Oracle Migration Workbench for MS SQL Server and Sybase Adaptive Server Reference Guide

Release 1.2.5.0.0 for Windows NT and Windows 95/98

December 1999

Part No. Z26179-01

This reference guide describes how to migrate from MS SQL Server 6.5, MS SQL Server 7.0, and Sybase Adaptive Server 11 to Oracle8 or Oracle8*i*.

Oracle Migration Workbench Reference for MS SQL Server and Sybase Adaptive Server, Release 1.2.5.0.0

Part No. Z26179-01

Release 1.2.5.0.0

Contents

3 Triggers and Stored Procedures

4 Distributed Environments

Send Us Your Comments

Oracle Migration Workbench for MS SQL Server and Sybase Adaptive Server Reference Guide, Release 1.2.5.0.0 for Windows NT and Windows 95/98

Part No. Z26179-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Email - mwbinfo@ie.oracle.com
- FAX - +353-1-803-1899
- Postal service:
  Documentation Manager
  Migration Technology Group
  Oracle Corporation
  Eastpoint Business Park
  Dublin 3
  Ireland

If you would like a reply, please give your name, address, and telephone number below.

Preface

The *Oracle Migration Workbench for MS SQL Server and Sybase Adaptive Server Reference Guide* provides detailed information about migrating a database from MS SQL Server 6.5, MS SQL Server 7.0, and Sybase Adaptive Server 11 to Oracle8 or Oracle8*i*. 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 MS SQL Server, Sybase Adaptive Server, and Oracle and outlines how those differences are handled by the Migration Workbench during the conversion process.

Audience

This guide is intended for anyone who is involved in converting an MS SQL Server or Sybase Adaptive Server database to Oracle using the Migration Workbench.

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 MS SQL Server or Sybase Adaptive Server.

How the Oracle Migration Workbench for MS SQL Server and Sybase Adaptive Server Reference Guide is Organized

This reference guide is organized as follows:

Chapter 1, "Overview"

Introduces the Migration Workbench and outlines features of this tool.

Chapter 2, "Databases"

Contains detailed information about the differences between data types, data storage concepts, schema objects, and the data manipulation language in MS SQL Server, Sybase Adaptive Server, and Oracle.

Chapter 3, "Triggers and Stored Procedures"

Introduces triggers and stored procedures, and compares T-SQL and PL/SQL language elements and constructs in MS SQL Server, Sybase Adaptive Server, and Oracle.

Chapter 4, "Distributed Environments"

Describes when and why distributed environments are used, and discusses application development tools.

Chapter 5, "Migrating Temporary Tables to Oracle"

Describes how to emulate temporary tables in Oracle8.

How to Use This Reference Guide

Every reader of this reference guide should read Chapter 1, "Overview" as that chapter provides an introduction to the concept and terminology of the Migration Workbench.

Conventions Used in This Reference Guide

The following typographic conventions are used in this reference guide:

Convention	Description
UPPERCASE	Uppercase text indicates case-insensitive filenames or directory names, commands, command keywords, initializing parameters, data types, table names, or object names. Enter text exactly as spelled; it need not be in uppercase.
[UPPERCASE]	Key names are represented by uppercase letters enclosed in brackets, as square in [RETURN].
Italicized Characters	Italicized Italic type in text indicates the complete names of documents, emphasizes a single word or short phrase, indicates variables, or indicates the first instance of an important word or phrase.

Convention	Description
UPPERCASE	Uppercase text indicates case-insensitive filenames or directory names, commands, command keywords, initializing parameters, data types, table names, or object names. Enter text exactly as spelled; it need not be in uppercase.
Code Examples	Monospace text distinguishes examples of commands and statements from the rest of the text. Monospace text must be entered exactly as shown. Example statements may include punctuation, such as commas or quotation marks. All punctuation in example statements is required. All example statements terminate with a semicolon (;). Depending on the application, a semicolon or other terminator may or may not be required to end a statement.
UPPERCASE in Code Examples	Uppercase words in example statements indicate the keywords within Oracle SQL. When you issue statements, however, keywords are not case sensitive.
lowercase in Code Examples	Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.
Bold	Boldface type in text indicates emphasis with stress, a term defined in the text or the glossary or in both locations, or case-sensitive filenames or directory names.
>	Right-facing angle brackets appear in navigation paths to indicate movement from one Web page to another.
{ }	Curly braces indicate that one of the enclosed arguments is required. Do not enter the braces themselves.
[ ]	Square brackets indicate that the enclosed arguments are optional. Do not enter the brackets themselves.
|	A vertical bar separates alternative items that may be optional or required. Do not type the vertical bar.
...	Ellipses indicate that the preceding item can be repeated. You can enter an arbitrary number of similaritems. In code fragments, an ellipsis means that code not relevant to the discussion has been omitted. Do not type the ellipsis.
SQL*Plus Prompts	The SQL*Plus prompt, SQL>, appears in SQL statement and SQL*Plus command examples. Enter your response at the prompt. Do not enter the text of the prompt, SQL>, in your response.

Convention	Description
UPPERCASE	Uppercase text indicates case-insensitive filenames or directory names, commands, command keywords, initializing parameters, data types, table names, or object names. Enter text exactly as spelled; it need not be in uppercase.
MS-DOS Prompts	The MS-DOS prompt, >, appears in MS-DOS command examples. Enter your response at the prompt. Do not enter the prompt in your response.
Storage Measurements	Storage measurements use these abbreviations: K, for kilobyte which equals 1024 bytes M, for megabyte which equals 1 048 576 bytes G, for gigabyte which equals 1 073 741 824 bytes

1

Overview

This chapter introduces the Oracle Migration Workbench (Migration Workbench) under the following headings:

Introduction

The Migration Workbench is a tool that simplifies the process of migrating data and applications from an MS SQL Server 6.5, MS SQL Server 7.0, or Sybase Adaptive Server 11 environment to Oracle8 or Oracle8*i*. The Migration Workbench 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:** MS SQL Server is used in this document to refer to both MS SQL Server 6.5 and MS SQL Server 7.0 unless otherwise stated.

Product Description

The Migration Workbench allows you to migrate an MS SQL Server or Sybase Adaptive Server database to an Oracle8 or Oracle8*i* database. The Migration Workbench employs an intuitive and informative User Interface and a series of wizards to simplify the migration process. To ensure portability, all components of the Migration Workbench are written in Java.

The Migration Workbench uses a repository to store migration information. This allows you to query the initial state of the application before migration. By initially loading the migratable components of the application system into a repository, you can work independently of the production application.

Furthermore, the Migration Workbench saves useful dependency information about the components being converted. For example, the Migration Workbench 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

The Migration Workbench allows you to:

- Migrate a complete MS SQL Server 6.5, MS SQL Server 7.0, or Sybase Adaptive Server 11 database to Oracle8 or Oracle8*i*.
- 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 MS SQL Server or Sybase Adaptive 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.

Terminology

The following terms are used to describe the Migration Workbench:

Application System is the database schema and application files that have been developed for a database environment other than Oracle. For example, MS SQL Server 6.5, MS SQL Server 7.0, or Sybase Adaptive Server 11.

Capture Wizard is an intuitive wizard that takes a snapshot of the data dictionary of the source database, loads it into the Source Model, and creates the Oracle Model.

Migration Wizard is an intuitive wizard that helps you migrate the source database to Oracle.

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.

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

Migration Workbench is the graphical tool that allows migration of an application system to an Oracle database environment.

Workbench Repository is the area in an Oracle database used to store the persistent information necessary for the Migration Workbench to migrate an application system.

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 the Migration Workbench. The source database is a database other than Oracle, for example, MS SQL Server.

Destination Database is the Oracle database to which the Migration Workbench migrates the data dictionary of the source database.

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

Oracle Model is a 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 will look when generated in an Oracle environment.

Navigator Pane is the part of the Migration Workbench User Interface that contains the tree views representing the Source Model and the Oracle Model.

Properties Pane is the part of the Migration Workbench 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 Migration Workbench User Interface that contains informational, error, or warning messages describing the progress of the migration process.

2

Databases

This chapter includes the following sections:

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

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

Table 2–1 Schema Objects in Oracle and MS SQL Server/Sybase

Oracle	MS SQL Server/Sybase
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, MS SQL Server, and Sybase. Many Oracle reserved words are valid object or column names in MS SQL Server and Sybase. For example, DATE is a reserved word in Oracle, but it is not a reserved word in MS SQL Server and Sybase. Therefore, no column is allowed to have the name DATE in Oracle, but a column can be named DATE in MS SQL Server or Sybase. 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 your object name is not a reserved word from either database.

For a list of reserved words in Oracle, see the *Oracle8i SQL Reference, Release 2 (8.1.6)* (Part Number A76989-01).

Table Design Considerations

This section discusses the many table design issues that you need to consider when converting MS SQL Server or Sybase 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 outlines conversion considerations for the following data types:

- DATETIME Data Types
- IMAGE and TEXT Data Types (Binary Large Objects)
- MS SQL Server and Sybase User-Defined Data Types

DATETIME Data Types

The date/time precision in MS SQL Server and Sybase is 1/300th of a second; in Oracle, the precision is one second. All three databases store point-in-time values

for DATE and TIME data types. In MS SQL Server and Sybase, the DATETIME data type stores date and time values that are accurate to 1/300th of a second. Oracle uses the DATE data type and stores date and time values that are accurate to one second.

For applications that require finer date/time precision than seconds, the table design must include an INTEGER column with each DATE column. Oracle needs this additional column to store the value of the sequence along with the DATE value, in order to store the sub-second information.

As an alternative, if an MS SQL Server or Sybase 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 will suffice:

Original Table Design

MS SQL Server/Sybase:

```
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 MS SQL Server or Sybase table, you can keep the same table design for the Oracle database.

Revised Table Design

MS SQL Server/Sybase:

```
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 MS SQL Server or Sybase 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, nanoseconds, etc.). The precision of the MS SQL Server and Sybase DATETIME data type is 1/300th of a second; the precision of the Oracle DATE data type is 1 second.

The MS SQL Server and Sybase DATETIME data type can be converted to a higher precision in Oracle using one of the following methods:

- The GET_TIME function in the system-defined DBMS_UTILITY package returns the time in 100ths of a second. The MS SQL Server or Sybase DATETIME data type can be converted to Oracle as a combination of DATE column and a NUMBER column, where the NUMBER column holds the time returned by the DBMS_UTILITY.GET_TIME function.
- The following statement is similar to the DBMS_UTILITY.GET_TIME function, and also returns the time in 100ths of a second:

```
SQL> SELECT hsecs FROM v$timer;
```

IMAGE and TEXT Data Types (Binary Large Objects)

The physical and logical storage methods for IMAGE and TEXT data differ from Oracle to MS SQL Server and Sybase. In MS SQL Server and Sybase, 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 MS SQL Server and Sybase 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.

MS SQL Server and Sybase User-Defined Data Types

This MS SQL Server and Sybase 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.

While migrating to Oracle PL/SQL, you must determine the base data type for each user-defined data type, to find the equivalent PL/SQL data type. Note that 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 MS SQL Server or Sybase. 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 MS SQL Server or Sybase. 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. Please refer to *Oracle8i Concepts, Release 2 (8.1.6)* (Part Number A76965-01), for details of the functionality that is possible for referential integrity constraints.

Unique Key Constraints

You can define a unique key for a table in MS SQL Server or Sybase. 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 MS SQL Server and Sybase to Oracle.

Check Constraints

Check constraints can be defined in a CREATE TABLE statement or an ALTER TABLE statement in MS SQL Server or Sybase. 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 will pass all the phone numbers that resemble the following:
(650)506-7000

This rule will fail 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 MS SQL Server and Sybase databases map one-to-one with Oracle check constraints. You can implement the column-level check constraints from the MS SQL Server or Sybase database to Oracle table-level check constraints. While converting the regular expressions, convert all simple regular expressions to check constraints in Oracle. MS SQL Server and Sybase 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 MS SQL Server, Sybase, and Oracle databases. Specifically, this chapter contains the following information:

- A table showing the base MS SQL Server and Sybase 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 MS SQL Server/Sybase

MS SQL Server/Sybase	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 and2^31. Or, place appropriate constraints such as: STATE_NO between 1 and 50

Table 2–2 Data Types in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Description	Oracle	Comments
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.
REAL	Four-byte, single-precision floating point number. This column has 7-digit precision. The range of values and the actual representation is platform dependent. This can result in incorrect interpretation if data is moved between platforms.	FLOAT	You may want to add a check constraint to constrain range of values. Also, 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.
FLOAT	A floating point number. This column has 15-digit precision.	FLOAT	You may want to add a check constraint to constrain range of values. Also, 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.
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.	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.

Table 2–2 Data Types in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Description	Oracle	Comments
CHAR(n)	Fixed-length string of exactly n 8-bit characters, blank padded. Synonym for CHARACTER. 0 < n < 256 for MS SQL Server 6.5 and Sybase. 0 < n < 8000 for MS SQL Server 7.0.	CHAR(n)	Pro*C client programs must use mode=ansi to have characters interpreted correctly for string comparison, mode=oracle otherwise.
VARCHAR(n)	Varying-length character string. 0 < n < 256 for MS SQL Server 6.5 and Sybase. 0 < n < 8000 for MS 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 MS SQL Server 6.5 and Sybase. 0 < n < 8000 for MS SQL Server 7.0.	RAW(n)/BLOB	
VARBINARY(n)	Varying length binary string of up to n 8-bit bytes. 0 < n < 256 for MS SQL Server 6.5 and Sybase. 0 < n < 8000 for MS SQL Server 7.0.	RAW(n)/BLOB	

Table 2–2 Data Types in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	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 through 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	The precision of DATE in Oracle and DATETIME in MS SQL Server and Sybase 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 MS SQL Server and Sybase, the column of type DATETIME can contain UNIQUE values because the DATETIME precision in MS SQL Server and Sybase 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.
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 Data Types in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Description	Oracle	Comments
MONEY	A monetary value represented as an integer portion and a decimal fraction, and stored as two 4-byte integers. Accuracy to the nearest 1/10,000. Data of this type should have a preceding dollar ($) sign when input. In the absence of the "$" sign, MS SQL Server and Sybase create the value as a float.	NUMBER(19,4)	MS SQL Server and Sybase input MONEY data types as a numeric data type with a preceding dollar sign ($) as in the following example, select * from table_x where y > $5.00 You must remove the "$" 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.
SMALLMONEY	Same as MONEY but constrained to be within a range.	NUMBER(10,4)	Since the range is -214,748.3648 to 214,748.364, NUMBER(10,4) suffices for this field.
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 incremented 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 MS SQL Server 6.5 and Sybase. NVARCHAR(128) in MS SQL Server 7.0.	VARCHAR2(30) and VARCHAR2(128) respectively.	

TEXT and IMAGE data types in MS SQL Server and Sybase follow the rules listed below:

- 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.

Recommendations

In addition to the data types listed in Table 2-2, users can define their own data types in MS SQL Server and Sybase 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.

Data types can easily be mapped from MS SQL Server and Sybase to Oracle with the equivalent data types listed in the above table. The Migration Workbench converts user-defined data types to their base type. You can defined 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 MS SQL Server, Sybase, and Oracle databases.

Specifically, it contains the following information:

- A table comparing the data storage concepts of MS SQL Server, Sybase, and Oracle databases
- Recommendations based on the information listed in the table

Data Storage Concepts Table

Table 2–3 Data Storage Concepts in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Database Devices:	**Datafiles:**
A database device is mapped to the specified physical disk files.	One or more datafiles are created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the datafiles 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 datafile cannot change in size. This limitation does not exist in Oracle.
Page:	**Data Block:**
Many pages constitute a database device. Each page contains a certain number of bytes.	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.
Extent:	**Extent:**
Eight pages make one extent. Space is allocated to all the databases in increments of one extent at a time.	An extent is a specific number of contiguous data blocks, obtained in a single allocation.
N/A	**Segments:**
	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 datafiles.

Table 2–3 Data Storage Concepts in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Segments (corresponds to Oracle Tablespace): 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. The following segments are created along with the database: ▫ System segment Stores the system tables. ▫ Log segment Stores the transaction log. ▫ Default segment All other database objects are stored on this segment unless specified otherwise. Segments are subsets of database devices.	**Tablespace (corresponds to MS SQL Server and Sybase Segments):** 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. **Tablespace Extent:** An extent is a specific number of contiguous data blocks within the same tablespace. **Tablespace Segments:** 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. There are four different types of segments as follows: ▫ 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. ▫ Index segment Each index has an index segment that stores all of its data. ▫ 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.

Table 2–3 Data Storage Concepts in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
	■ 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.
Log Devices: These are logical devices assigned to store the log. The database device to store the logs can be specified while creating the database.	**Redo Log Files:** 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.

Table 2–3 Data Storage Concepts in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Database Devices: 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. 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: ▪ Allocated to the free space available to a database ▪ Allocated to store specific user objects ▪ Used to store the transaction log of a database ▪ Labeled as default device to create and alter database objects 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. 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.	N/A
Dump Devices These are logical devices. A database dump is stored on these devices. The DUMP DATABASE command uses the dump device to dump the database.	N/A

Table 2–3 Data Storage Concepts in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N/A	**Control Files:** Each database has a control file. This file records the physical structure of the database. It contains the following information: ▪ database name ▪ names and locations of a database's datafiles and redo log files ▪ time stamp of database creation 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 MS SQL Server and Sybase.

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, MS SQL Server, and Sybase all have a way to control the physical placement of a database object. In MS SQL Server and Sybase, 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 MS SQL Server or Sybase to Oracle. The decisions that were made when defining the storage of the database objects for MS SQL Server or Sybase should also apply for Oracle. Especially important are initial object sizes and physical object placement.

Schema Objects

This section compares the following MS SQL Server, Sybase, and Oracle schema objects:

- Alias
- Database
- Database Link
- Data and Hash Cluster
- Defaults
- Index
- Privilege
- Profile
- Role
- Rule
- Sequence
- Snapshot
- Synonym
- Tables
- Tablespace
- User
- View

Each schema object is compared in separate tables based on create, alter, drop, grant, revoke, and truncate where applicable. Most tables are divided into the following four sections:

- syntax
- description
- permissions
- examples

Each table is followed by a recommendations section that contains important information about conversion implications.

Alias

This section contains the following tables for the schema object Alias:

- Create
- Drop

Create

Table 2–4 Comparison of Creating the Alias Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
sp_addalias login_id usr_nm_inside_db	Oracle does not have aliases. Similar functionality is provided by roles. See the Roles section for more information in this regard.
Description:	N / A
The purpose of an alias is to allow an individual to access the database as another database user without the DBA having to add him or her as a user in the database. The SP_ADDALIAS system procedure creates a row in the SYSALTERNATES table, which keeps all information about aliases. An alias may be set up for any user. This allows more than one user ID to have the same set of privileges as the base user ID.	
Permissions:	N / A
The SA or DBO can add an alias.	
Example:	N / A

`sp_addalias user1 dbo`
`sp_addalias user2 dbo`
Both user1 and user2 can act as DBO.	

Recommendations:

A user account can be created for each alias set up on MS SQL Server or Sybase. The same privileges as the base user ID can be granted to this account.

Drop

Table 2–5 Comparison of Dropping the Alias Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
`sp_dropalias login_id`	Oracle does not have aliases. Functionality is provided by roles. See the Roles section for more information in this regard.
Description:	N/A
The SP_DROPALIAS system procedure allows you to drop an alias.	
Permissions:	N/A
The SA or DBO can drop an alias.	
Example:	N/A
`sp_dropalias user1`	

Recommendations:

Oracle does not have aliases. The MS SQL Server and Sybase alias dropping information is provided for reference.

Database

This section contains the following tables for the schema object Database:

- Create
- Alter
- Drop

Create

Table 2–6 Comparison of Creating the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`CREATE DATABASE database_name`
`[ON {DEFAULT | database_`
`device}[= size]`
`[, database_device [= size]]`
`...]`
`[LOG ON database_device [=`
`size]`
`[, database_device [=`
`size]]...]`
	Syntax:
	`CREATE DATABASE database_name`
	`[CONTROLFILE REUSE]`
	`[LOGFILE [GROUP integer] file_`
	`definition`
	`[, [GROUP integer] file_`
	`definition]...]`
	`[MAXLOGFILES integer]`
	`[MAXLOGMEMBERS] integer]`
	`[MAXLOGHISTORY] integer]`
	`[DATAFILE file_definition`
	`[,file_definition]...]`
	`[MAXDATAFILES integer]`
	`[MAXINSTANCES integer]`
	`[ARCHIVELOG | NOARCHIVELOG]`
	`[EXCLUSIVE]`
	`[CHARACTER SET charset]`

Table 2–6 Comparison of Creating the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
Each server can manage up to 32767 databases.	One server controls one database. One server instance comprises the system global area (SGA) and a few processes. The SGA is used to store data blocks and parsed SQL statements (including PL/SQL blocks) in shared memory.
The master database contains all the necessary information about all the other databases created by the users. The model database is used as a template to create other databases. The `tempdb` is a temporary database that can be used as a working storage by all the users on the server.	A given database consists of a system tablespace and one or more user-defined tablespaces.
	Each of the tablespaces may have its own storage, backup, and recovery strategy.
A database can be assigned as a default database to the user. Note that in Oracle a tablespace can be assigned as a default tablespace to the users. In order to use an object in another database, the object name has to be fully specified in order to reference that object. This can be seen in the following example: `database_name.owner_name.object_name`	The CREATE DATABASE command makes a database ready for initial use. It clears the database files and loads initial database tables required by the RDBMS. **Caution:** If you use CREATE DATABASE on an existing database, you will destroy the database.
A database is an atomic storage unit for administration, backup, and recovery.	The control file stores the physical structure of the database. The file definition takes the following form: `'file' [SIZE integer [K | M] [REUSE]]`
CREATE DATABASE creates the specification of the database in the master database (system tables) and creates the database itself as a copy of the model database. A 2M database is created on the default device when the values are not specified.	SIZE is the number of bytes set aside for this file. The suffix K multiplies the value by 1024 and suffix M multiplies it by 1048576. SIZE and REUSE together tell Oracle to reuse the file if it already exists, or create it if it does not exist. SIZE without REUSE creates a file if one does not already exist, but returns an error if a file does exist. Without SIZE, the file must already exist. LOGFILE names the files to be used as the redo log files. Mirrored log files are recommended for full protection against media failure. MAXLOGFILES overrides the LOG_FILES parameter specified in the init.orafile, and defines the maximum number of redo log files that can be created for this database.

Table 2–6 Comparison of Creating the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description (continued):	**Description (continued):**
The optional ON clause lets you put the database on a specific device. In MS SQL Server and Sybase, the physical devices are initialized using the DISK INIT command. This command attaches a logical device name to physical devices. These logical device names are used in the CREATE DATABASE statement. The LOG ON option creates the transaction log (the SYSLOGS table) on a separate device, and improves performance. If the LOG ON option is omitted, the transaction log is created on the same device as the data tables.	ARCHIVELOG and NOARCHIVELOG define the way redo log files are used when the database is first created. NOARCHIVELOG is the default, meaning that redo log files are reused without saving the contents elsewhere. This provides instance recovery, but does not provide recovery from a media failure. ARCHIVELOG forces redo log files to be archived so you can recover from media failure. ARCHIVELOG also supports instance recovery. DATAFILE names the datafiles used by the database. These files exist in the SYSTEM tablespace. MAXDATAFILES is the maximum number of datafiles that can be created for this database. MAXINSTANCES overrides the INSTANCES parameter in the **init.ora file, and sets the maximum number of simultaneous instances that can mount and open this database.** The optional REUSE clause tells the server to destroys the contents of the named file and associate this file to the database in the context. EXCLUSIVE is optional and means that all databases are created to allow only one instance that has an exclusive access to the database and allows only one instance of exclusive access to any database created. Use the ALTER DATABASE DISMOUNT and ALTER DATABASE MOUNT PARALLEL commands to allow multiple instances to access the database.

Table 2–6 Comparison of Creating the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Permissions:	**Permissions:**
The SA can grant permission to use the CREATE DATABASE command. The SA usually retains the CREATE DATABASE permission, and changes the database owner of the database by using the SP_CHANGEDBOWNER system procedure in the new database.	You must have the OSDBA role enabled in order to issue this command.
Examples:	**Example:**

`CREATE DATABASE my_database`
`ON DEFAULT =`
	`CREATE DATABASE my_database`
	`LOGFILE`
	`GROUP 1`
	`('test_log1a', 'test_log1b') SIZE 500K,`
	`GROUP 2`
	`('test_log2a', 'test_log2b') SIZE 500K`
	`DATAFILE 'test_system' SIZE 10M`

Recommendations:

Because of the conceptual differences between the two databases, the best approach is to create the database manually in Oracle. Sizing and backup methods chosen for MS SQL Server and Sybase often apply nearly as well for Oracle. Care should be taken to ensure that sizing and backup information learned by using MS SQL Server or Sybase is passed on to the Oracle database.

MS SQL Server and Sybase applications that use two or more databases on the same server usually translate to one Oracle database with several tablespaces and a different user for each tablespace.

In most cases, the MS SQL Server or Sybase application should be converted so that one server in MS SQL Server or Sybase is converted to one Oracle database instance.

The ability to backup / restore individual databases in MS SQL Server or Sybase is provided by creating one Oracle tablespace for each MS SQL Server or Sybase database. Then DBAs can perform the same backup / restore by tablespace on Oracle that they could by database in MS SQL Server or Sybase.

The ability of MS SQL Server and Sybase to keep logical sets of tables together in databases is accomplished in Oracle by creating tablespaces.

The use of tablespaces in Oracle can provide all the same benefits of multiple databases per server in MS SQL Server and Sybase. However, if there are several completely unrelated databases in the same server in MS SQL Server or Sybase, it may make sense to split them into completely different database instances in Oracle.

Alter

Table 2–7 Comparison of Altering the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: ALTER DATABASE database {ADD FILE <filespec> [,...n] [TO FILEGROUP filegroup_name] |ADD LOG FILE <filespec> [,...n] |REMOVE FILE logical_file_name |ADD FILEGROUP filegroup_name |REMOVE FILEGROUP filegroup_ name |MODIFY FILE <filespec> |MODIFY FILEGROUP filegroup_ name filegroup_property }	**Syntax:** ALTER DATABASE [database_name] {ADD LOGFILE [THREAD integer] [GROUP integer] file_definition [,file_ definition]...| ADD LOGFILE MEMBER file [REUSE][, file [REUSE]...] TO {GROUP integer |(file[,file]...) | file} | DROP LOGFILE {GROUP integer|(file[,file]...)|file} | DROP LOGFILE MEMBER file[,file] | RENAME file TO file | NOARCHIVELOG | ARCHIVELOG | MOUNT [EXCLUSIVE | PARALLEL] | OPEN [RESETLOGS | NORESETLOGS] | ENABLE [PUBLIC] THREAD integer | DISABLE THREAD integer | BACKUP CONTROLFILE TO file [REUSE] | DATAFILE file {ONLINE|OFFLINE [DROP]} | CREATE DATAFILE file[,file] [AS file_spec[,file_spec]...] | RENAME GLOBAL_NAME TO database [.domain]| RECOVER recover_clause | SET {DBMAC {ON | OFF} | DBHIGH = string | DBLOW = string}}

Table 2–7 Comparison of Altering the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
ALTER DATABASE is used to add, remove, or modify files or file groups from the database. The database size can also be altered by changing the value of the "database size" configuration variable by using the SP_CONFIGURE system procedure. The reconfigure command should be used to apply the changes.	The log file is assigned to a thread, either explicitly with the THREAD clause or to the thread assigned to the current Oracle instance. Use the THREAD parameter only if you are using Oracle with the parallel server option in parallel mode. The file definition takes the following form: 'file' [SIZE integer [K | M] [REUSE]] SIZE is the number of bytes set aside for this file. The suffix K multiplies the value by 1024 and suffix M multiplies it by 1048576. SIZE and REUSE together tell Oracle to reuse the file if it already exists, or to create it if it does not exist. SIZE without REUSE creates a file if one does not already exist, but returns an error if a file does exist. Without SIZE, the file must already exist. A GROUP is a collection of log files. You can add a GROUP by listing the log files or naming them with an integer. If a mirrored redo log is used, groups of online redo log files can be created. Each member in a GROUP is exactly the same size. The members in the GROUP are multiple copies of the redo log created to protect the log against losing the drive with the log file. ADD LOGFILE MEMBER adds new files to an existing log file group. DROP FILE drops an existing redo log file group. DROP LOGFILE MEMBER drops one or more members of a log file group. RENAME changes the name of the existing database or log file. When you first create the database, it is mounted in exclusive mode, meaning only its creator has access to it. To allow multiple instances, use the MOUNT PARALLEL command on a loosely-coupled cluster configuration. This command is available only for the parallel server version of Oracle. After mounting the database, OPEN it.

Table 2–7 Comparison of Altering the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
The ALTER DATABASE command in MS SQL Server or Sybase can only alter the size of the database. Two megabytes of space is added to the default device of the database, if the size is not specified. The minimum increase you can specify is one megabyte in size. The segments are automatically extended to the additional space. It is important to back up the MASTER database after each use of the ALTER DATABASE command. This ensures that the recovery is easier and safer if the MASTER database is damaged.	RESETLOGS resets the redo logs, cleaning out all the redo log entries when you OPEN the database. Use NORESETLOGS to leave the logs intact when you OPEN the database. You can ENABLE or DISABLE a thread. PUBLIC makes the thread available to any instance not requesting a specific thread. Use DATAFILE to make the datafile ONLINE or OFFLINE. You can CREATE a new datafile to replace a lost or damaged datafile. RENAME GLOBAL_NAME changes the name of the database. Specify the domain to tell Oracle where the database is located on the network. Use RECOVER to recover the database. This command performs media recovery for a lost database.
Permissions: The DBO or SA can use this command. The SA can grant privileges on this command to other users. The DBO or SA must be using the MASTER database to execute this command.	**Permissions:** The user issuing this command needs the ALTER DATABASE privilege.

Example:
 	
To add 3MB to the log device:
 	
`ALTER DATABASE my_database`
`MODIFY FILE`
`(NAME = logdevice_name`
`SIZE = 3MB)`
 	
The logdevice_name is the name of the log device specified while creating the database with LOG ON option.
 	
To allocate an additional device to the database:
 	
ALTER DATABASE database_name ON additional_device_name
	Example:
	
	`ALTER DATABASE`
	`ADD LOGFILE GROUP 10`
	`('log1c', 'log2c') SIZE 3M`
	
	`ALTER DATABASE DROP LOGFILE MEMBER`
	`'LOG3C'`
	
	`ALTER DATABASE DROP LOGFILE GROUP 3`
	
	`ALTER DATABASE`
	`RENAME FILE 'log1a', 'log2a'`
	`TO 'log1c', 'log2c'`

Recommendations:

Oracle functionality exceeds that of MS SQL Server and Sybase. There should be no conversion implications.

Drop

Table 2–8 Comparison of Dropping the Database Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `DROP DATABASE database_name[, database_name ...]`	**Syntax:** Oracle does not have a separate command to drop a database. **Caution:** If you use the CREATE DATABASE command on an existing database, you will destroy the database.
Description: DROP DATABASE deletes the database and all the objects in it from the server, frees the storage space that had been allocated for it, and deletes the related information from the system tables in the MASTER database. This command will not work if the database is in use. DROP DATABASE does not drop the server. It only drops individual databases. It does not remove devices or clean up operating system files.	N / A
Permissions: The DBO alone is allowed to execute this command. The DBO must be using the MASTER database to execute this command.	N / A
Example: `DROP DATABASE my_temp_dbs`	N / A

Recommendations:

Oracle does not have a command to drop a database because there will only be one database per instance. The process for removing an Oracle database is the same as the process for removing an MS SQL Server or Sybase server. The CREATE DATABASE command destroys an existing database if it has the same name as the database being created.

If a database is considered equivalent to a tablespace in Oracle, the DROP TABLESPACE command in Oracle is equivalent to the DROP DATABASE command in MS SQL Server or Sybase.

Database Link

This section contains the following tables for the schema object Database Link:

- Create
- Drop

Create

Table 2–9 Comparison of Creating the Database Link Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
SP_ADDLINKED SERVER	`CREATE [PUBLIC] DATABASE LINK link CONNECT TO user IDENTIFIED BY password USING 'connect_string'`
Description:	**Description:**
Allows queries against databases accessible via OLE DB data sources.	A database link is a named object that describes a path from one database to another. These objects are used in distributed database environment. PUBLIC links are available to all users except those who have created a private link with the same name. `connect_string` is the name and the location of the remote database that can be accessed through SQL*Net. Remote tables can be accessed just like the local tables, except that the table name must be suffixed by @link. The DBA can set the maximum number of simultaneous links that can be created.

Table 2–9 Comparison of Creating the Database Link Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Permissions:** Any user with the CREATE DATABASE LINK system privilege can create private database links. Any user with the CREATE PUBLIC DATABASE LINK system privilege can create public database links. Also, users must the have CREATE SESSION system privilege on a remote database.

N / A	**Examples:**
	`CREATE DATABASE LINK sales.hq.acme.com`
	`CONNECT TO scott IDENTIFIED BY tiger`
	`USING 'D:BOSTON-MFG'`

Recommendations:

An MS SQL Server or Sybase server can support one or more databases, and all these databases can be accessed from one another by fully qualifying the object names. In many applications there is a layer that translates the object names to the actual object names with complete reference (along with the server_name, database_name, owner_name). Database links should be created for all the different servers in the Oracle application environment so that the layer mentioned would simply return the object name for Oracle installations.

The MS SQL Server or Sybase system catalogs hold information about the servers known to the local server. These tables can be read and corresponding database links can be created.

In Oracle, database links are used in distributed database environments. A two-phase commit operation is frequently needed in distributed database environments. MS SQL Server and Sybase only have a programmatic two-phase commit, which is very complex and impractical to use. Because of Oracle's straightforward transparent two-phase commit, distributed database applications are more practical in an Oracle environment.

Oracle allows links to other heterogeneous databases via Oracle Gateway technology.

Drop

Table 2–10 Comparison of Dropping the Database Link Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase do not have database links.	`DROP [PUBLIC] DATABASE LINK link`
N/A	**Description:**
	This command drops the specified database link from the database.
N/A	**Permissions:**
	You can only drop a database link in your own schema. You must have the DROP PUBLIC DATABASE LINK system privilege to drop a PUBLIC database link.
N/A	**Examples:**
	`DROP DATABASE LINK sales.hq.acme.com`

Recommendations:

This command has no effect on the conversion process. Table 2-10 is provided for reference only.

Data and Hash Cluster

This section contains the following tables for the Data and Hash Cluster schema object:

- Create
- Alter
- Drop

Create

Table 2–11 Comparison of Creating the Data and Hash Cluster Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: Table clusters and hash clusters are not supported in MS SQL Server or Sybase.	**Syntax:** CLUSTER [user.]cluster (column data type [, column data type]...) [INITRANS integer] [MAXTRANS integer] [PCTFREE integer] [PCTUSED integer] [SIZE integer[K|M]] [STORAGE storage] [TABLESPACE tablespace] [INDEX|[HASH IS column] HASHKEYS integer]

Table 2–11 Comparison of Creating the Data and Hash Cluster Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description:** Clusters are an optional method of storing tabledata in which one or more tables are physically stored together because they share common columns and are often used together. Clusters require at least one cluster column for each of the tables. These must have the same data type and size, but are not required to have the same name. For the tables in a cluster, rows with the same cluster column values are kept together on disk in the same area, the same logical blocks. Clusters can improve performance when the tables are joined on the cluster columns. Disk access time improves because the related rows are physically stored together. The related columns of the tables in a cluster make up the indexed cluster key. This command creates a cluster of two or more tables. Tables are added to the cluster using CREATE TABLE with the cluster clause. This command commits pending changes to the database. Each distinct value in each cluster column is stored only once, regardless of whether it occurs once or many times in the tables and rows. Tables with LONG columns cannot be clustered. SIZE sets the size in bytes for a logical storage block and should be the average amount of space needed to store all the rows from all the clustered tables that are associated with a single cluster key. If SIZE exceeds the physical block size, Oracle uses the physical block size instead. The cluster is indexed by default. You must create an index on the cluster key before putting any data in the cluster. If you specify the hash cluster, Oracle uses a hash function to store and locate the rows of the tables. HASH IS lets you create your own hash value as a column of the table. Otherwise, Oracle uses an internal hash function based on the columns of the cluster key. HASHKEYS creates the hash cluster and specifies the number of hash values, rounded to the nearest prime number. The minimum value is 2.

Table 2–11 Comparison of Creating the Data and Hash Cluster Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N/A	**Permissions:** You must have the CREATE CLUSTER system privilege to create a cluster in your own schema. You must have the CREATE ANY CLUSTER system privilege to create a cluster in another user's schema.
N/A	**Examples:** `CREATE CLUSTER personnel ( department_number NUMBER) SIZE 512 HASHKEYS 500 STORAGE (INITIAL 100k NEXT 50k PCTINCREASE 10)`

Recommendations:

Clusters improve the performance of certain queries, but they can negatively affect the performance of the INSERT and UPDATE operations and other queries.

Use clusters to store the relatively static tables that need to be joined frequently by using a specific key.

Table clusters and hash clusters should be examined as a possible performance improvement, but are not necessary in a conversion from MS SQL Server or Sybase to Oracle.

Use hashing to reduce I/O when locating rows with an equality condition.

Alter

Table 2–12 Comparison of Altering the Data and Hash Cluster Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
Table clusters and hash clusters are not supported in MS SQL Server or Sybase.
	Syntax:
	`ALTER CLUSTER [user.]cluster`
	`{INITRANS integer|`
	`MAXTRANS integer|`
	`PCTFREE integer|`
	`PCTUSED integer|`
	`SIZE integer[K|M] |`
	`STORAGE storage|`
	`ALLOCATE EXTENT[(SIZE integer[K|M]) | (DATAFILE 'filename') | (INSTANCE integer)]}`

N/A	**Description:** The ALTER CLUSTER redefines future storage allocations or allocates an extent for the specified cluster. SIZE determines how many cluster keys are stored in data blocks allocated to the cluster. You can only change the SIZE parameter for an indexed cluster, not for a hash cluster. ALLOCATE EXTENT explicitly allocates a new extent for the cluster. The user can only allocate a new extent for an indexed cluster, not a hash cluster.
N/A	**Permissions:** User can alter their own clusters. Any user with the ALTER ANY CLUSTER system privilege can alter another user's cluster.

N/A	**Examples:**
	`ALTER CLUSTER scott.customer`
	`SIZE 512`
	`STORAGE (MAXEXTENTS 25)`

Recommendations:

Table clusters and hash clusters should be examined as a possible performance improvement, but are not necessary in a conversion from MS SQL Server or Sybase to Oracle.

Drop

Table 2–13 Table 2-13 Comparison of Dropping the Data and Hash Cluster Schema Objects in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
Table clusters and hash clusters are not supported in MS SQL Server or Sybase.	`DROP CLUSTER [user.]cluster [INCLUDING TABLES [CASCADE CONSTRAINTS]]`
N / A	**Description:**
	This command removes the specified cluster from the database. INCLUDING TABLES drops all tables that belong to the cluster. If the user omits this clause and the cluster still contains tables, Oracle returns an error and does not drop the cluster.
	CASCADE CONSTRAINTS drops all referential integrity constraints from tables outside the cluster that refer to primary and unique keys in the tables of the cluster. If the user omits this option and such referential integrity constraints exist, Oracle returns an error and does not drop the cluster.
N / A	**Permissions:**
	Users can drop their own clusters. Any user with the DROP ANY CLUSTER system privilege can drop another user's cluster.
N / A	**Examples:**
	`DROP CLUSTER geography INCLUDING TABLES CASCADE CONSTRAINTS`

Recommendations:

Table clusters and hash clusters should be examined as a possible performance improvement, but are not necessary in a conversion from MS SQL Server or Sybase to Oracle.

Defaults

This section contains the following tables for the schema object Defaults:

- Create
- Drop

Create

Table 2–14 Comparison of Creating the Default Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`CREATE DEFAULT [owner.]default_name AS value|constant_expression`
`sp_bindefault default_name, {"table.column" | data type_name}`
	Syntax:
	Defaults are specified as part of the CREATE TABLE or ALTER TABLE statement.

Description: Defaults in MS SQL Server and Sybase are created as separate objects. These defaults are bound to individual columns of the table. The expression used as a default value can be a constant literal or a built-in function that returns a constant value. For example: `user_name(), getdate())`	**Description:** You can specify the default value for a column with the DEFAULT constraint in the table creation / alteration statement. The expression used in the DEFAULT constraint can be a constant literal or a built-in function that returns a constant value. For example, SYSDATE.
Permissions: The DBO has the CREATE DEFAULT permission and can transfer it to other users.	N / A

Example:
`CREATE DEFAULT user_def AS user_id()`
`sp_bindefault user_def , "table1.uid_column"`
	N / A

Recommendations:

The implementation of defaults in MS SQL Server, Sybase, and Oracle is conceptually very similar.

Defaults in MS SQL Server and Sybase can use built-in functions. These functions have to be parsed and replaced by equivalent Oracle functions. If the equivalent function is not available, you may want to make the column as NULL allowed and update it with the default value from within a trigger.

Drop

Table 2–15 Comparison of Dropping the Default Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `DROP DEFAULT [owner.]default_name [, [owner.]default_name ...]`	**Syntax:** Defaults are specified as part of the CREATE TABLE or ALTER TABLE statement.
Description: A default cannot be dropped if it is currently bound to a column or a user-defined data type. Use SP_UNBINDEFAULT to unbind the default.	N / A
Permissions: Only the default owner can issue this command.	N / A
Example: `DROP DEFAULT user_def`	N / A

Recommendations:

Replace DROP DEFAULT statements with ALTER TABLE statements.

Index

This section contains the following tables for the schema object Index:

- Create
- Alter

- Drop

Create

Table 2–16 Comparison of Creating the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX index_name ON [[database.]owner.]table (col_name [, col_name]...) [WITH {FILLFACTOR = x, IGNORE_DUP_KEY, [IGNORE_DUP_ROW|ALLOW_DUP_ROW]}] ON segment_name`	**Syntax:** `CREATE INDEX [user.] index_name ON {[user.]table (col_name[ASC | DESC] [,col_name[ASC | DESC]...) | CLUSTER [user.]cluster} [INITRANS integer] [MAXTRANS integer] [PCTFREE integer] [STORAGE storage] [TABLESPACE tablespace] [NOSORT]`

Table 2–16 Comparison of Creating the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description: Index names in MS SQL Server and Sybase must be unique for the table. Two tables in MS SQL Server and Sybase can have indexes of the same name. IGNORE_DUP_KEY causes the server to ignore a row that would violate a unique key. It does not give any error message. IGNORE_DUP_ ROW causes the server to ignore a row that is a duplicate of an existing row. It does not give any error message. ALLOW_DUP_ROW allows duplicate rows in the table. The FILLFACTOR decides how full each page becomes while creating a new index on existing data. The server must split the pages when they fill up; the FILLFACTOR percentage thus affects performance. FILLFACTOR can be changed globally for all indexes by setting it with SP_CONFIGURE. The FILLFACTOR value should be smaller when the corresponding table is dynamic and is capable of growing. ON segment_name names the segment to which the index is assigned.	**Description:** Each user schema must have a unique index name for each Oracle database. No two indexes created by the same user can have the same name. Oracle automatically creates unique indexes to enforce unique column constraints. PCTFREE is the percentage of space left free in the index for new entries and the updates. TABLESPACE names the tablespace to which the index is assigned. NOSORT reduces the time to create an index if, and only if, the values in the column being indexed are already in ascending order. Oracle does not balance indexes on its own. This calls for the maintenance of indexes on tables which have very high numbers of INSERTs and DELETEs. The indexes on the dynamic tables need to be dropped and created again from time to time, to ensure the same average response time. A query which can be satisfied from just the index does not need the actual table rows. This occurs when the query selects only columns included in the index key. **Cluster:** CLUSTER is the cluster key indexed for a cluster. Clusters must have their keys indexed for their associated tables to be accessed. **Non-clustered Indexes:** All Oracle indexes are non-clustered indexes.

Table 2–16 Comparison of Creating the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description (continued): A query which can be satisfied from just the index does not need the actual table rows. This occurs when the query selects only columns included in the index key. **Clustered Indexes:** In MS SQL Server and Sybase, a clustered index is a b-tree index. The leaf pages contain the data. The index forces the data to be stored in physically sorted order, that is, sequentially. A table can have only one clustered index. A clustered index in MS SQL Server or Sybase is always a UNIQUE index. In most applications, a clustered index is created on the primary key for a table. **Non-clustered Indexes:** In MS SQL Server and Sybase, a non-clustered index is a b-tree index. The leaf pages contain pointers to the data. The data is stored in a random order. A table can have multiple non-clustered indexes. These indexes can be UNIQUE if specified. A table can have up to 249 non-clustered indexes.	
Permissions: The CREATE INDEX permission defaults to the table owner and cannot be transferred.	**Permissions:** Users can index their own tables. Any valid database user with the INDEX privilege on the table or the CREATE ANY INDEX system privilege can also create an index.
Example: `CREATE UNIQUE NONCLUSTERED INDEX c_temp_ix ON table customer (col1, col2) WITH FILLFACTOR = 20 IGNORE_DUP_KEY ON segment index_seg`	**Example:** `CREATE INDEX c_temp_ix ON customer(col1 DESC,col2) NOSORT`

Recommendations:

Index names in MS SQL Server and Sybase are only required to be unique for each table. In Oracle they must be unique for each user, regardless of the table the index is on. Change the non-unique index names when moving them to Oracle.

Clustered indexes should be replaced by primary keys in Oracle.

UNIQUE non-clustered indexes translate to UNIQUE column constraints.

Oracle never ignores rows being inserted or updated. It either performs the INSERT or UPDATE or gives an error. If MS SQL Server or Sybase indexes were created with IGNORE_DUP_KEY or IGNORE_DUP_ROW, a note should be made that the application needs to change to handle the error.

ALLOW_DUP_ROW functionality is supported in Oracle provided no other constraints are violated.

Alter

Table 2–17 Comparison of Altering the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase have no command comparable to ALTER INDEX.	`ALTER INDEX [user.]index_name {INITRANS integer | MAXTRANS integer | STORAGE storage}`
N/A	**Description:**
	The ALTER INDEX command is used to change future storage allocation for data blocks in an index.
	INITRANS and MAXTRANS change the values of these parameters for the index. See the Tables section for a description of these parameters.
	STORAGE changes the storage parameters for the index.
N/A	**Permissions:**
	The index owner can alter the index. Other users must have the ALTER ANY INDEX system privilege to alter an index.

Table 2–17 Comparison of Altering the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example:	**Example:**

N/A	`ALTER INDEX scott.ix_cust`
	`INITRANS 5`
	`STORAGE (NEXT 100K)`

Recommendations:

This command has no effect on the conversion process. The information is provided for reference only.

Drop

Table 2–18 Comparison of Dropping the Index Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
`DROP INDEX [table.] index_name` `[, [table.] index_name ...]`	`DROP INDEX [user.]index_name`
Description:	**Description:**
The DROP INDEX command drops the specified index.	The DROP INDEX command drops the specified index. It commits pending changes to the database.
Permissions:	**Permissions:**
By default, the index owner has the DROP INDEX permission which is not transferable.	This command can be issued by the owner of the index. A user must have the DROP ANY INDEX system privilege to drop an index from another user's schema.
Example:	**Example:**
`DROP INDEX test_tabl.test_index`	`DROP INDEX test_index`

Recommendations:

If applications drop multiple indexes with one DROP INDEX command, they need to be converted into multiple DROP INDEX commands in Oracle.

Privilege

This section contains the following tables for the schema object Privilege:

- Grant
- Revoke

Grant

Table 2–19 Comparison of Granting the Privilege Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
System Privileges: `GRANT {system_privilege [, system_privilege]...| ALL} TO {user [,user]...| group[,group]... | PUBLIC}`	System Privileges: `GRANT {system_privilege [, system_privilege]...| role[,role]...} TO {user [,user]...| role[,role]...|PUBLIC} [WITH ADMIN OPTION]`

Object Privileges :
`GRANT {object_privilege[, object_privilege]...| ALL}`
 	
`ON object [(col_list)] TO {user [,user]...| group[,group]... | PUBLIC}`
	Object Privileges:
	`GRANT {object_privilege[, object_privilege]...| ALL[PRIVILEGES]} [(column[,column]...)]`
	
	`ON [user.]object TO {user [,user]...| role[,role]...|PUBLIC} [WITH GRANT OPTION]`

Table 2–19 Comparison of Granting the Privilege Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
System privileges are the privileges granted to create and manage various schema objects. These commands also include the system administrative commands. Object privileges are the privileges granted on the various operations on the schema objects. The column list is applicable only when the object corresponds to a table or a view. SELECT and UPDATE privileges can be column-specific. MS SQL Server and Sybase allow anti-grants. Anti-grants are revoke statements that do not have a corresponding, preceding, explicit grant statement. For example, you can issue the REOVKE SELECT ON SYSLOGINS.PASSWORD FROM PUBLIC command to disallow access to the PASSWORD column in the SYSLOGINS table. This statement need not be preceded by the GRANT SELECT ON SYSLOGINS.PASSWORD TO PUBLIC command. This is frequently used to revoke part of a privilege which was granted previously. It allows system administrators to quickly do things such as "Give everyone EXCEPT Bob access to table X".	System privileges are the privileges granted to create and manage various schema objects themselves. These commands also include the system administrative commands. WITH ADMIN OPTION lets the granted user or role (the grantee) grant the system privilege or role to other grantees. The grantee can also use its option to alter or drop a granted role. Object privileges are the privileges granted on the various operations on the schema objects. The column list is applicable only when the object corresponds to a table or a view. The INSERT and UPDATE privileges can be column-specific. Object privileges include any or all of the following privileges: SELECT, INSERT, DELETE, UPDATE, EXECUTE, ALTER, REFERENCE, and INDEX. WITH GRANT OPTION passes along the right to grant the granted object privileges to another user or role. GRANTs on synonyms become grants on the underlying object that the synonym references. GRANT can be specified to a list of columns when granting the INSERT, REFERENCE, or UPDATE privileges. Oracle only supports additive privileges. This means that the REVOKE statement should have a corresponding GRANT statement. There is no way to REVOKE part of a privilege. You must only GRANT the specific privileges desired.

Table 2–19 Comparison of Granting the Privilege Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Permissions:	**Permissions:**
The SA grants system privileges and object owners grant object privileges.	To grant a system privilege, the grantor must either have been granted the system privilege with the ADMIN OPTION or have been granted the GRANT ANY PRIVILEGE system privilege. To grant object privileges, the object must be in the grantor's own schema or the grantor must have been granted the object privileges with the GRANT option.
Examples:	**Examples:**

`GRANT ALL ON items TO PUBLIC`
`GRANT SELECT ON items(column1) TO mary`
`GRANT EXECUTE ON st_proc1 TO mary`
`GRANT CREATE TABLE, CREATE PROCEDURE TO jim`
	`GRANT ALL ON items TO PUBLIC`
	`GRANT SELECT ON items TO mary`
	`GRANT EXECUTE ON st_proc1 TO mary`
	`GRANT CREATE TABLE, CREATE PROCEDURE TO jim`
	`GRANT SELECT, UPDATE ON items TO jim`
	`GRANT CREATE SESSION, CREATE VIEW, CREATE SYNONYM TO basic_role WITH ADMIN OPTION;`
	`GRANT basic_role TO mary,jim,tom`

Recommendations:

In MS SQL Server and Sybase, both grants and revokes are recorded in the system catalogues. When a user attempts an operation against the object, MS SQL Server and Sybase check to see if the user was granted authorization either directly (e.g., "GRANT SELECT ON X TO mary", to public, or via group.

They also check to see if the user was explicitly revoked access from the object at the user, public, or group level. If a permission is revoked, the REVOKE overrides all GRANTs issued. For example, the following statements are executed in MS SQL Server or Sybase:

```
GRANT SELECT ON X TO public;
REVOKE SELECT ON X FROM bob;
```

When user "bob" attempts to select on object "X" in the database, MS SQL Server or Sybase sees if bob, public, or the group of which Bob is currently a member, has select on the object (they do) and it is not true that bob, public, or the group of which Bob is currently a member has been revoked select (Bob has). Bob cannot access object X but everyone else can.

Oracle does not allow this kind of granting and revoking because it results in an unmanageable tangle of grants and revokes. Oracle enforces the idea that if a privilege is granted to public, then everyone has the privilege, without exceptions. If it is not accessible to everyone, then it should not be granted to public.

While converting the privileges from MS SQL Server or Sybase to Oracle, all the anti-grants should be resolved before creating the DDL for Oracle. All the privileges should be additive privileges.

Revoke

Table 2–20 Table 2-20 Comparison of Revoking the Privilege Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**

System Privileges:
`REVOKE {ALL | system_privilege}`
`FROM {user [,user]...|`
`group[,group]...|PUBLIC}`
Object privileges :
`REVOKE`
`{object_privilege[, object_`
`privilege]...| ALL}`
`ON object [(col_list)]`
`FROM {user [,user]...|`
`group[,group]... | PUBLIC}`
	System Privileges:
	`REVOKE {system_privilege`
	`[, system_privilege]...`
	`| role[,role]...}`
	`FROM {user [,user]...|`
	`role[,role]...|PUBLIC}`
	Object Privileges:
	`REVOKE`
	`object_privilege[,object_`
	`privilege]...`
	`ON [user.]object`
	`FROM {user [,user]...|`
	`role[,role]...|PUBLIC}`
	`[CASCADE CONSTRAINTS]`

Table 2–20 Table 2-20 Comparison of Revoking the Privilege Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description: MS SQL Server and Sybase allow anti-grants. Anti-grants are REVOKE statements that do not have a corresponding, preceding, explicit grant statement. For example, one can issue the REVOKE SELECT ON SYSLOGINS.PASSWORD FROM PUBLIC command to revoke access to the PASSWORD column of the SYSLOGINS table. This statement need not be preceded with the GRANT SELECT ON SYSLOGINS.PASSWORD TO PUBLIC command. This is frequently used to revoke part of a privilege which was granted previously. It allows system administrators to quickly do things such as "Give everyone EXCEPT Bob access to table X".	**Description:** The system privileges REVOKE command takes privileges and roles away from users or privileges away from roles. Any system privilege may be revoked. The REVOKE command takes object privileges on a specific object away from a user or role. Oracle supports additive privileges only. This means that the REVOKE statement must have a corresponding GRANT statement. There is no way to revoke part of a privilege. You must only grant the specific privileges desired. The CASCADE CONSTRAINTS statement drops any referential integrity constraints defined by the user or by users granted the role. This applies to a REFERENCES privilege.
Permissions: The SA can revoke the system privileges and the object owners can revoke the object privileges.	**Permissions:** A user can issue a REVOKE statement provided the user has the corresponding GRANT permissions.

Examples:
`REVOKE SELECT ON table2 FROM PUBLIC`
 	
`REVOKE SELECT ON table3(password_col) FROM PUBLIC`
 	
`REVOKE CREATE TABLE, CREATE VIEW FROM jim`
	Examples:
	`REVOKE basic_role FROM mary,jim,tom`
	
	`REVOKE SELECT ON items FROM PUBLIC`

Recommendations:

While converting the privileges from MS SQL Server or Sybase to Oracle, all the anti-grants should be resolved before creating the DDL for Oracle.

All the privileges should be additive privileges.

Profile

This section contains the following tables for the schema object Profile:

- Create
- Alter
- Drop

Create

Table 2–21 Comparison of Creating the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: MS SQL Server and Sybase do not have profiles. MS SQL Server and Sybase use database resources to set the global values for various resources such as the number of connections to the server. The maximum number of connections allowed to the server can be specified in the application environment.	**Syntax:** `CREATE PROFILE profile LIMIT {SESSIONS_PER_USER | CPU_PER_SESSION | CPU_PER_CALL | CONNECT_TIME | IDLE_TIME | LOGICAL_READS_PER_SESSION | LOGICAL_READS_PER_CALL | COMPOSITE_LIMIT | PRIVATE_SGA} {integer [K|M] |UNLIMITED |DEFAULT}`

Table 2–21 Comparison of Creating the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description:** The CREATE PROFILE command creates a set of limits on the use of the database resources. When the profile is associated with the user, you can control what the user does by those limits. SESSIONS_PER_USER limits the user to a specified number of concurrent SQL sessions. CPU_PER_SESSION limits the CPU time for a parse, execute, or fetch call in hundredths of seconds. CONNECT_TIME limits THE elapsed time of a session in minutes. IDLE_TIME disconnects a user after this number of minutes; this does not apply when a query is running. LOGICAL_READS_PER_SESSION limits the number of blocks read per session to the specified number. LOGICAL_READS_PER_CALL limits the number of blocks read for parse, execute, or fetch calls. PRIVATE_SGA limits the amount of space the user can allocate in the SGA as private. COMPOSITE_LIMIT limits the total resource cost for a session, in service units, based on a weighted sum of CPU, connect time, logical reads, and private SGA resources. UNLIMITED means there is no limit on a particular resource. DEFAULT picks up the limit from DEFAULT profile, which can be changed with the ALTER PROFILE command.
N / A	**Permissions:** You must have the CREATE PROFILE system privilege to create a profile.

Table 2–21 Comparison of Creating the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

N / A	**Example:**
	`CREATE PROFILE clerk LIMIT`
	`SESSIONS_PER_USER 2`
	`CPU_PER_SESSION unlimited`
	`CPU_PER_CALL 6000`
	`LOGICAL_READS_PER_SESSION unlimited`
	`LOGICAL_READS_PER_CALL 100`
	`IDLE_TIME 30`
	`CONNECT_TIME 480`

Recommendations:

Although profiles are not required in converting from MS SQL Server or Sybase to Oracle, they should be investigated and used wherever possible to aid the DBA in controlling system use.

Alter

Table 2–22 Comparison of Altering the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
MS SQL Server and Sybase do not have profiles.
	Syntax:
	`ALTER PROFILE profile LIMIT`
	`{SESSIONS_PER_USER |`
	`CPU_PER_SESSION |`
	`CPU_PER_CALL |`
	`CONNECT_TIME |`
	`IDLE_TIME |`
	`LOGICAL_READS_PER_SESSION |`
	`LOGICAL_READS_PER_CALL |`
	`COMPOSITE_LIMIT integer |`
	`UNLIMITED |`
	`DEFAULT |`
	`PRIVATE_SGA} {integer`
	`[K|M]|UNLIMITED| DEFAULT}`

Table 2–22 Comparison of Altering the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description:** The ALTER PROFILE command allows the user to modify a particular profile setting. See CREATE PROFILE for option information.
N / A	**Permissions:** You must have the ALTER PROFILE system privilege to alter a profile.

N / A	**Example:**
	`ALTER PROFILE clerk LIMIT`
	`CPU_PER_CALL default`
	`LOGICAL_READS_PER_SESSION 20000`

Recommendations:

Although profiles are not required in converting from MS SQL Server and Sybase to Oracle, they should be investigated and used wherever possible to aid the DBA in controlling system use.

Drop

Table 2–23 Comparison of Dropping the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: MS SQL Server and Sybase do not have profiles.	**Syntax:** `DROP PROFILE profile [CASCADE]`
N / A	**Description:** The DROP PROFILE command drops thespecified profile from the database. CASCADE de-assigns the profile from any users to whom it is assigned and automatically assigns the DEFAULT profile instead. You must specify this option to drop a profile that is currently assigned to users.

Table 2–23 Comparison of Dropping the Profile Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N/A	**Permissions:** You must have the DROP PROFILE system privilege to remove a profile from a database.
N/A	**Example:** `DROP PROFILE engineer CASCADE`

Recommendations:

Although profiles are not required in converting from MS SQL Server and Sybase to Oracle, they should be investigated and used wherever possible to aid the DBA in controlling system use.

Role

This section contains the following tables for the schema object Role:

- Create
- Alter
- Drop

Create

Table 2–24 Comparison of Creating the Role Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`sp_addgroup group_name`
`sp_addrole`
	Syntax:
	`CREATE ROLE role`
	`[NOT IDENTIFIED|`
	`IDENTIFIED [BY password|EXTERNALLY]]`

Table 2–24 Comparison of Creating the Role Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
The concept of roles exists in MS SQL Server and Sybase. Roles are similar to groups in previous releases. A group is a schema object declared as a group of users. A group called PUBLIC is defined in the MODEL database. Each new database is created with a group called PUBLIC. Every user automatically becomes a member of the group PUBLIC.	The CREATE ROLE command creates a named role or set of privileges. When the role is granted to the user, all the privileges of that role are granted to the user. Oracle automatically creates several roles. For example, the CONNECT, RESOURCE, and DBA roles provide compatibility with prior Oracle versions. The EXP_FULL_DATABASE and IMP_FULL_ DATABASE roles allow the user to use the import and export utilities.
Create new groups with SP_ ADDGROUP and assign a user to a new group with SP_CHANGEGROUP system procedures.	Create the role using the CREATE ROLE statement and then grant privileges to the role using GRANT statements. Use roles to group and easily assign privileges to many users.
A user can be a member of only one group other than PUBLIC at any given time. An individual must register with the database as two different users such as MGR or CLK in order to work in different groups. In this case, one user ID is a member of the MANAGER group, and another is a member of the CLERK group.	Roles can be granted to other roles to define a hierarchy of permissions. A user may be assigned to multiple roles and can switch between roles. The SET ROLE command can be used to enable a particular role. Once enabled, all the privileges associated with the role are enabled. This is useful to simulate real-life roles, such as manager, clerk, etc.
MS SQL Server and Sybase do not differentiate between the privileges granted to the group and to an individual user. This allows the users to create a view or stored procedure that refers to an object granted to them via their group privilege.	When the user wants to access something that the role allows, enable the role using the SET ROLE command. Oracle gives a different treatment to the privileges granted to a role as opposed to directly granted to a user.
Permissions:	**Permissions:**
The DBO can execute this command.	You must have the CREATE ROLE system privilege to create a role.
Examples:	**Examples:**
`sp_addgroup teller`	`CREATE ROLE teller` `IDENTIFIED BY cashflow`

Recommendations:

Oracle roles and MS SQL Server or Sybase groups/roles can all be granted privileges. This concept is similar in MS SQL Server, Sybase, and Oracle because it is used mainly to give a set of privileges to a set of users.

In MS SQL Server and Sybase, you make all the users members of one group and grant a set of privileges to the group. In Oracle, you create a role with a set of privileges and then grant this role to a number of users.

To replicate the functionality of groups, create a role for each MS SQL Server or Sybase group and the privileges granted to each group are granted to the corresponding role. The roles are then assigned to each user. Each Oracle user would be assigned to the following two roles only:

CONNECT and the user's corresponding MS SQL Server or Sybase group.

If the MS SQL Server or Sybase application uses groups to grant the privileges and the privileges are required to create views and stored procedures, you must grant the privileges in the Oracle application to individual users as well as to roles. This is necessary because Oracle does not allow the user to build objects that refer to the objects which the user was given access to through roles.

Since many roles can be assigned to a person, it may be advisable to investigate the richer functionality of Oracle roles to see if more logical groupings of privileges are possible. Furthermore, since roles may be assigned other roles, you may find it more convenient to create a hierarchy of roles suitable for each application.

Alter

Table 2–25 Comparison of Altering the Role Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `sp_changegroup new_group_name user_name_inside_db`	**Syntax:** `ALTER ROLE role {NOT IDENTIFIED | IDENTIFIED [BY password | EXTERNALLY]} GRANT role[,role]... TO {user [,user]...| role[,role]...|PUBLIC} REVOKE role[,role]... FROM {user [,user]| role[,role]|PUBLIC}` The SET ROLE command can be used to switch between roles.
Description: This command changes the named user from the current group to the new group.	**Description:** The ALTER ROLE command lets the user modify the password for the roles already created with CREATE ROLE. The GRANT/REVOKE PRIVILEGE statements are used to change assignments of roles.
Permissions: The DBO can change the group of other database users. The users cannot do this themselves.	**Permissions:** You must be granted either ROLE with ADMIN OPTIONS or have the ALTER ANY ROLE system privilege to alter a role.

Examples:
`sp_changegroup new_grp user1`
`sp_changegroup "public" user1`
This removes the user user1 from the existing group without assigning user1 to any other group.
	Examples:
	`ALTER ROLE teller IDENTIFIED BY letter`

Recommendations:

Oracle roles and MS SQL Server or Sybase groups can both be granted privileges. Each MS SQL Server or Sybase user can only belong to one group, but each Oracle user can have many roles. This concept is similar in MS SQL Server, Sybase, and

Oracle databases because it is used mainly to give a set of privileges to a set of users. In MS SQL Server and Sybase, you make all the users members of one group and give a set of privileges to the group. In Oracle, you create a role with a set of privileges and then grant this role to a group of users.

To replicate the functionality of groups, create a role for each MS SQL Server or Sybase group and the privileges granted to each group are granted to the corresponding role. The roles are then assigned to each user. Each Oracle user would be assigned the following two roles only: CONNECT and the user's corresponding MS SQL Server or Sybase group.

In Oracle, the SET ROLE command can be used to switch between roles.

If the MS SQL Server or Sybase application uses groups to grant the privileges and the privileges are required to create views and stored procedures, you must grant the privileges in the Oracle application to individual users as well as to roles.

Drop

Table 2–26 Comparison of Dropping the Role Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`sp_dropgroup group_name`
`sp_droprole role_name`
	Syntax:
	`DROP ROLE [user.]role`

Description: The `sp_dropgroup` command drops the specified group. The group cannot be dropped if it has users attached to it as members.	**Description:** The DROP ROLE command drops the specified role. It commits pending changes to the database.
Permissions: The DBO can change the group of other database users. The users cannot do this themselves.	**Permissions:** The user must have been granted the role with the ADMIN option or have the DROP ANY ROLE system privilege to use this command.
Example: `sp_dropgroup accountant`	**Example:** DROP ROLE accountant

Recommendations:

Oracle functionality directly matches or exceeds that of MS SQL Server and Sybase. There should be no conversion implications.

Rule

This section contains the following tables for the schema object Rule:

- Create
- Drop

Create

Table 2–27 Comparison of Creating the Rule Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`CREATE RULE [owner.]rule_name AS condition_expression`
 	
`sp_bindrule rule_name, {"table.column"|data type_name} [,future_only]`
	Syntax:
	Oracle allows check constraints in tables for simple business rules. Triggers may be used for more complex rules.

Description: CREATE RULE creates data integrity constraints in the database. SP_BINDRULE binds the rule to the columns. Rules in MS SQL Server and Sybase are a special kind of stored procedure. Rules can define complex data integrity constraints. A rule is bound to the database column. A rule can be bound to a particular column of a table or it can be bound to a user-defined data type. Domains are implemented using rules. The MS SQL Server or Sybase rule can refer to one or more columns in the table because they are implemented in a manner similar to the stored procedures.	N / A
Permissions: The DBO has the CREATE RULE permission by default, and can transfer it to the other users.	N / A

Table 2–27 Comparison of Creating the Rule Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Examples:
`CREATE RULE col1_rule`
`AS @col1_var > 100`
	N/A

Recommendations:

Oracle allows check constraints in tables for simple business rules. Triggers may be used for more complex rules.

The LIKE clause in MS SQL Server and Sybase rules can accept wildcard characters and ranges of values while the Oracle LIKE clause accepts only wildcard characters. If the rule has ranges of values (it uses regular expressions), it can be translated using a combination of SUBSTR and TRANSLATE in an Oracle check constraint.

MS SQL Server and Sybase integrity constraints can be implemented as check constraints in Oracle. See the Tables section for information about check in table constraints.

The column reference in the MS SQL Server or Sybase rule definition is not a column name. A rule is bound to the database column. The @var_name should be parsed out and replaced by this column name if you convert the RULEs to check constraint.

Drop

Table 2–28 Comparison of Dropping the Rule Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`sp_unbindrule rule_name,`
`{"table.column"|datatype_name}`
`[,future_only]`
`DROP RULE [owner.]rule_name`
`[,[owner.]rule_name ...]`
	Syntax:
	Oracle can DROP/DISABLE a check constraint by using the ALTER TABLE command.

Table 2–28 Comparison of Dropping the Rule Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	N / A
SP_UNBINDRULE detaches the rule from the columns and must be used before DROP RULE can be implemented.	
DROP RULE drops a rule from the database.	
Permissions:	N / A
The rule owner has the DROP RULE permission by default. This privilege is not transferable.	
Examples:	N / A
DROP RULE col1_rule	

Recommendations:

Sequences are very useful and are very efficient. You should replace the equivalent code in MS SQL Server or Sybase that generates unique IDs with references to sequences.

Sequence

This sections contains the following tables for the schema object Sequence:

- Create
- Alter
- Drop

Create

Table 2–29 Comparison of Creating the Sequence Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**

Sequences are not implemented in MS SQL Server or Sybase.	`CREATE SEQUENCE [user.]sequence`
	`[INCREMENT BY integer]`
	`[START WITH integer]`
	`[MAXVALUE integer | NOMAXVALUE]`
	`[MINVALUE integer | NOMINVALUE]`
	`[CYCLE | NOCYCLE]`
	`[CACHE integer | NOCACHE]`
	`[ORDER | NOORDER]`

Description:	**Description:**
The user has to build the logic to create sequence numbers whenever necessary. This is not a trivial task in multi-user applications. The common way to implement this is with a table of sequence names and last values. Whenever a number is needed, the application must lock a row in the table, SELECT the current value, UPDATE it, and unlock the row. The problem with this approach is that it causes applications to wait while another user is getting the next value. This single-threads the application and limits the performance to the speed of this operation.	A sequence generates a serial list of unique numbers. Sequence numbers are independent of tables, so the same sequence can be used for one or more tables. Oracle implements sequences internally so that blocking does not occur. The default value for INCREMENT BY is 1. A positive value causes ascending increments, a negative value causes decrements. START WITH is the number to start the sequence. START WITH defaults to MAXVALUE for descending sequence and MINVALUE for ascending sequences. CYCLE restarts a sequence when MINVALUE or MAXVALUE is reached. CACHE allows a pre-allocated set of sequence numbers to be kept in the memory. ORDER guarantees that the sequence numbers are assigned to the instances requesting them in the order the requests are received.
N / A	**Permissions:** To create a sequence in your own schema, you must have the CREATE SEQUENCE privilege. To create a sequence in another user's schema, you must have the CREATE ANY SEQUENCE system privilege.

Table 2–29 Comparison of Creating the Sequence Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Example:** `CREATE SEQUENCE eseq INCREMENT BY 10`

Alter

Table 2–30 Comparison of Altering the Sequence Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
Sequences are not implemented in MS SQL Server or Sybase.
	Syntax:
	`ALTER SEQUENCE [user.]sequence`
	`{INCREMENT BY integer |`
	`{MAXVALUE integer | NOMAXVALUE} |`
	`{MINVALUE integer | NOMINVALUE} |`
	`{CYCLE | NOCYCLE} |`
	
	`{CACHE integer | NOCACHE} |`
	`{ORDER | NOORDER} |`

N / A	**Description:** Use ALTER SEQUENCE to alter the sequence definition.
N / A	**Permissions:** If the sequence is in your own schema, you can alter the sequence. Otherwise you must have the ALTER privilege on the sequence or the ALTER ANY SEQUENCE system privilege.
N / A	**Examples:** `ALTER SEQUENCE eseq CYCLE CACHE 5`

Recommendations:

ALTER SEQUENCE does not allow you to set the next value a sequence will generate. To set the next value of a sequence, either change the increment, select the next value, and change the increment back, or drop and recreate the sequence.

Drop

Table 2–31 Comparison of Dropping the Sequence Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
Sequences are not implemented in MS SQL Server or Sybase.	`DROP SEQUENCE [user.]sequence`
N/A	**Description:**
	This command drops the specified sequence from the database.
N/A	**Permissions:**
	If the sequence is in your own schema you can drop the sequence. Otherwise, you must have the DROP ANY SEQUENCE system privilege.
N/A	**Example:**
	`DROP SEQUENCE elly.eseq`

Recommendations:

This command does not affect database conversion. The information is provided for reference only.

Snapshot

This section contains the following table for the schema object Snapshot:

- Create

Create

Table 2–32 Comparison of Creating the Snapshot Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase do not support snapshots.	Refer to *Oracle8i Replication Management API Reference, Release 2 (8.1.6)* (Part Number: A76958-01) for information about DBMS_REPCAT.CREATE_SNAPSHOT.

Table 2–32 Comparison of Creating the Snapshot Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Replication in MS SQL Server and Sybase is not as mature as replication in Oracle.	**Description:** Snapshots provide an automatic method for table replication. There are two types of snapshots: read-only and updateable. A snapshot log is a table associated with the master table of a snapshot that tracks changes to the master table. You can have only one log per master table. A simple snapshot selects data from a single master table using a simple query. A complex snapshot selects data using a GROUP BY, CONNECT BY, subquery, join, or set operation in the query.
N/A	**Permissions:** User with the CREATE SNAPSHOT privilege can create snapshots in their own shema. A user with the CREATE ANY SNAPSHOT privilege can create snapshots in any user's schema.
N/A	**Examples:**

Recommendations:

While converting a distributed database application from MS SQL Server or Sybase to Oracle, you should look for situations requiring constant availability of remote information and handle them using table snapshots. Refer to *Oracle8i Replication Management API Reference, Release 2 (8.1.6)* (Part Number: A76958-01) for more information about the DBMS_REPCAT package.

Synonym

This section contains the following tables for the schema object Synonym:

- Create
- Drop

Create

Table 2–33 Comparison of Creating the Synonym Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
MS SQL Server and Sybase do not support synonyms.
Views that do SELECT * FROM are good candidates for conversion to synonyms.
	Syntax:
	`CREATE [PUBLIC] SYNONYM`
	`[user.]synonym`
	`FOR [user.]object [@database_link]`

N / A	**Description:** A synonym is an alias for object names. A synonym is not an object itself, but a direct reference to an object. Synonyms are used to mask the real name and owner of an object, provide public access to an object, provide location transparency for the objects of a remote database, and simplify the SQL statements for database users. PUBLIC makes the synonym available to all users. Without PUBLIC, other users must prefix the synonym with the owner name. Synonyms can be created on tables, views, stored procedures, and other synonyms. @database_link links to a remote database. The synonym refers to an object in the remote database as specified by the database link.
N / A	**Permissions:** You must have the CREATE SYNONYM privilege to create a private synonym in your own schema. You must have the CREATE ANY SYNONYM privilege to create a private synonym in another user's schema. You must have the CREATE PUBLIC SYNONYM privilege to create a PUBLIC synonym.

Table 2–33 Comparison of Creating the Synonym Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

N / A	**Examples:**
	`CREATE PUBLIC SYNONYM our_custs FOR customer`
	`CREATE SYNONYM custs FOR customer`

Recommendations:

This command does not affect database conversion. The information is provided for reference only.

Oracle uses synonyms to build location transparency for objects in distributed database applications.

Drop

Table 2–34 Comparison of Dropping the Synonym Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase do not support synonyms.	DROP [PUBLIC] SYNONYM [schema.]synonym
N / A	**Description**: This command removes the specified synonym from the database.
N / A	**Permissions:** To drop a private synonym, either the synonym must be in your own schema or you must have the DROP ANY SYNONYM system privilege. To drop a public synonym, either the synonym must be in your own schema or you must have the DROP ANY PUBLIC SYNONYM system privilege.
N / A	**Examples:** `DROP SYNONYM our_custs`

Recommendations:

This command does not affect database conversion. The information is provided for reference only.

Tables

This section contains the following tables for the schema object Tables:

- Create
- Alter
- Drop
- Truncate

Create

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: CREATE TABLE [[database.]user.]table (column data type [NOT NULL|NULL] [,column data type [NOT NULL|NULL]]...) [ON segment_name]	**Syntax:** CREATE TABLE [user.]table [table_constraint] ({column data type [DEFAULT clause] [column_constraint] | table_constraint} [,{column data type [DEFAULT clause] [column_constraint] | table_constraint}]...) [CLUSTER cluster(column [,column]...) | [[INITRANS integer] [MAXTRANS integer] [PCTFREE integer] [PCTUSED integer] [STORAGE storage] [TABLESPACE tablespace]] [ENABLE enable | DISABLE disable] [AS query]

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
MS SQL Server 6.5 and Sybase are case-sensitive for object names by default.	Oracle is case insensitive for object names.
	Oracle is case insensitive for object names.
If the user is not specified, user defaults to the user issuing this command.	If the user is not specified, it defaults to the user issuing this command.
MS SQL Server 6.5 and Sybase default to NOT NULL for columns.	Oracle defaults to NULL for columns.
(Table Description is continued after the Column Constraint and Table Constraint sections below.)	DEFAULT specifies a value to be assigned to the column if a row is inserted without a value for this column. The value can be a simple literal or the result of an expression. The expression, however, cannot include a reference to a column, to LEVEL, or to ROWNUM.
	The AS clause creates the rows of the new table through the returned query rows. The columns and types of the query must match those defined in the CREATE TABLE statement.
	(Table Description is continued after the Column Constraint and Table Constraint sections below.)
Column Constraints	**Column Constraints**
Syntax:	**Syntax:**
See the Rule section.	`[CONSTRAINT constraint]` `{[NOT] NULL |` `{UNIQUE	PRIMARY KEY}|` `REFERENCES [user.]table [(column)]` `[ON DELETE CASCADE] |` `CHECK (condition)}`  `[[USING INDEX[PCTFREE integer	INITRANS integer	MAXTRANS integer	TABLESPACE tablespace	STORAGE storage]]` `[EXCEPTIONS INTO [user.] table] |` `DISABLE]`

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Column Constraints	**Column Constraints**
Description:	**Description**:
See the Rule section.	CONSTRAINT is an optional name assigned to this constraint. NULL permits NULL values and is the default. The NOT NULL clause specifies that every row must have a value for this column. UNIQUE forces column values to be unique. You can use only one primary key on a table. An index enforces the unique or primary key, and the USING INDEX clause and its options specify the storage characteristics of that index. REFERENCES identifies this column as a foreign key from [user.]table [(column)]. If you omit the column, this implies that the name in the user.table defaults to the primary key in this table. ON DELETE CASCADE maintains referential integrity automatically by removing foreign key rows in the dependent tables if you remove the primary key row in this table. CHECK assures that the values for this column pass a condition such as the following: `Amt number(12,2) CHECK (Amt >= 0)` This condition may be any valid expression that tests TRUE or FALSE. It can contain functions, any columns from this table, and literals. EXCEPTIONS INTO specifies a table into which Oracle puts rows that violate an enabled integrity constraint. This table must be local and must exist before you use this option. DISABLE disables the integrity constraint when it is created. You can enable it later with the ENABLE clause in CREATE or ALTER TABLE.

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Table Constraints	**Table Constraints**
Syntax:	**Syntax:**
See the Rule section.	[CONSTRAINT constraint] {[NOT] NULL | {UNIQUE|PRIMARY KEY} (column[,column]...) | FOREIGN KEY (column[,column]...) REFERENCES [user.]table [(column[,column]...) [ON DELETE CASCADE] | CHECK (condition)} [[USING INDEX[PCTFREE integer | INITRANS integer | MAXTRANS integer | TABLESPACE tablespace | STORAGE storage]] [EXCEPTIONS INTO [user.] table] | DISABLE]
Table Constraints	**Table Constraints**
Description:	**Description:**
See the Rule section.	Table constraints are similar to column constraints, except that you can reference multiple columns with a single constraint. For example, a table constraint can declare three columns as PRIMARY KEY.

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Table Description (continued):	**Table Description (continued):**

The ON segment_name clause places the table on a specific segment. This logical device must already be assigned to the database with either CREATE DATABASE or ALTER DATABASE constructs. The table is created on the default segment if the ON segment_name clause is not specified.	CLUSTER includes this table in the named cluster.
	
	INITRANS specifies the initial number of transactions that can update a data block concurrently. The default value is 1. Every transaction takes space in the data block itself until the transaction is completed.
	
	MAXTRANS sets the maximum number of transactions that can update a data block concurrently.
	
	Use TABLESPACE to specify the name of the tablespace on which this table is created.
	
	Whenever Oracle inserts a row into a table, it first checks how much space is available in the current data block. If the size of the row leaves less than PCTFREE percent in the block, it puts the row in a newly allocated block instead. The default value for PCTFREE is 10.
	
	PCTUSED defaults to 40. This is the minimum percentage of available space in a block that will allow the insertion of new rows in this data block. The other space available is used for updating the existing rows.
	
	The STORAGE clause has various sub-clauses as follows:
	
	`STORAGE ( [INITIAL integer]`
	`[NEXT integer]`
	`[PCTINCREASE integer]`
	`[MINEXTENTS integer]`
	`[MAXEXTENTS integer])`
	
	INITIAL allocates the first extent of space to the object.
	
	NEXT is the size of the extent allocated after the initial extent has been filled.
	
	PCTINCREASE controls the rate of growth of extents beyond the second. If this is set to 0, every additional extent will be of the same size as the second extent, specified by NEXT.

Table 2–35 Comparison of Creating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
	Table Description (continued): MINEXTENTS defaults to 1, meaning that when the object is created, only one initial extent is allocated. A number larger than 1 creates that many total extents, and all of these are allocated to the object when the object is created. MAXEXTENTS sets the total number of extents that can be allocated. The default value is 99. The ENABLE and DISABLE clauses enable and disable constraints.
Permissions: The DBO has the CREATE TABLE permission by default and can transfer it to other users.	**Permissions:** You must have the CREATE TABLE privilege to create a table in your own schema. You must have the CREATE ANY TABLE privilege to create a table in another user's schema.

Examples:
`CREATE TABLE test_tbl`
`(col1 int NOT NULL,`
`col2 char(10) NULL)`
 	
`CREATE TABLE table3`
`(col1 my_type NOT NULL,`
`col2 varchar(100) NULL)`
	Examples:
	CREATE TABLE test_tb1
	(col1 NUMBER(6) NOT NULL PRIMARY KEY,
	col2 VARCHAR2(10) DEFAULT "ab")
	TABLESPACE tabs_1
	STORAGE (
	INITIAL 400K
	NEXT 400K
	MAXEXTENTS 5
	PCTINCREASE 0)

Recommendations:

The conceptual definition of tables is the same in MS SQL Server, Sybase, and Oracle.

Reserved Words

Be aware that table names and column names in MS SQL Server and Sybase can be reserved words in Oracle.

Defaults
Convert MS SQL Server and Sybase defaults to be created as part of the table creation in Oracle.

NULL/Not NULL
CREATE TABLE should always specify the NULL / NOT NULL constraint as the default is the opposite in MS SQL Server, Sybase, and Oracle.

Row-Migration
Row migration occurs when a row is updated and increases in size until it no longer fits in the block at which time it must be moved to another block. Migration may also occur when rows grow in size and PCTFREE is not set correctly.

Row-Chaining
Row-chaining can occur when one row exceeds the size of one data block and has to be stored as a chain of data blocks. This affects all types of operations on this table. Oracle users can set the block size for their database to avoid such situations.

If MS SQL Server or Sybase tables have large record sizes, it may be necessary to increase the block size in Oracle to avoid row chaining problems.

Unique Keys
Define unique keys for columns defined in MS SQL Server and Sybase as unique non-clustered index.

Storage
The storage specifications for index must be added manually.

Data Types
MS SQL Server and Sybase data types should be translated to equivalent Oracle data types. See the Data Types section of this chapter for a table of equivalent data types.

Alter

Table 2–36 Comparison of Altering the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
 	
`ALTER TABLE`
`[[database.]owner.]table`
`ADD column data type NULL`
`[[, column data type NULL]...]`
	Syntax:
	
	`ALTER TABLE [user.]table`
	`{ADD ({column data type [DEFAULT clause]`
	`[column_constraint] |`
	`table_constraint}`
	`[,{column data type [DEFAULT clause]`
	`[column_constraint] |`
	`table_constraint}]...) |`
	`MODIFY ({column data type [DEFAULT clause]`
	`[column_constraint] |`
	`table_constraint}`
	`[,{column data type [DEFAULT clause]`
	`[column_constraint] |`
	`table_constraint}]...) |`
	`DROP`
	`{PRIMARY KEY|`
	`UNIQUE(column[,column]...) |`
	`CONSTRAINT constraint_name}`
	`[CASCADE] |`
	
	`PCTFREE integer |`
	`PCTUSED integer |`
	`INITRANS integer |`
	`MAXTRANS integer |`
	`STORAGE storage |`
	`ALLOCATE EXTENT`
	`[({SIZE integer[K|M] | DATAFILE file | INSTANCE integer})] }`
	`[ENABLE enable|DISABLE disable]`

Table 2–36 Comparison of Altering the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
This command can be used to add columns at the end of the table. Columns of type BIT cannot be added to an existing table.	ADD allows the user to add a column with a column constraint and a default value to the end of an existing table, or to add a table constraint to the table's definition. Table constraints and column constraints follow the same format in CREATE TABLE. MODIFY changes an existing column. The changes include: n A change in column type or decrease in the size provided every row contains NULL value for this column. n A NOT NULL column may be added to a table with no rows. n An existing column can be modified to NOT NULL if it has a non-NULL value in every row. n Increasing the length of a NOT NULL column without specifying NULL leaves it as NOT NULL. n Views that reference a table with SELECT FROM will not work after the column is added to the table unless they are dropped and re-created. n An existing column can be modified to have a DEFAULT or a column constraint. DROP allows the user to drop a column constraint or a table constraint. ALLOCATE EXTENT lets the user explicitly allocate a new extent. ENABLE and DISABLE, enable and disable constraints.

Table 2–36 Comparison of Altering the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Permissions:	**Permissions:**
The table owner and DBO can use the ALTER TABLE command. The DBO must impersonate the table owner with the SETUSER command before using the ALTER TABLE command.	The table owner can alter the table. A user must have the ALTER privilege on the table or the ALTER ANY TABLE system privilege to alter a table.
Example:	**Examples:**

`ALTER TABLE students`
`ADD extra_curr_act varchar(255)`
`NULL`
	`ALTER TABLE students`
	`ADD (extra_curr_act varchar(255))`
	
	`PCTFREE 30`
	`PCTUSED 60`
	
	`ALTER TABLE items ADD(CONSTRAINT c1`
	`CHECK (unit_price < total_price))`

Recommendations:

Oracle functionality exceeds that of MS SQL Server and Sybase. There should be no conversion implications.

Drop

Table 2–37 Comparison of Dropping the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**

`DROP TABLE`
`[[database.]owner.]table`
`[, [[database.]owner.]table`
`...]`
	`DROP TABLE [user.]table`
	`[CASCADE CONSTRAINTS]`

Table 2–37 Comparison of Dropping the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
The DROP TABLE command removes the table definition and all of its data, indexes, triggers, and permissions from the database.	The DROP TABLE command drops the table and commits pending changes to the database. All indexes, triggers, and grants associated with the table are dropped. Objects depending on the dropped table are marked invalid and cease to work. CASCADE CONSTRAINTS drops all referential integrity constraints referring to keys in the dropped table.
Permissions:	**Permissions:**
The table owner and DBO can use the DROP TABLE command. The DBO must impersonate the table owner with the SETUSER command before using the DROP TABLE command.	Only the table owner can drop a table. A user must have the DROP ANY TABLE system privilege to drop a table.
Example:	**Examples:**
`DROP TABLE test_tab1`	`DROP TABLE test_tab1 CASCADE CONSTRAINTS`

Recommendations:

Convert MS SQL Server and Sybase applications that drop multiple tables with one DROP TABLE command into multiple DROP TABLE commands in Oracle.

Truncate

Table 2–38 Comparison of Truncating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
`TRUNCATE TABLE [[database.]owner.]table`	`TRUNCATE TABLE [user.]table [DROP | REUSE STORAGE]`

Table 2–38 Comparison of Truncating the Table Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
TRUNCATE TABLE removes all rows from the table. This statement is not responded to by a DELETE TRIGGER. This statement is not logged and therefore cannot activate the trigger.	TRUNCATE removes all rows from the table. DROP STORAGE deallocates space from the deleted rows. REUSE STORAGE leaves the space allocated for the new rows in the table. The TRUNCATE command is faster than a DELETE command as it does not generate the rollback information, does not fire any delete triggers, and does not record any information in the snapshot log. In addition, TRUNCATE does not invalidate the objects depending on the deleted rows or the privileges on the table. You cannot roll back the TRUNCATE statement.
Permissions:	**Permissions:**
Only the table owner can issue this command.	The table owner can truncate the table. A user must have the DELETE ANY TABLE system privilege to truncate a table.
Example:	**Example:**
`TRUNCATE TABLE test_tab1`	`TRUNCATE TABLE test_tab1`

Recommendations:

Oracle functionality directly matches or exceeds that of MS SQL Server and Sybase. There should be no conversion implications.

Tablespace

This section contains the following tables for the schema object Tablespace:

- Create
- Alter
- Drop

Create

Table 2–39 Comparison of Creating the Tablespace Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
MS SQL Server and Sybase do not have tablespaces. See the Data Storage Concepts section of this chapter for more information.
	Syntax:
	`CREATE TABLESPACE tablespace`
	`DATAFILE file_definition [, file_definition]...`
	`[DEFAULT STORAGE storage]`
	`[ONLINE | OFFLINE]`

N / A	**Description:** DEFAULT STORAGE defines the default storage for all objects created in this tablespace. The ONLINE clause, which is the default, indicates that the tablespace becomes available as soon as it is created. The OFFLINE clause prevents access to the tablespace until the ALTER TABLESPACE clause changes it to ONLINE.
N / A	**Permissions:** Only the DBA can create the tablespaces. You must have the CREATE TABLESPACE system privilege to create a tablespace.

N / A	**Example:**
	`CREATE TABLESPACE rb_segs`
	`DATAFILE 'datafile_1' SIZE 50M`
	`DEFAULT STORAGE (INITIAL 50K NEXT 50K MINEXTENTS 2MAXEXTENTS 50 PCTINCREASE 0)`
	`OFFLINE`

Recommendations:

Tablespaces have some features in common with MS SQL Server and Sybase "databases" and some features in common with MS SQL Server and Sybase "segments".

As part of the conversion process, the structure of the Oracle database must be determined. It should be based on how databases and devices are used in MS SQL Server and Sybase.

Tablespaces and Databases
Oracle tablespaces and MS SQL Server and Sybase databases are similar in the following respects:

- Provide backup features
- Can be brought off-line independently of other parts of the server
- Provide a method for grouping related tables together

Tablespaces and Segments
Oracle tablespaces and MS SQL Server and Sybase segments both provide the control over the physical location of the database objects.

However, Oracle provides features that MS SQL Server and Sybase do not, such as default storage information and usage quotas.

Alter

Table 2–40 Comparison of Altering the Tablespace Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: MS SQL Server and Sybase do not have tablespaces. See the Data Storage Concepts section of this chapter for more information.	**Syntax:** `ALTER TABLESPACE tablespace` `{ADD DATAFILE file_definition [,file_definition] |` `RENAME DATAFILE file [,file]... TO file [,file] |` `DEFAULT STORAGE storage |` `ONLINE |` `OFFLINE [NORMAL	TEMPORARY|IMMEDIATE]|` `{BEGIN	END} BACKUP}`

Table 2–40 Comparison of Altering the Tablespace Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description:** ADD DATAFILE adds a file or series of files to the tablespace. The database file names and sizes of these files are specified in file_definition. RENAME DATAFILE ...TO changes the name of an existing tablespace file. The tablespace should be offline while the renaming takes place RENAME DATAFILE ...TO does not actually rename the files, but only associates their new names with this tablespace. DEFAULT STORAGE defines the default storage for all future objects created in this tablespace. ONLINE brings the tablespace on-line, and the OFFLINE clause takes the tablespace off-line, either without waiting for its users to logoff (IMMEDIATE), or after the users have logged off (NORMAL). BEGIN BACKUP assures that all database files in this tablespace are backed up the next time you do a system backup. You can execute the BEGIN BACKUP clause at any time. The END BACKUP clause indicates that the system backup is finished. If the tablespace is on-line, any system backup must also back up archive redo logs. If the tablespace is off-line, system backup of archive redo logs is unnecessary.
N / A	**Permissions:** You must have the ALTER TABLESPACE system privilege to perform any of this command's operations. If you have to MANAGE TABLESPACE system privilege, you can only take the tablespace online or offline and begin or end a backup.

N / A	**Example:**
	`ALTER TABLESPACE rb_segs`
	`ADD DATAFILE 'extra_file' SIZE 1M`

Drop

Table 2–41 Comparison of Dropping the Tablespace Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase do not have tablespaces. See the Data Storage Concepts section of this chapter for more information.	`DROP TABLESPACE tablespace [INCLUDING CONTENTS]`
N / A	**Description:**
	This command drops the specified tablespace.
	INCLUDING CONTENTS allows you to drop a tablespace that contains data. Without this option, you can only drop empty tablespaces. Tablespace should be offline in order to execute this command successfully
N / A	**Permissions:**
	You must have the DROP TABLESPACE system privilege to drop a tablespace.
N / A	**Example:**
	`DROP TABLESPACE test_tbspc`

User

This section contains the following tables for the schema object User:

- Create
- Alter
- Drop

Create

Table 2–42 Comparison of Creating the User Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
This is a two-step process. First the user is given access to the server, and then to the specific database within that server.
1. User is given access to server:
`sp_addlogin "login_id"`
`[, password ]`
`[, default_database ]`
`[, default_language ]`
2. User is given access to the database. The user is considered to be a member of the PUBLIC group.
`sp_adduser login_id`
`[,usr_nm_inside_db ]`
`[,group_name]`
	Syntax:
	`CREATE USER user`
	`IDENTIFIED {BY password | EXTERNALLY}`
	`[DEFAULT TABLESPACE tablespace]`
	`[TEMPORARY TABLESPACE tablespace]`
	`[QUOTA {integer [K | M] | UNLIMITED}`
	`ON tablespace]`
	`[PROFILE profile]`

Table 2–42 Comparison of Creating the User Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
These commands give the user access to the server and then to the specific database within that server. 1. The default password is NULL and the default database is the MASTER database. The Password is not encrypted and is stored in the SYSLOGINS table. It is accessible to the SA. 2. The user is registered as the user of a database with the SP_ADDUSER procedure. The user remains a member of the PUBLIC group even if made a member of another group.	This command creates a user account that lets the user log into the database with a certain set of privileges and storage settings. When first created, the user has no privileges. You must GRANT roles and privileges to the user. The privilege to CREATE SESSION is the minimal privilege required for any user in order to log on. EXTERNALLY specifies that access to the database is verified through the operating system security. DEFAULT TABLESPACE is the tablespace in which the user creates objects. TEMPORARY TABLESPACE stores temporary objects created for the user's operations. You can put the QUOTA clause on either the default or temporary tablespaces to limit the amount of space, in bytes, that a user can allocate. PROFILE assigns a named profile to the user to limit usage of database resources. Oracle assigns the DEFAULT profile to the user if the profile is not specified while creating the user.
Permissions:	**Permissions:**
1. The SA can add the server login. 2. The DBO can add a user to the database.	You must have the CREATE USER system privilege to create a user.
Example:	**Example:**
1. `sp_addlogin "whales", whales_pass, pubs` 2. `sp_adduser whales name_in_db group`	`CREATE USER whales IDENTIFIED BY whales_pass`

Recommendations:

MS SQL Server, Sybase, and Oracle treat the individual users in a very similar way, and the conversion is straightforward.

The user SA in MS SQL Server and Sybase is roughly equivalent to the user SYSTEM in Oracle. The user who is the DBO in MS SQL Server or Sybase can be converted to have DBA privileges in Oracle.

Alter

Table 2–43 Comparison of Altering the User Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
MS SQL Server and Sybase do not support ALTER USER. However, the user can make the following comparable changes:
The password can be changed using the SP_PASSWORD system procedure after the user account is set up.
The default database can be changed using the SP_DEFAULTDB system procedure.
The default language can be changed using the DP_DEFAULTLANGUAGE system procedure.
	Syntax:
	`ALTER USER user`
	`{IDENTIFIED {BY password |`
	`EXTERNALLY}|`
	`DEFAULT TABLESPACE tablespace|`
	`TEMPORARY TABLESPACE tablespace|`
	`QUOTA {integer [K | M] | UNLIMITED}`
	`ON tablespace|`
	`PROFILE profile|`
	`DEFAULT ROLE {role[,role]... | ALL`
	`[EXCEPT role[,role]] | NONE}}`

N/A	**Description:** This command can be used to change a user's password or the DEFAULT or TEMPORARY tablespace. ALTER USER can also change the quota, the resource profile, or the default role.
N/A	**Permissions:** You must have to ALTER USER system privilege to alter the characteristics of a database user. However, you can change your own password without this privilege.

N/A	**Examples:**
	`ALTER USER scott IDENTIFIED BY lion`
	`DEFAULT TABLESPACE tstest`

Recommendations:

There should be no conversion implications.

Drop

Table 2–44 Comparison of Dropping the User Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**

Drop the user account from the server:
`sp_droplogin login_id`
Drop the user from the database:
`sp_dropuser usr_nm_inside_db`
	`DROP USER user [CASCADE]`

Description:	**Description:**
These two commands are used to drop the user from the server and the database. The DBO cannot be dropped using the SP_DROPUSER command.	This command drops the specified user. It commits pending changes to the database. CASCADE drops all the objects in the user's schema before dropping the user, and you must specify CASCADE if the user has any objects in the schema.
Permissions:	**Permissions:**
The SA can drop the user account from the server. The DBO can drop the user from the database.	The user must have the DROP USER system privilege to drop a user.
Example:	**Example:**

Drop the user from the database:
`sp_dropuser bradley`
Drop the user account from the server:
`sp_droplogin bradley`
	`DROP USER bradley CASCADE`

Recommendations:

Oracle functionality is similar to that of MS SQL Server and Sybase. There should be no conversion implications.

View

This section contains the following tables for the schema object View:

- Create
- Alter
- Drop

Create

Table 2–45 Comparison of Creating the View Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `CREATE VIEW [owner.]view_name [(column_name [, column_name] ...)] AS select_statement`	**Syntax:** `CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW [user.]view_name [(alias[,alias]...) ] AS select_statement [WITH CHECK OPTION [CONSTRAINT constraint]]`

Table 2–45 Comparison of Creating the View Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
INSERTs and UPDATEs are allowed on a view provided only one of the base tables is undergoing change. View cannot be created on temporary table. You cannot create a trigger or build an index on a view. You cannot update a view with the following clauses in the `select_statement`: ORDER BY, COMPUTE, DISTINCT, and SELECT INTO. The select_statement can refer to one or more tables and other views. If a view definition statement includes aggregate functions (GROUP BY), and the query on the view does not include the aggregate columns, the GROUP BY is ignored. If a view is defined with an outer join, and query includes a condition on a column from the inner table of the outer join, all rows from the inner table are returned. Rows that do not meet the condition, display NULL values in the respective columns of that row. This result is not the same as it would be when two tables are joined with an outer join.	You can use VIEW to UPDATE and DELETE rows if the view is based on a single table and its query does not contain the GROUP BY clause, the DISTINCT clause, group functions, or references to the pseudo-column ROWNUM. You can update views containing other pseudo-columns or expressions, provided they are not referenced in the statement UPDATE. INSERT rows using VIEW if the view is based on a single table and if its query does not contain the GROUP BY clause, the DISTINCT clause, group functions, or references to any pseudo-columns, or any expressions. If the underlying table is altered to add more columns after the creation of the view, views that reference the table withSELECT*FROM must be dropped and recreated to see the new columns. You cannot create a trigger or build an index on a view. OR REPLACE recreates a view if it already exists. The FORCE option creates the view regardless of whether the tables to which the view refers exist or whether the user has privileges on them. The user still cannot execute the view but can create it. The NOFORCE option creates the view only if the base tables exist and the user has privileges on them.

Table 2–45 Comparison of Creating the View Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
	Description: (continued) If `alias` is specified, the view uses the alias as the name of the corresponding column in that query. If an alias is not specified, the view inherits the column name from the query. In the latter case, each column in a query must have a unique name. AS select_statement identifies the columns of the tables and other views that are to appear in this view. The WHERE clause in the select_ statement determines which rows are to be retrieved. The WITH CHECK OPTION clause restricts inserts and updates performed through the view. This prevents them from creating rows that the view cannot itself select, based on the WHERE clause of the CREATE VIEW statement. This option may be used in a view that is based on another view. However, if the underlying view also has a WITH CHECK OPTION clause, it is ignored. CONSTRAINT is an optional name given to CHECK OPTION.
Permissions: The DBO has the CREATE VIEW permission by default and can transfer it to other users.	**Permissions:** You must have the CREATE VIEW system privilege to create a view in your own schema. You must have the CREATE ANY VIEW system privilege to create a view in another user's schema. FORCE / NOFORCE clauses dictate whether a user needs privileges on the base tables.

Examples:
 	
`CREATE VIEW sales_staff AS`
`select empno, ename, deptno`
`from emp`
`where deptno = 10`
	Example:
	
	`CREATE OR REPLACE VIEW sales_staff AS`
	`select empno, ename, deptno from emp`
	`where deptno = 10`
	`WITH CHECK OPTION`
	`CONSTRAINT sales_staff_cnst`

Recommendations:

View text may include some MS SQL Server and Sybase-specific SQL constructs which must be converted manually. Also, the MS SQL Server and Sybase views that use GROUP BY and aggregates need one or more view equivalents to get the same results in Oracle.

Also see the INSERT, SELECT, UPDATE, and DELETE statements in the Data Manipulation Language section of this chapter for more information in this regard.

Alter

Table 2–46 Comparison of Altering the View Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
MS SQL Server and Sybase do not have a command comparable to ALTER VIEW.	ALTER VIEW [user.]view COMPILE
N / A	**Description:** This command is used to explicitly recompile a view that is invalid. Explicit recompilation allows you to locate recompilation errors prior to runtime. You may want to explicitly recompile a view after altering one of its base tables to ensure that the alteration does not affect the view or other objects that depend on it. Note that this command does not change the definition of an existing view. To redefine a view you must use the CREATE VIEW command with the OR REPLACE option.
N / A	**Permissions:** You can issue this command on your own view. You must have the ALTER ANY TABLE system privilege to issue this command on the view of another user's schema.
N / A	**Examples:** `ALTER VIEW user1.test_view COMPILE`

Recommendations:

ALTER VIEW has no effect on database conversion. This information is provided for reference only.

Drop

Table 2–47 Comparison of Dropping the View Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `DROP VIEW [owner.]view_name [[,owner.]view_name ...]`	**Syntax:** `DROP VIEW [user.]view_name`
Description: This command drops the specified view from the database.	**Description:** This command drops the specified view and commits pending changes to the database. Views and synonyms built on dropped view are marked invalid and cease to work.
Permissions: Only the view owner can issue this command.	**Permissions:** The owner of the view can drop it. You must have the DROP ANY VIEW system privilege to drop a view from the database.
Examples: `DROP VIEW test_view`	**Examples:** `DROP VIEW test_view`

Recommendations:

MS SQL Server and Sybase applications that drop multiple views with one DROP VIEW command must have those commands converted to multiple

DROP VIEW commands in Oracle.

Data Manipulation Language

This section uses tables to compare the syntax and description of Data Manipulation Language (DML) elements in the MS SQL Server, Sybase, 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

Connecting to the Database

The statement illustrated in the following table connects a user to a database.

Table 2–50 Connecting to the Database in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`USE database_name`
	Syntax:
	`CONNECT user_name/password`
	`SET role`

Table 2–50 Connecting to the Database in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	
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.	

Recommendations:

This concept of connecting to a database is conceptually different in the MS SQL Server, Sybase, and Oracle databases. An MS SQL Server or Sybase 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 the following table retrieves rows from one or more tables or views.

Table 2–51 SELECT Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: 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]}	**Syntax:** 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]}

Table 2–51 SELECT Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
DISTINCT eliminates the duplicate rows.	DISTINCT eliminates the duplicate rows.
The INTO clause and the items that follow it in the command syntax are optional, because MS SQL Server and Sybase allow SELECT statements without FROM clauses as can be seen in the following example: `SELECT getdate()`	The `INSERT INTO <table> SELECT FROM...` construct allows you to insert the results of the SELECT statement into a table.
SELECT...INTO allows you to insert the results of the SELECT statement into a table.	`COLUMN ALIAS` is defined by putting the alias directly after the selected COLUMN.

SELECT_LIST can contain a SELECT statement in the place of a column specification as follows:
`SELECT d.empno, d.deptname,`
`empname = (SELECT ename FROM emp`
`WHERE enum = d.empno)`
`FROM dept d`
`WHERE deptid = 10`
	If you use TABLE ALIAS, the TABLE must always be referenced using the ALIAS.

The above example also shows the format for the column alias. `ALIAS = selected_column`	You can also retrieve data from SYNONYMS.
COMPUTE attaches computed values at the end of the query. These are called row_ aggregates.	EXPRESSION could be a column name, a literal, a mathematical computation, a function, several functions combined, or one of several PSEUDO-COLUMNS.
Outer joins are implemented as follows: `WHERE tab1.col1 *= tab2.col1;`	Outer joins are implemented as follows: `WHERE tab1.col1 = tab2.col1 (+);`
where all values from `TAB1` will be returned even if `TAB2` does not have a match or `WHERE tab1.col1 =* tab2.col1;`	Where all values from `TAB1` will be returned even if `TAB2` does not have a match or `WHERE tab1.col1 (+) = tab2.col1;`
where all values from `TAB2` will be returned even if `TAB1` does not have a match.	where all values from `TAB2` will be returned even if `TAB1` does not have a match.
If a GROUP BY clause is used, all non-aggregate select columns are needed.	If a GROUP BY clause is used, all non-aggregate select columns must be in a GROUP BY clause.
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.	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 MS SQL Server and Sybase.

SELECT Statements without FROM Clauses:

MS SQL Server and Sybase support 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 one above.

Translate the above query to:

```
SELECT sysdate FROM dual;
```

SELECT INTO Statement:

The MS SQL Server and Sybase 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 MS SQL Server or Sybase 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
```

Subqueries in Place of Columns:

In MS SQL Server and Sybase, a SELECT statement may appear anywhere that a column specification appears. Oracle does not support this non-ANSI extension to ANSI SQL. Change the subquery in the SELECT list either by using a DECODE statement or by dividing the query into two different queries.

Use the following SALES table as a basis for the examples below:

Year	Qty	Amount
1993	1	1.3
1993	2	1.4
1993	3	3
1993	4	2.3

MS SQL Server/Sybase:

If you want to select the year, q1 amount, q2 amount, q3 amount, and q4 as a row, MS SQL Server and Sybase accept the following query:

```
SELECT distinct year,
 q1 = (SELECT amt FROM sales
       WHERE qtr=1 AND year = s.year),
 q2 = (SELECT amt FROM sales
       WHERE qtr=2 AND year = s.year),
 q3 = (SELECT amt FROM sales
       WHERE qtr=3 AND year = s.year),
 q4 = (SELECT amt FROM sales
       WHERE qtr=4 AND year = s.year)
FROM sales s
```

In Oracle:

In this example, replace the SELECT statements with DECODE so that the query functions as normal. The DECODE function is much faster than MS SQL Server and Sybase subqueries. Translate the above query to the following for Oracle:

```
SELECT year,
DECODE( qtr, 1, amt, 0 ) q1,
DECODE( qtr, 2, amt, 0 ) q2,
DECODE( qtr, 3, amt, 0 ) q3,
DECODE( qtr, 4, amt, 0 ) q4
FROM sales s;
```

If you cannot convert your query using the above method, create views and base the query on the views rather than on the original tables.

For example, consider the following query in MS SQL Server and Sybase:

```
SELECT name,
sumlength = (SELECT sum(length) FROM syscolumns WHERE id = t.id),
```

```
count_indexes = (SELECT count(*) FROM sysindexes WHERE id = t.id)
FROM sysobjects t
```

This query returns the sum of the lengths of the columns of a table and the number of indexes on that table. This is best handled in Oracle by using some views.

Convert this to the following in Oracle:

```
CREATE view V1 ( sumlength, oid ) as
SELECT sum(length), id FROM syscolumns
GROUP BY  id

CREATE view V2 ( count_indexes, oid ) AS
SELECT count(*), id FROM sysindexes
GROUP BY  id

SELECT name, sumlength, count_indexes
FROM sysobjects t, v1, v2
WHERE t.id = v1.oid
AND t.id = v2.oid
```

Comparing Subqueries to Subqueries:

MS SQL Server and Sybase also allow a SELECT statement in the WHERE clause. For example, consider the following statement from MS SQL Server or Sybase:

```
SELECT empname, deptname
FROM emp, dept
WHERE emp.empno = 100
  AND(SELECT security_code
       FROM employee_security
       WHERE empno = emp.empno) =
      (SELECT security_code
       FROM security_master
       WHERE sec_level = dept.sec_level)
```

Convert this to the ANSI-standard statement below for Oracle:

```
SELECT empname, deptname
FROM emp, dept
WHERE emp.empno = 100
  AND EXISTS (SELECT security_code
              FROM employee_security es
              WHERE es.empno = emp.empno
                AND es.security_code =
                    (SELECT security_code
                     FROM security_master
```

```
WHERE sec_level =
      dept.sec_level));
```

Column Aliases:

Convert column aliases from the following MS SQL Server or Sybase syntax:

```
SELECT employees=col1 FROM tab1
```

to the following Oracle syntax:

```
SELECT col1 employees FROM tab1
```

Note: MS SQL Server and Sybase also support 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.

Outer JOIN Syntax:

Convert the outer JOIN syntax from the MS SQL Server or Sybase syntax to the Oracle syntax.

In addition to these, there are many implications due to the differences in the implementation of the special clauses such as GROUP BY, functions, joins. These are discussed later in this chapter.

SELECT with GROUP BY Statement

Table 2–52 SELECT with GROUP BY Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Server	Oracle
Syntax:	**Syntax**:
See the SELECT Statement section.	See the SELECT Statement section.

Table 2–52 SELECT with GROUP BY Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Server	Oracle
Description:	**Description**:
Non-aggregate SELECT columns must be in a GROUP BY clause.	All non-aggregate SELECT columns must be in a GROUP BY clause.

INSERT Statement

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

Table 2–53 INSERT Statement in Oracle and MS SQL Server/Sybase

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

Recommendations:

INSERT statements in MS SQL Server and Sybase 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 MS SQL Server-specific functions must be replaced with the equivalent Oracle constructs.

Note: Oracle lets you create functions that directly match most MS SQL Server and Sybase functions.

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

UPDATE Statement

The statement illustrated in the following table updates the data in a table or the data in a table referenced by a view.

Table 2–54

MS SQL Server	Oracle

Syntax:
`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]`
	Syntax:
	`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}]`

Description: 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. Updates through multi-table views can modify only columns in one of the underlying tables.	**Description:** 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. 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. Updates can only be done on single table views.

Recommendations:

There are two ways to convert UPDATE statements with FROM clauses as indicated below.

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 MS SQL Server or Sybase:

```
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 MS SQL Server or Sybase:

```
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 the following table removes rows from tables and rows from tables referenced in views.

Table 2–55 DELETE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`DELETE [FROM]`
`[[database.]owner.]{table | view}`
`[FROM`
`[[database.]owner.]{table | view}`
`[, [[database.]owner.]{table | view}]...]`
`[WHERE where_clause]`
	Syntax:
	`DELETE [FROM] [user.]{table | view}`
	`[@dblink]`
	`[alias]`
	
	`[WHERE where_clause]`

Description: The first FROM in DELETE FROM is optional. The second FROM clause is an MS SQL Server or Sybase 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. Deletes can only be performed through single table views.	**Description**: FROM is optional. ALIAS can be specified for the table name as a correlation name, which can be used in the condition. Deletes can only be performed through single table views

Remove Second FROM Clause:

Remove the second FROM clause from the DELETE statements.

Convert the following MS SQL Server or Sybase 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 MS SQL Server or Sybase 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

Comparison Operators

The following table compares the operators used in the MS SQL Server, Sybase, and Oracle databases. Comparison operators are used in WHERE clauses and COLUMN check constraints / rules to compare values

Table 2–56 Comparison Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase Only	Oracle Only
Equal to	=		

Table 2–56 Comparison Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase Only	Oracle Only
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
Not equal to a member of set	NOT IN != ANY <> ANY		!= SOME <> SOME
Less than a member of set	< ANY		< SOME

Table 2–56 Comparison Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase Only	Oracle Only
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 MS SQL Server or Sybase:

```
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]%"
```

Method 1 - Eliminating use of [ ]:

Use this method with the SUBSTR () function if possible.

```
SELECT title
from titles
where substr (titles,1,1) in
```

```
('A', 'B', 'C', 'D', 'E', 'F')
```

Method 2 - Eliminating use of [ ]:

The second method uses the % construct.

```
SELECT title
FROM titles
WHERE (title like 'A%'
   OR  title like 'B%'
   OR  title like 'C%'
   OR  title like 'D%'
   OR  title like 'E%'
   OR  title like 'F%')
```

3. Change NULL constructs:

The following table shows that in Oracle, NULL is never equal to NULL. Change the all = NULL constructs to IS NULL to retain the same functionality.

Table 2–57 Changing NULL Constructs

NULL Construct	MS SQL Server/Sybase	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 MS SQL Server or Sybase:

```
WHERE col1 = NULL
```

Convert it as follows for Oracle:

```
WHERE col1 IS NULL
```

Arithmetic Operators

Table 2–58 Arithmetic Operators in Oracle and MS SQL Server/Sybase

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

Recommendations:

Replace any Modulo functions in MS SQL Server or Sybase with the mod() function in Oracle.

String Operators

Table 2–59 String Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase 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 MS SQL Server and Sybase, 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 2–60 Set Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase 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 2–61 Bit Operators in Oracle and MS SQL Server/Sybase

Operator	Same in All Three Databases	MS SQL Server/Sybase Only	Oracle Only
bit and		&	
bit or		|	
bit exclusive or		^	
bit not		~	

Recommendations:

Oracle enables you to write your own procedures to perform bitwise operations.

If you have the following MS SQL Server or Sybase construct:

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

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

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

Built-In Functions

Character Functions

Table 2–62 Character Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle	Description
ascii(char)	ascii(char)	Returns the ASCII equivalent of the character.
char(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 *Oracle8i SQL Reference, Release 2 (8.1.6) (Part Number A76989-01) 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.

Table 2–62 Character Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle	Description
patindex(pattern,	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.
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)	rpad(' ', 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	

Table 2–62 Character Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle	Description
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 2–63 Comparison Operators in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	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.

Recommendations:

Note: The above functions tables list all the MS SQL Server and Sybase 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 MS SQL Server and Sybase functions.

Date Functions

Table 2–64 Date Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	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.
`datediff(dd, datetime1,datetime2)`	`date2-date1`	Returns the difference between the dates specified by the datetime1 and datetime2 variables. This difference is calculated in the number of days.
`datediff(mm, datetime1,datetime2)`	`months_between ( date2, date1)`	Returns the difference between the dates specified by the datetime1 and datetime2 variables. This difference is calculated in the number of months.
`datediff(yy, datetime1,datetime2)`	`(date2-date1) /365.254`	Returns the difference between the dates specified by the datetime1 and datetime2 variables. This difference is calculated in the number of years.

Table 2–64 Date Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle	Description
`datename (datepart, date)`	`to_char(date, format)`	Returns the specified part of the date as an integer. The MS SQL Server and Sybase 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 MS SQL Server or Sybase. See the Data Types section of this chapter for more information about conversion of the DATETIME data type.
`datepart(datepart, date)`	`to_char(date, format)`	Returns the specified part of the date as a character string (name). The MS SQL Server and Sybase 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 MS SQL Server or Sybase.
`getdate()`	`sysdate`	Returns the system date.

Recommendations:

The above table lists all the MS SQL Server and Sybase 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 MS SQL Server and Sybase 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 2–65 Mathematical Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
abs(n)	abs(n)
acos(n)	acos(n)
asin(n)	
atan(n)	atan(n)
atn2(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 2–65 Mathematical Functions in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
sqrt(n)	sqrt(n)
tan(n)	tan(n)

Recommendations:

The above table lists all the MS SQL Server and Sybase 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 MS SQL Server and Sybase 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

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 2–66 Locking in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
MS SQL Server and Sybase locking is fully automatic and does not require intervention by users. MS SQL Server and Sybase apply 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, MS SQL Server and Sybase reads block the modifications to the data.** **Update locks**: 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. **Intent locks**: MS SQL Server and Sybase locking is fully automatic and does not require intervention by users. MS SQL Server and Sybase apply 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, MS SQL Server and Sybase reads block the modifications to the data. **Extent locks:** 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. A list of active locks for the current server can be seen with SP_LOCK system procedure.	Oracle locking is fully automatic and does not require intervention by users. Oracle features the following categories of locks: Data locks (DML locks) to protect data.The "table locks" lock the entire table and "row locks" lock individual rows. Dictionary locks (DDL locks) to protect the structure of objects. Internal locks to protect internal structures, such as files. DML operations can acquire data locks at two different levels; one for specific rows and one for entire tables. Row-level locks: 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. Table-level data locks can be held in any of the following modes: Row share table lock (RW): 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. Row exclusive table lock (RX): 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. **Share lock (SL):** Share row exclusive lock(SRX) **Exclusive lock (X):** The dynamic performance table V$LOCK keeps the information about locks.

Recommendations:

In MS SQL Server and Sybase, 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 MS SQL Server or Sybase 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 MS SQL Server or Sybase logical transactions are automatically translated to Oracle logical transactions, the transactions explained above that execute properly in MS SQL Server and Sybase as they are serialized will cause a deadlock in Oracle. These transactions should be identified and serialized to avoid the deadlock. These transactions are serialized in MS SQL Server and Sybase as INSERT, UPDATE, and DELETE statements block other statements.

Row-Level Versus Page-Level Locking

Table 2–67 Row-Level Versus Page-Level Locking in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
MS SQL Server 6.5 and Sybase do not have a row-level locking feature. MS SQL Server 6.5 and Sybase apply 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. MS SQL Server 7.0 implements a form of row-level locking. MS 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.	Oracle has a row-locking feature. Only one row is locked when a DML statement is changing the row.

Recommendations:

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

Read Consistency

Table 2–68 Read Consistency in Oracle and MS SQL Server/Sybase

MS SQL Server	Oracle
MS SQL Server and Sybase provide 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. 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.	Read consistency as supported by Oracle does the following: n 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 n Ensures that reads of database data do not wait for other reads or writes of the same data n Ensures that writes of database data do not wait for reads of the same data n Ensures that writes wait for other writes only if they attempt to update identical rows in concurrent transactions To provide read consistency, Oracle creates a read-consistent set of data when a table is being read and simultaneously updated. Read consistency functions as follows: 1. When an update occurs, the original datavalues changed by the update are recorde in rollback segments. 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 afteranother user's transaction is committed reflect the changes made by the transaction. You can specify that a transaction be read only using the following command: `SET TRANSACTION READ ONLY`

Logical Transaction Handling

Table 2–69

MS SQL Server/Sybase	Oracle

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 MS SQL Server and Sybase, transactions are not implicit. Start logical transaction with a BEGIN TRANSACTION clause. Logical transactions can be committed or rolled back as follows.
`BEGIN TRANSACTION [transaction_name]`
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:
`SAVE TRANSACTION savepoint_name`
Roll back to the specified SAVEPOINT with the following command:
`ROLLBACK TRAN <savepoint_name>`
MS SQL Server and Sybase allow 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 MS SQL Server and Sybase 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.
Completed transactions are written to the database device at CHECKPOINT. A CHECKPOINT writes all dirty pages to the disk devices since the last CHECKPOINT.
Calls to remote procedures are executed independently of any transaction in which they are included.
	Statements are not automatically committed to the database. The COMMIT WORK statement is required to commit the pending changes to the database.
	Oracle transactions are implicit. This means that the logical transaction starts as soon as data changes in the database.
	COMMIT WORK commits the pending changes to the database.
	ROLLBACK undoes all the transactions after the last COMMIT WORK statement.
	Savepoints can be set in transactions with the following command:
	`SET SAVEPOINT savepoint_name`
	The following command rolls back to the specified SAVEPOINT:
	`ROLLBACK <savepoint_name>`
	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.
	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

Recommendations:

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

Oracle transactions are always implicit, which means that individual statements are not committed automatically. When converting an MS SQL Server or Sybase 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 MS SQL Server or Sybase for the application.

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

3

Triggers and Stored Procedures

This chapter includes the following sections:

Introduction

MS SQL Server and Sybase store 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 the *PL/SQL User's Guide and Reference, Release 2 (8.1.6) (Part Number A77069-01).*

The following topics are discussed in this section:

Triggers

MS SQL Server and Sybase 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.

MS SQL Server and Sybase allow INSERT, UPDATE, and DELETE triggers. Triggers typically need access to the before image and after image of the data that is being changed. MS SQL Server and Sybase achieve 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 MS SQL Server and Sybase trigger code is written to enforce referential integrity. MS SQL Server and Sybase 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 MS SQL Server or Sybase 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 the *Oracle8i Application Developer's Guide - Fundamentals, Release 2 (8.1.6)* (Part Number A76939-01).

Stored Procedures

Stored procedures provide a powerful way to code the application logic that can be stored with the server. MS SQL Server, Sybase, 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 MS SQL Server and Sybase 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 this section:

- Methods Used to Send Data to Clients
- Individual SQL Statements
- Logical Transaction Handling
- Error Handling within the Stored Procedure

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

Methods Used to Send Data to Clients

Different relational database management systems (RDBMSs) use different methods to send data to clients. For example, in MS SQL Server and Sybase the server sends data to the client in the form of a byte-stream. The client is responsible for retrieving all the data from the communication channel before sending another request to the server. In Oracle, the client can issue one or more SQL statements on the same network connection, and the system global area (SGA) stores all the data retrieved from the database. The server sends the data to the client as requested and the client sends a FETCH request on the connection whenever it is ready for the next set of results. This section discusses the different methods used to send data to clients under the following headings:

- Output Variables
- Results Sets: The MS SQL Server and Sybase Method of Sending Data to the Client
- Oracle: Cursor Variables for Returning Query Results
- MS SQL Server and Sybase: Multiple Results Sets
- MS SQL Server and Sybase: Cursors

Output Variables

MS SQL Server, Sybase, and Oracle can all send data to clients by means of output variables.

Results Sets: The MS SQL Server and Sybase Method of Sending Data to the Client

Many MS SQL Server and Sybase applications rely on the SQL Server-specific stream-based data return method called "result sets". Oracle is optimized to return data more efficiently when the data is requested using an ANSI-standard SQL SELECT statement, as compared to any proprietary stored procedure method. Therefore, the best design decision is to use stored procedures for data processing and SELECT statements for queries.

In Oracle, the use of cursor variables allows client programs to retrieve well-structured result sets.

To send even a single row back to the client from the stored procedure, MS SQL Server and Sybase can use result sets instead of an ANSI-standard method.

For example:

```
CREATE PROCEDURE get_emp_rec @empid INT
AS
        SELECT  fname, lname, loginid, addr, title, dept, mgrid
        FROM employee
        WHERE empid = @empid
```

The above procedure can be converted to an Oracle PL/SQL procedure as follows:

```
CREATE OR REPLACE PROCEDURE get_emp_rec
(empid  IN      NUMBER,
 fname  OUT     VARCHAR2,
 lname  OUT     VARCHAR2,
 loginid        OUT     VARCHAR2,
 addr   OUT     VARCHAR2,
 title  OUT     VARCHAR2,
 dept   OUT     NUMBER,
 mgrid  OUT     NUMBER)
AS
BEGIN
        SELECT  fname, lname, loginid, addr, title, dept, mgrid
        INTO    fname, lname, loginid, addr, title, dept, mgrid
        FROM    employee
        WHERE   empid = empid;
END;
```

Output variables are a structured way of sending data from server to client. Output variables allow the caller to see the results in a predictable manner, as the structure of the output variable is predefined. This method also allows encapsulation of behavior of the stored procedures.

Output variables offer the following benefits:

- Facilitate better structuring of code
- Allow the caller to see the results in a structured and predictable way, as the structure of the output variables is well defined
- Allow encapsulation of behavior of the called routine

If a third-party user interface product uses the result set capability of MS SQL Server or Sybase, make sure that the same functionality can be made available to the Oracle database. For example, PowerBuilder can use result sets to populate the data windows.

Note that although many client programs (for example, Oracle Call Interface (OCI), precompilers, SQL*Module, or SQL*Plus) can recognize cursor variables, most Open Database Connectivity (ODBC) drivers cannot. One solution when using such

ODBC drivers is to identify the code that produces the result set, and move this code online in the client program. The Oracle8*i* ODBC Driver release 8.1.5.4.0 and later releases support result sets.

In the following example, an MS SQL Server or Sybase stored procedure returns a result set with multiple rows:

```
CREATE PROCEDURE make_loginid
BEGIN
        update employee
        set loginid = substring(fname,1,1) + convert(varchar(7),empid)
           select fname, lname, loginid from employee
END
```

This procedure sends all the qualifying rows to the client as a continuous data stream. To further process the rows, the client program must retrieve the rows one after another from the communication channel.

The following piece of the DB-Library / C code executes the above procedure and prints each row to the screen.

```
main()
{

/*      Data structure dbproc is conceptually very similar
                to CDA data structure used in Oracle's OCI/C programs */
        dbcmd(dbproc, "exec make_loginid");
        /*      The above command sets the command buffer with the
                transact-sql command that needs to be executed. */

        dbsqlexec(dbproc);
        /*      This command causes the parsing and execution of the
                SQL command on the server side. */

        dbresults(dbproc);
        /*      This command puts the result rows onto the
                communications channel. */

        /*The following while loop retrieves the result rows one after the other
           by calling the function dbnextrow repeatedly. Note that this
          implementation is cursor implementation through DB-Library functions.
*/
        while (dbnextrow(dbproc) != NO_MORE_ROWS)
        {
                dbprrow(dbproc);
             /* This function prints the retrieved row to the standard output.
```

```
*/
        }
```

Such MS SQL Server and Sybase stored procedures can be migrated to the Oracle PL/SQL stored procedures or packages in different ways, as follows:

1. Place the final SELECT statement, which should return the result rows, in the client program. The Oracle client can fetch the result rows from the server as a multi-row array, and the entire process is very efficient.
2. Make use of PL/SQL tables. The SELECT statement in this case is part of the stored procedure code and the columns in the result rows are stored in PL/SQL tables. These tables are available to the client program as output variables from the stored procedures.
3. This method is the default method used by the Migration Workbench. This method is applicable only when it is extremely necessary to simulate the looping mechanism of the MS SQL Server or Sybase client to retrieve the result rows. This process is not recommended in Oracle because for each row that has to be retrieved, a FETCH request must be sent to the server from the client, thus creating more network traffic. In this case, an MS SQL Server or Sybase stored procedure is converted to a package and a member procedure. A cursor is defined with the package body; this cursor is equivalent to the SELECT statement associated with the result set. The first call to the procedure opens the cursor. Subsequent calls fetch and send the next row back to the client in the form of output parameters. Once the last row has been fetched, the cursor is closed.

Examples of these different Oracle solutions to the result set problem are presented below:

1. If the SELECT statement is made part of the client code, the PL/SQL code for the `make_loginid` procedure is as follows:

```
CREATE OR REPLACE PROCEDURE make_loginid
AS
BEGIN
        update employee
        set loginid = substr(lname,1,1)
                ||
                substr(to_char(empid),1,7);
END;
```

The following SELECT statement becomes part of the client code:

```
select fname, lname, loginid from employee
```

The following PL/SQL code shows how to migrate the `make_loginid` procedure to Oracle by using PL/SQL tables as output parameters:

```
CREATE OR REPLACE PACKAGE make_loginid_pkg
IS
BEGIN
        DECLARE EmpFnameTabType IS TABLE OF
                employee.fname %TYPE
                INDEX BY BINARY_INTEGER;
        DECLARE EmpLnameTabType IS TABLE OF
                employee.lname %TYPE
                INDEX BY BINARY_INTEGER;
        DECLARE EmpLoginidTabType IS TABLE OF
                employee.loginid %TYPE
                INDEX BY BINARY_INTEGER;
        emp_fname_tab   EmpFnameTabType;
        emp_lname_tab   EmpLnameTabType;
        emp_loginid_tab EmpLoginidTabType;
        PROCEDURE make_loginid
                (emp_fname_tab  OUT     EmpFnameTabType,
                 emp_lname_tab  OUT     EmpLnameTabType,
                 emp_loginid_tab        OUT     EmpLoginidTabType);
END make_loginid_pkg;
```

The package body definition is as follows:

```
CREATE OR REPLACE PACKAGE BODY make_loginid_pkg
IS
BEGIN
        PROCEDURE make_loginid
                (emp_fname_tab  OUT     EmpFnameTabType,

                 emp_lname_tab  OUT     EmpLnameTabType,
                 emp_loginid_tab        OUT     EmpLoginidTabType)
        AS
        DECLARE i BINARY_INTEGER := 0;
        BEGIN
                update employee
                set loginid = substr(fname,1,1)
                        ||
                        substr(to_char(empid),1,7);
                FOR emprec IN (select fname,lname,loginid
                         from employee) LOOP
                        i := i + 1;
```

```
                        emp_fname_tab[i] = emprec.fname;
                        emp_lname_tab[i] = emprec.lname;
                        emp_loginid_tab[i] = emprec.loginid;
                END LOOP;
        END make_loginid;
END make_loginid_pkg;
```

This procedure updates the PL/SQL tables with the data. This data is then available to the client after the execution of this packaged procedure.

2. The following packaged procedure sends the rows one after the other to the client upon each call to the packaged procedure. The `make_loginid_pkg.update_loginid` procedure must be executed once and the `make_loginid_pkg.fetch_emprec` procedure must be executed in a loop to fetch the rows one after another from the client program.

The package definition is as follows:

```
CREATE OR REPLACE PACKAGE make_loginid_pkg
IS
BEGIN
PROCEDURE update_loginid;
PROCEDURE fetch_emprec
                done_flag       IN OUT  INTEGER,
                nrows   IN OUT  INTEGER,
                fname   OUT             VARCHAR2,
                lname   OUT             VARCHAR2,
                loginid OUT             VARCHAR2);
END make_loginid_pkg;
```

The package body definition is as follows:

```
CREATE OR REPLACE PACKAGE BODY make_loginid_pkg
IS
BEGIN
CURSOR emprec IS
        select fname, lname, loginid
        from employee;
PROCEDURE update_loginid
IS
BEGIN
        update employee
        set loginid =   substr(fname,1,1) ||
                        substr(to_char(loginid),1,7);
END update_loginid;
```

```
PROCEDURE fetch_emprec
                done_flag       IN OUT  INTEGER,
                nrows   IN OUT  INTEGER,
                fname   OUT             VARCHAR2,
                lname   OUT             VARCHAR2,
                loginid OUT             VARCHAR2)
IS
BEGIN
        IF NOT emprec%ISOPEN THEN
                OPEN emprec;
                nrows := 0;
        END IF;
        done_flag := 0;
        FETCH emprec INTO fname, lname, loginid;
                IF emprec%NOTFOUND THEN
                CLOSE emprec;
                        done_flag := 1;
        ELSE
                nrows := nrows + 1;
        ENDIF;
END fetch_emprec;

END make_loginid_pkg;
```

Oracle: Cursor Variables for Returning Query Results

Oracle allows you to define a cursor variable to return query results. This cursor variable is similar to the user-defined record type and array type. The cursor stored in the cursor variable is like any other cursor. It is a reference to a work area associated with a multi-row query. It denotes both the set of rows and a current row in that set. The cursor referred to in the cursor variable can be opened, fetched from, and closed just like any other cursor.

There is a difference; since it is a PL/SQL variable, it can be passed into and out of procedures like any other PL/SQL variable. As a result, procedures which use cursor variables are easily reusable. You can easily see what the output of the procedure will be from looking at the procedure definition, and the same procedure can be used to return the results of a SELECT statement to a calling client program. Cursor variables can even be the return value of a function. The cursor variables preserve well-structured programming concepts while allowing the client routine to retrieve result sets.

Typically, the cursor would be declared in a client program (for example, OCI, precompilers, SQL*Module, or SQL*Plus) and then passed as an IN OUT parameter

to the PL/SQL procedure. The procedure then opens the cursor based on a SELECT statement. The calling program performs the FETCHs from the cursor, including the possibility of using ARRAY FETCH to retrieve multiple rows in one network message, and closes the cursor when it is done.

Pro*C Client:

```
...
struct emp_record {
        char ename[11];
        float sal;
}emp_record;
SQL_CURSOR c;

EXEC SQL EXECUTE
        BEGIN
                emp_package.open_emp(:c,1);
        END;
END-EXEC;
...
/* fetch loop until done */
EXEC SQL FETCH :c INTO :emp_record;
...
CLOSE :c;
...
```

Oracle Server:

```
CREATE OR REPLACE PACKAGE emp_package IS
        TYPE emp_part_rec IS RECORD
        (ename emp.ename%type, sal emp.sal%type);
        TYPE emp_cursor IS REF CURSOR
                RETURN emp_part_rec;
        PROCEDURE open_emp (c_emp IN OUT emp_cursor,
                        select_type IN NUMBER);
END emp_package;

CREATE OR REPLACE PACKAGE BODY emp_package IS
PROCEDURE open_emp (c_emp IN OUT emp_cursor,
                        select_type IN NUMBER) IS
        BEGIN
                IF select_type=1 THEN
                        OPEN c_emp FOR SELECT ename, sal FROM EMP
                                WHERE COMM IS NOT NULL;
                ELSE
```

```
                    OPEN c_emp FOR SELECT ename, sal FROM EMP;
                END IF;
        END open_emp;
END emp_package;
```

MS SQL Server and Sybase: Multiple Results Sets

MS SQL Server and Sybase stored procedures can return multiple different result sets to the calling routine.

For example, consider the following procedure:

```
CREATE PROCEDURE example_proc
AS
BEGIN

SELECT empno, empname, empaddr FROM emp
WHERE empno BETWEEN 1000 and 2000

SELECT empno, deptno, deptname FROM emp, dept
WHERE emp.empno = dept.empno
AND emp.empno BETWEEN 1000 and 2000

END
```

This procedure returns two different result sets. The client is responsible for processing the results. To convert MS SQL Server or Sybase multiple result sets to Oracle, pass one more cursor variable to the stored procedure to open a second cursor; the client program then looks at both cursor variables for data. However, it can be difficult to track all the result sets in a single procedure. It is recommended that you just use one result set, that is, one cursor variable per procedure, if possible.

MS SQL Server and Sybase: Cursors

Cursors allow row-by-row operations on a given result set. MS SQL Server and Sybase provide ANSI-standard SQL syntax to handle cursors. The additional DECLARE CURSOR, OPEN, FETCH, CLOSE, and DEALLOCATE CURSOR clauses are included in T-SQL. Using these statements you can achieve cursor manipulation in a stored procedure. After FETCHing the individual row of a result set, this current row can be modified with extensions provided with UPDATE and DELETE statements.

The UPDATE statement syntax is as follows:

```
update <table_name>
```

```
set <column_name> = <expression>
from <table1>, <table_name>
where current of <cursor name>
The DELETE statement syntax is as follows:
delete from <table_name>
where current of <cursor name>
MS SQL Server and Sybase cursors map one-to-one with Oracle cursors.
```

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, MS SQL Server and Sybase 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:

MS SQL Server and Sybase:

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

Oracle:

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

MS SQL Server and Sybase:

```
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 MS SQL Server and Sybase, 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 MS SQL Server and Sybase, 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 MS SQL Server and Sybase, 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 MS SQL Server or Sybase 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. Note that this coding style is required only to simulate MS SQL Server or Sybase 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 an MS SQL Server or Sybase 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 will give 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 MS SQL Server or Sybase 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 MS SQL Server or Sybase, so that it can be converted to the Oracle RAISE_APPLICATION_ERROR statement.

Customized Error Messages

MS SQL Server and Sybase allow 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 MS SQL Server or Sybase 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 will serve a two-fold purpose: it will ensure that the system is more portable across different types of database servers, and it will give 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 section in Chapter 2 for a list of MS SQL Server and Sybase data types and their equivalent Oracle data types.

Composite Data Types

MS SQL Server and Sybase do 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 MS SQL Server, Sybase, 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 four sections

- Syntax
- Description
- Permissions
- Examples

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

See the Schema Objects section in Chapter 2 for information about database schema objects.

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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Syntax:
`CREATE PROCEDURE procedure [@formal_parameter formal_ parameter_data type [OUTPUT] [= default_value] [,@formal_parameter formal_ parameter_datatype [OUTPUT] [= default_value]] ...`
 	
`AS [BEGIN] procedural_statements [END]`
	Syntax:
	`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] [DEFAULT default_value]] ... [)] IS | AS [local_variable data type;]... BEGIN PL/SQL statements | PL/SQL blocks END;`

Table 3–2 Comparison of Creating the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Description:	**Description:**
The CREATE PROCEDURE statement creates the named stored procedure in the database.	The OR REPLACE keywords replace the procedure by the new definition if it already exists.
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.	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.
The keyword AS indicates the start of the body of the procedure.	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.
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.	The BEGIN and END keywords enclose the body of the procedure.
See the T-SQL and PL/SQL Language Elements section of this chapter for more information about the constructs allowed in T-SQL procedures.	
Permissions:	**Permissions**:
You must have the CREATE PROCEDURE system privilege to create the stored procedures	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.

Table 3–2 Comparison of Creating the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example:	**Example**:
`CREATE PROCEDURE myproc @cust char(30)= space(30), @cust_id int OUTPUT, @param3 datetime OUTPUTAS BEGIN DECLARE @local_var1 int, @local_var2 datetime SELECT @local_var2 = getdate() SELECT @param3 = @local_var2 SELECT @local_var1 = customer_id FROM customer WHERE customer = @cust SELECT @cust_id = @local_var1 END`	`CREATE OR REPLACE PROCEDURE sam.credit (acc_no IN NUMBER DEFAULT 0, acc IN VARCHAR2, amount IN NUMBER, return_status OUT NUMBER) AS BEGIN UPDATE accounts SET balance = balance + amount WHERE account_id = acc_no; EXCEPTION WHEN SQL%NOTFOUND THEN RAISE_APPLICATION_ERROR (-20101, 'Error updating accounts table'); END`

Recommendations:

Functionally identical parts can be identified in the T-SQL procedure and PL/SQL procedure structure. It is, therefore, easy to automate the conversion of most of the constructs from MS SQL Server or Sybase to Oracle.

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

Drop

Table 3–3 Comparison of Dropping the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax:**
`DROP PROCEDURE procedure`	`DROP PROCEDURE [schema.]procedure`
Description:	**Description**:
The procedure definition is deleted from the data dictionary. All the objects that reference this procedure must have references to this procedure removed	When a procedure is dropped, Oracle invalidates all the local objects that reference the dropped procedure

Table 3–3 Comparison of Dropping the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Permissions:	**Permissions**:
Procedure owners can drop their own procedures. A DBO can drop any procedure.	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
Example:	**Example**:
`DROP PROCEDURE myproc`	`DROP PROCEDURE sam.credit;`

Recommendations:

The above 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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: `EXEC [@return_value = ] procedure [[@formal_parameter = ] {@actual_parameter | constant_literal} [OUT]] [,[[@formal_parameter = ] {@actual_parameter | constant_literal} [OUT]]] ...`	**Syntax**: `procedure [([{actual_parameter | constant_literal | formal_parameter => {actual_parameter | constant_literal} }] [,{actual_parameter | constant_literal | formal_parameter => {actual_parameter | constant_literal} }] .... )]`

Table 3–4 Comparison of Executing the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Description:
 	
MS SQL Server and Sybase 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.
 	
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.
 	
Positional notation:
The actual parameters are supplied to the procedure in the same order as the formal parameters in the procedure definition.
 	
Named notation:
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:
 	
`@formal_parameter = @actual_parameter`
 	
A constant literal can be specified in the place of the following:
 	
`'@actual_parameter ' as:`
`@formal_parameter = 10`
 	
The keyword OUT should be specified if the procedure has to return the value of that parameter to the calling routine as OUTPUT.
	Description:
	
	Oracle PL/SQL procedures send data back to the calling routine by means of OUT parameters. Oracle offers FUNCTIONS 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.
	
	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.
	
	Positional notation:
	The actual parameters are supplied to the procedure in the same order as the formal parameters in the procedure definition.
	
	Named notation:
	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:
	
	`formal_parameter => actual_parameter`
	
	A constant literal can be specified in the place of the following:
	
	'actual_parameter' as:
	`formal_parameter => 10`
	
	If the formal_parameter is specified as OUT or IN OUT in the procedure definition, the value will be made available to the calling routine after the execution of the procedure

Table 3–4 Comparison of Executing the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Permissions: 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.	**Permissions** 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

Example:
Positional notation:
`EXEC GetEmplName @EmpID`
`EXEC @status = GetAllDeptCodes`
`EXEC @status = UpdateEmpSalary @EmpID, @EmpName`
`EXEC UpdateEmpSalary 13000,'Joe Richards'`
Named notation:
`EXEC UpdateEmpSalary @Employee = @EmpName, @Employee_Id = @EmpID`
Mixed notation:
`EXEC UpdateEmpSalary @EmpName, @Employee_Id = @EmpID`
`EXEC UpdateEmpSalary @Employee = @EmpName, @EmpID`
	Example:
	Positional notation:
	`credit (accno, accname, amt, retstat);`
	Named notation:
	`credit (acc_no => accno, acc => accname, amount => amt, return_status => retstat)`
	Mixed notation (where positional notation must precede named notation):
	`credit (accno, accname, amount => amt, return_status => retstat)`

Alter

Table 3–5 Comparison of Altering the Procedure Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax**:

The system procedure SP_ RECOMPILE recompiles the named stored procedure. For example:
`ALTER PROCEDURE <procedure name>`
`|RECOMPILE`
`|ENCRYPT`
`|RECOMPILE, ENCRYPT`
	`ALTER PROCEDURE [schema.]procedure COMPILE`

Description:	**Description:**
This command causes the recompilation of the procedure. Procedures that become invalid for some reason should be recompiled explicitly using this command.	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
Permissions:	**Permissions**:
The owner of the procedure can issue this command	The procedure must be in the user's schema or the user must have the ALTER ANY PROCEDURE privilege to use this command
Example:	**Example**:
`sp_recompile my_proc`	`ALTER PROCEDURE sam.credit COMPILE;`

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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: In MS SQL Server and Sybase, a stored procedure can be easily converted to a function in Oracle because the stored procedure in MS SQL Server or Sybase 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	**Syntax**: `CREATE [OR REPLACE] FUNCTION [user.]function [(parameter [OUT] data type[,(parameter [IN OUT] data type]...)] RETURN data type { IS | AS } block`
N/A	**Description**: The OR REPLACE keywords replace the function with the new definition if it already exists. 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. 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 the Data Types section in Chatper 2, Database for more information about data types.
N/A	**Permissions:** 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.

Table 3–6 Comparison of Creating the Function Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N/A	**Example**: `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;`

Drop

Table 3–7 Comparison of Dropping the Function Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N/A	**Syntax**: `DROP FUNCTION [schema.]function`
N/A	**Description**: When a function is dropped, Oracle invalidates all the local objects that reference the dropped function.
N/A	**Permissions**: 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
N/A	**Example**: `DROP FUNCTION sam.credit;`

Execute

Table 3–8 Comparison of Executing the Function Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Syntax**: `function [({actual_parameter | constant_ literal}...)]`
N / A	**Description**: Functions can return an atomic value to the calling routine using the RETURN statement. A function can be called as part of an expression. This is a very powerful concept. All the MS SQL Server and Sybase 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.
N / A	**Permissions**: 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.

N / A	**Example**:
	`1) IF sal_ok (new_sal, new_title) THEN ....  END IF;`
	`2) promotable:= sal_ok(new_sal, new_title) AND (rating>3);`
	where sal_ok is a function that returns a BOOLEAN value.

Alter

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

MS SQL Server	Oracle
N/A	**Syntax:** `ALTER FUNCTION [schema.]function COMPILE`
N/A	**Description:** 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	**Permissions:** 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	**Example:** `ALTER FUNCTION sam.credit COMPILE`

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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax**:
MS SQL Server and Sybase do not support this concept.	`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]`
N/A	**Description**:
	This is the external or public part of the package.
	CREATE PACKAGE sets up the specification for a PL/SQL package which can be a group of procedures, functions, exception, variables, constants, and cursors.
	Functions and procedures of the package can share data through variables, constants, and cursors.
	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.
N/A	**Permissions**:
	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.

Table 3–10 Comparison of Creating the Package Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

N / A	**Example**:
	`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;`

Drop

Table 3–11 Comparison of Dropping the Package Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax: MS SQL Server and Sybase do not support this concept.	**Syntax**: `DROP PACKAGE [BODY] [schema.]package`
N / A	**Description**: 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.

Table 3–11 Comparison of Dropping the Package Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Permissions**: 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.
N / A	**Example**: `DROP PACKAGE emp_actions;`

Alter

Table 3–12

MS SQL Server	Oracle
Syntax: MS SQL Server and Sybase do not support this concept.	**Syntax**: `ALTER PACKAGE [user.]package COMPILE [PACKAGE | BODY]`
N / A	**Description**: Packages that become invalid for some reason should be recompiled explicitly using this command. 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. PACKAGE, the default option, recompiles the package body and specification. BODY recompiles only the package body. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
N / A	**Permissions**: The package must be in the user's schema or the user must have the ALTER ANY PROCEDURE privilege to use this command.

Table 3–12

MS SQL Server	Oracle
N / A	**Example**: `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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax**:
MS SQL Server and Sybase do not support this concept.	`CREATE [OR REPLACE] PACKAGE BODY [schema.]package {IS | AS} pl/sql_package_body`

Table 3–13 Comparison of Creating the Package Body Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description**: 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. See the *PL/SQL User's Guide and Reference, Release 2 (8.1.6)* (Part Number A77069-01) for more information on writing package bodies**.**
N / A	**Permissions**: 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.

Table 3–13 Comparison of Creating the Package Body Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

N / A	**Example:**
	`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;`

Drop

Table 3–14 Comparison of Dropping the Package Body Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax**:
MS SQL Server and Sybase do not support this concept.	`DROP PACKAGE [BODY] [schema.]package`

Table 3–14 Comparison of Dropping the Package Body Schema Object in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
N / A	**Description**: 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	**Permissions**: 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	**Example**: `DROP PACKAGE BODY emp_actions;`

Alter

Table 3–15

MS SQL Server/Sybase	Oracle
Syntax:	**Syntax**:
MS SQL Server and Sybase do not support this concept.	`ALTER PACKAGE [user.]package COMPILE [PACKAGE | BODY]`

Table 3–15

MS SQL Server/Sybase	Oracle
N / A	**Description**: Packages that become invalid for some reason should be recompiled explicitly using this command. 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. PACKAGE, the default option, recompiles the package body and specification. BODY recompiles only the package body. Explicit recompilation eliminates the need for implicit recompilation and prevents associated runtime compilation errors and performance overhead.
N / A	**Permissions**: The package must be your own schema or you must have the ALTER ANY PROCEDURE privilege to use this command.
N / A	**Example**: `ALTER PACKAGE emp_actions COMPILE BODY`

T-SQL Versus PL/SQL Constructs

This section provides information about the MS SQL Server and Sybase constructs and equivalent Oracle constructs generated by the Migration Workbench. 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
- SELECT Statement as Part of the SELECT List
- SELECT Statement with GROUP BY Clause
- Column Aliases
- UPDATE with FROM Statement
- DELETE with FROM Statement
- Temporary Tables
- Result Set (Converted Using a Cursor Variable)
- Cursor Handling
- Transaction Handling Statements

You will find listed the syntax for the MS SQL Server and Sybase constructs and their Oracle equivalents, as well as comments about conversion considerations.

Note that the procedures in the Oracle column are the direct output of the Migration Workbench. These PL/SQL procedures have more lines of code compared to the source MS SQL Server and Sybase procedures because these PL/SQL procedures are converted to emulate MS SQL Server and Sybase functionality. The PL/SQL procedures written from scratch for the same functionality in Oracle would be much more compact.

Also, note that the PL/SQL procedures generated by the Migration Workbench indicate the manual conversion required by adding appropriate commands.

In general, the Migration Workbench deals with the MS SQL Server and Sybase 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.

- MS SQL Server-specific constructs are converted into PL/SQL constructs if the equivalent constructs are available in PL/SQL.
- Some MS 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 MS 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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
CREATE PROC proc1 AS RETURN 0	CREATE OR REPLACE FUNCTION PROC1 RETURN INTEGER AS StoO_selcnt INTEGER; StoO_error INTEGER; StoO_rowcnt INTEGER; StoO_errmsg VARCHAR2(255); StoO_sqlstatus INTEGER; BEGIN RETURN 0; END PROC1; /

Comments

An MS SQL Server or Sybase stored procedure can be converted to a stored procedure, a function, or a package in Oracle. The output depends upon the option used when running the Migration Workbench.

The Migration Workbench automatically adds what is necessary to simulate MS SQL Server or Sybase functionality. In the example in Table 3-16 above, the Migration Workbench added the following three variables:

```
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
```

These variables are needed in the EXCEPTION clause in the PL/SQL procedures that must be added for each SQL statement to emulate MS SQL Server or Sybase functionality. See the SELECT Statement topic in this section for clarification of the purpose of these variables.

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

Parameter Passing

Table 3–17 Comparison of Parameter Passing in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
CREATE PROC proc1 @x int=-1, @y money, @z bit OUT, @a char(20) = 'TEST' AS RETURN 0	CREATE OR REPLACE FUNCTION PROC1(x INTEGER DEFAULT -1, y NUMBER , z IN OUT NUMBER, a CHAR DEFAULT 'TEST') RETURN INTEGER AS StoO_selcnt INTEGER; StoO_error INTEGER; StoO_rowcnt INTEGER; StoO_errmsg VARCHAR2(255); StoO_sqlstatus INTEGER; BEGIN RETURN 0; END PROC1; /

Comments

Parameter passing is almost the same in MS SQL Server, Sybase, 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.

MS SQL Server and Sybase data types are converted to Oracle base data types. For example, all MS SQL Server and Sybase 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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
DECLARE
@x int,
@y money,
@z bit,
@a char(20)
RETURN 0
GO
	CREATE OR REPLACE FUNCTION PROC1
	RETURN INTEGER
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	x INTEGER;
	y NUMBER;
	z NUMBER;
	a CHAR(20);
	BEGIN
	RETURN 0;
	END PROC1;
	/

Comments

MS SQL Server, Sybase, and Oracle follow similar rules for declaring local variables.

The Migration Workbench 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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Example 1:
`CREATE PROC proc1 @Flag int = 0`
`AS`
`BEGIN`
`DECLARE @x int`
`IF ( @Flag=0 )`
`SELECT @x = -1`
`ELSE`
`SELECT @x = 10`
`END`
	Example 1:
	`CREATE OR REPLACE PROCEDURE PROC1(`
	`Flag INTEGER DEFAULT 0)`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error INTEGER;`
	`StoO_rowcnt INTEGER;`
	`StoO_errmsg VARCHAR2(255);`
	`StoO_sqlstatus INTEGER;`
	`x INTEGER;`
	`BEGIN`
	`IF (PROC1.Flag = 0) THEN`
	`PROC1.x := -1;`
	`ELSE`
	`PROC1.x := 10;`
	`END IF;`
	`END;`
	`/`

Table 3–19 Comparison of IF Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example 2:	**Example 2:**

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
	CREATE OR REPLACE PROCEDURE PROC1(
	Flag CHAR DEFAULT ' ')
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	x INTEGER;
	BEGIN
	IF (PROC1.Flag = ' ') THEN
	PROC1.x := -1;
	ELSE
	IF (PROC1.Flag = 'a') THEN
	PROC1.x := 10;
	ELSE
	IF (PROC1.Flag = 'b') THEN
	PROC1.x := 20;
	END IF;
	END IF;
	END IF;
	END;
	/

Table 3–19 Comparison of IF Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example 3:	**Example 3:**

```
CREATE PROC proc1
AS
BEGIN
DECLARE @x int
IF EXISTS ( SELECT * FROM
table2 )
 SELECT @x = -1
END
```

```
CREATE OR REPLACE PROCEDURE PROC1
AS
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus  INTEGER;
x       INTEGER;
BEGIN

        BEGIN
        StoO_selcnt := 0;
        StoO_error  := 0;
        StoO_rowcnt := 0;
        SELECT 1 INTO StoO_selcnt
        FROM DUAL
        WHERE EXISTS (
                SELECT  *
                 FROM TABLE2);
     StoO_rowcnt := SQL%ROWCOUNT;
     EXCEPTION
           WHEN OTHERS THEN
              StoO_selcnt := 0;
              StoO_error := SQLCODE;
             StoO_errmsg := SQLERRM;
        END;
        IF StoO_selcnt != 0 THEN
                PROC1.x :=  -1;
        END IF;
END;
/
```

Table 3–19 Comparison of IF Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Example 4:
CREATE PROC proc1 @basesal money, @empid int
AS
BEGIN
IF (select sal from emp where empid = @empid) < @basesal
UPDATE emp
SET sal_flag = -1
WHERE empid = @empid
END
	Example 4:
	CREATE OR REPLACE PROCEDURE PROC1(
	basesal NUMBER ,
	empid INTEGER)
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	BEGIN
	StoO_selcnt := 0;
	StoO_error := 0;
	StoO_rowcnt := 0;
	SELECT 1 INTO StoO_selcnt
	FROM DUAL
	WHERE (
	SELECT SAL
	FROM EMP
	WHERE EMPID = PROC1.empid)<PROC1.basesal;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	IF StoO_selcnt != 0 THEN
	BEGIN
	StoO_error := 0;
	StoO_rowcnt := 0;
	UPDATE EMP
	SET SAL_FLAG = -1
	WHERE EMPID = PROC1.empid;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END IF;
	END;
	/

Comments

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

If EXISTS(...) in MS SQL Server and Sybase 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 above.

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 above.

RETURN Statement

Table 3–20 Comparison of RETURN Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
@x int
AS
IF @x = -1
RETURN 25022
ELSE
RETURN 25011
	CREATE OR REPLACE FUNCTION PROC1(
	x INTEGER)
	RETURN INTEGER
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	IF PROC1.x = -1 THEN
	RETURN 25022;
	ELSE
	RETURN 25011;
	END IF;
	END PROC1;
	/

Comments

A RETURN statement is used to return a single value back to the calling program and works the same in both databases. MS SQL Server and Sybase 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. For this reason, the value is commented out if the MS SQL Server or Sybase procedure is converted to a PL/SQL procedure, but not commented out if converted to a PL/SQL function. The Migration Workbench does this automatically.

RAISERROR Statement

Table 3–21 Comparison of RAISERROR Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
CREATE PROC proc1 AS RAISERROR 12345 "No Employees found"	CREATE OR REPLACE PROCEDURE PROC1 AS StoO_selcnt INTEGER; StoO_error INTEGER; StoO_rowcnt INTEGER; StoO_errmsg VARCHAR2(255); StoO_sqlstatus INTEGER; BEGIN raise_application_error(-20999, 12345 ‖ '-' ‖ "No Employees Found"); END PROC1; /

Comments

MS SQL Server and Sybase use 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.

The Migration Workbench copies the error code and error message from a RAISERROR statement and places them in the RAISE_APPLICATION_ERROR statement appended to the error message.

EXECUTE Statement

Table 3–22 Comparison of EXECUTE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
EXEC SetExistFlag
EXEC SetExistFlag yes=@yes,
@Status OUT
EXEC @Status = RecordExists
EXEC SetExistFlag @yes
	CREATE OR REPLACE PROCEDURE PROC1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	BEGIN
	SETEXISTFLAG;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	BEGIN
	SETEXISTFLAG(=>PROC1.yes,
	PROC1.Status);
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error :=
	SQLCODE;
	StoO_errmsg :=
	SQLERRM;
	END;
	BEGIN
	PROC1.Status:=RECORDEXISTS;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	BEGIN
	SETEXISTFLAG(PROC1.yes);
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END PROC1;
	/

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. If a procedure is converted to a PL/SQL function, make sure to assign the RETURN value to a variable when calling it (see the call to RecordExists in Table 3-22 above).

The Migration Workbench converts the parameter-calling convention to be either positional, named, or mixed. For information on parameter-calling conventions, see the Schema Objects section in this chapter.

WHILE Statement

Table 3–23 Comparison of WHILE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example 1:	**Example 1:**

```
CREATE PROC proc1
 @i int
 AS
        WHILE @i > 0

        BEGIN
         print 'Looping inside
WHILE....'
         SELECT @i = @i + 1
        END
```

```
CREATE OR REPLACE PROCEDURE PROC1(
in_i    IN INTEGER )
      AS

StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus INTEGER;
i       INTEGER;
BEGIN
    PROC1.i := PROC1.in_i;
           <<i_loop1>>
     WHILE PROC1.i > 0 LOOP
BEGIN
DBMS_OUTPUT.PUT_LINE('Looping inside
while.....') ;
  PROC1.i :=  PROC1.i + 1;
                    END;
END LOOP;
END PROC1;
   /
```

Table 3–23 Comparison of WHILE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Example 2:
CREATE PROC proc1
@i int,
@y int
AS
WHILE @i > 0
BEGIN
print 'Looping inside WHILE....'
SELECT @i = @i + 1
END
	Example 2:
	CREATE OR REPLACE PROCEDURE PROC1(
	in_i IN INTEGER ,
	y INTEGER)
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	i INTEGER;
	BEGIN
	PROC1.i := PROC1.in_i;
	<<i_loop1>>
	WHILE PROC1.i > 0 LOOP
	BEGIN
	DBMS_OUTPUT.PUT_LINE('Looping inside while.....') ;
	PROC1.i := PROC1.i + 1;
	END;
	END LOOP;
	END PROC1;
	/

Table 3–23 Comparison of WHILE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
Example 3:	**Example 3:**

```
CREATE PROC proc1
 AS
 DECLARE @sal money
 SELECT @sal = 0
 WHILE EXISTS(SELECT * FROM emp
where sal < @sal )
 BEGIN
  SELECT @sal = @sal + 99

  DELETE emp
  WHERE sal < @sal
 END
 GO
```

```
 CREATE OR REPLACE PROCEDURE PROC1
 AS
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus  INTEGER;
sal     NUMBER;
 BEGIN
      PROC1.sal :=  0;
         <<i_loop1>>
         WHILE 1 = 1 LOOP
  BEGIN
BEGIN
StoO_selcnt := 0;
StoO_error  := 0;
SELECT 1 INTO StoO_selcnt FROM DUAL
WHERE (EXISTS (
SELECT  *
FROM EMP
WHERE SAL < PROC1.sal));
EXCEPTION
 WHEN OTHERS THEN
StoO_selcnt := 0;
StoO_error := SQLCODE;
StoO_errmsg := SQLERRM;
   END;
         IF StoO_selcnt != 1 THEN
            EXIT;
            END IF;
       PROC1.sal :=  PROC1.sal + 99;
            BEGIN
 StoO_error   := 0;
 StoO_rowcnt  := 0;
 DELETE  EMP
      WHERE SAL < PROC1.sal;
  StoO_rowcnt := SQL%ROWCOUNT;
  EXCEPTION
      WHEN OTHERS THEN
       StoO_error  := SQLCODE;
       StoO_errmsg := SQLERRM;
          END;
            END;
             END LOOP;
               END PROC1;
/
```

Table 3–23 Comparison of WHILE Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

Example 4:

```
CREATE PROC proc1
 AS

 DECLARE @sal money

 WHILE (SELECT count (*) FROM emp
) > 0
 BEGIN
 SELECT @sal = max(sal) from emp
  WHERE stat = 1

   DELETE emp
  WHERE sal < @sal
 END
 GO
```

Example 4:

```
CREATE OR REPLACE PROCEDURE PROC1
  AS
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus  INTEGER;
        sal     NUMBER;
BEGIN
      <<i_loop1>>
WHILE 1 = 1 LOOP
BEGIN

BEGIN
StoO_selcnt := 0;
StoO_error  := 0;
SELECT 1 INTO StoO_selcnt FROM DUAL
WHERE ((
SELECT  COUNT(*)
FROM EMP)>0);
EXCEPTION
WHEN OTHERS THEN
StoO_selcnt := 0;
StoO_error := SQLCODE;
StoO_errmsg := SQLERRM;
END;
IF StoO_selcnt != 1 THEN
EXIT;
END IF;
BEGIN
StoO_rowcnt := 0;
StoO_selcnt := 0;
StoO_error  := 0;
 SELECT   MAX(SAL)
INTO PROC1.sal FROM EMP
WHERE STAT = 1;
   StoO_rowcnt := SQL%ROWCOUNT;
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
   StoO_rowcnt := 2;
 WHEN OTHERS THEN
  StoO_rowcnt := 0;
  StoO_selcnt := 0;
  StoO_error := SQLCODE;
  StoO_errmsg := SQLERRM;
   END;
  BEGIN
 StoO_error   := 0;
 StoO_rowcnt  := 0;
DELETE  EMP
    WHERE SAL < PROC1.sal;
   StoO_rowcnt := SQL%ROWCOUNT;
EXCEPTION
 WHEN OTHERS THEN
 StoO_error  := SQLCODE;
StoO_errmsg := SQLERRM;
END;
END;
END LOOP;
END PROC1;
/
```

Comments

The Migration Workbench can convert most WHILE constructs. However, the CONTINUE within a WHILE loop in MS SQL Server and Sybase does not have a direct equivalent in PL/SQL. It is simulated using the GOTO statement with a label. Because the Migration Workbench is a single-pass parser, it adds a label statement at the very beginning of every WHILE loop (see Example 2 in Table 3-23 above).

GOTO Statement

Table 3–24 Comparison of GOTO Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1 @Status int
AS
DECLARE @j int
IF @Status = -1
GOTO Error
 	
SELECT @j = -1
Error:
SELECT @j = -99
	CREATE OR REPLACE PROCEDURE PROC1(
	Status INTEGER)
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	j INTEGER;
	BEGIN
	IF PROC1.Status = -1 THEN
	GOTO ERROR;
	END IF;
	PROC1.j := -1;
	<<ERROR>>
	PROC1.j := 99;
	END PROC1;
	/

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 MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

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.'
	CREATE OR REPLACE PROCEDURE proc1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	i_x INTEGER;
	BEGIN
	BEGIN
	SELECT count(*)
	INTO i_x
	FROM emp;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN TOO_MANY_ROWS THEN
	StoO_rowcnt := 2;
	WHEN OTHERS THEN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	END;
	IF StoO_rowcnt = 0 THEN
	DBMS_OUTPUT.PUT_LINE
	('No rows found.') ;
	END IF;
	IF StoO_error = 0 THEN
	DBMS_OUTPUT.PUT_LINE('No errors.') ;
	END IF;
	END;
	/

Comments

@@rowcount is converted to StoO_rowcnt, which takes its value from the PL/SQL cursor attribute SQL%ROWCOUNT.

@@error is converted to StoO_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. This requires that the Migration Workbench add a local exception block around every SQL statement

and a few PL/SQL statements. Other global variables are converted with a warning message. These may need to be converted manually.

ASSIGNMENT Statement

Table 3–26 Comparison of ASSIGNMENT Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
DECLARE @x int
SELECT @x = -1
SELECT @x=sum(salary) FROM
employee
	CREATE OR REPLACE PROCEDURE PROC1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	x INTEGER;
	BEGIN
	PROC1.x := -1;
	BEGIN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := 0;
	
	SELECT SUM(SALARY)
	INTO PROC1.x FROM
	EMPLOYEE;
	StoO_rowcnt :=
	SQL%ROWCOUNT;
	
	EXCEPTION
	WHEN TOO_MANY_ROWS THEN
	StoO_rowcnt := 2;
	WHEN OTHERS THEN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END PROC1;
	/

Comments

Assignment in MS SQL Server and Sybase 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, or a NULL value is assigned to the variable as can be seen in the following example:

```
SELECT empno INTO empno
FROM employee
WHERE ename = 'JOE RICHARDS'
```

SELECT Statement

Table 3–27 Comparison of SELECT Statement in Oracle and MS SQL Server/Sybase

MS SQL Server	Oracle

Example 1:
`CREATE PROC proc1`
`AS`
`SELECT ename FROM employee`
	Example 1:
	`CREATE OR REPLACE PACKAGE PROC1Pkg AS`
	`TYPE RT1 IS RECORD (`
	`ENAME`
	`EMPLOYEE.ENAME%TYPE`
	`);`
	`TYPE RCT1 IS REF CURSOR RETURN RT1;`
	`END;`
	`/`
	`CREATE OR REPLACE PROCEDURE PROC1(`
	`RC1 IN OUT PROC1Pkg.RCT1)`
	`AS`
	`StoO_selcnt INTEGER;`
	`StoO_error INTEGER;`
	`StoO_rowcnt INTEGER;`
	`StoO_errmsg VARCHAR2(255);`
	`StoO_sqlstatus INTEGER;`
	`BEGIN`
	`OPEN RC1 FOR`
	`SELECT ENAME FROM EMPLOYEE;`
	`END PROC1;`
	`/`

Table 3–27 Comparison of SELECT Statement in Oracle and MS SQL Server/Sybase

MS SQL Server	Oracle
Example 2:	**Example 2**

```
CREATE PROC proc1
AS
DECLARE @name char(20)
SELECT @name = ename FROM
employee
IF @@rowcount = 0
 RETURN 25022
```

```
CREATE OR REPLACE FUNCTION PROC1
RETURN INTEGER
AS
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus  INTEGER;
name    CHAR(20);
BEGIN
        BEGIN
                StoO_rowcnt := 0;
                StoO_selcnt := 0;
                StoO_error  := 0;

                SELECT   ENAME
      INTO PROC1.name FROM EMPLOYEE;
      StoO_rowcnt := SQL%ROWCOUNT;

        EXCEPTION
           WHEN TOO_MANY_ROWS THEN
                StoO_rowcnt := 2;
                    WHEN OTHERS THEN
               StoO_rowcnt := 0;
               StoO_selcnt := 0;
              StoO_error := SQLCODE;
             StoO_errmsg := SQLERRM;
        END;
        IF StoO_rowcnt = 0 THEN
                RETURN 25022;
        END IF;
END PROC1;
/
```

Comments

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

MS SQL Server, Sybase, and Oracle can all pass data to the client using output parameters in the stored procedures. MS SQL Server and Sybase use another method known as "Result Sets" to transfer the data from the server to client. The examples discussed here do not return multiple rows to the client.

In Example 1, the procedure returns a single row result set to the client which is converted to a PL/SQL procedure that returns a single row using the output parameters.

Example 1:

A SELECT statement is converted into a SELECT...INTO clause and the extra parameter "i_oval1" is added to the procedure definition. Since the Migration Workbench does not currently look up the data types on the Oracle server, it sets the default data type to VARCHAR2.

Note: In Oracle, the query should return only one row or the TOO_MANY_ROWS exception is raised and the data value is not assigned to the variables. To return more than one row, refer to the example on RESULT SETS later in this section.

In MS SQL Server and Sybase, if the SELECT statement that assigns value to a variable returns more than one value, the last value that is returned is assigned to the variable.

Example 2:

The second example illustrates fetching data into a local variable. Since this is straightforward, the Migration Workbench handles it successfully.

Note: MS SQL Server-specific SQL statements should be converted manually. The Migration Workbench handles ANSI-standard SQL statements only.

SELECT Statement as Part of the SELECT List

Table 3–28 Comparison of SELECT Statement as Part of the SELECT List in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase

```
CREATE PROC proc1
AS
DECLARE @x int
DECLARE @y char(20)
SELECT @x = col1, @y = (select
name from emp)
FROM table1
```

Oracle

```
CREATE OR REPLACE PROCEDURE PROC1
AS
StoO_selcnt     INTEGER;
StoO_error      INTEGER;
StoO_rowcnt     INTEGER;
StoO_errmsg     VARCHAR2(255);
StoO_sqlstatus  INTEGER;
x       INTEGER;
y       CHAR(20);
temp_var1       VARCHAR2(255);
BEGIN
/****** Subqueries in select list is not
supported in Oracle. *******/
/****** MANUAL CONVERSION MIGHT BE REQUIRED
*******/
        BEGIN
        StoO_error   := 0;
        StoO_rowcnt  := 0;
        SELECT NAME
        INTO temp_var1
         FROM EMP;
        StoO_rowcnt := SQL%ROWCOUNT;
        EXCEPTION
        WHEN TOO_MANY_ROWS THEN
            StoO_StoO_rowcnt  := 2;
        WHEN OTHERS THEN
            StoO_StoO_rowcnt  := 0;
            StoO_error  := SQLCODE;
            StoO_errmsg := SQLERRM;
        END;
        BEGIN
               StoO_rowcnt := 0;
                StoO_selcnt := 0;
                StoO_error  := 0;
      SELECT   COL1,  temp_var1
   INTO PROC1.x, PROC1.y FROM TABLE1;
    StoO_rowcnt := SQL%ROWCOUNT;
             EXCEPTION
             WHEN TOO_MANY_ROWS THEN
                 StoO_rowcnt := 2;
             WHEN OTHERS THEN
              StoO_rowcnt := 0;
              StoO_selcnt := 0;
              StoO_error := SQLCODE;
              StoO_errmsg := SQLERRM;
        END;
END PROC1;
/
```

Comments

The MS SQL Server and Sybase SELECT statement with a subquery as part of the SELECT list cannot be converted to PL/SQL using the Migration Workbench. Manual changes are needed to convert this type of SELECT statement.

The Migration Workbench writes appropriate comments in the output PL/SQL procedures and the subqueries are omitted.

SELECT Statement with GROUP BY Clause

Table 3–29 Comparison of SELECT Statement with GROUP BY Clause in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
DECLARE @ename char(20)
DECLARE @salary int
SELECT @ename=ename,
@salary=salary FROM emp
WHERE salary > 100000
GROUP BY deptno
	CREATE OR REPLACE PROCEDURE PROC1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	ename CHAR(20);
	salary INTEGER;
	BEGIN
	BEGIN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := 0;
	
	SELECT ENAME, SALARY
	INTO PROC1.ename,
	PROC1.salary FROM EMP
	WHERE SALARY > 100000
	GROUP BY DEPTNO;
	StoO_rowcnt := SQL%ROWCOUNT;
	
	EXCEPTION
	WHEN TOO_MANY_ROWS THEN
	StoO_rowcnt := 2;
	WHEN OTHERS THEN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END PROC1;
	/

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.

The Migration Workbench converts this type of SELECT statement to PL/SQL. The equivalent PL/SQL statement, however, will give an error in Oracle.

Column Aliases

Table 3–30 Comparison of Column Aliases in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
CREATE PROC proc1 @Status int=0 AS SELECT x=sum(salary) FROM employee	CREATE OR REPLACE PROCEDURE PROC1(Status INTEGER DEFAULT 0, RC1 IN OUT PROC1Pkg.RCT1) AS StoO_selcnt INTEGER; StoO_error INTEGER; StoO_rowcnt INTEGER; StoO_errmsg VARCHAR2(255); StoO_sqlstatus INTEGER; BEGIN OPEN RC1 FOR SELECT SUM(SALARY) "X" FROM EMPLOYEE; END PROC1; /

Comments

The Migration Workbench can convert MS SQL Server-specific column aliases to the equivalent Oracle format. No manual changes are required.

UPDATE with FROM Statement

Table 3–31 Comparison of UPDATE with FROM Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
UPDATE table1
SET col1 = 1
FROM table1, table2
WHERE table1.id =
table2.id
	CREATE OR REPLACE PROCEDURE proc1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	BEGIN
	BEGIN
	UPDATE table1
	SET
	col1 = 1 **/* FROM table1,table2 --**
	MANUAL CONVERSION */
	WHERE table1.id = table2.id;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	END;
	END;

Comments

An UPDATE with a FROM clause cannot be converted. Instead, the Migration Workbench provides a comment indicating that manual conversion is required.

There are two ways to convert UPDATE with a FROM statements, and these are illustrated below.

Method 1:

Use the subquery in the SET clause if columns are being updated to values coming from a different table. For example, consider the following T/SQL statement:

```
UPDATE titles
SET pub_id = publishers.pub_id
FROM titles, publishers
WHERE titles.title like 'C%'
AND publishers.pub_name = 'new age'
```

Convert this statement to the following PL/SQL statement 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:

Use the subquery in the WHERE clause for all other UPDATE...FROM statements. For example, consider the following T/SQL statement:

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

Convert this statement to the following PL/SQL statement in Oracle :

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

DELETE with FROM Statement

Table 3–32 Comparison of DELETE with FROM Statement in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
DELETE FROM table1
FROM table1, table2
WHERE table1.id =
table2.id
	CREATE OR REPLACE PROCEDURE PROC1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	UF1_rowid ROWID;
	UF1_oval1 TABLE1.COL1%TYPE;
	
	CURSOR UF1_cursor IS
	SELECT TABLE1.ROWID, 1 FROM
	TABLE1, TABLE2
	
	WHERE TABLE1.ID = TABLE2.ID
	FOR UPDATE OF TABLE1.COL1;
	BEGIN
	
	OPEN UF1_cursor;
	LOOP
	FETCH UF1_cursor INTO UF1_rowid, UF1_
	oval1;
	EXIT WHEN UF1_cursor%NOTFOUND;
	BEGIN
	StoO_error := 0;
	StoO_rowcnt := 0;
	UPDATE TABLE1 SET COL1 = UF1_oval1
	WHERE ROWID = UF1_rowid;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END LOOP;
	CLOSE UF1_cursor;
	END PROC1;
	/

Comments

A DELETE with FROM..FROM clause must be converted manually.

While converting DELETE with FROM..FROM clause, remove the second FROM clause. For example consider the following T/SQL statement:

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

Convert the above statement to the following PL/SQL statement in Oracle:

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

Temporary Tables

Table 3–33 Comparison of Temporary Tables in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
SELECT col1, col2
INTO #Tab
FROM table1
WHERE table1.id = 100
	CREATE OR REPLACE PROCEDURE PROC1
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	/*CONVERTING SELECT INTO temp_Tab*/
	/*TO INSERT INTO temp_Tab*/
	BEGIN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := 0;
	INSERT INTO temp_Tab
	SELECT
	USERENV('SESSIONID'), COL1, COL2 FROM
	TABLE1
	WHERE TABLE1.ID = 100;
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN TOO_MANY_ROWS THEN
	StoO_rowcnt := 2;
	WHEN OTHERS THEN
	StoO_rowcnt := 0;
	StoO_selcnt := 0;
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	END PROC1;
	/

Comments

Temporary tables are supported by Oracle8*i*. The Migration Workbench utilizes this feature in Oracle8*i*.

Also, SELECT..INTO..#TEMPTAB is converted to an INSERT statement. You must make manual changes to ensure that rows are unique to a particular session and all

the rows for that session are deleted at the end of the operation. This requires that you add an extra column to the table definition and the value of `USERENV('session_id')` for all the rows inserted. At the end, delete all rows for that `session_id`. If many procedures use the same temp table in the same session, `SEQUENCE` can be used to make sure that the rows are unique to a particular `session_id/SEQUENCE` combination.

Result Set (Converted Using a Cursor Variable)

Command Option -M

Table 3–34

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
SELECT col1, col2
FROM table1
	CREATE OR REPLACE PACKAGE PROC1Pkg
	AS
	TYPE RT1 IS RECORD (
	COL1
	TABLE1.COL1%TYPE,
	COL2
	TABLE1.COL2%TYPE
	);
	TYPE RCT1 IS REF CURSOR RETURN RT1;
	END;
	/
	
	CREATE OR REPLACE PROCEDURE PROC1(
	RC1 IN OUT PROC1Pkg.RCT1)
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	OPEN RC1 FOR
	SELECT COL1, COL2 FROM
	TABLE1;
	END PROC1;
	/

Comments

Convert an MS SQL Server or Sybase procedure that returns a multi-row result set to a PL/SQL packaged function by selecting the appropriate parse option in the property sheet for a stored procedure.

The T-SQL SELECT statement is converted to a cursor and a cursor variable is added as an OUT parameter to return the data back to the calling program. Use the cursor referenced by the cursor variable to fetch the result rows.

For more details on how Result Sets are handled by the Migration Workbench, see T-SQL and PL/SQL Language Elements section in this chapter.

Note: The conversion to a packaged function does not work in all cases. Carefully check the input source and decide whether it can be converted to a packaged function. Also check the output for accuracy.

Cursor Handling

Table 3–35 Comparison of Cursor Handling Result Set in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

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
	CREATE OR REPLACE PROCEDURE CURSOR_
	DEMO
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	empno INTEGER;
	ename CHAR(100);
	sal NUMBER;
	CURSOR CURSOR_1 IS
	SELECT EMPNO, ENAME, SAL
	FROM EMP;
	
	BEGIN
	OPEN CURSOR_1;
	
	CURSOR_1 INTO
	cursor_demo.empno, cursor_
	demo.ename, cursor_demo.sal;
	
	IF CURSOR_1%NOTFOUND THEN
	StoO_sqlstatus := 2;
	ELSE
	StoO_sqlstatus := 0;
	END IF;
	CLOSE CURSOR_1;
	
	/*[SPCONV-ERR(xxx)]:Deallocate Cursor
	is not supported*/
	
	NULL;
	
	END CURSOR_DEMO;
	/

Comments

MS SQL Server and Sybase introduced cursors in T-SQL. Syntactical conversion of cursors from MS SQL Server or Sybase to Oracle is very straightforward. Note that in PL/SQL, deallocation of cursors is not required as it happens transparently.

Transaction Handling Statements

Table 3–36 Comparison of Transaction-Handling Statements in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle

CREATE PROC proc1
AS
BEGIN TRAN tran1
 	
UPDATE table1
SET id = id + 1
WHERE name = 'Event'
 	
IF @@Rowcount != 1
BEGIN
ROLLBACK TRAN tran1
RETURN 25700
END
 	
COMMIT TRAN tran1
RETURN 0
	CREATE OR REPLACE FUNCTION PROC1
	RETURN INTEGER
	AS
	StoO_selcnt INTEGER;
	StoO_error INTEGER;
	StoO_rowcnt INTEGER;
	StoO_errmsg VARCHAR2(255);
	StoO_sqlstatus INTEGER;
	BEGIN
	SAVEPOINT TRAN1;
	BEGIN
	StoO_error := 0;
	StoO_rowcnt := 0;
	UPDATE TABLE1
	SET ID = ID + 1
	
	WHERE NAME = 'Event';
	StoO_rowcnt := SQL%ROWCOUNT;
	EXCEPTION
	WHEN OTHERS THEN
	StoO_error := SQLCODE;
	StoO_errmsg := SQLERRM;
	END;
	IF StoO_rowcnt != 1 THEN
	BEGIN
	ROLLBACK TO SAVEPOINT TRAN1;
	RETURN 25700;
	END;
	END IF;
	COMMIT WORK;
	RETURN 0;
	END PROC1;
	/

Comments

The Migration Workbench does a one-to-one mapping when converting MS SQL Server and Sybase transaction commands to their Oracle equivalents. For more details about how transactions are handled in Oracle, see the Transaction-Handling Semantics topic later in this chapter.

Note: Make sure that the functionality remains the same, as the transaction models may differ in MS SQL Server, Sybase, and Oracle.

T-SQL and PL/SQL Language Elements

T-SQL is the MS SQL Server and Sybase 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
- Sending Data to the Client: Result Sets
- DDL Constructs within MS SQL Server and Sybase Stored Procedures

Transaction Handling Semantics

MS SQL Server and Sybase

MS SQL Server and Sybase offer two different transaction models: the ANSI-standard implicit transaction model and the explicit transaction model.

MS SQL Server and Sybase provide 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 MS SQL Server and Sybase, the default is `isolation level 1`.

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

If the MS SQL Server or Sybase application implements ANSI-standard chained (implicit) transactions with `isolation level 3`, the application will migrate smoothly to Oracle because Oracle implements the ANSI-standard implicit transaction model, which ensures repeatable reads.

In a non-ANSI standard application, MS SQL Server and Sybase 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. Note, however, that 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 nesting levels of the nested transactions.

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 col1='value_1';
SAVEPOINT first_sp;
UPDATE test_table SET col1='value_2';
ROLLBACK TO SAVEPOINT first_sp;
COMMIT;   /* col1 is 'value_1'*/
```

Conversion Preparation Recommendations

Logical transactions are handled differently in MS SQL Server, Sybase, and Oracle. In MS SQL Server and Sybase, 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, MS SQL Server and Sybase require that transactions in stored procedures be allowed to nest, whereas Oracle does not support transaction nesting.

The following table compares MS SQL Server and Sybase to Oracle transaction-handling statements:

Table 3–37 Comparison of Transaction-Handling Statements in Oracle and MS SQL Server/Sybase

MS SQL Server/Sybase	Oracle
BEGIN TRAN	
BEGIN TRAN tran_1	SAVEPOINT tran_1
COMMIT TRAN	COMMIT
(for the transaction with nest level=1)	
COMMIT TRAN	
(for the transaction with nest level>1)	
COMMIT TRAN tran_1	COMMIT
(for the transaction with nest level=1)	
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, the Migration Workbench cannot determine the nest level of the current transaction-handling statement. The variable `@@trancount` is a runtime environment variable.

Table 3-38 shows the currently implemented MS SQL Server and Sybase to Oracle conversion strategy for the transaction-handling statements

Table 3–38 Conversion Strategy for Transaction-Handling Statements

MS SQL Server/Sybase	Oracle
BEGIN TRAN	/*BEGIN TRAN >>> statement ignored <<<*/
BEGIN TRAN tran_1	SAVEPOINT tran_1;

Table 3–38 Conversion Strategy for Transaction-Handling Statements

MS SQL Server/Sybase	Oracle
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 MS SQL Server and Sybase, make the changes so that at least the outermost transaction is controlled in the client application.

Exception-Handling and Error-Handling Semantics

MS SQL Server and Sybase

In MS SQL Server and Sybase, 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.

MS SQL Server and Sybase allow 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 MS SQL Server or Sybase 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 the T-SQL Versus PL/SQL Constructs section in this chapter for more information about the extra code required to simulate MS SQL Server or Sybase behavior.

If the RAISERROR statement in an MS SQL Server or Sybase 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 will serve a two-fold purpose: it will ensure that the system is portable, and it gives the administrator centralized control over the error messages.

Special Global Variables

MS SQL Server and Sybase

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.

`@@trancount`:
Keeps track of the nesting level for the nested transactions.

`@@rowcount`:
The number of rows affected by the most recently executed T-SQL statement.

`@@servername`:
The name of the local MS SQL Server or Sybase server.

`@@transtate`:
The current state of the transaction.

`@@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 TL/SQL procedure is in chained, or implicit transaction mode.

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 the Data Manipulation Language section in Chapter 2 for a discussion of MS SQL Server, Sybase, and Oracle operators.

Built-in Functions

See the Data Manipulation Language section in Chapter 2 for a discussion of built-in functions in MS SQL Server, Sybase, and Oracle.

Sending Data to the Client: Result Sets

Single Result Set

MS SQL Server and Sybase stored procedures can return data to the client by means of a Result Set. A SELECT statement that does not assign values to the local variables sends the data to the client in the form of byte-stream.

In a case where a third-party user interface product uses the result set capability of MS SQL Server or Sybase, consult with the vendor to make sure that the same functionality is available for the Oracle database.

The following example procedure sends the data out as a result set. More appropriately, an OUTPUT parameter holding the value "YES" or "NO" (depending upon the evaluation of <condition>) or a function returning "YES" or "NO" should have been used.

```
CREATE PROCEDURE x
AS
BEGIN
...
...
IF <condition> THEN
  SELECT "YES"
ELSE
  SELECT "NO"
END
```

Multiple Result Sets

Avoid MS SQL Server or Sybase stored procedures that return multiple result sets to the calling routine.

The following procedure returns two different result sets, which the client is responsible for processing:

```
CREATE PROCEDURE example_proc
AS
BEGIN

SELECT empno,empname, empaddr FROM emp
WHERE empno BETWEEN 1000 and 2000
SELECT empno,deptno, deptname FROM emp, dept
WHERE empno.empno = dept.empno
AND emp.empno BETWEEN 1000 and 2000

END
```

Recommendations

Some alternatives to simulating the result set in PL/SQL procedures are presented below:

- Packaged procedures with PL/SQL tables as output parameters
- This is an extension of the first method. Instead of fetching one row at a time, now we fetch many rows (ARRAY FETCH) at a time and assign the values to PL/SQL tables. These tables are available to the client after the execution of the procedure.
- Packaged procedures with a cursor variable as output parameter
- This alternative is possible in Oracle. Oracle allows you to define a cursor type variable to clearly return query results. This cursor type variable is similar to the user-defined record type and array variable. The cursor stored in the cursor variable is like any other cursor. It is a reference to a work area associated with a multi-row query. It denotes both the set of rows and a current row in that set. The cursor referred to in the cursor variable can be opened, fetched from, and closed just like any other cursor. Since it is a PL/SQL variable, it can be passed into and out of procedures like any other PL/SQL variable. This is a more direct equivalent to the result set in MS SQL Server and Sybase.
- Procedure or function that populates a temporary table with result set rows
- This temporary table has an additional column to hold the SESSION_ID of the current session to keep the rows separate for each session of the user. The client program can then retrieve the rows from this temporary table with a simple SELECT statement.
- The Migration Workbench adopts the third option to convert the result set.

About Converting a T-SQL Procedure with a Result Set

Method 1

A T-SQL procedure with a result set may need some manual changes after conversion to an Oracle package with a member function. The problems are described in detail below.

For example, consider the following T-SQL procedure:

```
CREATE PROC test_proc
AS
BEGIN
        T-SQL block1
        T-SQL block2
        SELECT statement corresponding to the result set
END
```

This procedure executes two T-SQL blocks before executing the SELECT statement associated with the result set. The procedure is converted to an Oracle package as follows:

```
CREATE OR REPLACE PACKAGE BODY test_proc_pkg
AS
BEGIN
        FUNCTION test_proc;
END;
CREATE OR REPLACE PACKAGE BODY test_proc_pkg
AS
BEGIN
        cursor declaration for the SELECT statement associated with the result
set in the source T-SQL procedure;
        FUNCTION test_proc
        RETURN INTEGER
        AS
        BEGIN
                PL/SQL version of T-SQL block1;
                PL/SQL version of T-SQL block2;
                FETCH loop for the cursor declared in the package body;
        END;
END;
```

The two T-SQL blocks in the source T-SQL procedure are executed only once when the procedure is called, and the result set is sent to the client.

In Oracle client, to simulate the fetching of the result set, the TEST_PROC_ PKG.TEST_PROC function must be called repeatedly until all the rows from the

cursor are fetched. The two PL/SQL blocks in the function are executed with each call to the function. This behavior differs from that in the source application.

You must manually separate the code associated with the FETCH loop for the cursor for the result set from the remaining code in the procedure. Changes to the client have to be made so that the rest of the procedure's code is called in accurate sequence with the repeated calls to the function returning rows from the result set.

The final Oracle package should be as follows:

```
CREATE OR REPLACE PACKAGE BODY test_proc_pkg
AS
BEGIN
        PROCEDURE proc1;
        FUNCTION test_proc;
END;
CREATE OR REPLACE PACKAGE BODY test_proc_pkg
AS
BEGIN
        cursor declaration for the SELECT statement associated with the result
set in the source T-SQL procedure;
        PROCEDURE proc1
        AS
        BEGIN
                PL/SQL version of T-SQL block1;
                PL/SQL version of T-SQL block2;
        END;
        FUNCTION test_proc
        RETURN INTEGER
        AS
        BEGIN
                FETCH loop for the cursor declared in the package body;
        END;
END;
```

The client should call the TEST_PROC_PKG.PROC1 procedure before repeatedly calling the TEST_PROC.PKG.TEXT_PROC function in order to achieve functionality similar to the source T-SQL procedure.

The variables that are common to these two parts should be either declared globally within the package body or should be passed as parameters to the procedure and the function.

DDL Constructs within MS SQL Server and Sybase Stored Procedures

MS SQL Server and Sybase allow 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.

Most of the T-SQL DDL constructs give syntax errors. You must remove the DDL statements from the T-SQL source to convert the T-SQL procedure to PL/SQL using the Migration Workbench.

The following DDL statements are ignored by the Migration Workbench. The statements appear commented in the output with a message "statement ignored."

```
CREATE TABLE
DROP TABLE
CREATE VIEW
DROP VIEW
CREATE INDEX
DROP INDEX
```

4

Distributed Environments

This chapter includes the following sections:

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 SQL statement outlined above can be changed to the following:

```
SELECT * FROM emp;
```

MS SQL Server and Sybase and Remote Objects

MS SQL Server and Sybase require 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.

MS SQL Server and Sybase do 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, MS SQL Server, and Sybase.

The MS SQL Server or Sybase 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 MS SQL Server 6.5 and Sybase 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 characteristics listed above, MS 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 MS SQL Server and Sybase, conversion of distributed applications from MS SQL Server or Sybase 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 MS SQL Server and Sybase applications are written using ODBC application programming interfaces (APIs) or Visual Basic. DB-Library is widely used to develop 3GL applications with MS SQL Server or Sybase as the backend.

Since Oracle provides ODBC connectivity, it is possible to convert ODBC-based MS SQL Server or Sybase applications to work with an Oracle backend.

If a Visual Basic application is written with ODBC as the connection protocol to access MS SQL Server or Sybase, it is possible to modify and fix the Visual Basic application to work with an Oracle backend.

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

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

5

Migrating Temporary Tables to Oracle

Temporary tables are available in Oracle8*i*. However, because Oracle8*i* temporary tables differ from MS SQL Server temporary tables you should still replace or emulate temporary tables within Oracle to ease migrations from MS SQL Server.

The emulation of temporary tables has been simplified by using temporary tables instead of permanent tables. See the Oracle8*i* temporary table syntax for Example 2 in the Implementation of Temporary Tables as Permanent Tables section.

This chapter discusses temporary tables under the following headings:

- Temporary Table Usage
- Replace Temporary Tables
- Emulate Temporary Tables
- Definition of temp_table_catalog
- Package Body temp_table

Temporary Table Usage

In MS SQL Server and Sybase, temporary tables are used to:

- Simplify Coding
- Simulate Cursors when Processing Data from Multiple Tables
- Improve Performance In a Situation Where Multi-Table Joins are Needed
- Associate Rows from Multiple Queries in One Result Set (UNION)
- Eliminate Re-Querying Data Needed for Joins
- Consolidate the Data for Decision Support Data Requirements

Simplify Coding

Instead of writing complicated multi-table join queries, temporary tables allow a query to be broken into different queries, where result sets of one query are stored in a temporary table and subsequent queries join this temporary table with actual database tables.

This type of code can be converted to Oracle as follows:

- Rewrite the queries to use multi-table joins
- Create permanent temporary tables
- Tune the complicated query using the parallel query option

MS SQL Server and Sybase:

```
WHILE @cur_dt > @start_dt
BEGIN
INSERT #TEMP1
        SELECT @cur_dt
        SELECT @cur_dt = dateadd(dd, -7, @cur_dt)
END
/******** create a temp table *****/
INSERT #TEMP2
SELECT t2.col1,
                t4.col2,
                " ",
                t5.col3,
           t2.col4,
                t3.col5,
                t2.col6,
                t2.col7,
                t4.col8,
                t4.col9
FROM
        db1..TABLE1 t1,
        db2..TABLE2 t2,
        db2..TABLE3 t3,
        db2..TABLE4 t4,
        db1..TABLE5 t5
WHERE   t1.col10 =@col10
AND     t1.col11 = @flag1
AND             t1.col2 = t4.col2
AND             t1.col2 = t5.col2
AND             t2.col4 between @start_col4 and @end_col4
AND             t3.col5 between @start_col5 and @end_col5
```

```
AND             t3.col12 = @flag2
AND             t2.col13 = @flag1
AND             t4.col2 like @col2
AND             t4.col14 = @flag3
AND             t4.col12 = @flag2
AND             t2.col1 = t4.col1
AND             t3.col1 = t2.col1
AND             t4.col1 = t3.col1
AND             t5.col2 like @col2
AND             t4.col2 = t5.col2
AND             t4.col15 = t5.col15
AND             t5.col3 like @var1
AND             t2.col6 <= @end_dt
AND             (t2.col7 >= @start_dt OR t2.col7 = NULL)
AND     t4.col8 <=@end_dt
UPDATE TABLE4
SET     t4.col2 = col16
FROM    #TEMP2 t1, db2..TABLE4 t4
WHERE   t1.col1 = t4.col1
AND             t4.col12 = @flag2
AND             t4.col14 = @flag4
```

Oracle Pseudo Code:

```
Use a PL/SQL table to simulate #TEMP1
For the INSERT #TEMP2 statement
Declare a cursor with the same SELECT statement
        (as used in MS SQL Server and Sybase)
For the UPDATE statement do the following:
loop
        fetch the cursor
        if cursor not found
           then exit ;
        end if ;
        -- update TABLE4 for each row that matches the criteria
        -- Note : i_col17 and i_col1 are local PL/SQL variables
                                which are populated by each fetch
        UPDATE TABLE4
        SET     col2 = i_col17
        WHERE   col1 = i_col1
        AND             col12 = @flag2
        AND             col14 = @flag4
end loop
```

Simulate Cursors when Processing Data from Multiple Tables

Oracle supports cursors, so this type of code can be converted to Oracle using cursors.

The following code is part of a procedure written in MS SQL Server. Compare it with the Oracle example (much simpler coding) that performs the same function.

MS SQL Server:

```
...
SELECT * INTO #emp FROM emp WHERE emp.dept = 10
SELECT @cnt = @@rowcount
WHILE @cnt > 0
BEGIN
        SELECT @name = name, @emp_id = emp_id
        FROM #emp
        WHERE emp_id = (SELECT MAX (emp_id) FROM #emp)
                    /* process this row */
        DELETE FROM #emp WHERE emp_id = @emp_id
        SELECT @cnt = @cnt -1
END
...
```

Oracle:

```
FOR emp_rec IN (SELECT name, emp_id FROM emp WHERE dept = 10)
        LOOP  /*process emp_rec.name and emp_rec.emp_id*/
        END LOOP
```

Improve Performance In a Situation Where Multi-Table Joins are Needed

In MS SQL Server and Sybase, you sometimes use temporary tables to avoid multi-table joins. These cases can be converted to Oracle, as Oracle performs complex multi-table queries more efficiently than MS SQL Server and Sybase.

See the sample code provided in the To Simplify Coding section for more information in this regard.

Associate Rows from Multiple Queries in One Result Set (UNION)

Oracle provides a UNION relational operator to achieve similar results.

MS SQL Server and Sybase:

```
INSERT #EMPL_TEMP
SELECT emp.empno
                dept.dept_no
                location.location_code
                emp.start_date
                emp.end_date
FROM    emp,
                dept ,
                location
WHERE   emp.empno = location.empno
AND             dept.deptno = emp.deptno
AND             dept.deptno = location.deptno
AND     emp.start_date BETWEEN @start_date AND @end_date
INSERT INTO #EMPL_TEMP VALUES ( 10000, 10, 15,getdate(),NULL )
...
```

Oracle:

```
SELECT  emp.empno
                dept.dept_no
                location.location_code
                emp.start_date
                emp.end_date
FROM    emp,
                dept ,
                location
WHERE   emp.empno = location.empno
AND     dept.deptno = emp.deptno
AND     dept.deptno = location.deptno
AND     emp.start_date BETWEEN i_start_date AND i_end_date
UNION
SELECT  10000,
                10,
                15,
                SYSDATE,
                NULL
FROM    DUAL
```

Eliminate Re-Querying Data Needed for Joins

Permanent tables can be created in Oracle to hold the data. The data in these tables can be deleted at the end of processing. If no COMMIT is performed and no DDL is issued, the records in these tables will not be recorded in the database. If a COMMIT is performed, the records from these tables can be deleted at the end of the process. Records in these tables can be kept separate for different users by having an additional column that holds a SESSION_ID.

If it is not possible to create the tables ahead of time, tables can be created dynamically with Oracle, using the DBMS_SQL package. In dynamically-created tables, the extra SESSION_ID columns are no longer needed, and space management issues such as fragmentation are eliminated. Performance may be affected, but deleting a large number of rows from a permanent temporary table also affects performance. In dynamic SQL, tables can be truncated or dropped.

MS SQL Server and Sybase:

```
INSERT #EMPL_TEMP
SELECT  emp.empno
                dept.dept_no
                emp.start_date
                emp.end_date
FROM    emp,
                dept ,
WHERE   emp.empno = dept.deptno
AND     emp.start_date BETWEEN @start_date AND @end_date
....
....
/* Later in the code, one needs to select from the temp table
   only, it is not necessary to do a join of EMP and DEPT */
SELECT  * FROM  #EMPL_TEMP
```

Oracle:

```
SELECT  emp.empno
                dept.dept_no
                emp.start_date
                emp.end_date
FROM    emp,
                dept ,
WHERE   emp.empno = dept.deptno
AND     emp.start_date BETWEEN i_start_date AND i_end_date  ;
/* The above join has to be performed every time one needs to get this result set
*/
```

Consolidate the Data for Decision Support Data Requirements

You often need to consolidate data across servers in a distributed database environment. You can use predefined views to consolidate this type of data. Oracle snapshots can replicate the data from remote databases. In addition, you can create permanent tables for MS SQL Server or Sybase temporary tables if queries need to perform joins against these tables.

Replace Temporary Tables

You should replace temporary tables to give the best performance in Oracle. You should always try to replace temporary tables with standard Oracle SQL. To do this, you must first determine the function of the temporary table. The function of the temporary table will be one of the following:

- To store an intermediate result
- To collect data

Emulate Temporary Tables

If it is not possible to replace temporary tables, you should emulate them as follows:

- Use PL/SQL tables to emulate temporary tables
- Create temporary tables as ordinary tables whenever they are needed.
- Create permanent tables and maintain them for multiple users.

Implementation as PL/SQL Tables

Temporary tables can be implemented as a PL/SQL table of records. Although this concept is quite appealing, you cannot use SQL on a PL/SQL table. Therefore, this concept is limited to simple uses of temporary tables. However, for simple uses of temporary tables, you should always consider replacing these temporary tables completely with standard SQL.

Implications of Creating Temporary Tables Dynamically

Since temporary tables can be created by any session "on the fly", you may have multiple instances of the same temporary table within one schema. As this type of multiple instance is not possible in Oracle, you should attach the SESSION_ID to the table name to make it unique. The result is a variable table name, which requires that all accesses to that table must be created with dynamic SQL. This process would complicate all types of migration tools.

As all DDL operations have an implicit commit, the creation of a temporary table would disturb the transactional behavior of the migrated application. The programs would have to be changed so that the creation of a temporary table always occurs at the start of a transaction. This process would also complicate migration tools.

Implications of Creating Permanent Tables

Currently, several users can share one table. Therefore, you need to maintain an additional column in the table for the SESSION_ID. As the SESSION_ID is unique in the lifetime of a database, there will be no access conflicts. The enforcement of the SESSION_ID can be accomplished with a view and a trigger. The cleanup in this option may be slower, as you must now delete rows and cannot do a simple DROP TABLE. You can execute this operation asynchronously with the JOBQUEUE package, or use the TRUNCATE TABLE command whenever you are the only user of the table. To avoid bottlenecks on the temporary tables, it is possible to create multiple incarnations of them and point the users via private synonyms. Also, the upcoming SQL3 Standard implements temporary tables as permanent tables, which have an incarnation per session.

These arguments show that the permanent table option is the best choice.

Implementation of Temporary Tables as Permanent Tables

The migration utility must first extract from the source database code all commands which create a temporary table.

The following MS SQL Server / Sybase T-SQL examples illustrate two types of such commands:

Example 1

```
CREATE TEMP TABLE tmpdate(
FromDt datetime year to minute,
ToDt datetime year to minute);
```

Example 2

```
SELECT aaufromdt date
from anforord aau, order ord, case cas, casetype ctp
where ctp.ctp_id  = CtpId
and ctp.ctpambukz = "N"
and cas.ctp_id    =  ctp.ctp_id
and ord.cas_id    = cas.cas_id
and aau.ord_id    = ord.ord_id
and cas.casgtg    = "Y"
```

```
and ordstozt is null
INTO temp tmpfromdate;
```

All such statements must be modified as follows:

- Change the syntax to Oracle syntax.
- Identify and substitute alternative values for Oracle reserved words.
- Prefix the name of the temporary table with "temp_".

When you have completed these steps, Example 1 type statements may be executed.

For statements of the same type as Example 2, you must also perform the following steps:

- Remove all bind variables, such as CtpId, and replace them with constants.
- Embed the statement in the following wrapper and execute it:

```
create table temp_<temptable>
as select *
from (<original statement>)
where 1=0; -- or similar logic to create the table without any rows
```

The complete Oracle code for Example 2 is as follows:

```
create table temp_tmpfromdate
as select * from
(
SELECT aaufromdt inf_date
from anforord aau, order ord, case cas, casetype ctp
where ctp.ctp_id  = 'X' -- CtpId
and ctp.ctpambukz = 'N'
and cas.ctp_id    =  ctp.ctp_id
and ord.cas_id    = cas.cas_id
and aau.ord_id    = ord.ord_id
and cas.casgtg    = 'Y'
and ordstozt is null)
where 0=1;
```

Oracle8*i* Temporary Tables

Oracle8*i* temporary table data is not visible across sessions so the SESSION_ID column is not required.

The Oracle8*i* temporary table syntax for Example 2 is as follows:

```
create table global temporary temp_<temptable> on commit preserve rows
```

```
as select * from (<original statement>)
where 1=0
```

The Migration Workbench does the following when it encounters a temporary table in a stored procedure or trigger:

- Generates the DDL to create the table
- Renames the table to `temp_tmpfromdate`
- Checks the column names for reserved words
- Adds the SESSION_ID column (if Oracle8*i* temporary tables are not being used)

With this setup, you can use the table `tmpfromdate` as if it is available once per session.

Maintenance of Temporary Tables

To maintain the temporary tables, you need a dictionary table temp_table_catalog (see Definition of temp_table_catalog) and the supporting package temp_table (see Package Body temp_table). The temp_table package performs all maintenance for temporary tables. To generate it, you need the following grants:

```
grant select on v_$session to <xxx>;
grant execute on dbms_sql to <xxx>;
grant execute on dbms_lock to <xxx>;
grant create public synonym to <xxx>;
grant create view to <xxx>;
grant create trigger to <xxx>;
```

The available functionality is explained in the comments of the package temp_table as follows:

```
create or replace PACKAGE temp_table IS

        procedure convert_to_temp (table_name in varchar2,
                                   use_dbms_output in boolean default
false);
--
--      Convert an ordinary table to a temporary table.
--

        procedure register (table_name in varchar2);
--      Register the usage of temporary table in temp_table_catalog
```

```
--      This procedure will be called out of the pre-insert trigger
--      on the temporary table.
        procedure drop_temp_table (table_name in varchar2);
--      Check usage in temp_table_catalog, delete the data of the
--      session and unregister the table

        procedure cleanup_session;
--      Find all temporary table usages of the session, delete or truncate
--      the temporary table and unregister the usage.
--      This procedure commits!

    END;
```

Definition of temp_table_catalog

```
create table temp_table_catalog
(session_id number,
 table_name varchar2(30),
 constraint temp_table_catalog_pk
    primary key (session_id, table_name))
```

Package Body temp_table

```
create or replace PACKAGE BODY temp_table IS

                last_table      varchar2(30) := ;  -- Store the last used
                                        -- object for the register procedure
--  The constant use_truncate enables the use of the truncate command on
--  temporary tables. Change it to false if that is not desired.
                use_truncate    constant boolean := true;

        procedure parse_sql (user_cursor in number,
                             sql_text in varchar2) is
        begin
                dbms_sql.parse (user_cursor, sql_text, dbms_sql.v7);
        exception
           when others then
                raise_application_error (-20100, 'Parsing Error ' ||
                        to_char (sqlcode) || ' at ' ||
                        to_char (dbms_sql.last_error_position + 1) ||
```

```
                        ' starting with: ' ||
                        substr (sql_text, dbms_sql.last_error_position + 1, 30)
||
                        '...', true);
        end;

        procedure execute_sql (sql_text in varchar2) is
                ignore  number;
                user_cursor     number;
        begin
                user_cursor := dbms_sql.open_cursor;
                parse_sql (user_cursor, sql_text);
                ignore := dbms_sql.execute (user_cursor);
                dbms_sql.close_cursor(user_cursor);
        exception
                when others then
                        if dbms_sql.is_open(user_cursor) then
                                dbms_sql.close_cursor(user_cursor);
                        end if;
                        raise;
        end;

        function get_lock_id (object_name in varchar2)
--
-- This function returns the lock_id for a specific object.
-- It is calculated as the object_id from oracle + 1000000
--
        return number is
                object_number   number;
        begin
                select object_id
                into   object_number
                from   user_objects uo
                where  uo.object_name = get_lock_id.object_name
                and    uo.object_type = 'VIEW';
                return object_number + 1000000;
        exception
--  Object not found ==> Raise error
           when no_data_found then
                raise_application_error (-20100, 'Object ' ||
                                         object_name || ' does not exists');
        end;


        procedure convert_to_temp (table_name in varchar2,
```

```
                                        use_dbms_output in boolean default false) is
--
--      Convert an ordinary table to a temporary table.
--
                sql_stmt        varchar2 (32000);
                col_sep         varchar2 (2) := null;
                con_list        varchar2 (100) := 'session_id';
                sel_table       varchar2 (30);
                procedure add (s in varchar2)
                is
--  Print one line of SQL code on sql_stmt or dbms_output
                begin
                        if use_dbms_output then
                                dbms_output.put_line (chr (9) || s);
                        else
                                sql_stmt := sql_stmt || chr (10) || s;
                        end if;
                end add;
                procedure execute_immediate
                as
                begin
                    if ( use_dbms_output ) then
                        dbms_output.put_line( '/' );
                    else
                        execute_sql (sql_stmt);
                        dbms_output.put_line(
                            substr( sql_stmt, 2, instr( sql_stmt,chr(10),2)-2 )
);
                        sql_stmt := NULL;
                    end if;
                end;
        begin
                if ( use_dbms_output ) then
                        sel_table := upper (table_name);
                else
                        sel_table := 'TEMP_' || upper (table_name);
                end if;
-- Rename the table to temp_XXX


                add ('rename ' || table_name);
                add ('to temp_' || table_name);
                execute_immediate;
-- In the next step we need to add the support for the sessionid column.
-- The column will be added with the following statement:
```

```
                add ('alter table temp_' || table_name);
                add ('add session_id number not null');
                execute_immediate;
-- Create a view for the original table
                add ('create view ' || table_name);
                add ('as select ');
                for col_rec in
                        (select column_name, table_name
                        from user_tab_columns
                        where table_name = sel_table
                        and   column_name != 'SESSION_ID'
                        order by column_id) loop
                        add (col_sep || col_rec.column_name);
                        col_sep := ', ';
                end loop;
                add (' from temp_' || table_name);
                add ('where session_id = userenv (''sessionid'')');
                execute_immediate;
-- To allow public access we need to create a public synonym and
-- grant public access.
                add ('create public synonym ' || table_name);
                add ('for ' || table_name);
                execute_immediate;
                add ('grant select, insert, update, delete');
                add ('on ' || table_name);
                add ('to public');
                execute_immediate;
-- To maintain the session_id information a pre-insert - per row trigger
-- will be created.
                add ('create trigger temp_' || table_name || '_bir');
                add ('before insert');
                add ('on temp_' || table_name);
                add ('for each row');
                add ('begin');
                add ('  :new.session_id := userenv (''sessionid'');');
                add ('end;');
                execute_immediate;
-- To register the usage of a temporary table for a specific session.
-- The procedure register has to be called in a pre-insert -
-- per statement trigger.
                add ('create trigger temp_' || table_name || '_bis');
                add ('before insert');
                add ('on temp_' || table_name);
                add ('begin');
                add ('  temp_table.register (''' || upper (table_name) ||
```

```
                        ''');');
                add ('end;');
                execute_immediate;
        end;

        procedure register (table_name in varchar2)is
--
--      Register the usage of temporary table in temp_table_catalog
--      This procedure may be called out of the pre-insert trigger
--      on the temporary table.
--
                dummy           varchar2(1);
                return_value    number;
                lock_id         number;
        begin
--  Check if we just registered the table
                if last_table = table_name then
                   return;
                end if;
                last_table := table_name;
-- Check if we have ever registered the table for our session
                begin
                        select 'x' into dummy
                        from   temp_table_catalog ttc
                        where  ttc.table_name = register.table_name
                        and    session_id = userenv ('sessionid');
                exception
                   when no_data_found then
-- If it is not registered, register the usage

                        insert into temp_table_catalog
                        values (userenv ('sessionid'), table_name);
-- and put out the share lock with a timeout of 5 seconds
                        if use_truncate then
                                lock_id := get_lock_id (table_name);
                                return_value :=
                                dbms_lock.request (lock_id,
                                                   dbms_lock.s_mode, 5,
FALSE);
                                if return_value not in (0, 4) then
                                        raise_application_error (-20100,
                                        'Unknown Error in DBMS_LOCK: ' ||
                                        to_char (return_value));
                                end if;
                        end if;
```

```
            end;
        end;
```

Index

S

T

U

V

W