
- [Built-In Functions](#)
  - [Character Functions](#)
  - [Date Functions](#)
  - [Mathematical Functions](#)
- [Locking Concepts and Data Concurrency Issues](#)
  - [Locking](#)
  - [Row-Level Versus Page-Level Locking](#)
  - [Read Consistency](#)
- [Logical Transaction Handling](#)

## Connecting to the Database

The statement illustrated in [Table 2–5](#) connects a user to a database.

**Table 2–5 Connecting to the Database in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>USE database_name</pre>	<p><b>Syntax:</b></p> <pre>CONNECT user_name/password SET role</pre>
<p><b>Description:</b></p> <p>A default database is assigned to each user. This database is made current when the user logs on to the server. A user executes the USE DATABASE_NAME command to switch to another database.</p>	

**Recommendations:**

This concept of connecting to a database is conceptually different in the Microsoft SQL Server and Oracle databases. A Microsoft SQL Server user can log on to the server and switch to another database residing on the server, provided the user has privileges to access that database. An Oracle Server controls only one database, so here the concept of a user switching databases on a server does not exist. Instead, in Oracle a user executes the SET ROLE command to change roles or re-issues a CONNECT command using a different user\_name.

**SELECT Statement**

The statement in [Table 2–6](#) retrieves rows from one or more tables or views.

**Table 2-6** *SELECT Statements in Oracle and Microsoft SQL Server*

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre> SELECT [ALL   DISTINCT] {select_list}   [INTO [owner.]table]   [FROM [owner.]{table   view}[alias]   [HOLDLOCK]   [, [owner.]{table   view }[alias]   [HOLDLOCK]]...]   [WHERE condition]   [GROUP BY [ALL] aggregate_free_expression   [, aggregate_free_expression]...]   [HAVING search_condition]   [UNION [ALL] SELECT...]   [ORDER BY {[owner.]{table   view }.}column     select_list_number   expression]   [ASC   DESC]   [, {[owner.]{table   view }.}column     select_list_number   expression]   [ASC   DESC]...]   [COMPUTE row_aggregate(column)   [, row_aggregate(column)...]   [BY column [, column...]]]   [FOR BROWSE]   The individual element in the select list   is as follows:   [alias = ]   { *   [owner.]{table   view}.*   SELECT ...     {[owner.]table.column   constant_literal     expression}   [alias]} </pre>	<p><b>Syntax:</b></p> <pre> SELECT [ALL   DISTINCT] {select_list} FROM [user.]{table   view } [@dblink] [alias] [, [user.]{table   view3} [@dblink] [alias]...] [WHERE condition] [CONNECT BY condition [START WITH condition]] [GROUP BY aggregate_free_expression [, aggregate_free_expression]...] [HAVING search_condition] [ {UNION [ALL]   INTERSECT   MINUS} SELECT ... ] [ORDER BY {expression   position} [ASC   DESC]...] [FOR UPDATE [OF {[user.]{table   view}.}column [, {[user.]{table   view}.}column... ] [noWAIT] ] The individual element in the select list is as follows: { *   [owner.]{table   view   snapshot   synonym}.*   {[owner.]table.column   constant_literal   expression } alias]} </pre>

**Table 2–6 (Cont.) SELECT Statements in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Description:</b></p> <p>DISTINCT eliminates the duplicate rows.</p> <p>The INTO clause and the items that follow it in the command syntax are optional, because Microsoft SQL Server allows SELECT statements without FROM clauses as can be seen in the following example:</p> <pre>SELECT getdate()</pre> <p>SELECT...INTO allows you to insert the results of the SELECT statement into a table.</p> <p>SELECT_LIST can contain a SELECT statement in the place of a column specification as follows:</p> <pre>SELECT d.empno, d.deptname, empname = (SELECT ename FROM emp            WHERE ename = d.empno) FROM dept d WHERE deptid = 10</pre> <p>The preceding example also shows the format for the column alias.</p> <pre>ALIAS = selected_column</pre> <p>COMPUTE attaches computed values at the end of the query. These are called row_aggregates.</p> <p>If a GROUP BY clause is used, all non-aggregate select columns are needed.</p> <p>FOR BROWSE keywords are used to get into browse mode. This mode supports the ability to perform updates while viewing data in an OLTP environment. It is used in front-end applications using DB-Library and a host programming language. Data consistency is maintained using the TIMESTAMP field in a multi-user environment. The selected rows are not locked; other users can view the same rows during the transaction. A user can update a row if the TIMESTAMP for the row is unchanged since the time of selection.</p>	<p><b>Description:</b></p> <p>DISTINCT eliminates the duplicate rows.</p> <p>The INSERT INTO &lt;table&gt; SELECT FROM... construct allows you to insert the results of the SELECT statement into a table.</p> <p>COLUMN ALIAS is defined by putting the alias directly after the selected COLUMN.</p> <p>If you use TABLE ALIAS, the TABLE must always be referenced using the ALIAS.</p> <p>You can also retrieve data from SYNONYMS.</p> <p>EXPRESSION could be a column name, a literal, a mathematical computation, a function, several functions combined, or one of several PSEUDO-COLUMNS.</p> <p>If a GROUP BY clause is used, all non-aggregate select columns must be in a GROUP BY clause.</p> <p>The FOR UPDATE clause locks the rows selected by the query. Other users cannot lock these row until you end the transaction. This clause is not a direct equivalent of the FOR BROWSE mode in Microsoft SQL Server.</p>

### **SELECT Statements without FROM Clauses:**

Microsoft SQL Server supports SELECT statements that do not have a FROM clause. This can be seen in the following example

```
SELECT getdate()
```

Oracle does not support SELECTs without FROM clauses. However, Oracle provides the DUAL table which always contains one row. Use the DUAL table to convert constructs such as the preceding one.

Translate the preceding query to:

```
SELECT sysdate FROM dual;
```

### **SELECT INTO Statement:**

The Microsoft SQL Server SELECT INTO statement can insert rows into a table. This construct, which is part SELECT and part INSERT, is not supported by ANSI. Replace these statements with INSERT...SELECT statements in Oracle.

If the Microsoft SQL Server construct is similar to the following:

```
SELECT col1, col2, col3
INTO target_table
FROM source_table
WHERE where_clause
```

you should convert it to the following for Oracle:

```
INSERT into target_table
SELECT col1, col2, col3
FROM source_table
WHERE where_clause
```

### **Column Aliases:**

Convert column aliases from the following Microsoft SQL Server syntax:

```
SELECT employees=col1 FROM table
```

to the following Oracle syntax:

```
SELECT col1 employees FROM table
```

---



---

**Note:** Microsoft SQL Server also supports Oracle-style column aliases.

---



---

**Table Aliases:**

Remove table aliases (also known as correlation names) unless they are used everywhere.

**Compute:**

Replace the `COMPUTE` clause with another `SELECT`. Attach the two sets of results using the `UNION` clause.

**SELECT with GROUP BY Statement**

Table 2-7 compares the `SELECT` with `GROUP BY` statement in Oracle to the same statement in Microsoft SQL Server.

**Table 2-7** *SELECT with GROUP BY Statement in Oracle and Microsoft SQL Server*

Microsoft SQL Server/Server	Oracle
<b>Syntax:</b> See the <code>SELECT</code> Statement section.	<b>Syntax:</b> See the <code>SELECT</code> Statement section.
<b>Description:</b> Non-aggregate <code>SELECT</code> columns must be in a <code>GROUP BY</code> clause.	<b>Description:</b> All non-aggregate <code>SELECT</code> columns must be in a <code>GROUP BY</code> clause.

**INSERT Statement**

The statements illustrated in the following table add one or more rows to the table or view.

**Table 2–8 INSERT Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>INSERT [INTO] [[database.]owner.] {table   view}{(column [, column]...)}{VALUES (expression [,expression]... )   query}</pre>	<p><b>Syntax:</b></p> <pre>INSERT INTO [user.]{table   view}[@dblink]{(column [, column]...)}{VALUES (expression [, expression]... )   query...};</pre>
<p><b>Description:</b></p> <p>INTO is optional.</p> <p>Inserts are allowed in a view provided only one of the base tables is undergoing change.</p>	<p><b>Description:</b></p> <p>INTO is required.</p> <p>Inserts can only be done on single table views.</p>

**Recommendations:**

INSERT statements in Microsoft SQL Server must be changed to include an INTO clause if it is not specified in the original statement.

The values supplied in the VALUES clause in either database may contain functions. The Microsoft SQL Server-specific functions must be replaced with the equivalent Oracle constructs.

---

**Note:** Oracle lets you create functions that directly match most Microsoft SQL Server functions.

---

Convert inserts that are inserting into multi-table views in Microsoft SQL Server to insert directly into the underlying tables in Oracle.

**UPDATE Statement**

The statement illustrated in [Table 2–9](#) updates the data in a table or the data in a table referenced by a view.

**Table 2–9 UPDATE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>UPDATE [[database.]owner.] {table   view} SET [[[database.]owner.] {table.   view.}] column = expression   NULL   (select_statement) [, column = expression   NULL   (select_statement)]... [FROM [[database.]owner.]table   view [, [[database.]owner.]table   view]... [WHERE condition]</pre>	<p><b>Syntax:</b></p> <pre>UPDATE [user.]{table   view} [@dblink] SET [[ user.] {table.   view.}] { column = expression   NULL   (select_ statement) [, column = expression   NULL   (select_statement)...]   (column [, column]...) = (select_statement)} [WHERE {condition   CURRENT OF cursor}]</pre>
<p><b>Description:</b></p> <p>The FROM clause is used to get the data from one or more tables into the table that is being updated or to qualify the rows that are being updated.</p> <p>Updates through multi-table views can modify only columns in one of the underlying tables.</p>	<p><b>Description:</b></p> <p>A single subquery may be used to update a set of columns. This subquery must select the same number of columns (with compatible data types) as are used in the list of columns in the SET clause.</p> <p>The CURRENT OF cursor clause causes the UPDATE statement to effect only the single row currently in the cursor as a result of the last FETCH. The cursor SELECT statement must have included in the FOR UPDATE clause.</p> <p>Updates can only be done on single table views.</p>

**Recommendations:**

There are two ways to convert UPDATE statements with FROM clauses, as indicated in the following sections.

**Method 1 - Convert UPDATE statements with FROM clauses:**

Use the subquery in the SET clause if columns are being updated to values coming from a different table.

Convert the following in Microsoft SQL Server:

```
update titles
SET pub_id = publishers.pub_id
```

```
FROM titles, publishers
WHERE titles.title LIKE 'C%'
AND publishers.pub_name = 'new age'
```

to the following in Oracle:

```
UPDATE titles

SET pub_id =
( SELECT a.pub_id
  FROM publishers a
  WHERE publishers.pub_name = 'new age'
)
WHERE titles.title like 'C%'
```

### **Method 2 - Convert UPDATE statements with FROM clauses:**

Use the subquery in the WHERE clause for all other UPDATE...FROM statements.

Convert the following in Microsoft SQL Server:

```
UPDATE shipping_parts
SET qty = 0
FROM shipping_parts sp, suppliers s
WHERE sp.supplier_num = s.supplier_num
AND s.location = "USA"
```

to the following in Oracle:

```
UPDATE shipping_parts
SET qty = 0
WHERE supplier_num IN (
SELECT supplier_num
FROM suppliers WHERE location = 'USA')
```

## **DELETE Statement**

The statement illustrated in [Table 2-10](#) removes rows from tables and rows from tables referenced in views.

**Table 2–10 DELETE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>DELETE [FROM] [[database.]owner.]{table   view} [FROM [[database.]owner.]{table   view} [, [[database.]owner.]{table   view}]...] [WHERE where_clause]</pre>	<p><b>Syntax:</b></p> <pre>DELETE [FROM] [user.]{table   view} [@dblink] [alias] [WHERE where_clause]</pre>
<p><b>Description:</b></p> <p>The first FROM in DELETE FROM is optional.</p> <p>The second FROM clause is a Microsoft SQL Server extension that allows the user to make deletions based on the data in other tables. A subquery in the WHERE clause serves the same purpose.</p> <p>Deletes can only be performed through single table views.</p>	<p><b>Description:</b></p> <p>FROM is optional.</p> <p>ALIAS can be specified for the table name as a correlation name, which can be used in the condition.</p> <p>Deletes can only be performed through single table views</p>

**Remove Second FROM Clause:**

Remove the second FROM clause from the DELETE statements.

Convert the following Microsoft SQL Server query:

```
DELETE
FROM sales
FROM sales, titles
WHERE sales.title_id = titles.title_id
AND titles.type = 'business'
```

to the following in Oracle:

```
DELETE
FROM sales
WHERE title_id in
( SELECT title_id
  FROM titles
  WHERE type = 'business')
```

)

Remove the second FROM even if the WHERE contains a multi-column JOIN.

Convert the following Microsoft SQL Server query:

```
DELETE
FROM sales
FROM sales, table_x
WHERE sales.a = table_x.a
      AND sales.b = table_x.b
      AND table_x.c = 'd'
```

to the following in Oracle:

```
DELETE
FROM sales
WHERE ( a, b ) in
( SELECT a, b
  FROM table_x
  WHERE c = 'd' )
```

## Operators

This section compares the different operators used in the Microsoft SQL Server and Oracle databases. It includes the following subsections:

- [Comparison Operators](#)
- [Arithmetic Operators](#)
- [String Operators](#)
- [Set Operators](#)
- [Bit Operators](#)

### Comparison Operators

[Table 2–11](#) compares the operators used in the Microsoft SQL Server, and Oracle databases. Comparison operators are used in WHERE clauses and COLUMN check constraints or rules to compare values.

**Table 2–11 Comparison Operators in Oracle and Microsoft SQL Server**

<b>Operator</b>	<b>Same in Both Databases</b>	<b>Microsoft SQL Server Only</b>	<b>Oracle Only</b>
Equal to	=		
Not equal to	!=		^=
	<>		
Less than	<		
Greater than	>		
Less than or equal to	<=	!>	
Greater than or equal to	>=	!<	
Greater than or equal to x and less than or equal to y	BETWEEN x AND y		
Less than x or greater than y	NOT BETWEEN x AND y		
Pattern Matches	LIKE 'a%'	LIKE'a[x-z]'	LIKE 'a\%'
a followed by 0 or more characters	LIKE 'a_'	LIKE'a[^x-z]'	ESCAPE '\'
a followed by exactly 1 character			
a followed by any character between x and z			
a followed by any character except those between x and z			
a followed by %			
Does not match pattern	NOT LIKE		
No value exists	IS NULL		
A value exists	IS NOT NULL		
At least one row returned by query	EXISTS (query)		
No rows returned by query	NOT EXISTS (query)		
Equal to a member of set	IN =ANY		= SOME

**Table 2–11 (Cont.) Comparison Operators in Oracle and Microsoft SQL Server**

Operator	Same in Both Databases	Microsoft SQL Server Only	Oracle Only
Not equal to a member of set	NOT IN ANY <> ANY	!=	!= SOME <> SOME
Less than a member of set	< ANY		< SOME
Greater than a member of set	> ANY		> SOME
Less than or equal to a member of set	<= ANY	!> ANY	<= SOME
Greater than or equal to a member of set	>= ANY	!< ANY	>= SOME
Equal to every member of set	= ALL		
Not equal to every member of set	!= ALL ALL	<>	
Less than every member of set	< ALL		
Greater than every member of set	> ALL		
Less than or equal to every member of set	<= ALL	!> ALL	
Greater than or equal to every member of set	>= ALL	!< ALL	

**Recommendations:**

1. Convert all !< and !> to >= and <=

Convert the following in Microsoft SQL Server:

```
WHERE col1 !< 100
```

to this for Oracle:

```
WHERE col1 >= 100
```

2. Convert like comparisons which use [ ] and [^]

```
SELECT title
FROM titles
```

```
WHERE title like "[A-F]%"
```

### 3. Change NULL constructs:

Table 2–12 shows that in Oracle, NULL is never equal to NULL. Change the all = NULL constructs to IS NULL to retain the same functionality.

**Table 2–12 Changing NULL Constructs**

NULL Construct	Microsoft SQL Server	Oracle
where col1 = NULL	depends on the data	FALSE
where col1 != NULL	depends on the data	TRUE
where col1 IS NULL	depends on the data	depends on the data
where col1 IS NOT NULL	depends on the data	depends on the data
where NULL = NULL	TRUE	FALSE

If you have the following in Microsoft SQL Server:

```
WHERE col1 = NULL
```

Convert it as follows for Oracle:

```
WHERE col1 IS NULL
```

## Arithmetic Operators

**Table 2–13 Arithmetic Operators in Oracle and Microsoft SQL Server**

Operator	Same in Both Databases	Microsoft SQL Server Only	Oracle Only
Add	+		
Subtract	-		
Multiply	*		
Divide	/		
Modulo	v	%	mod(x, y)

### Recommendations:

Replace any Modulo functions in Microsoft SQL Server with the mod() function in Oracle.

## String Operators

**Table 2–14** *String Operators in Oracle and Microsoft SQL Server*

Operator	Same in Both Databases	Microsoft SQL Server Only	Oracle Only
Concatenate	s	+	
Identify Literal	'this is a string'	"this is also a string"	

### Recommendations:

Replace all addition of strings with the || construct.

Replace all double quotes string identifiers with single quote identifiers.

In Microsoft SQL Server, an empty string (") is interpreted as a single space in INSERT or assignment statements on VARCHAR data. In concatenating VARCHAR, CHAR, or TEXT data, the empty string is interpreted as a single space. The empty string is never evaluated as NULL. You must bear this in mind when converting the application.

## Set Operators

**Table 2–15** *Set Operators in Oracle and Microsoft SQL Server*

Operator	Same in Both Databases	Microsoft SQL Server Only	Oracle Only
Distinct row from either query	UNION		
All rows from both queries	UNION ALL		
All distinct rows in both queries	d		INTERSECT
All distinct rows in the first query but not in the second query	d		MINUS

## Bit Operators

**Table 2–16** *Bit Operators in Oracle and Microsoft SQL Server*

Operator	Same in Both Databases	Microsoft SQL Server Only	Oracle Only
bit and		&	
bit or			
bit exclusive or		^	
bit not		~	

### Recommendations:

Oracle enables you to write the procedures to perform bitwise operations.

If you have the following Microsoft SQL Server construct:

```
X | Y : (Bitwise OR)
```

You could write a procedure called `dbms_bits.or(x,y)` and convert the preceding construct to the following in Oracle:

```
dbms_bits.or(x,y)
```

## Built-In Functions

This section compares the different functions used in the Microsoft SQL Server and Oracle databases. It includes the following subsections:

- [Character Functions](#)
- [Miscellaneous Functions](#)
- [Date Functions](#)
- [Mathematical Functions](#)

### Character Functions

**Table 2–17** *Character Functions in Oracle and Microsoft SQL Server*

Microsoft SQL Server	Oracle	Description
<code>ascii(char)</code>	<code>ascii(char)</code>	Returns the ASCII equivalent of the character.

**Table 2–17 (Cont.) Character Functions in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>	<b>Description</b>
chr(integer_expression)	chr(integer_expression)	Converts the decimal code for an ASCII character to the corresponding character.
charindex(specified_exp, char_string)	instr(specified_exp, char_string, 1, 1)	Returns the position where the specified_exp first occurs in the char_string.
convert(data type, expression, [format])	to_char, to_number, to_date, to_label, chartorowid, rowtochar, hextochar, chartohex	Converts one data type to another using the optional format. The majority of the functionality can be matched. Refer to <i>Oracle Database SQL Language Reference</i> for more information.
datalength(expression)	g	Computes the length allocated to an expression, giving the result in bytes.
difference(character_exp, character_exp)	d	Returns the numeric difference of the SOUNDEX values of the two strings.
isnull(variable, new_value)	nvl(variable, new_value)	If the value of the variable is NULL, the new_value is returned.
lower(char_exp)	lower(char_exp)	Converts uppercase characters to lowercase characters.
ltrim(char_exp)	ltrim(char_exp)	Truncates trailing spaces from the left end of char_exp.
patindex(pattern, column_name)	column_name)	Returns the position of the pattern in the column value. The pattern can have wild characters. This function also works on TEXT and BINARY data types.
replicate(char_exp, n)	rpad(char_exp, length(char_exp)*n, "")	Produces a string with char_exp repeated n times.
reverse(char_string)		Reverses the given char_string.

**Table 2-17 (Cont.) Character Functions in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>	<b>Description</b>
right(char_exp, n)	substr(char_exp, (length(char_exp)	Returns the part of the string starting at the position given by n from the right and extending up to the end of the string.
rtrim(char_exp)	rtrim(char_exp)	Truncates the trailing spaces from the right end of char_exp.
soundex(exp)	soundex(exp)	Returns phonetically similar expressions to the specified exp.
space(int_exp)	rpads(' ', int_exp-1, '')	Generates a string with int_exp spaces.
str(float_exp, length)	to_char(float_ exp)stuff(char_exp, start, length, replace_ str)substr(char_exp, 1, start)    replace_str   substr(char_exp, start+length)	Replaces a substring within char_exp with replace_str.
substring(char_exp, start, length)	substr(char_exp, start, length)	Replaces a substring within char_exp with replace_str.
Works on IMAGE and TEXT data types	Does not work with LONG and LONG_RAW data types	
textptr(column_name)	d	Returns a pointer as a varbinary(16) data type for a named IMAGE or TEXT column.
textvalid("column_name", text_pointer)	h	Returns 1 if the specified text_pointer is valid for the specified column_name. The column must be of type TEXT or IMAGE.
upper(char_exp)	upper(char_exp)	Converts lowercase characters to uppercase characters.

## Miscellaneous Functions

**Table 2–18 Comparison Operators in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle	Description
datalength(expression)	lengthb	Computes the length allocated to an expression, giving the result in bytes.
isnull(variable, new_value)	nvl(variable, new_value)	If the value of the variable is NULL, the new_value is returned.

---

**Note:** The preceding functions tables list all the Microsoft SQL Server character manipulation functions. They do not list all the Oracle functions. There are many more Oracle character manipulation functions that you can use.

---

### Defining Functions in Oracle:

Oracle adds the ability to define functions. With this feature you can create Oracle functions that match the name and function of Microsoft SQL Server functions.

## Date Functions

**Table 2–19 Date Functions in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle	Description
dateadd(dd, int_exp, datetime_var)	date+int_exp requires conversion of int_exp to a number of days	Adds the int_exp number of days to the date contained in datetime_var.
dateadd(mm, int_exp, datetime_var)	add_months(date, int_exp) or date+int_exp requires conversion of int_exp to a number of days	Adds the int_exp number of months to the date contained in datetime_var.
dateadd(yy, int_exp, datetime_var)	date+int_exp requires conversion of int_exp to a number of days	Adds the int_exp number of years to the date contained in datetime_var.

**Table 2–19 (Cont.) Date Functions in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle	Description
<code>datediff (dd, datetime1,datetime2)</code>	<code>date2-date1</code>	Returns the difference between the dates specified by the <code>datetime1</code> and <code>datetime2</code> variables. This difference is calculated in the number of days.
<code>datediff (mm, datetime1,datetime2)</code>	<code>months_between (date2, date1)</code>	Returns the difference between the dates specified by the <code>datetime1</code> and <code>datetime2</code> variables. This difference is calculated in the number of months.
<code>datediff (yy, datetime1,datetime2)</code>	<code>(date2-date1) /365.254</code>	Returns the difference between the dates specified by the <code>datetime1</code> and <code>datetime2</code> variables. This difference is calculated in the number of years.
<code>datetime (datepart, date)</code>	<code>to_char(date, format)</code>	Returns the specified part of the date as an integer. The Microsoft SQL Server DATETIME has a higher precision than Oracle DATE. For this reason, it is not always possible to find an equivalent format string in Oracle to match the datepart in Microsoft SQL Server. See the Data Types section of this chapter for more information about conversion of the DATETIME data type.
<code>datepart(datepart, date)</code>	<code>to_char(date, format)</code>	Returns the specified part of the date as a character string (name). The Microsoft SQL Server DATETIME has a higher precision than Oracle DATE. For this reason, it is not always possible to find an equivalent format string in Oracle to match the datepart in Microsoft SQL Server.
<code>getdate()</code>	<code>sysdate</code>	Returns the system date.

**Recommendations:**

The preceding table lists all the Microsoft SQL Server date manipulation functions. It does not list all the Oracle date functions. There are many more Oracle date manipulation functions that you can use.

It is recommended that you convert most date manipulation functions to "+" or "-" in Oracle.

Oracle adds the ability to define functions. With this feature you can create Oracle functions that match the name and functionality of all Microsoft SQL Server functions. This is a useful feature, where users can call a PL/SQL function from a SQL statement's SELECT LIST, WHERE clause, ORDER BY clause, and HAVING clause. With the parallel query option, Oracle executes the PL/SQL function in parallel with the SQL statement. Hence, users create parallel logic.

## Mathematical Functions

**Table 2–20** *Mathematical Functions in Oracle and Microsoft SQL Server*

<b>Microsoft SQL Server</b>	<b>Oracle</b>
abs(n)	abs(n)
acos(n)	acos(n)
asin(n)	
atan(n)	atan(n)
atan2(n,m)	
ceiling(n)	ceil(n)
cos(n)	cos(n)
cot(n)	
degrees(n)	
exp(n)	exp(n)
floor(n)	floor(n)
log(n)	ln(n)
log10(n)	log(base,number)
pi()	
power(m,n)	power(m,n)
radians(n)	
rand(n)	
round(n[,m])	round(n[,m])
sign(n)	sign(n)
sin(n)	sin(n)

**Table 2–20 (Cont.) Mathematical Functions in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
sqrt(n)	sqrt(n)
tan(n)	tan(n)

**Recommendations:**

The preceding table lists all the Microsoft SQL Server number manipulation functions. It does not list all the Oracle mathematical functions. There are many more Oracle number manipulation functions that you can use.

Oracle adds the ability to define functions. With this feature you can create Oracle functions that match the name and functionality of all Microsoft SQL Server functions. This is the most flexible approach. Users can write their own functions and execute them seamlessly from a SQL statement.

Oracle functions listed in the table work in SQL as well as PL/SQL.

## Locking Concepts and Data Concurrency Issues

This section compares locking and transaction handling in the Microsoft SQL Server and Oracle databases. It includes the following subsections:

- [Locking](#)
- [Row-Level Versus Page-Level Locking](#)
- [Read Consistency](#)
- [Logical Transaction Handling](#)

### Locking

Locking serves as a control mechanism for concurrency. Locking is a necessity in a multi-user environment because more than one user at a time may be working with the same data.

**Table 2–21 Locking in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>
<p>Microsoft SQL Server locking is fully automatic and does not require intervention by users.</p> <p>Microsoft SQL Server applies exclusive locks for INSERT, UPDATE, and DELETE operations. When an exclusive lock is set, no other transaction can obtain any type of lock on those objects until the original lock is in place.</p> <p>For non-update or read operations, a shared lock is applied. If a shared lock is applied to a table or a page, other transactions can also obtain a shared lock on that table or page. However, no transaction can obtain an exclusive lock. Therefore, Microsoft SQL Server reads block the modifications to the data.</p> <p><b>Update locks:</b></p> <p>Update locks are held at the page level. They are placed during the initial stages of an update operation when the pages are being read. Update locks can co-exist with shared locks. If the pages are changed later, the update locks are escalated to exclusive locks.</p>	<p>Oracle locking is fully automatic and does not require intervention by users. Oracle features the following categories of locks:</p> <p>Data locks (DML locks) to protect data. The "table locks" lock the entire table and "row locks" lock individual rows.</p> <p>Dictionary locks (DDL locks) to protect the structure of objects.</p> <p>Internal locks to protect internal structures, such as files.</p> <p>DML operations can acquire data locks at two different levels; one for specific rows and one for entire tables.</p> <p>Row-level locks:</p> <p>An exclusive lock is acquired for an individual row on behalf of a transaction when the row is modified by a DML statement. If a transaction obtains a row level lock, it also acquires a table (dictionary) lock for the corresponding table. This prevents conflicting DDL (DROP TABLE, ALTER TABLE) operations that would override data modifications in an on-going transaction.</p>

**Table 2–21 (Cont.) Locking in Oracle and Microsoft SQL Server**

<b>Microsoft SQL Server</b>	<b>Oracle</b>
<p><b>Intent locks:</b></p> <p>Microsoft SQL Server locking is fully automatic and does not require intervention by users. Microsoft SQL Server applies exclusive locks for INSERT, UPDATE, and DELETE operations. When an exclusive lock is set, no other transaction can obtain any type of lock on those objects until the original lock is in place. For non-update or read operations, a shared lock is applied. If a shared lock is applied to a table or a page, other transactions can also obtain a shared lock on that table or page. However, no transaction can obtain an exclusive lock. Therefore, Microsoft SQL Server reads block the modifications to the data.</p> <p><b>Extent locks:</b></p> <p>Extent locks lock a group of eight database pages while they are being allocated or freed. These locks are held during a CREATE or DROP statement, or during an INSERT that requires new data or index pages.</p> <p>A list of active locks for the current server can be seen with SP_LOCK system procedure.</p>	<p>Table-level data locks can be held in any of the following modes:</p> <p>Row share table lock (RW):</p> <p>This indicates that the transaction holding the lock on the table has locked rows in the table and intends to update them. This prevents other transactions from obtaining exclusive write access to the same table by using the LOCK TABLE table IN EXCLUSIVE MODE statement. Apart from that, all the queries, inserts, deletes, and updates are allowed in that table.</p> <p>Row exclusive table lock (RX):</p> <p>This generally indicates that the transaction holding the lock has made one or more updates to the rows in the table. Queries, inserts, deletes, updates are allowed in that table.</p> <p><b>Share lock (SL):</b></p> <p>Share row exclusive lock(SRX)</p> <p><b>Exclusive lock (X):</b></p> <p>The dynamic performance table V\$LOCK keeps the information about locks.</p>

**Recommendations:**

In Microsoft SQL Server, SELECT statements obtain shared locks on pages/rows. This prevents other statements from obtaining an exclusive lock on those pages/rows. All statements that update the data need an exclusive lock. This means that the SELECT statement in Microsoft SQL Server blocks the UPDATE statements as long as the transaction that includes the SELECT statement does not commit or rollback. This also means that two transactions are physically serialized whenever one transaction selects the data and the other transaction wants to change the data first and then select the data again. In Oracle, however, SELECT statements do not block UPDATE statements, since the rollback segments are used to store the changed data before it is updated in the actual tables. Also, the reader of the data is

never blocked in Oracle. This allows Oracle transactions to be executed simultaneously.

If Microsoft SQL Server logical transactions are automatically translated to Oracle logical transactions, the preceding transactions that execute properly in Microsoft SQL Server as they are serialized cause a deadlock in Oracle. These transactions should be identified and serialized to avoid the deadlock. These transactions are serialized in Microsoft SQL Server as INSERT, UPDATE, and DELETE statements block other statements.

## Row-Level Versus Page-Level Locking

**Table 2–22** *Row-Level Versus Page-Level Locking in Oracle and Microsoft SQL Server*

<b>Microsoft SQL Server</b>	<b>Oracle</b>
Microsoft SQL Server does not have a row-level locking feature.	Oracle has a row-locking feature. Only one row is locked when a DML statement is changing the row.
Microsoft SQL Server applies a page-level lock, which effectively locks all rows on the page, whenever any row in the page is being updated. This is an exclusive lock whenever the data is being changed by DML statements.	
Microsoft SQL Server 7.0 implements a form of row-level locking.	
Microsoft SQL Server 7.0 escalates locks at row level to page level automatically.	
SELECT statements are blocked by exclusive locks that lock an entire page.	

### **Recommendations:**

No changes are required to take advantage of the row-level locking feature of Oracle.

## Read Consistency

**Table 2–23** *Read Consistency in Oracle and Microsoft SQL Server*

Microsoft SQL Server	Oracle
<p>Microsoft SQL Server provides the HOLDLOCK function for transaction-level read consistency. Specifying a SELECT with HOLDLOCK puts a shared lock on the data. More than one user can execute a SELECT with HOLDLOCK at the same time without blocking each other.</p> <p>When one of the users tries to update the selected data, HOLDLOCK blocks the update until the other users commit, rollback, or attempt an update and a deadlock occurs. This means that HOLDLOCK prevents other transactions from updating the same data until the current transaction is in effect.</p>	<p>Read consistency as supported by Oracle does the following:</p> <ul style="list-style-type: none"> <li>■ Ensures that the set of data seen by a statement is consistent at a single point-in-time and does not change during statement execution</li> <li>■ Ensures that reads of database data do not wait for other reads or writes of the same data</li> <li>■ Ensures that writes of database data do not wait for reads of the same data</li> <li>■ Ensures that writes wait for other writes only if they attempt to update identical rows in concurrent transactions</li> </ul> <p>To provide read consistency, Oracle creates a read-consistent set of data when a table is being read and simultaneously updated.</p> <p>Read consistency functions as follows:</p> <ol style="list-style-type: none"> <li>1. When an update occurs, the original data values changed by the update are recorded in rollback segments.</li> <li>2. While the update remains part of an uncommitted transaction, any user that reads the modified data views the original data values. Only statements that start after another user's transaction is committed reflect the changes made by the transaction.</li> </ol> <p>You can specify that a transaction be read only using the following command:</p> <pre>SET TRANSACTION READ ONLY</pre>

## Logical Transaction Handling

**Table 2–24 Logical Transaction Handling in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p>After completion, any statement not within a transaction is automatically committed. A statement can be a batch containing multiple T-SQL statements that are sent to the server as one stream. The changes to the database are automatically committed after the batch executes. A ROLLBACK TRAN statement subsequently executed has no effect. In Microsoft SQL Server, transactions are not implicit. Start logical transaction with a BEGIN TRANSACTION clause. Logical transactions can be committed or rolled back as follows.</p> <pre>BEGIN TRANSACTION [transaction_name]</pre> <p>Use COMMIT TRAN to commit the transaction to the database. You have the option to specify the transaction name. Use ROLLBACK TRAN to roll back the transaction. You can set savepoints to roll back to a certain point in the logical transaction using the following command:</p> <pre>SAVE TRANSACTION savepoint_name</pre> <p>Roll back to the specified SAVEPOINT with the following command:</p> <pre>ROLLBACK TRAN &lt;savepoint_name&gt;</pre>	<p>Statements are not automatically committed to the database. The COMMIT WORK statement is required to commit the pending changes to the database.</p> <p>Oracle transactions are implicit. This means that the logical transaction starts as soon as data changes in the database.</p> <p>COMMIT WORK commits the pending changes to the database.</p> <p>ROLLBACK undoes all the transactions after the last COMMIT WORK statement.</p> <p>Savepoints can be set in transactions with the following command:</p> <pre>SET SAVEPOINT savepoint_name</pre> <p>The following command rolls back to the specified SAVEPOINT:</p> <pre>ROLLBACK &lt;savepoint_name&gt;</pre> <p>Two-phase commit is automatic and transparent in Oracle. Two-phase commit operations are needed only for transactions which modify data on two or more databases.</p>

**Table 2–24 (Cont.) Logical Transaction Handling in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p>Microsoft SQL Server allows you to nest BEGIN TRAN/COMMIT TRAN statements. When nested, the outermost pair of transactions creates and commits the transaction. The inner pairs keep track of the nesting levels. A ROLLBACK command in the nested transactions rolls back to the outermost BEGIN TRAN level, if it does not include the name of the SAVEPOINT. Most Microsoft SQL Server applications require two-phase commit, even on a single server. To see if the server is prepared to commit the transaction, use PREPARE TRAN in two-phase commit applications.</p>	<p>When a CHECKPOINT occurs, the completed transactions are written to the database device. A CHECKPOINT writes all dirty pages to the disk devices that have been modified since last checkpoint</p>
<p>Completed transactions are written to the database device at CHECKPOINT. A CHECKPOINT writes all dirty pages to the disk devices since the last CHECKPOINT.</p>	
<p>Calls to remote procedures are executed independently of any transaction in which they are included.</p>	

**Recommendations:**

Transactions are not implicit in Microsoft SQL Server. Therefore, applications expect that every statement they issue is automatically committed if it is executed.

Oracle transactions are always implicit, which means that individual statements are not committed automatically. When converting a Microsoft SQL Server application to an Oracle application, care needs to be taken to determine what constitutes a transaction in that application. In general, a COMMIT work statement needs to be issued after every "batch" of statements, single statement, or stored procedure call to replicate the behavior of Microsoft SQL Server for the application.

In Microsoft SQL Server, transactions may also be explicitly begun by a client application by issuing a BEGIN TRAN statement during the conversion process.



---

---

## Triggers and Stored Procedures

This chapter describes the differences between Microsoft SQL Server and Oracle. It contains the following sections:

- [Triggers](#)
- [Stored Procedures](#)
- [Data Types](#)
- [Schema Objects](#)
- [T/SQL Versus PL/SQL Constructs](#)
- [T/SQL and PL/SQL Language Elements](#)

### Triggers

Microsoft SQL Server database triggers are AFTER triggers. This means that triggers are fired after the specific operation is performed. For example, the INSERT trigger fires after the rows are inserted into the database. If the trigger fails, the operation is rolled back.

Microsoft SQL Server allows INSERT, UPDATE, and DELETE triggers. Triggers typically need access to the before image and after image of the data that is being changed. Microsoft SQL Server achieves this with two temporary tables called INSERTED and DELETED. These two tables exist during the execution of the trigger. These tables and the table for which the trigger is written have the exact same structure. The DELETED table holds the before image of the rows that are undergoing change because of the INSERT/UPDATE/DELETE operation, and the INSERTED table holds the after image of these rows. If there is an error, the triggers can issue a rollback statement.

Most of the Microsoft SQL Server trigger code is written to enforce referential integrity. Microsoft SQL Server triggers are executed once per triggering SQL statement (such as INSERT, UPDATE, or DELETE). If you want some actions to be performed for each row that the SQL statement affects, you must code the actions using the INSERTED and DELETED tables.

Oracle has a rich set of triggers. Oracle also provides triggers that fire for events such as INSERT, UPDATE, and DELETE. You can also specify the number of times that the trigger action is to be executed. For example, once for every row affected by the triggering event (such as might be fired by an UPDATE statement that updates many rows), or once for the triggering statement (regardless of how many rows it affects).

A ROW trigger is fired each time that the table is affected by the triggering event. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. A STATEMENT trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects.

Oracle triggers can be defined as either BEFORE triggers or AFTER triggers. BEFORE triggers are used when the trigger action should determine whether the triggering statement should be allowed to complete. By using a BEFORE trigger, you can avoid unnecessary processing of the triggering statement and its eventual rollback in cases where an exception is raised.

As combinations, there are four different types of triggers in Oracle:

- BEFORE STATEMENT trigger
- BEFORE ROW trigger
- AFTER STATEMENT trigger
- AFTER ROW trigger

It is sometimes necessary to create a ROW trigger or a STATEMENT trigger to achieve the same functionality as the Microsoft SQL Server trigger. This occurs in the following cases:

- The triggering code reads from its own table (mutating).
- The triggering code contains group functions.

In the following example, the group function AVG is used to calculate the average salary:

```
SELECT AVG(inserted.salary)
FROM inserted a, deleted b
```

```
WHERE a.id = b.id;
```

This would be converted to Oracle by creating an AFTER ROW trigger to insert all the updated values into a package, and an AFTER STATEMENT trigger to read from the package and calculate the average.

For examples of Oracle triggers, see *Oracle Database 2 Day Developer's Guide*.

## Stored Procedures

Microsoft SQL Server stores triggers and stored procedures with the server. Oracle stores triggers and stored subprograms with the server. Oracle has three different kinds of stored subprograms, namely functions, stored procedures, and packages. For detailed discussion on all these objects, see *Oracle Database PL/SQL Language Reference*.

Stored procedures provide a powerful way to code the application logic that can be stored with the server. Microsoft SQL Server and Oracle all provide stored procedures.

The language used to code these objects is a database-specific procedural extension to SQL. In Oracle it is PL/SQL and in Microsoft SQL Server it is Transact SQL (T/SQL). These languages differ to a considerable extent. The individual SQL statements and the procedural constructs, such as `if-then-else`, are similar in both versions of the procedural SQL. Considerable differences can be found in the following areas discussed in these sections:

- [Individual SQL Statements](#)
- [Logical Transaction Handling](#)
- [Error Handling Within the Stored Procedure](#)

This section also considers various components of typical Microsoft SQL Server stored procedures and suggests ways to design them in order to avoid conversion problems. By applying the standards described in this section to the coding, you can convert your stored procedures from Microsoft SQL Server to Oracle.

### Individual SQL Statements

In individual SQL statements, you should try to follow ANSI-standard SQL whenever possible. However, there are cases where you need to use database-specific SQL constructs, mostly for ease of use, simplicity of coding, and performance enhancement. For example, Microsoft SQL Server constructs such as

the following are SQL Server-specific, and cannot be converted to Oracle without manual intervention:

```
update <table_name>
set ...
from <table1>, <table_name>
where...
```

The manual intervention required to convert statements such as this can be seen in the following examples:

**Microsoft SQL Server:**

```
DELETE sales
FROM sales, titles
WHERE sales.title_id = titles.title_id
AND titles.type = 'business'
```

**Oracle:**

```
DELETE sales
FROM sales, titles
WHERE sales.title_id = titles.title_id
AND titles.type = 'business'
```

**Microsoft SQL Server:**

```
UPDATE titles
SET price = price + author_royalty
FROM titles, title_author
WHERE titles.title.id = title_author.title_id
```

**Oracle:**

```
UPDATE titles O
SET price = ( SELECT (O.price + I.author_royalty)
              FROM title_author I
              WHERE I.title_id = O.title_id)
WHERE EXISTS (SELECT 1
              FROM title_author
              WHERE title_author.title_id = O.title_id) ;
```

All the ANSI-standard SQL statements can be converted from one database to another using automatic conversion utilities.

## Logical Transaction Handling

In Microsoft SQL Server, transactions are explicit by definition. This implies that an individual SQL statement is not part of a logical transaction by default. A SQL statement belongs to a logical transaction if the transaction explicitly initiated by the user with a `BEGIN TRANSACTION` (or `BEGIN TRAN`) statement is still in effect. The logical transaction ends with a corresponding `COMMIT TRANSACTION` (or `COMMIT TRAN`) or `ROLLBACK TRANSACTION` (or `ROLLBACK TRAN`) statement. Each SQL statement that is not part of a logical transaction is committed on completion.

In Oracle, transactions are implicit as set by the ANSI standard. The implicit transaction model requires that each SQL statement is part of a logical transaction. A new logical transaction is automatically initiated when a `COMMIT` or `ROLLBACK` command is executed. This also implies that data changes from an individual SQL statement are not committed to the database after execution. The changes are committed to the database only when a `COMMIT` statement is run. The differences in the transaction models impact the coding of application procedures.

### Transaction-Handling Statements

For client/server applications, it is recommended that you make the transaction-handling constructs part of the client procedures. The logical transaction is always defined by client users, and they should control it. This strategy is also more suitable for distributed transactions, where the two-phase commit operations are necessary. Making the transaction-handling statements a part of the client code serves a two-fold purpose; the server code is more portable, and the distributed transactions can be independent of the server code. Try to avoid using the `BEGIN TRAN`, `ROLLBACK TRAN`, and `COMMIT TRAN` statements in the stored procedures. In Microsoft SQL Server, transactions are explicit. In Oracle, transactions are implicit. If the transactions are handled by the client, the application code residing on the server can be independent of the transaction model.

## Error Handling Within the Stored Procedure

Oracle PL/SQL checks each SQL statement for errors before proceeding with the next statement. If an error occurs, control immediately jumps to an exception handler. This avoids you having to check the status of every SQL statement. For example, if a `SELECT` statement does not find any rows in the database, an exception is raised, and the code to deal with this error is executed.

In Microsoft SQL Server, you need not check for errors after each SQL statement. Control is passed to the next statement, irrespective of the error conditions generated by the previous statement. It is your responsibility to check for errors after the execution of each SQL statement. Failure to do so may result in erroneous results.

In Oracle, to simulate the behavior of Microsoft SQL Server and to pass the control to the next statement regardless of the status of execution of the previous SQL statement, you must enclose each SQL statement in an equivalent PL/SQL block. This block must deal with all possible exceptions for that SQL statement. This coding style is required only to simulate Microsoft SQL Server behavior. An Oracle PL/SQL procedure ideally has only one exception block, and all error conditions are handled in that block.

Consider the following code in a Microsoft SQL Server stored procedure:

```
begin

    select @x = col1 from table1 where col2 = @y
    select @z = col3 from table2 where col4 = @x

end
```

In this code example, if the first SELECT statement does not return any rows, the value of @x could be UNDEFINED. If the control is passed on to the next statement without raising an exception, the second statement returns incorrect results because it requires the value of @x to be set by an earlier statement. In a similar situation, Oracle PL/SQL raises a NO\_DATA\_FOUND exception if the first statement fails.

### **RAISERROR Statement**

The Microsoft SQL Server RAISERROR statement does not return to the calling routine. The error code and message is passed to the client, and the execution of the stored procedure continues further. The Oracle RAISE\_APPLICATION\_ERROR statement returns to the calling routine. As a standard, a RETURN statement must appear after the RAISERROR statement in Microsoft SQL Server, so that it can be converted to the Oracle RAISE\_APPLICATION\_ERROR statement.

### **Customized Error Messages**

Microsoft SQL Server allows you to customize the error messages using a system table. The system procedures allow the user to add error messages to the system. Adding error messages to the Microsoft SQL Server system table is not desirable because there is no equivalent on the Oracle system. This can be avoided by

maintaining a user-defined error messages table, located in the centralized database. Standard routines can be written to add the error message to the table and retrieve it whenever necessary. This method serves a two-fold purpose: it ensures that the system is more portable across different types of database servers, and it gives the administrator centralized control over the error messages.

## Data Types

This section provides information about data types under the following headings:

- [Local Variable](#)
- [Server Data Types](#)
- [Composite Data Types](#)

### Local Variable

T/SQL local variables can be any server data type except TEXT and IMAGE. PL/SQL local variables can be any server data type including the following:

- BINARY\_INTEGER
- BOOLEAN

PL/SQL local variables can also be either of the following composite data types allowed by PL/SQL:

- RECORD
- TABLE

### Server Data Types

See the "[Data Types](#)" on page 2-8 for a list of Microsoft SQL Server data types and their equivalent Oracle data types.

### Composite Data Types

Microsoft SQL Server does not have composite data types.

**Table 3–1 Composite Data Types in Oracle**

Oracle	Comments
RECORD	You can declare a variable to be of type RECORD. Records have uniquely named fields. Logically related data that is dissimilar in type can be held together in a record as a logical unit.
TABLE	PL/SQL tables can have one column and a primary key, neither of which can be named. The column can belong to any scalar data type. The primary key must belong to type BINARY_INTEGER.

## Schema Objects

This section compares the following Microsoft SQL Server and Oracle schema objects:

- [Procedure](#)
- [Function](#)
- [Package](#)
- [Package Body](#)

Each schema object is compared in separate tables based on create, drop, execute and alter, where applicable. The tables are divided into the following sections:

- Syntax
- Description
- Permissions
- Examples

Some tables are followed by a recommendations section that contains important information about conversion implications.

## Procedure

This section provides the following tables for the schema object procedure :

- [Create](#)
- [Drop](#)
- [Execute](#)
- [Alter](#)

## Create

**Table 3–2 Comparison of Creating the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>CREATE PROCEDURE procedure [@formal_ parameter formal_parameter_data type [OUTPUT] [= default_value] [,@formal_parameter formal_parameter_data type [OUTPUT][= default_value]] ...  AS BEGIN] procedural_statements [END]</pre>	<p><b>Syntax:</b></p> <pre>CREATE [OR REPLACE] PROCEDURE [schema.]procedure [(] [formal_parameter [IN   OUT   IN OUT] formal_parameter_data type] [DEFAULT default_value] [,formal_parameter [IN   OUT   IN OUT]formal_parameter_data type] [DEFAULTdefault_value]]... [)] IS   AS [local_ variable data type;]... BEGIN PL/SQL statements   PL/SQL blocks END;</pre>
<p><b>Description:</b></p> <p>The CREATE PROCEDURE statement creates the named stored procedure in the database.</p> <p>You can optionally specify the parameters passed to the procedure as OUTPUT. Values of OUTPUT variables are available to the calling routine after the procedure is executed. The parameters specified without the OUTPUT keyword are considered as input parameters.</p> <p>The keyword AS indicates the start of the body of the procedure.</p> <p>The BEGIN and END keywords that enclose the stored procedure body are optional; all the procedural statements contained in the file after AS are considered part of the stored procedure if BEGIN and END are not used to mark blocks.</p> <p>See the T/SQL and PL/SQL Language Elements section of this chapter for more information about the constructs allowed in T/SQL procedures.</p>	<p><b>Description:</b></p> <p>The OR REPLACE keywords replace the procedure by the new definition if it already exists.</p> <p>The parameters passed to the PL/SQL procedure can be specified as IN (input), OUT (output only), or IN OUT (input and output). In the absence of these keywords, the parameter is assumed to be the "IN" parameter.</p> <p>The keyword IS or AS indicates the start of the procedure. The local variables are declared after the keyword IS or AS and before the keyword BEGIN.</p> <p>The BEGIN and END keywords enclose the body of the procedure.</p>

**Table 3–2 (Cont.) Comparison of Creating the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Permissions:</b></p> <p>You must have the CREATE PROCEDURE system privilege to create the stored procedures</p>	<p><b>Permissions:</b></p> <p>To create a procedure in your own schema, you must have the CREATE PROCEDURE system privilege. To create a procedure in another user's schema, you must have the CREATE ANY PROCEDURE system privilege.</p>

**Recommendations:**

Functionally identical parts can be identified in the T/SQL procedure and PL/SQL procedure structure. Therefore, you can automate the conversion of most of the constructs from Microsoft SQL Server to Oracle.

OR REPLACE keywords in an Oracle CREATE PROCEDURE statement provide an elegant way of recreating the procedure. In Microsoft SQL Server, the procedure must be dropped explicitly before replacing it.

**Drop****Table 3–3 Comparison of Dropping the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>DROP PROCEDURE procedure</p>	<p><b>Syntax:</b></p> <p>DROP PROCEDURE [schema.]procedure</p>
<p><b>Description:</b></p> <p>The procedure definition is deleted from the data dictionary. All the objects that reference this procedure must have references to this procedure removed</p>	<p><b>Description:</b></p> <p>When a procedure is dropped, Oracle invalidates all the local objects that reference the dropped procedure</p>
<p><b>Permissions:</b></p> <p>Procedure owners can drop their own procedures. A DBO can drop any procedure.</p>	<p><b>Permissions:</b></p> <p>The procedure must be in the schema of the user or the user must have the DROP ANY PROCEDURE system privilege to execute this command</p>
<p><b>Example:</b></p> <p>DROP PROCEDURE myproc</p>	<p><b>Example:</b></p> <p>DROP PROCEDURE sam.credit;</p>

**Recommendations:**

The preceding statement does not have any effect on the conversion process. This information is provided for reference only.

**Execute**

**Table 3–4 Comparison of Executing the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <pre>EXEC [@return_value = ] procedure [[@formal_parameter = ] {@actual_ parameter   constant_literal} [OUT]] [, [[@formal_parameter = ] {@actual_ parameter   constant_literal} [OUT]]] ...</pre>	<p><b>Syntax:</b></p> <pre>procedure [[({actual_parameter   constant_literal   formal_parameter =&gt; {actual_parameter   constant_literal} })] [, {actual_parameter   constant_literal   formal_parameter =&gt; {actual_parameter   constant_literal} })] ... )]</pre>
<p><b>Description:</b></p> <p>Microsoft SQL Server stored procedures can only return integer values to the calling routine using the RETURN statement. In the absence of a RETURN statement, the stored procedure still returns a return status to the calling routine. This value can be captured in the "return_value" variable.</p> <p>The formal_parameter is the parameter in the procedure definition. The actual_parameter is defined in the local block which calls the procedure supplying the value of the actual parameter for the respective formal parameter. The association between an actual parameter and formal parameter can be indicated using either positional or named notation.</p>	<p><b>Description:</b></p> <p>Oracle PL/SQL procedures send data back to the calling routine by means of OUT parameters. Oracle offers <b>functions</b> that are a different type of schema objects. Functions can return an atomic value to the calling routine using the RETURN statement. The RETURN statement can return value of any data type.</p> <p>The formal_parameter is the parameter in the procedure definition. The actual_parameter is defined in the local block which calls the procedure supplying the value of the actual parameter for the respective formal parameter. The association between an actual parameter and formal parameter can be indicated using either positional or named notation.</p>

**Table 3–4 (Cont.) Comparison of Executing the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Positional notation:</b> The actual parameters are supplied to the procedure in the same order as the formal parameters in the procedure definition.</p>	<p><b>Positional notation:</b> The actual parameters are supplied to the procedure in the same order as the formal parameters in the procedure definition.</p>
<p><b>Named notation:</b> The actual parameters are supplied to the procedure in an order different than that of the formal parameters in the procedure definition by using the name of the formal parameter as:</p>	<p><b>Named notation:</b> The actual parameters are supplied to the procedure in an order different than that of the formal parameters in the procedure definition by using the name of the formal parameter as:</p>
<pre>@formal_parameter = @actual_ parameter</pre>	<pre>formal_parameter =&gt; actual_parameter</pre>
<p>A constant literal can be specified in the place of the following:</p>	<p>A constant literal can be specified in the place of the following:</p>
<pre>'@actual_parameter ' as: @formal_parameter = 10</pre>	<p>as:</p> <pre>formal_parameter =&gt; 10</pre>
<pre>@formal_parameter = 10</pre>	<p>If the formal_parameter is specified as OUT or IN OUT in the procedure definition, the value is made available to the calling routine after the execution of the procedure.</p>
<p>The keyword OUT should be specified if the procedure has to return the value of that parameter to the calling routine as OUTPUT.</p>	
<p><b>Permissions:</b> The user should have the EXECUTE permission on the stored procedure. The user need not have explicit privileges to access the underlying objects referred to within the stored procedure.</p>	<p><b>Permissions</b> The user should have the EXECUTE privilege on the named procedure. The user need not have explicit privileges to access the underlying objects referred to within the PL/SQL procedure</p>

**Table 3–4 (Cont.) Comparison of Executing the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Example:</b></p> <p><b>Positional notation:</b></p> <pre>EXEC GetEmpName @EmpID EXEC @status = GetAllDeptCodes EXEC @status = UpdateEmpSalary @EmpID, @EmpName EXEC UpdateEmpSalary 13000, 'Joe Richards'</pre> <p><b>Named notation:</b></p> <pre>EXEC UpdateEmpSalary @Employee = @EmpName, @Employee_Id = @EmpID</pre> <p><b>Mixed notation:</b></p> <pre>EXEC UpdateEmpSalary @EmpName, @Employee_Id = @EmpID EXEC UpdateEmpSalary @Employee = @EmpName, @EmpID</pre>	<p><b>Example:</b></p> <p><b>Positional notation:</b></p> <pre>credit (accno, accname, amt, retstat);</pre> <p><b>Named notation:</b></p> <pre>credit (acc_no =&gt; accno, acc =&gt; accname, amount =&gt; amt, return_status =&gt; retstat)</pre> <p><b>Mixed notation (where positional notation must precede named notation):</b></p> <pre>credit (accno, accname, amount =&gt; amt, return_status =&gt; retstat)</pre>

**Alter**

**Table 3–5 Comparison of Altering the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>The system procedure SP_RECOMPILE recompiles the named stored procedure. For example:</p> <pre>ALTER PROCEDURE &lt;procedure name&gt;   RECOMPILE   ENCRYPT   RECOMPILE, ENCRYPT</pre>	<p><b>Syntax:</b></p> <pre>ALTER PROCEDURE [schema.]procedure COMPILE</pre>

**Table 3–5 (Cont.) Comparison of Altering the Procedure Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Description:</b></p> <p>This command causes the recompilation of the procedure. Procedures that become invalid for some reason should be recompiled explicitly using this command.</p>	<p><b>Description:</b></p> <p>This command causes the recompilation of the procedure. Procedures that become invalid for some reason should be recompiled explicitly using this command. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead</p>
<p><b>Permissions:</b></p> <p>The owner of the procedure can issue this command</p>	<p><b>Permissions:</b></p> <p>The procedure must be in the user's schema or the user must have the ALTER ANY PROCEDURE privilege to use this command</p>
<p><b>Example:</b></p> <pre>sp_recompile my_proc</pre>	<p><b>Example:</b></p> <pre>ALTER PROCEDURE sam.credit COMPILE;</pre>

## Function

This section provides the following tables for the schema object Function:

- [Create](#)
- [Drop](#)
- [Execute](#)
- [Alter](#)

## Create

**Table 3–6 Comparison of Creating the Function Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>In Microsoft SQL Server, you can convert a stored procedure to a function in Oracle because the stored procedure in Microsoft SQL Server can RETURN an integer value to the calling routine using a RETURN statement. A stored procedure returns a status value to the calling routine even in the absence of a RETURN statement. The returned status is equal to ZERO if the procedure execution is successful or NON-ZERO if the procedure fails for some reason. The RETURN statement can return only integer values</p> <p>N/A</p>	<p><b>Syntax:</b></p> <pre>CREATE [OR REPLACE] FUNCTION [user.]function [(parameter [OUT] data type[, (parameter [IN OUT] data type)...]) RETURN data type {IS AS} block</pre> <p><b>Description:</b></p> <p>The OR REPLACE keywords replace the function with the new definition if it already exists.</p> <p>Parameters passed to the PL/SQL function can be specified as "IN" (input), "OUT" (output), or "IN OUT" (input and output). In the absence of these keywords the parameter is assumed to be IN.</p> <p>RETURN data type specifies the data type of the function's return value. The data type can be any data type supported by PL/SQL. See "<a href="#">Data Types</a>" on page 2-8 for more information about data types.</p> <p><b>Permissions:</b></p> <p>To create a function in your own schema, you must have the CREATE PROCEDURE system privilege. To create a function in another user's schema, you must have the CREATE ANY PROCEDURE system privilege.</p>
N/A	

**Table 3–6 (Cont.) Comparison of Creating the Function Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Example:</b></p> <pre>CREATE FUNCTION get_bal (acc_no IN NUMBER) RETURN NUMBER IS     acc_bal NUMBER(11,12); BEGIN     SELECT balance     INTO acc_bal     FROM accounts     WHERE account_id = acc_no; RETURN(acc_bal); END;</pre>

## Drop

**Table 3–7 Comparison of Dropping the Function Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Syntax:</b></p> <pre>DROP FUNCTION [schema.]function</pre>
N/A	<p><b>Description:</b></p> <p>When a function is dropped, Oracle invalidates all the local objects that reference the dropped function.</p>
N/A	<p><b>Permissions:</b></p> <p>The function must be in the schema of the user or the user must have the DROP ANY PROCEDURE system privilege to execute this command</p>
N/A	<p><b>Example:</b></p> <pre>DROP FUNCTION sam.credit;</pre>

## Execute

**Table 3–8 Comparison of Executing the Function Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Syntax:</b></p> <pre>function [(actual_parameter   constant_literal)...]</pre>
N/A	<p><b>Description:</b></p> <p>Functions can return an atomic value to the calling routine using the RETURN statement.</p> <p>A function can be called as part of an expression. This is a very powerful concept. All the Microsoft SQL Server built-in functions can be coded using PL/SQL, and these functions can be called like any other built-in functions in an expression, starting with Oracle.</p>
N/A	<p><b>Permissions:</b></p> <p>You should have the EXECUTE privilege on the function to execute the named function. You need not have explicit privileges to access the underlying objects that are referred to within the PL/SQL function.</p>
N/A	<p><b>Example:</b></p> <pre>1) IF sal_ok (new_sal, new_title) THEN     ...     END IF;</pre> <pre>2) promotable:=     sal_ok(new_sal, new_title) AND     (rating&gt;3);</pre> <p>where sal_ok is a function that returns a BOOLEAN value.</p>

## Alter

**Table 3–9 Comparison of Altering the Function Schema Object in Oracle and Microsoft SQL Server 7.0**

Microsoft SQL Server	Oracle
N/A	<p><b>Syntax:</b></p> <pre>ALTER FUNCTION [schema.]function COMPILE</pre>

**Table 3–9 (Cont.) Comparison of Altering the Function Schema Object in Oracle and Microsoft SQL Server 7.0**

Microsoft SQL Server	Oracle
N/A	<b>Description:</b> This command causes the recompilation of a function. Functions become invalid if the objects that are referenced from within the function are dropped or altered. Functions that become invalid for some reason should be recompiled explicitly using this command. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
N/A	<b>Permissions:</b> The function must be in the user's schema or the user must have the ALTER ANY PROCEDURE privilege to use this command
N/A	<b>Example:</b> <pre>ALTER FUNCTION sam.credit COMPILE</pre>

## Package

This section provides the following tables for the schema object Package:

- [Create](#)
- [Drop](#)
- [Alter](#)

## Create

**Table 3–10 Comparison of Creating the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>Microsoft SQL Server does not support this concept.</p>	<p><b>Syntax:</b></p> <pre>CREATE [OR REPLACE] PACKAGE [user.]package {IS   AS} {variable_declaration   cursor_specification   exception_declaration   record_declaration   plsql_ table_declaration   procedure_specification   function_specification   [{variable_declaration   cursor_specification   exception_declaration   record_ declaration   plsql_table_declaration   procedure_ specification   function_specification}; ]...} END [package]</pre>
N/A	<p><b>Description:</b></p> <p>This is the external or public part of the package.</p> <p>CREATE PACKAGE sets up the specification for a PL/SQL package which can be a group of procedures, functions, exception, variables, constants, and cursors.</p> <p>Functions and procedures of the package can share data through variables, constants, and cursors.</p> <p>The OR REPLACE keywords replace the package by the new definition if it already exists. This requires recompilation of the package and any objects that depend on its specification.</p>
N/A	<p><b>Permissions:</b></p> <p>To create a package in the user's own schema, the user must have the CREATE PROCEDURE system privilege. To create a package in another user's schema, the user must have the CREATE ANY PROCEDURE system privilege.</p>

**Table 3–10 (Cont.) Comparison of Creating the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Example:</b></p> <pre> CREATE PACKAGE emp_actions AS   -- specification   TYPE EmpRecTyp IS RECORD (emp_id INTEGER, salary REAL);   CURSOR desc_salary (emp_id NUMBER) RETURN EmpRecTyp;  PROCEDURE hire_employee   (ename CHAR,   job CHAR,   mgr NUMBER,   sal NUMBER,   comm NUMBER,   deptno NUMBER); PROCEDURE fire-employee (emp_id NUMBER); END emp_actions; </pre>

## Drop

**Table 3–11 Comparison of Dropping the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>Microsoft SQL Server does not support this concept.</p>	<p><b>Syntax:</b></p> <pre>DROP PACKAGE [BODY] [schema.]package</pre>
N/A	<p><b>Description:</b></p> <p>The BODY option drops only the body of the package. If you omit BODY, Oracle drops both the body and specification of the package. If you drop the body and specification of the package, Oracle invalidates any local objects that depend on the package specification.</p> <p>schema . is the schema containing the package. If you omit schema, Oracle assumes the package is in your own schema.</p> <p>When a package is dropped, Oracle invalidates all the local objects that reference the dropped package.</p>

**Table 3–11 (Cont.) Comparison of Dropping the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Permissions:</b></p> <p>The package must be in the schema of the user or the user must have the DROP ANY PROCEDURE system privilege to execute this command.</p>
N/A	<p><b>Example:</b></p> <pre>DROP PACKAGE emp_actions;</pre>

**Alter**

**Table 3–12 Comparison of Altering the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Syntax:</b></p> <p>Microsoft SQL Server does not support this concept.</p>	<p><b>Syntax:</b></p> <pre>ALTER PACKAGE [user.]package COMPILE [PACKAGE   BODY]</pre>
N/A	<p><b>Description:</b></p> <p>Packages that become invalid for some reason should be recompiled explicitly using this command.</p> <p>This command causes the recompilation of all package objects together. You cannot use the ALTER PROCEDURE or ALTER FUNCTION commands to individually recompile a procedure or function that is part of a package.</p> <p>PACKAGE, the default option, recompiles the package body and specification.</p> <p>BODY recompiles only the package body.</p> <p>Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.</p>
N/A	<p><b>Permissions:</b></p> <p>The package must be in the user's schema or the user must have the ALTER ANY PROCEDURE privilege to use this command.</p>

**Table 3–12 (Cont.) Comparison of Altering the Package Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<b>Example:</b> ALTER PACKAGE emp_actions COMPILE PACKAGE

## Package Body

This section provides the following tables for the schema object Package Body:

- [Create](#)
- [Drop](#)
- [Alter](#)

### Create

**Table 3–13 Comparison of Creating the Package Body Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Syntax:</b> Microsoft SQL Server does not support this concept.	<b>Syntax:</b> CREATE [OR REPLACE] PACKAGE BODY [schema.] package {IS   AS} pl/sql_package_body
N/A	<b>Description:</b> This is the internal or private part of the package. CREATE PACKAGE creates the body of a stored package. OR REPLACE recreates the package body if it already exists. If you change a package body, Oracle recompiles it. schema. is the schema to contain the package. If omitted, the package is created in your current schema. package is the of the package to be created. pl/sql_package_body is the package body which can declare and define program objects. For more information on writing package bodies, see the <i>PL/SQL User's Guide and Reference, Release 1 (9.0.1)</i> .

**Table 3–13 (Cont.) Comparison of Creating the Package Body Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Permissions:</b></p> <p>To create a package in your own schema, you must have the CREATE PROCEDURE privilege. To create a package in another user's schema, you must have the CREATE ANY PROCEDURE privilege.</p>
N/A	<p><b>Example:</b></p> <pre> CREATE PACKAGE BODY emp_actions AS -- body   CURSOR desc_salary (emp_id NUMBER)     RETURN EmpRecTyp IS     SELECT empno, sal FROM emp     ORDER BY sal DESC;   PROCEDURE hire_employee     (ename  CHAR,      job    CHAR,      mgr    NUMBER,      sal    NUMBER,      comm   NUMBER,      deptno NUMBER) IS   BEGIN     INSERT INTO emp VALUES       (empno_seq.NEXTVAL, ename,        job, mgr, SYSDATE, sal,        comm, deptno);   END hire_employee;    PROCEDURE fire_employee     (emp_id  NUMBER) IS   BEGIN     DELETE FROM emp     WHERE empno = emp_id;   END fire_employee;  END emp_actions; </pre>

## Drop

**Table 3–14 Comparison of Dropping the Package Body Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Syntax:</b> Microsoft SQL Server does not support this concept.	<b>Syntax:</b> <pre>DROP PACKAGE [BODY] [schema.]package</pre>
N/A	<b>Description:</b> The BODY option drops only the body of the package. If you omit BODY, Oracle drops both the body and specification of the package. If you drop the body and specification of the package, Oracle invalidates any local objects that depend on the package specification.  schema . is the schema containing the package. If you omit schema ., Oracle assumes the package is in your own schema.  When a package is dropped, Oracle invalidates all the local objects that reference the dropped package.
N/A	<b>Permissions:</b> The package must be in the your own schema or you must have the DROP ANY PROCEDURE system privilege to execute this command.
N/A	<b>Example:</b> <pre>DROP PACKAGE BODY emp_actions;</pre>

## Alter

**Table 3–15 Comparison of Altering the Package Body Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Syntax:</b> Microsoft SQL Server does not support this concept.	<b>Syntax:</b> <pre>ALTER PACKAGE [user.]package COMPILE [PACKAGE   BODY]</pre>

**Table 3–15 (Cont.) Comparison of Altering the Package Body Schema Object in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
N/A	<p><b>Description:</b></p> <p>Packages that become invalid for some reason should be recompiled explicitly using this command.</p> <p>This command causes the recompilation of all package objects together. You cannot use the ALTER PROCEDURE or ALTER FUNCTION commands to individually recompile a procedure or function that is part of a package.</p> <p>PACKAGE, the default option, recompiles the package body and specification.</p> <p>BODY recompiles only the package body.</p> <p>Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.</p>
N/A	<p><b>Permissions:</b></p> <p>The package must be your own schema or you must have the ALTER ANY PROCEDURE privilege to use this command.</p>
N/A	<p><b>Example:</b></p> <pre>ALTER PACKAGE emp_actions COMPILE BODY</pre>

## T/SQL Versus PL/SQL Constructs

This section provides information about the Microsoft SQL Server constructs and equivalent Oracle constructs generated by SQL Developer. The conversions of the following constructs are discussed in detail:

- [CREATE PROCEDURE Statement](#)
- [Parameter Passing](#)
- [DECLARE Statement](#)
- [IF Statement](#)
- [RETURN Statement](#)
- [RAISERROR Statement](#)
- [EXECUTE Statement](#)
- [WHILE Statement](#)

- [GOTO Statement](#)
- [@@Rowcount and @@Error Variables](#)
- [ASSIGNMENT Statement](#)
- [SELECT Statement with GROUP BY Clause](#)
- [Column Aliases](#)
- [UPDATE with FROM Statement](#)
- [DELETE with FROM Statement](#)
- [Temporary Tables](#)
- [Cursor Handling](#)

Listed is the syntax for the Microsoft SQL Server constructs and their Oracle equivalents, as well as comments about conversion considerations.

The procedures in the Oracle column are the direct output of SQL Developer. In general, SQL Developer deals with the Microsoft SQL Server T/SQL constructs in one of the following ways:

- The ANSI-standard SQL statements are converted to PL/SQL because it supports ANSI-standard SQL.
- Microsoft SQL Server-specific constructs are converted into PL/SQL constructs if the equivalent constructs are available in PL/SQL.
- Some Microsoft SQL Server-specific constructs are ignored and appropriate comments are incorporated in the output file.
- Constructs that require manual conversion are wrapped around with proper comments in the output file.
- For Microsoft SQL Server-specific constructs that result in syntax errors, an appropriate error message is displayed including the line number.

## CREATE PROCEDURE Statement

**Table 3–16 Comparison of CREATE PROCEDURE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
CREATE PROC proc1 AS RETURN 0	CREATE OR REPLACE FUNCTION proc1 RETURN NUMBER AS BEGIN RETURN 0; END;

### Comments

The REPLACE keyword is added to replace the procedure, function, or package if it already exists.

The procedure is translated to an Oracle function because it returns a value.

## Parameter Passing

**Table 3–17 Comparison of Parameter Passing in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
CREATE PROC proc1 @x int=-1, @y money, @z bit OUT, @a char(20) = 'TEST' AS RETURN 0	CREATE OR REPLACE FUNCTION proc1 ( v_x IN NUMBER DEFAULT -1, v_y IN NUMBER, v_z OUT NUMBER, v_a IN CHAR DEFAULT 'TEST' ) RETURN NUMBER AS BEGIN RETURN 0; END;

### Comments

Parameter passing is almost the same in Microsoft SQL Server and Oracle. By default, all the parameters are INPUT parameters, if not specified otherwise.

The value of the INPUT parameter cannot be changed from within the PL/SQL procedure. Thus, an INPUT parameter cannot be assigned any values nor can it be passed to another procedure as an OUT parameter. In Oracle, only IN parameters can be assigned a default value.

The @ sign in a parameter name declaration is removed in Oracle.

In Oracle, the parameter data type definition does not include length/size.

Microsoft SQL Server data types are converted to Oracle base data types. For example, all Microsoft SQL Server numeric data types are converted to NUMBER and all alphanumeric data types are converted to VARCHAR2 and CHAR in Oracle.

## DECLARE Statement

**Table 3–18 Comparison of DECLARE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
CREATE PROC proc1	CREATE OR REPLACE PROCEDURE proc1
AS	AS
DECLARE	v_x NUMBER(10,0);
@x int,	v_y NUMBER(19,2);
@y money,	v_z NUMBER(1,0);
@z bit,	v_a CHAR(20);
@a char(20)	BEGIN
RETURN 0	RETURN;
GO	END;

### Comments

Microsoft SQL Server and Oracle follow similar rules for declaring local variables.

SQL Developer overrides the scope rule for variable declarations. As a result, all the local variables are defined at the top of the procedure body in Oracle.

## IF Statement

**Table 3–19 Comparison of IF Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Example 1:</b></p> <pre>CREATE PROC proc1 @Flag int = 0 AS BEGIN DECLARE @x int IF ( @Flag=0 )     SELECT @x = -1 ELSE     SELECT @x = 10 END</pre>	<p><b>Example 1:</b></p> <pre>CREATE OR REPLACE PROCEDURE proc1 (     v_Flag IN NUMBER DEFAULT 0 ) AS     v_x NUMBER(10,0); BEGIN     IF ( v_Flag = 0 ) THEN         v_x := -1;     ELSE         v_x := 10;     END IF</pre>
<p><b>Example 2:</b></p> <pre>CREATE PROC proc1 @Flag char(2) = '' AS BEGIN DECLARE @x int IF ( @Flag='' )     SELECT @x = -1 ELSE IF (@Flag = 'a')     SELECT @x = 10 ELSE IF (@Flag = 'b')     SELECT @x = 20 END</pre>	<p><b>Example 2:</b></p> <pre>CREATE OR REPLACE PROCEDURE proc1 (     v_Flag IN CHAR DEFAULT '' ) AS     v_x NUMBER(10,0); BEGIN     IF ( v_Flag = '' ) THEN         v_x := -1;     ELSE         IF ( v_Flag = 'a' ) THEN             v_x := 10;         ELSE             IF ( v_Flag = 'b' ) THEN                 v_x := 20;             END IF;         END IF;     END IF; END IF; END;</pre>

**Table 3–19 (Cont.) Comparison of IF Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Example 3:</b></p> <pre>CREATE PROC proc1 AS BEGIN DECLARE @x int IF EXISTS ( SELECT * FROM table2 ) SELECT @x = -1 END</pre>	<p><b>Example 3:</b></p> <pre>CREATE OR REPLACE PROCEDURE proc1 AS v_x NUMBER(10,0); v_temp NUMBER(1, 0) := 0; BEGIN SELECT 1 INTO v_temp FROM DUAL WHERE EXISTS ( SELECT * FROM table2 );  IF v_temp = 1 THEN v_x := -1; END IF; END;</pre>
<p><b>Example 4:</b></p> <pre>CREATE PROC proc1 @basesal money, @empid int AS BEGIN IF (select sal from emp where empid = @empid) &lt; @basesal UPDATE emp SET sal_flag = -1 WHERE empid = @empid END</pre>	<p><b>Example 4:</b></p> <pre>CREATE OR REPLACE PROCEDURE proc1 ( v_basesal IN NUMBER, v_empid IN NUMBER ) AS v_temp NUMBER(1, 0) := 0; BEGIN SELECT 1 INTO v_temp FROM DUAL WHERE ( SELECT sal FROM emp WHERE empid = v_empid ) &lt; v_ basesal;  IF v_temp = 1 THEN UPDATE emp SET sal_flag = -1 WHERE empid = v_empid; END IF; END;</pre>

**Comments**

IF statements in Microsoft SQL Server and Oracle are nearly the same except in the following two cases:

- If EXISTS(...) in Microsoft SQL Server does not have an equivalent PL/SQL construct. Therefore, it is converted to a SELECT INTO WHERE EXISTS clause and an IF statement as shown in Example 3 in Table 3-19.
- IF (SELECT... ) with comparison does not have an equivalent PL/SQL construct. Therefore, it is converted to a SELECT INTO...WHERE... clause, as shown in Example 4 in Table 3-19.

## RETURN Statement

**Table 3-20 Comparison of RETURN Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
CREATE PROC proc1	CREATE OR REPLACE FUNCTION proc1
@x int	(
AS	v_x IN NUMBER
IF @x = -1	)
RETURN 25022	AS
ELSE	BEGIN
RETURN 25011	IF v_x = -1 THEN
	RETURN 25022;
	ELSE
	RETURN 25011;
	END IF;
	END;

### Comments

A RETURN statement is used to return a single value back to the calling program and works the same in both databases. Microsoft SQL Server can return only the numeric data type, while Oracle can return any of the server data types or the PL/SQL data types.

In a PL/SQL procedure, a RETURN statement can only return the control back to the calling program without returning any data. SQL Developer translates procedures returning values to functions automatically.

## RAISERROR Statement

**Table 3–21 Comparison of RAISERROR Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS     RAISERROR 12345 "No Employees found"</pre>	<pre>CREATE OR REPLACE PROCEDURE PROC1 AS BEGIN     raise_application_error(-20999, 'No Employees found'); END PROC1;</pre>

### Comments

Microsoft SQL Server uses RAISERROR to notify the client program of any error that occurred. This statement does not end the execution of the procedure, and the control is passed to the next statement.

PL/SQL provides similar functionality with RAISE\_APPLICATION\_ERROR statements. However, it ends the execution of the stored subprogram and returns the control to the calling program. It is equivalent to a combination of RAISERROR and a RETURN statement.

## EXECUTE Statement

**Table 3–22 Comparison of EXECUTE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS EXEC SetExistFlag EXEC SetExistFlag yes=@yes, @Status OUT EXEC @Status = RecordExists EXEC SetExistFlag @yes</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS BEGIN     SetExistFlag;     SetExistFlag(yes=&gt;v_yes, Status);     Status:=RecordExists;     SetExistFlag(v_yes); END proc1;</pre>

### Comments

The EXECUTE statement is used to execute another stored procedure from within a procedure. In PL/SQL, the procedure is called by its name within the PL/SQL block.

SQL Developer converts the parameter-calling convention to be either positional, named, or mixed. For information on parameter-calling conventions, see "[Schema Objects](#)" on page 3-8.

## WHILE Statement

**Table 3–23 Comparison of WHILE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Example 1:</b>	<b>Example 1:</b>
<pre>CREATE PROC proc1   @i int AS     WHILE @i &gt; 0     BEGIN         print 'Looping inside WHILE....'         SELECT @i = @i + 1     END</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 (   iv_i IN NUMBER ) AS   v_i NUMBER(10,0) :=iv_i; BEGIN   WHILE v_i &gt; 0   LOOP     BEGIN       DBMS_OUTPUT.PUT_LINE('Looping inside WHILE....');       v_i := v_i + 1;     END;   END LOOP; END;</pre>

**Table 3–23 (Cont.) Comparison of WHILE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Example 2:</b>	<b>Example 2:</b>
<pre> CREATE PROC proc1   @i int,   @y int AS     WHILE @i &gt; 0     BEGIN         print 'Looping inside WHILE...'         SELECT @i = @i + 1     END </pre>	<pre> CREATE OR REPLACE PROCEDURE proc1 (   iv_i IN NUMBER,   v_y IN NUMBER ) AS   v_i NUMBER(10,0):=iv_i; BEGIN   WHILE v_i &gt; 0   LOOP     BEGIN       DBMS_OUTPUT.PUT_LINE('Looping inside WHILE...');       v_i := v_i + 1;     END;   END LOOP; END; </pre>

**Table 3–23 (Cont.) Comparison of WHILE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Example 3:</b>	<b>Example 3:</b>
<pre> CREATE PROC proc1 AS   DECLARE @sal money   SELECT @sal = 0   WHILE EXISTS(SELECT * FROM emp where sal &lt; @sal )   BEGIN     SELECT @sal = @sal + 99      DELETE emp     WHERE sal &lt; @sal   END GO </pre>	<pre> CREATE OR REPLACE PROCEDURE proc1 AS   v_sal NUMBER(19,2);   v_temp NUMBER(1, 0) := 0; BEGIN   v_sal := 0;   LOOP     v_temp := 0;     SELECT 1 INTO v_temp     FROM DUAL     WHERE EXISTS ( SELECT * FROM emp WHERE sal &lt; v_sal );     IF v_temp != 1 THEN       EXIT;     END IF;     BEGIN       v_sal := v_sal + 99;       DELETE emp       WHERE sal &lt; v_sal;     END;   END LOOP; END; </pre>

**Table 3–23 (Cont.) Comparison of WHILE Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<b>Example 4:</b>	<b>Example 4:</b>
<pre> CREATE PROC proc1 AS     DECLARE @sal money      WHILE (SELECT count (*) FROM emp ) &gt; 0     BEGIN         SELECT @sal = max(sal)         from emp         WHERE stat = 1          DELETE emp         WHERE sal &lt; @sal     END END GO </pre>	<pre> CREATE OR REPLACE PROCEDURE proc1 AS     v_sal NUMBER(19,2);     v_temp NUMBER(1, 0) := 0; BEGIN     LOOP         v_temp := 0;         SELECT 1 INTO v_temp         FROM DUAL         WHERE ( SELECT COUNT(*)         FROM emp ) &gt; 0;         IF v_temp != 1 THEN             EXIT;         END IF;         BEGIN             SELECT MAX(sal)             INTO v_sal             FROM emp             WHERE stat = 1;             DELETE emp             WHERE sal &lt; v_sal;         END;     END LOOP; END; </pre>

**Comments**

SQL Developer can convert most WHILE constructs. However, the CONTINUE within a WHILE loop in Microsoft SQL Server does not have a direct equivalent in PL/SQL. It is simulated using the GOTO statement with a label.

## GOTO Statement

**Table 3–24 Comparison of GOTO Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 @Status int AS DECLARE @j int         IF @Status = -1             GOTO Error          SELECT @j = -1 Error:         SELECT @j = -99</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 (     v_Status IN NUMBER ) AS     v_j NUMBER(10,0); BEGIN     IF v_Status = -1 THEN         GOTO Error;     END IF;     v_j := -1;     &lt;&lt;Error&gt;&gt;     v_j := -99; END;</pre>

### Comments

The GOTO <label> statement is converted automatically. No manual changes are required.

## @@Rowcount and @@Error Variables

**Table 3–25 Comparison of @@Rowcount and @@Error Variables in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre> CREATE PROC proc1 AS   DECLARE @x int   SELECT @x=count(*) FROM emp   IF @@rowcount = 0     print 'No rows found.'     IF @@error = 0       print 'No errors.'</pre>	<pre> CREATE OR REPLACE PROCEDURE proc1 AS   v_sys_error NUMBER := 0;   v_x NUMBER(10,0); BEGIN   BEGIN     SELECT COUNT(*)     INTO v_x     FROM emp ;   EXCEPTION     WHEN OTHERS THEN       v_sys_error := SQLCODE;   END;   IF SQL%ROWCOUNT = 0 THEN     DBMS_OUTPUT.PUT_LINE('No rows found.');</pre>
	<pre>   END IF;   IF v_sys_error = 0 THEN     DBMS_OUTPUT.PUT_LINE('No errors.');</pre>
	<pre>   END IF; END;</pre>

### Comments

@@rowcount is converted to the PL/SQL cursor attribute SQL%ROWCOUNT.

@@error is converted to v\_sys\_error, which contains the value returned by the SQLCODE function. The value returned by SQLCODE should only be assigned within an exception block; otherwise, it returns a value of zero.

## ASSIGNMENT Statement

**Table 3–26 Comparison of ASSIGNMENT Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS DECLARE @x int     SELECT @x = -1     SELECT @x=sum(salary) FROM employee</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS     v_x NUMBER(10,0); BEGIN     v_x := -1;     SELECT SUM(salary)     INTO v_x     FROM employee ; END;</pre>

### Comments

Assignment in Microsoft SQL Server is done using the SELECT statement, as illustrated in [Table 3–26](#).

PL/SQL assigns values to a variable as follows:

It uses the assignment statement to assign the value of a variable or an expression to a local variable. It assigns a value from a database using the SELECT . . INTO clause. This requires that the SQL returns only one row.

## SELECT Statement

**Table 3–27 Comparison of SELECT Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<p><b>Example 1</b></p> <pre>CREATE PROC proc1 AS SELECT ename FROM employee</pre>	<p><b>Example 1</b></p> <pre>CREATE OR REPLACE PROCEDURE proc1 (     cv_1 IN OUT SYS_REFCURSOR ) AS BEGIN     OPEN cv_1 FOR     SELECT ename     FROM employee ; END;</pre>

**Table 3–27 (Cont.) Comparison of SELECT Statement in Oracle and Microsoft SQL**

Microsoft SQL Server	Oracle
<b>Example 2</b>	<b>Example 2</b>
<pre>CREATE PROC proc1 AS DECLARE @name char(20)     SELECT @id = id     FROM employee  RETURN id</pre>	<pre>CREATE OR REPLACE FUNCTION proc1 AS     v_name CHAR(20); BEGIN     SELECT id     INTO v_id     FROM employee ; RETURN v_id; END;</pre>

**Comments**

Because of the differences in their architectures, Microsoft SQL Server stored procedures return data to the client program in a different way than Oracle.

Microsoft SQL Server and Oracle can all pass data to the client using output parameters in the stored procedures. Microsoft SQL Server 6.5 uses another method known as result sets to transfer the data from the server to client.

**SELECT Statement with GROUP BY Clause****Table 3–28 Comparison of SELECT Statement with GROUP BY Clause in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS DECLARE @ename char(20) DECLARE @salary int SELECT @ename=ename, @salary=salary FROM emp WHERE salary &gt; 100000 GROUP BY deptno</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS     v_ename CHAR(20);     v_salary NUMBER(10,0); BEGIN     SELECT ename,     salary     INTO v_ename,     v_salary     FROM emp     WHERE salary &gt; 100000     GROUP BY deptno; END;</pre>

### Comments

T/SQL allows GROUP BY statements where the column used in the GROUP BY clause does not need to be part of the SELECT list. PL/SQL does not allow this type of GROUP BY clause.

SQL Developer converts this type of SELECT statement to PL/SQL. However, the equivalent PL/SQL statement returns an error in Oracle.

## Column Aliases

**Table 3–29 Comparison of Column Aliases in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 @Status int=0 AS     SELECT x=sum(salary) FROM employee</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 (     v_Status IN NUMBER DEFAULT 0,     cv_1 IN OUT SYS_REFCURSOR ) AS BEGIN     OPEN cv_1 FOR         SELECT SUM(salary) x         FROM employee ; END;</pre>

### Comments

SQL Developer can convert Microsoft SQL Server-specific column aliases to the equivalent Oracle format. No manual changes are required.

## UPDATE with FROM Statement

**Table 3–30 Comparison of UPDATE with FROM Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS     UPDATE table1     SET col1 = 1     FROM table1, table2     WHERE table1.id = table2.id</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS BEGIN     UPDATE table1     SET ( col1 ) = ( SELECT 1     FROM table1 ,     table2     WHERE table1.id = table2.id ); END;</pre>

## DELETE with FROM Statement

**Table 3–31 Comparison of DELETE with FROM Statement in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS     DELETE FROM table1     FROM table1, table2     WHERE table1.id = table2.id</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS BEGIN     DELETE FROM table1     WHERE ROWID IN     (SELECT table1.ROWID     FROM table1, table2     WHERE table1.id = table2.id); END;</pre>

## Temporary Tables

**Table 3–32 Comparison of Temporary Tables in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC proc1 AS     SELECT col1, col2     INTO #Tab     FROM table1     WHERE table1.id = 100</pre>	<pre>CREATE OR REPLACE PROCEDURE proc1 AS BEGIN     DELETE FROM tt_Tab;      INSERT INTO tt_Tab (         SELECT col1,                col2         FROM table1         WHERE table1.id = 100 ); END;</pre>

### Comments

Temporary tables are supported by Oracle, and SQL Developer uses this feature. The DDL of the temporary table is extracted and generated as a standalone object.

## Cursor Handling

**Table 3–33 Comparison of Cursor Handling Result Set in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
<pre>CREATE PROC cursor_demo AS DECLARE @empno INT DECLARE @ename CHAR(100) DECLARE @sal FLOAT DECLARE cursor_1 CURSOR FOR SELECT empno, ename, sal FROM emp  OPEN cursor_1  FETCH cursor_1 INTO @empno, @ename, @sal  CLOSE cursor_1  DEALLOCATE CURSOR cursor_1</pre>	<pre>CREATE OR REPLACE PROCEDURE cursor_demo AS CURSOR cursor_1 IS SELECT empno, ename, sal FROM emp ; v_empno NUMBER(10,0); v_ename CHAR(100); v_sal NUMBER; BEGIN OPEN cursor_1; FETCH cursor_1 INTO v_empno,v_ename,v_ sal; CLOSE cursor_1; END;</pre>

### Comments

Microsoft SQL Server introduced cursors in T/SQL. Syntactical conversion of cursors from Microsoft SQL Server to Oracle is straightforward.

---

**Note:** In PL/SQL, deallocation of cursors is not required because it happens automatically.

---

## T/SQL and PL/SQL Language Elements

T/SQL is the Microsoft SQL Server procedural SQL language and PL/SQL is the Oracle procedural SQL language. This section discusses the following T/SQL and PL/SQL language elements:

- [Transaction Handling Semantics](#)
- [Exception-Handling and Error-Handling Semantics](#)
- [Special Global Variables](#)

- [Operators](#)
- [Built-in Functions](#)
- [DDL Constructs within Microsoft SQL Server Stored Procedures](#)

## Transaction Handling Semantics

### Microsoft SQL Server

Microsoft SQL Server offers two different transaction models: the ANSI-standard implicit transaction model and the explicit transaction model.

Microsoft SQL Server provides options to support ANSI-standard transactions. These options can be set or un-set using the SET command.

The following SET command sets the implicit transaction mode:

```
set chained on
```

The following SET command sets the isolation level to the desired level:

```
set transaction isolation level {1|3}
```

isolation level 1 prevents dirty reads. Isolation level 2 prevents un-repeatable reads. Isolation level 3 prevents phantoms. Isolation level 3 is required by ANSI standards. For Microsoft SQL Server, the default is isolation level 1.

To implement isolation level 3, Microsoft SQL Server applies HOLDLOCK to all the tables taking part in the transaction. In Microsoft SQL Server, HOLDLOCK, along with page-level locks, can block users for a considerable length of time, causing poor response time.

If the Microsoft SQL Server application implements ANSI-standard chained (implicit) transactions with isolation level 3, the application migrates smoothly to Oracle because Oracle implements the ANSI-standard implicit transaction model, which ensures repeatable reads.

In a non-ANSI standard application, Microsoft SQL Server transactions are explicit. A logical transaction has to be explicitly started with the statement BEGIN TRANSACTION. The transaction is committed with a COMMIT TRANSACTION or rolled back with a ROLLBACK TRANSACTION statement. The transactions can be named. For example, the following statement starts a transaction named

```
account_tran.  
BEGIN TRANSACTION account_tran
```

The explicit transaction mode allows nested transactions. However, the nesting is only syntactical. Only outermost `BEGIN TRANSACTION` and `COMMIT TRANSACTION` statements actually create and commit the transaction. This could be confusing as the inner `COMMIT TRANSACTION` does not actually commit.

The following example illustrates the nested transactions:

```
BEGIN TRANSACTION
  /* T/SQL Statements */
  BEGIN TRANSACTION
  /* T/SQL Statements */
    BEGIN TRANSACTION account_tran
    /* T/SQL Statements */
    IF SUCCESS
      COMMIT TRANSACTION account_tran
    ELSE
      ROLLBACK TRANSACTION account_tran
    END IF
  /* T/SQL Statements */
  IF SUCCESS
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
  END IF
  /* T/SQL Statements */
COMMIT TRANSACTION
```

When `BEGIN TRANSACTION` and `COMMIT TRANSACTION` statements are nested, the outermost pair creates and commits the transaction while the inner pairs only keep track of nesting levels. The transaction is not committed until the outermost `COMMIT TRANSACTION` statement is executed. Normally the nesting of the transaction occurs when stored procedures containing `BEGIN TRANSACTION` / `COMMIT TRANSACTION` statements call other procedures with transaction-handling statements. The global variable `@@trancount` keeps track of the number of currently active transactions for the current user. If you have more than one open transaction, you need to `ROLLBACK`, then `COMMIT`.

The named and unnamed inner `COMMIT TRANSACTION` statements have no effect. The inner `ROLLBACK TRANSACTION` statements without the name roll back the statements to the outermost `BEGIN TRANSACTION` statement and the current transaction is canceled. The named inner `ROLLBACK TRANSACTION` statements cancel the respective named transactions.

### **Oracle**

Oracle applies ANSI-standard implicit transaction methods. A logical transaction begins with the first executable SQL statement after a COMMIT, ROLLBACK, or connection to the database. A transaction ends with a COMMIT, ROLLBACK, or disconnection from the database. An implicit COMMIT statement is issued before and after each DDL statement. The implicit transaction model prevents artificial nesting of transactions because only one logical transaction per session can be in effect. The user can set SAVEPOINT in a transaction and roll back a partial transaction to the SAVEPOINT.

For example:

```
UPDATE test_table SET coll='value_1';
SAVEPOINT first_sp;
UPDATE test_table SET coll='value_2';
ROLLBACK TO SAVEPOINT first_sp;
COMMIT; /* coll is 'value_1'*/
```

### Conversion Preparation Recommendations

Logical transactions are handled differently in Microsoft SQL Server and Oracle. In Microsoft SQL Server, transactions are explicit by default. Oracle implements ANSI-standard implicit transactions. This prevents a direct conversion from T/SQL transaction-handling statements to PL/SQL transaction-handling statements.

Also, Microsoft SQL Server requires that transactions in stored procedures be allowed to nest, whereas Oracle does not support transaction nesting.

[Table 3–34](#) compares Microsoft SQL Server to Oracle transaction-handling statements:

**Table 3–34 Comparison of Transaction-Handling Statements in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
BEGIN TRAN	
BEGIN TRAN tran_1	SAVEPOINT tran_1
COMMIT TRAN (for the transaction with nest level=1)	COMMIT
COMMIT TRAN (for the transaction with nest level>1)	
COMMIT TRAN tran_1 (for the transaction with nest level=1)	COMMIT

**Table 3–34 (Cont.) Comparison of Transaction-Handling Statements in Oracle and Microsoft SQL Server**

Microsoft SQL Server	Oracle
COMMIT TRAN tran_1 (for the transaction with nest level>1)	
ROLLBACK TRAN	ROLLBACK
ROLLBACK TRAN tran_1	ROLLBACK TO SAVEPOINT tran_1

At the time of conversion, SQL Developer cannot determine the nest level of the current transaction-handling statement. The variable @@trancount is a runtime environment variable.

Table 3–35 shows the currently implemented Microsoft SQL Server to Oracle conversion strategy for the transaction-handling statements

**Table 3–35 Conversion Strategy for Transaction-Handling Statements**

Microsoft SQL Server	Oracle
BEGIN TRAN	/*BEGIN TRAN >>> statement ignored <<<*/
BEGIN TRAN tran_1	SAVEPOINT tran_1;
COMMIT TRAN (for the transaction with nest level=1)	COMMIT WORK;
COMMIT TRAN (for the transaction with nest level>1)	COMMIT WORK;
COMMIT TRAN tran_1 (for the transaction with nest level=1)	COMMIT WORK;
COMMIT TRAN tran_1 (for the transaction with nest level>1)	COMMIT WORK;
ROLLBACK TRAN	ROLLBACK WORK;
ROLLBACK TRAN tran_1	ROLLBACK TO SAVEPOINT tran_1
SAVE TRAN tran_1	SAVEPOINT tran_1

Because of the difference in the way the two databases handle transactions, you may want to consider some reorganization of the transactions.

Try to design client/server applications so that the transaction-handling statements are part of the client code rather than the stored procedure code. This strategy should work because the logical transactions are almost always designed by the user and should be controlled by the user.

For the conversion of stored procedures, consider setting a `SAVEPOINT` at the beginning of the procedures, and roll back only to the `SAVEPOINT`. In Microsoft SQL Server, make the changes so that at least the outermost transaction is controlled in the client application.

## Exception-Handling and Error-Handling Semantics

### Microsoft SQL Server

In Microsoft SQL Server, you must check for errors after each SQL statement because control is passed to the next statement regardless of any error conditions generated by the previous statement. The client `ERROR_HANDLER` routine is invoked as a call-back routine if any server error occurs, and the error conditions can be handled in the call-back routine.

Stored procedures use the `RAISERROR` statement to notify the client of any error condition. This statement does not cause the control to return to the calling routine.

Microsoft SQL Server allows you to customize the error messages using a system table. The system procedures allow the user to add error messages to this table.

### Oracle

In Oracle, each SQL statement is automatically checked for errors before proceeding with the next statement. If an error occurs, control immediately jumps to an exception handler if one exists. This frees you from needing to check the status of every SQL statement. For example, if a `SELECT` statement does not find any row in the database, an exception is raised. The corresponding exception handler part of the block should include the code to deal with this error. The built-in `RAISE_APPLICATION_ERROR` procedure notifies the client of the server error condition and returns immediately to the calling routine.

Oracle places an implicit `SAVEPOINT` at the beginning of a procedure. The built-in `RAISE_APPLICATION_ERROR` procedure rolls back to this `SAVEPOINT` or the last committed transaction within the procedure. The control is returned to the calling routine.

The Oracle `RAISE_APPLICATION_ERROR` statement allows the user to customize the error message. If an exception is raised, `SQLCODE` is returned automatically by PL/SQL to the caller. It keeps propagating until it is handled.

### Recommendations

To simulate Microsoft SQL Server behavior in Oracle, you must enclose each SQL statement in an equivalent PL/SQL block. This block must deal with the exceptions that need to be trapped for the SQL statement.

See "[T/SQL Versus PL/SQL Constructs](#)" on page 3-25 for more information about the extra code required to simulate Microsoft SQL Server behavior.

If the RAISERROR statement in a Microsoft SQL Server stored procedure is immediately followed by the RETURN statement, these two statements can be converted to the Oracle RAISE\_APPLICATION\_ERROR statement.

You can customize error messages with the help of a user-defined table. You can write standard routines to add and retrieve error messages to this table. This method serves a two-fold purpose: it ensures that the system is portable, and it gives the administrator centralized control over the error messages.

## Special Global Variables

### Microsoft SQL Server

In Microsoft SQL Server, the following global variables are particularly useful in the conversion process:

`@@error:`

The server error code indicating the execution status of the most recently executed T/SQL statement. For code examples, see "[@@Rowcount and @@Error Variables](#)" on page 3-38.

`@@identity:`

Returns the last identity value generated by the statement. It does not revert to a previous setting due to ROLLBACKS or other transactions.

`@@rowcount:`

The number of rows affected by the most recently executed T/SQL statement. For code examples, see "[@@Rowcount and @@Error Variables](#)" on page 3-38.

`@@servername:`

The name of the local Microsoft SQL Server server.

`@@sqlstatus:`

The status information resulting from the last FETCH statements.

`@@tranchained:`

The current transaction mode of the T/SQL procedure. If `@@tranchained` returns 1, the T/SQL procedure is in chained, or implicit transaction mode.

`@@trancount:`

Keeps track of the nesting level for the nested transactions for the current user.

`@@transtate:`

The current state of the transaction.

### **Oracle**

`SQLCODE:`

The server error code indicating the execution status of the most recently executed PL/SQL statement.

`SQL%ROWCOUNT:`

The variable attached to the implicit cursor associated with each SQL statement executed from within the PL/SQL procedures. This variable contains the number of rows affected by the execution of the SQL statement attached to the implicit cursor.

### **Recommendations:**

The `@@error` variable has a direct equivalent in Oracle, and that is the `SQLCODE` function. The `SQLCODE` function returns the server error code.

The `SQL%ROWCOUNT` variable in Oracle is functionally equivalent to `@@rowcount`.

There are many more special global variables available with PL/SQL. Not all those variables are listed here. There are more special global variables available in T/SQL also. Not all those variables are listed here because they do not play a major role in the conversion process.

## **Operators**

See "[Data Manipulation Language](#)" on page 2-23 for a discussion of Microsoft SQL Server and Oracle operators.

## **Built-in Functions**

See "[Data Manipulation Language](#)" on page 2-23 for a discussion of built-in functions in Microsoft SQL Server and Oracle.

## **DDL Constructs within Microsoft SQL Server Stored Procedures**

Microsoft SQL Server allows DDL constructs to be part of the stored procedures. Oracle allows DDL statements as part of the dynamic SQL. Oracle issues an implicit COMMIT statement after each DDL statement.

SQL Developer currently converts DDL constructs within stored procedures into standalone DDL constructs. The DDL is removed from the body of the stored procedure and is created before the procedure. Support for dynamic SQL is planned for a future release.

---

---

# Distributed Environments

This chapter includes the following sections:

- [Distributed Environments](#)
- [Application Development Tools](#)

## Distributed Environments

A distributed environment is chosen for various applications where:

- The data is generated at various geographical locations and needs to be available locally most of the time.
- The data and software processing is distributed to reduce the impact of any particular site or hardware failure.

## Accessing Remote Databases in a Distributed Environment

When a relational database management system (RDBMS) allows data to be distributed while providing the user with a single logical view of data, it supports "location transparency." Location transparency eliminates the need to know the actual physical location of the data. Location transparency thus helps make the development of the application easier. Depending on the needs of the application, the database administrator (DBA) can hide the location of the relevant data.

To access a remote object, the local server must establish a connection with the remote server. Each server requires unique names for the remote objects. The methods used to establish the connection with the remote server, and the naming conventions for the remote objects, differ from database to database.

## Oracle and Remote Objects

Oracle allows remote objects (such as tables, views, and procedures) throughout a distributed database to be referenced in SQL statements using global object names. In Oracle, the global name of a schema object comprises the name of the schema that contains the object, the object name, followed by an "at" sign (@), and a database name. For example, the following query selects information from the table named `scott.emp` in the `SALES` database that resides on a remote server:

```
SELECT * FROM  
scott.emp@sales.division3.acme.com
```

A distributed database system can be configured so that each database within the system has a unique database name, thereby providing "effective" global object names.

Furthermore, by defining synonyms for remote object names, you can eliminate references to the name of the remote database. The synonym is an object in the local database that refers to a remote database object. Synonyms shift the responsibility of distributing data from the application developer to the DBA. Synonyms allow the DBA to move the objects as desired without impacting the application.

The synonym can be defined as follows:

```
CREATE PUBLIC SYNONYM emp FOR  
scott.emp@sales.division3.acme.com;
```

Using this synonym, the preceding SQL statement can be changed to the following:

```
SELECT * FROM emp;
```

## Microsoft SQL Server and Remote Objects

Microsoft SQL Server requires schema objects throughout a distributed database to be referenced in SQL statements by fully qualifying the object names. The complete name of a schema object has the following format:

```
server_name.database_name.object_owner_name.object_name
```

The `server_name` is the name of a remote server. The `database_name` is the name of a remote database on the remote server.

Microsoft SQL Server does not support the concept of synonyms or location transparency. In a distributed environment, objects cannot be moved around without impacting the application, as the developers must include the location of the object in the application code.

Most of the static queries tend to include the references to the remote server and remote database. Some applications maintain a user table to map the complete object names (including the remote server name and the database name) to dummy object names. The queries refer to these dummy object names. The translations are performed in real-time with the help of the map in the user table. This limitation precludes any common scheme of referring to remote objects that can work for Oracle and Microsoft SQL Server.

The Microsoft SQL Server Omni SQL Gateway server allows location transparency, but this requires that the schema definitions of all the databases participating in the distribution must be available with the Omni SQL Gateway server.

## Replication

Replication functionality in Microsoft SQL Server has the following characteristics:

- Unidirectional
- Table-based, not transaction-based
- No automatic conflict resolution (must be manual)
- Heterogeneous replication through Open Database Connectivity (ODBC)

In addition to the preceding characteristics, Microsoft SQL Server 7.0 replication provides heterogeneous replication through ODBC.

Oracle replication has richer replication functionality, which includes the following:

- Bi-directional
- Any database object can be replicated
- Automatic resynchronization
- Automatic conflict resolution
- Heterogeneous replication provided through gateways

Since Oracle distributed environment and replication support is a superset of Microsoft SQL Server, conversion of distributed applications from Microsoft SQL Server to Oracle is feasible.

## Application Development Tools

Several application development tools that are currently available use specific features of one of the various database servers; you may have to invest significant

effort to port these products to other database servers. With critical applications, it is sometimes best to develop and maintain a different set of application development tools that work best with the underlying database, as ODBC support is not adequate in such cases.

The majority of Microsoft SQL Server applications are written using ODBC application programming interfaces (APIs) or Visual Basic. DB-Library is widely used to develop 3GL applications with Microsoft SQL Server as the back end.

Since Oracle provides ODBC connectivity, it is possible to convert ODBC-based Microsoft SQL Server applications to work with an Oracle back end.

If a Visual Basic application is written with ODBC as the connection protocol to access Microsoft SQL Server, it is possible to modify and fix the Visual Basic application to work with an Oracle back end.

Many Visual Basic applications use VB-SQL which is DB-Library for Visual Basic. VB-SQL allows Visual Basic programs to access Microsoft SQL Server natively (as opposed to using ODBC). Such applications can also be converted to work with an Oracle back end, if you replace the VB-SQL database access routines with Oracle Objects for OLE.

Oracle provides a call interface known as Oracle Call Interface (OCI), which is functionally equivalent to the DB-Library API. Conversion of DB-Library applications to OCI applications is feasible.

---

## Disconnected Source Model Loading

The Disconnected Source Model Load feature of SQL Developer allows consultants to work on a customer's database migration without having to install and run SQL Developer at the customer site.

To perform the disconnected source model load option a customer must generate delimited flat files containing schema metadata from the database to be migrated. You generate the flat file by running a predefined SQL Developer script against the source database. The flat files are sent to a consultant who uses SQL Developer to load the metadata files into a source and Oracle model. You can then map this schema to Oracle.

### Generating Database Metadata Flat Files

Microsoft SQL Server databases use the Bulk Copy Program (BCP) to generate delimited metadata flat files. Predefined scripts installed with SQL Developer invoke the BCP, and generate the flat files for each database. The BCP outputs delimited metadata files from the database with a `.dat` extension. However, for a successful migration of a database the `.dat` metadata files are converted into XML files by SQL Developer. SQL Developer converts the `.dat` files when the source metadata files are selected during the capture phase of the migration, and outputs the generated `.xml` files to the same root directory as the source `.dat` files.

### Flat File Generation Scripts

The predefined script files are stored in the `%ORACLE_HOME%\Omw\DSML_Scripts\plugin` directory. Refer to the following table to locate the correct SQL Developer script:

**Table 5–1 Location and Name of Script Files and Name of Associated Files**

Plug-in	Directory Location	Script File Name	Associated Files
Microsoft SQL Server 6.5	%ORACLE_HOME%\DSML_scripts\sqlserver6	SS6_DSML_SCRIPT.BAT	CREATE_SS65_INDEX_TABLES.SQL DROP_SS65_INDEX_TABLES.SQL
Microsoft SQL Server 7	%ORACLE_HOME%\DSML_scripts\sqlserver7	SS7_DSML_SCRIPT.BAT	Not Applicable
Microsoft SQL Server 2000	%ORACLE_HOME%\DSML_scripts\sqlserver2000	SS2K_DSML_SCRIPT.BAT	Not Applicable

### Running the Scripts

To run a script file for a plugin from the %ORACLE\_HOME%\DSML\_scripts\*plugin* directory, use the following command line:

```
<script file name> <database> <password> <server>
```

For example, to run the Microsoft SQL Server 2000 script file to generate metadata flat files, use the following command:

```
SS2K_DSML_SCRIPT <database> <password> <server>
```

---

---

# Index

## A

---

accessing remote databases, 4-1  
AFTER triggers, 3-1  
application development tools, 4-3  
arithmetic operators, 2-37  
ASSIGNMENT statement, 3-39

## B

---

BEGIN TRAN statement, 3-5  
BEGIN TRANSACTION statement, 3-5  
bit operators, 2-39  
BLOBs, 2-5  
built-in functions, 2-39, 3-51

## C

---

CHAR(n) data type, 2-11  
character functions, 2-39  
check constraints, 2-7  
column aliases, 3-41  
column names, 2-3  
column-level CHECK constraint, 2-7  
COMMIT TRAN statement, 3-5  
COMMIT TRANSACTION statement, 3-5  
comparison operators, 2-34  
connecting to a database, 2-24  
control files, 2-23  
CREATE PROCEDURE statement, 3-27  
cursor handling, 3-44  
customized error messages, 3-6

## D

---

data block, 2-19  
data concurrency, 2-45  
data files, 2-19  
data manipulation language, 2-23  
data storage concepts, 2-18  
data type mappings, 2-8  
data types, 3-7  
data types, conversion considerations, 2-3  
database devices, 2-19  
date functions, 2-42  
DATETIME data type, 2-3, 2-13  
DDL constructs, 3-51

declarative referential integrity, 2-6  
DECLARE statement, 3-28  
DELETE statement, 2-32  
DELETE triggers, 3-1  
DELETE with FROM statement, 3-42  
destination database, 1-3  
Disconnected Source Model Load, 5-1  
distributed environments, 4-1

## E

---

entity integrity constraints, 2-6  
error handling, 3-5  
error-handling semantics, 3-49  
exception-handling semantics, 3-49  
EXECUTE statement, 3-32  
explicit transaction model, 3-45  
extent, 2-19

## F

---

features, 1-2  
Flat File Generation Scripts, 5-1  
FLOAT data type, 2-10  
function, schema object, 3-14  
functions, defining in Oracle, 2-42

## G

---

global variables, 3-38, 3-50  
GOTO statement, 3-37

## I

---

IF statement, 3-29  
IMAGE data type, 2-5  
implicit transaction model, 3-45  
individual SQL statements, 3-3  
INSERT statement, 2-29  
INSERT triggers, 3-1

## L

---

locking concepts, 2-45  
logical transaction, 3-5  
logical transaction handling, 2-50

## M

---

mathematical functions, 2-44  
metadata flat files  
    generating, 5-1  
miscellaneous functions, 2-42

## O

---

object names, 2-3  
operators, 2-34, 3-51  
Oracle Model, 1-3

## P

---

package body, 3-22  
package, schema object, 3-18  
page, 2-19  
page-level locking, 2-48  
parameter passing, 3-27  
PL/SQL and T-SQL constructs, comparison, 3-25  
PL/SQL and T-SQL, language elements, 3-44  
procedure, schema object, 3-8  
product description, 1-1

## R

---

RAISERROR statement, 3-6, 3-32  
read consistency, 2-49  
redo log files, 2-21  
referential integrity, 3-2  
referential integrity constraints, 2-6  
remote objects, Oracle, 4-2  
remote objects, SQL Server, 4-2  
replication, 4-3  
repository, 1-3  
reserved words, 2-3  
RETURN statement, 3-31  
ROLLBACK TRAN statement, 3-5  
ROLLBACK TRANSACTION statement, 3-5  
row-level locking, 2-48

## S

---

schema migration, 2-1  
schema object similarities, 2-1  
SELECT INTO statement, 2-28  
SELECT statement, 2-25, 3-39  
SELECT statement, with GROUP BY clause, 3-40  
SELECT statements without FROM clauses, 2-28  
SELECT with GROUP BY statement, 2-29  
set operators, 2-38  
source database, 1-3  
Source Model, 1-3  
special global variables, 3-50  
stored procedures, SQL Server, 3-3  
stored subprograms, Oracle, 3-3  
string operators, 2-38  
SYSNAME data type, 2-17

## T

---

table design considerations, 2-3  
table-level CHECK constraint, 2-7  
tablespace, 2-20  
temporary tables, comparison, 3-43  
TEXT data type, 2-5  
TIMESTAMP data type, 2-17  
transaction handling semantics, 3-45  
triggers, Oracle, 3-3  
triggers, SQL Server, 3-3  
T-SQL and PL/SQL constructs, comparison, 3-25  
T-SQL and PL/SQL, language elements, 3-44  
T-SQL local variables, 3-7

## U

---

unique keys, 2-6  
UPDATE statement, 2-30  
UPDATE triggers, 3-1  
UPDATE with FROM statement, 3-42  
user-defined types, SQL Server, 2-5

## V

---

VARCHAR(n) data type, 2-12

## W

---

WHILE statement, 3-33