



Integrated Cloud Applications & Platform Services



Oracle Database 12c R2: SQL Workshop II

Student Guide - Volume II

D80194GC20

Edition 2.0 | April 2017 | D98638

Learn more from Oracle University at education.oracle.com

Author

Apoorva Srinivas

**Technical Contributors
and Reviewers**

Nancy Greenberg

Suresh Rajan

Bryan Roberts

Sharath Bhujani

Graphic Designer

Kavya Bellur

Publishers

Asief Baig

Veena Narasimhan

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introduction

- Lesson Objectives 1-2
- Lesson Agenda 1-3
- Course Objectives 1-4
- Course Prerequisites 1-5
- Course Roadmap 1-6
- Lesson Agenda 1-10
- Tables Used in This Course 1-11
- Appendices and Practices Used in This Course 1-12
- Development Environments 1-13
- Introduction to Oracle Live SQL 1-14
- Lesson Agenda 1-15
- Review of Restricting Data 1-16
- Review of Sorting Data 1-17
- Review of SQL Functions 1-18
- Review of Single-Row Functions 1-19
- Review of Types of Group Functions 1-20
- Review of Using Subqueries 1-21
- Review of Managing Tables Using DML Statements 1-22
- Lesson Agenda 1-23
- Oracle Database SQL Documentation 1-24
- Additional Resources 1-25
- Summary 1-26
- Practice 1: Overview 1-27

2 Introduction to Data Dictionary Views

- Course Roadmap 2-2
- Objectives 2-3
- Lesson Agenda 2-4
- Why Data Dictionary? 2-5
- Data Dictionary 2-6
- Data Dictionary Structure 2-7
- How to Use Dictionary Views 2-9
- USER_OBJECTS and ALL_OBJECTS Views 2-10
- USER_OBJECTS View 2-11

Lesson Agenda 2-12
Table Information 2-13
Column Information 2-14
Constraint Information 2-16
USER_CONSTRAINTS: Example 2-17
Querying USER_CONS_COLUMNS 2-18
Lesson Agenda 2-19
Adding Comments to a Table 2-20
Quiz 2-21
Summary 2-22
Practice 2: Overview 2-23

3 Creating Sequences, Synonyms, and Indexes

Course Roadmap 3-2
Objectives 3-3
Lesson Agenda 3-4
E-Commerce Scenario 3-5
Database Objects 3-6
Referencing Another User's Tables 3-7
Sequences 3-8
CREATE SEQUENCE Statement: Syntax 3-9
Creating a Sequence 3-11
NEXTVAL and CURRVAL Pseudocolumns 3-12
Using a Sequence 3-14
SQL Column Defaulting Using a Sequence 3-15
Caching Sequence Values 3-16
Modifying a Sequence 3-17
Guidelines for Modifying a Sequence 3-18
Sequence Information 3-19
Lesson Agenda 3-20
Synonyms 3-21
Creating a Synonym for an Object 3-22
Creating and Removing Synonyms 3-23
Synonym Information 3-24
Lesson Agenda 3-25
Indexes 3-26
How Are Indexes Created? 3-27
Creating an Index 3-28
CREATE INDEX with the CREATE TABLE Statement 3-29
Function-Based Indexes 3-31
Creating Multiple Indexes on the Same Set of Columns 3-32

Creating Multiple Indexes on the Same Set of Columns: Example	3-33
Index Information	3-34
USER_INDEXES: Examples	3-35
Querying USER_IND_COLUMNS	3-36
Removing an Index	3-37
Quiz	3-38
Summary	3-39
Practice 3: Overview	3-40

4 Creating Views

Course Roadmap	4-2
Objectives	4-3
Lesson Agenda	4-4
Why Views?	4-5
Database Objects	4-6
What Is a View?	4-7
Advantages of Views	4-8
Simple Views and Complex Views	4-9
Lesson Agenda	4-10
Creating a View	4-11
Retrieving Data from a View	4-14
Modifying a View	4-15
Creating a Complex View	4-16
View Information	4-17
Lesson Agenda	4-18
Rules for Performing DML Operations on a View	4-19
Rules for Performing Modify Operations on a View	4-20
Rules for Performing Insert Operations Through a View	4-21
Using the WITH CHECK OPTION Clause	4-22
Denying DML Operations	4-23
Lesson Agenda	4-25
Removing a View	4-26
Quiz	4-27
Summary	4-28
Practice 4: Overview	4-29

5 Managing Schema Objects

Course Roadmap	5-2
Objectives	5-3
Lesson Agenda	5-4
Adding a Constraint Syntax	5-5

Adding a Constraint 5-6
Dropping a Constraint 5-7
Dropping a Constraint ONLINE 5-8
ON DELETE Clause 5-9
Cascading Constraints 5-10
Renaming Table Columns and Constraints 5-12
Disabling Constraints 5-13
Enabling Constraints 5-14
Constraint States 5-15
Deferring Constraints 5-16
Difference Between INITIALLY DEFERRED and INITIALLY IMMEDIATE 5-17
DROP TABLE ... PURGE 5-19
Lesson Agenda 5-20
Using Temporary Tables 5-21
Creating a Temporary Table 5-22
Lesson Agenda 5-23
External Tables 5-24
Creating a Directory for the External Table 5-25
Creating an External Table 5-27
Creating an External Table by Using ORACLE_LOADER 5-29
Querying External Tables 5-30
Creating an External Table by Using ORACLE_DATAPUMP: Example 5-31
Quiz 5-32
Summary 5-33
Practice 5: Overview 5-34

6 Retrieving Data by Using Subqueries

Course Roadmap 6-2
Objectives 6-3
Lesson Agenda 6-4
Retrieving Data by Using a Subquery as a Source 6-5
Lesson Agenda 6-7
Multiple-Column Subqueries 6-8
Column Comparisons 6-9
Pairwise Comparison Subquery 6-10
Nonpairwise Comparison Subquery 6-11
Lesson Agenda 6-12
Scalar Subquery Expressions 6-13
Scalar Subqueries: Examples 6-14
Lesson Agenda 6-15
Correlated Subqueries 6-16

Using Correlated Subqueries: Example 1	6-18
Using Correlated Subqueries: Example 2	6-19
Lesson Agenda	6-20
Using the EXISTS Operator	6-21
Find All Departments That Do Not Have Any Employees	6-23
Lesson Agenda	6-24
WITH Clause	6-25
WITH Clause: Example	6-26
Recursive WITH Clause	6-27
Recursive WITH Clause: Example	6-28
Quiz	6-29
Summary	6-30
Practice 6: Overview	6-31

7 Manipulating Data by Using Subqueries

Course Roadmap	7-2
Objectives	7-3
Lesson Agenda	7-4
Using Subqueries to Manipulate Data	7-5
Lesson Agenda	7-6
Inserting by Using a Subquery as a Target	7-7
Lesson Agenda	7-9
Using the WITH CHECK OPTION Keyword on DML Statements	7-10
Lesson Agenda	7-12
Correlated UPDATE	7-13
Using Correlated UPDATE	7-14
Correlated DELETE	7-16
Using Correlated DELETE	7-17
Summary	7-18
Practice 7: Overview	7-19

8 Controlling User Access

Course Roadmap	8-2
Objectives	8-3
Lesson Agenda	8-4
Controlling User Access	8-5
Privileges	8-6
System Privileges	8-7
Creating Users	8-8
User System Privileges	8-9
Granting System Privileges	8-10

Lesson Agenda	8-11
What is a Role?	8-12
Creating and Granting Privileges to a Role	8-13
Changing Your Password	8-14
Lesson Agenda	8-15
Object Privileges	8-16
Granting Object Privileges	8-18
Passing On Your Privileges	8-19
Confirming Granted Privileges	8-20
Lesson Agenda	8-21
Revoking Object Privileges	8-22
Quiz	8-24
Summary	8-25
Practice 8: Overview	8-26

9 Manipulating Data Using Advanced Queries

Course Roadmap	9-2
Objectives	9-3
Lesson Agenda	9-4
Explicit Default Feature: Overview	9-5
Using Explicit Default Values	9-6
Lesson Agenda	9-7
E-Commerce Scenario	9-8
Multitable INSERT Statements: Overview	9-9
Types of Multitable INSERT Statements	9-11
Multitable INSERT Statements	9-12
Unconditional INSERT ALL	9-14
Conditional INSERT ALL: Example	9-15
Conditional INSERT ALL	9-16
Conditional INSERT FIRST: Example	9-18
Conditional INSERT FIRST	9-19
Pivoting INSERT	9-20
Lesson Agenda	9-23
MERGE Statement	9-24
MERGE Statement Syntax	9-25
Merging Rows: Example	9-26
Lesson Agenda	9-29
FLASHBACK TABLE Statement	9-30
Using the FLASHBACK TABLE Statement	9-32
Lesson Agenda	9-33
Tracking Changes in Data	9-34

Flashback Query: Example 9-35
Flashback Version Query: Example 9-36
VERSIONS BETWEEN Clause 9-37
Quiz 9-38
Summary 9-40
Practice 9: Overview 9-41

10 Managing Data in Different Time Zones

Course Roadmap 10-2
Objectives 10-3
Lesson Agenda 10-4
E-Commerce Scenario 10-5
Time Zones 10-6
TIME_ZONE Session Parameter 10-7
CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP 10-8
Comparing Date and Time in a Session's Time Zone 10-9
DBTIMEZONE and SESSIONTIMEZONE 10-11
TIMESTAMP Data Types 10-12
TIMESTAMP Fields 10-13
Difference Between DATE and TIMESTAMP 10-14
Comparing TIMESTAMP Data Types 10-15
Lesson Agenda 10-16
INTERVAL Data Types 10-17
INTERVAL Fields 10-18
INTERVAL YEAR TO MONTH: Example 10-19
INTERVAL DAY TO SECOND Data Type: Example 10-21
Lesson Agenda 10-22
EXTRACT 10-23
TZ_OFFSET 10-24
FROM_TZ 10-26
TO_TIMESTAMP 10-27
TO_YMINTERVAL 10-28
TO_DSINTERVAL 10-29
Daylight Saving Time (DST) 10-30
Quiz 10-32
Summary 10-33
Practice 10: Overview 10-34

11 Oracle Cloud Overview

Lesson Objectives 11-2
Lesson Agenda 11-3

Introduction to Oracle Cloud	11-4
Oracle Cloud Services	11-5
Cloud Deployment Models	11-6
Lesson Agenda	11-7
Evolving from On-premises to Exadata Express	11-8
What is Exadata Express?	11-9
Exadata Express for Users	11-10
Exadata Express for Developers	11-11
Getting Started with Exadata Express	11-12
Oracle Exadata Express Cloud Service	11-13
Getting Started with Exadata Express	11-14
Managing Exadata	11-15
Service Console	11-16
Web Access through Service Console	11-17
Client Access Configuration through Service Console	11-18
Database Administration through Service Console	11-19
SQL Workshop	11-20
Connecting through Database Clients	11-22
Enabling SQL*Net Access for Client Applications	11-23
Downloading Client Credentials	11-24
Connecting Oracle SQL Developer	11-25
Connecting Oracle SQLcl	11-26
Summary	11-27

A Table Descriptions

B Using SQL Developer

Objectives	B-2
What is Oracle SQL Developer?	B-3
Specifications of SQL Developer	B-4
SQL Developer 3.2 Interface	B-5
Creating a Database Connection	B-7
Browsing Database Objects	B-10
Displaying the Table Structure	B-11
Browsing Files	B-12
Creating a Schema Object	B-13
Creating a New Table: Example	B-14
Using SQL Worksheet	B-15
Executing SQL Statements	B-19
Saving SQL Scripts	B-20
Executing Saved Script Files: Method 1	B-21

Executing Saved Script Files: Method 2	B-22
Formatting the SQL Code	B-23
Using Snippets	B-24
Using Snippets: Example	B-25
Using the Recycle Bin	B-26
Debugging Procedures and Functions	B-27
Database Reporting	B-28
Creating a User-Defined Report	B-29
Search Engines and External Tools	B-30
Setting Preferences	B-31
Resetting the SQL Developer Layout	B-33
Data Modeler in SQL Developer	B-34
Summary	B-35

C Using SQL*Plus

Objectives	C-2
SQL and SQL*Plus Interaction	C-3
SQL Statements Versus SQL*Plus Commands	C-5
SQL*Plus: Overview	C-6
Logging in to SQL*Plus	C-7
Displaying the Table Structure	C-8
SQL*Plus Editing Commands	C-10
Using LIST, n, and APPEND	C-12
Using the CHANGE Command	C-13
SQL*Plus File Commands	C-14
Using the SAVE and START Commands	C-15
SERVERROUTPUT Command	C-16
Using the SQL*Plus SPOOL Command	C-17
Using the AUTOTRACE Command	C-18
Summary	C-19

D Commonly Used SQL Commands

Objectives	D-2
Basic SELECT Statement	D-3
SELECT Statement	D-4
WHERE Clause	D-5
ORDER BY Clause	D-6
GROUP BY Clause	D-7
Data Definition Language	D-8
CREATE TABLE Statement	D-9
ALTER TABLE Statement	D-10

DROP TABLE Statement	D-11
GRANT Statement	D-12
Privilege Types	D-13
REVOKE Statement	D-14
TRUNCATE TABLE Statement	D-15
Data Manipulation Language	D-16
INSERT Statement	D-17
UPDATE Statement Syntax	D-18
DELETE Statement	D-19
Transaction Control Statements	D-20
COMMIT Statement	D-21
ROLLBACK Statement	D-22
SAVEPOINT Statement	D-23
Joins	D-24
Types of Joins	D-25
Qualifying Ambiguous Column Names	D-26
Natural Join	D-28
Equijoins	D-29
Retrieving Records with Equijoins	D-30
Additional Search Conditions Using the AND and WHERE Operators	D-31
Retrieving Records with Nonequijoins	D-32
Retrieving Records by Using the USING Clause	D-33
Retrieving Records by Using the ON Clause	D-34
Left Outer Join	D-35
Right Outer Join	D-36
Full Outer Join	D-37
Self-Join: Example	D-38
Cross Join	D-39
Summary	D-40

E Generating Reports by Grouping Related Data

Objectives	E-2
Review of Group Functions	E-3
Review of the GROUP BY Clause	E-4
Review of the HAVING Clause	E-5
GROUP BY with ROLLUP and CUBE Operators	E-6
ROLLUP Operator	E-7
ROLLUP Operator: Example	E-8
CUBE Operator	E-9
CUBE Operator: Example	E-10
GROUPING Function	E-11

GROUPING Function: Example E-12
GROUPING SETS E-13
GROUPING SETS: Example E-15
Composite Columns E-17
Composite Columns: Example E-19
Concatenated Groupings E-21
Concatenated Groupings: Example E-22
Summary E-23

F Hierarchical Retrieval

Objectives F-2
Sample Data from the EMPLOYEES Table F-3
Natural Tree Structure F-4
Hierarchical Queries F-5
Walking the Tree F-6
Walking the Tree: From the Bottom Up F-8
Walking the Tree: From the Top Down F-9
Ranking Rows with the LEVEL Pseudocolumn F-10
Formatting Hierarchical Reports Using LEVEL and LPAD F-11
Pruning Branches F-13
Summary F-14

G Writing Advanced Scripts

Objectives G-2
Using SQL to Generate SQL G-3
Creating a Basic Script G-4
Controlling the Environment G-5
The Complete Picture G-6
Dumping the Contents of a Table to a File G-7
Generating a Dynamic Predicate G-9
Summary G-11

H Oracle Database Architectural Components

Objectives H-2
Oracle Database Architecture: Overview H-3
Oracle Database Server Structures H-4
Connecting to the Database H-5
Interacting with an Oracle Database H-6
Oracle Memory Architecture H-8
Process Architecture H-10
Database Writer Process H-12

Log Writer Process	H-13
Checkpoint Process	H-14
System Monitor Process	H-15
Process Monitor Process	H-16
Oracle Database Storage Architecture	H-17
Logical and Physical Database Structures	H-19
Processing a SQL Statement	H-21
Processing a Query	H-22
Shared Pool	H-23
Database Buffer Cache	H-25
Program Global Area (PGA)	H-26
Processing a DML Statement	H-27
Redo Log Buffer	H-29
Rollback Segment	H-30
COMMIT Processing	H-31
Summary of the Oracle Database Architecture	H-33
Summary	H-34

I Regular Expression Support

Objectives	I-2
What Are Regular Expressions?	I-3
Benefits of Using Regular Expressions	I-4
Using the Regular Expressions Functions and Conditions in SQL and PL/SQL	I-5
What are Metacharacters?	I-6
Using Metacharacters with Regular Expressions	I-7
Regular Expressions Functions and Conditions: Syntax	I-9
Performing a Basic Search by Using the REGEXP_LIKE Condition	I-10
Replacing Patterns by Using the REGEXP_REPLACE Function	I-11
Finding Patterns by Using the REGEXP_INSTR Function	I-12
Extracting Substrings by Using the REGEXP_SUBSTR Function	I-13
Subexpressions	I-14
Using Subexpressions with Regular Expression Support	I-15
Why Access the nth Subexpression?	I-16
REGEXP_SUBSTR: Example	I-17
Using the REGEXP_COUNT Function	I-18
Regular Expressions and Check Constraints: Examples	I-19
Quiz	I-20
Summary	I-21



10

Managing Data in Different Time Zones

ORACLE

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Course Roadmap

Lesson 1: Introduction

Unit 1: Views, Sequences,
Synonyms and Indexes

Unit 2: Managing Database
Objects and Subqueries

Unit 3: User Access

**Unit 4: Advanced
Queries**

▶ Lesson 9: Manipulating Data Using Advanced
Queries

▶ **Lesson 10: Manipulating Data in Different
Time Zones**

You are here!

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Use data types similar to DATE that store fractional seconds and track time zones
- Use data types that store the difference between two datetime values
- Use the following datetime functions:
 - CURRENT_DATE
 - CURRENT_TIMESTAMP
 - LOCALTIMESTAMP
 - DBTIMEZONE
 - SESSIONTIMEZONE
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In this lesson, you learn how to use data types similar to DATE that store fractional seconds and track time zones. This lesson addresses some of the datetime functions available in the Oracle database.

Lesson Agenda

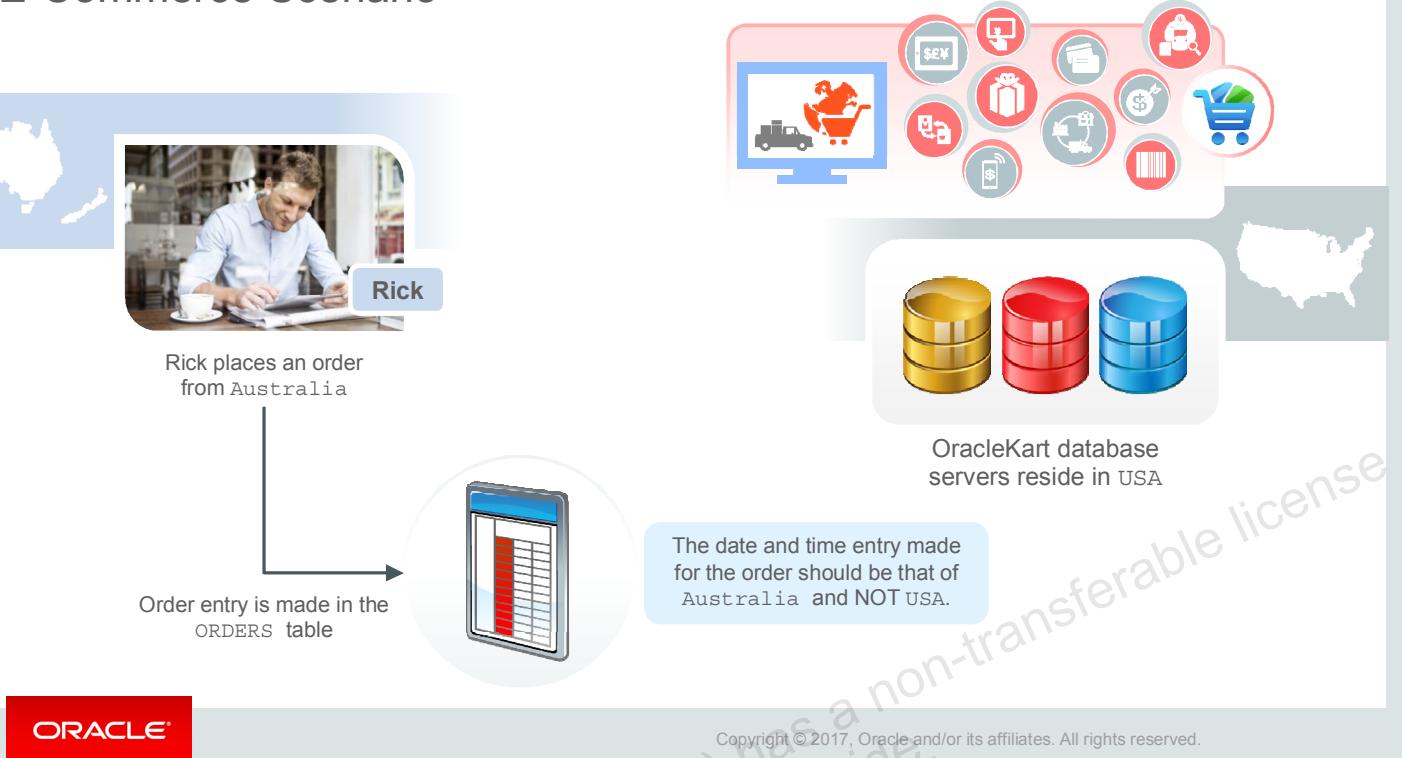
- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

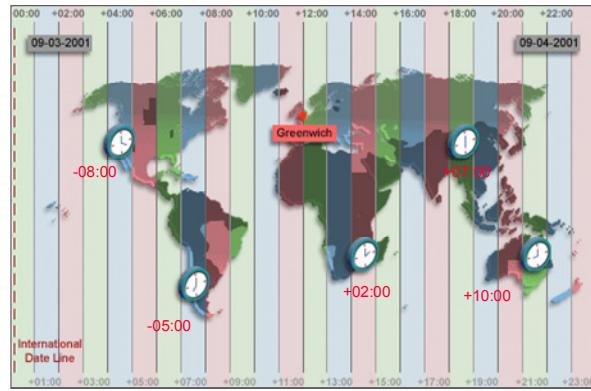
E-Commerce Scenario



Recall the OracleKart e-commerce company that is a global online shopping website. Their data center is located in the US. The customers of this site are spread out all around the globe. When a customer, Rick, who lives in Australia, places an order, the date and time of the order is recorded as per the US time zone. Due to this, the order delivery date is miscalculated and the order is delayed. How can this be avoided?

Ideally, the order date and time should be the local time of the place from which the customer placed the order (in this case, Australia). Nonetheless, if the time zone is not set, the date and time will be set to that of the server where the database resides. For example, if the database is in the US, then the date and time entry will be in the US time zone. This will cause problems while calculating the delivery time and returns for the customer. Hence, time zones are very useful for the e-commerce industry. Let us learn more about the different time zones and how to use them.

Time Zones



The image represents the time for each time zone when Greenwich time is 12:00.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Let us look into some of the basics of time zones:

The hours of the day are measured by the turning of the earth. The time of day at any particular moment depends on where you are. When it is noon in Greenwich, England, it is midnight along the International Date Line.

The earth is divided into 24 time zones, one for each hour of the day. The time along the prime meridian in Greenwich, England, is known as Greenwich Mean Time (GMT). GMT is now known as Coordinated Universal Time (UTC).

UTC is the time standard against which all other time zones in the world are referenced. It is the same all year round and is not affected by summer time or daylight saving time.

The meridian line is an imaginary line that runs from the North Pole to the South Pole. It is known as zero longitude and it is the line from which all other lines of longitude are measured. All time is measured relative to UTC and all places have a latitude (their distance north or south of the equator) and a longitude (their distance east or west of the Greenwich meridian).

TIME_ZONE Session Parameter

TIME_ZONE may be set to:

- An absolute offset
- Database time zone
- OS local time zone
- A named region

```
ALTER SESSION SET TIME_ZONE = '-05:00';
ALTER SESSION SET TIME_ZONE = dbtimezone;
ALTER SESSION SET TIME_ZONE = local;
ALTER SESSION SET TIME_ZONE = 'America/New_York';
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP

- CURRENT_DATE:
 - Returns the current date from the user session
 - Has a data type of DATE
- CURRENT_TIMESTAMP:
 - Returns the current date and time from the user session
 - Has a data type of TIMESTAMP WITH TIME ZONE
- LOCALTIMESTAMP:
 - Returns the current date and time from the user session
 - Has a data type of TIMESTAMP



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The CURRENT_DATE and CURRENT_TIMESTAMP functions return the current date and current time stamp, respectively. The data type of CURRENT_DATE is DATE. The data type of CURRENT_TIMESTAMP is TIMESTAMP WITH TIME ZONE.

The values returned display the time zone displacement of the SQL session executing the functions. Here, the time zone displacement is the difference (in hours and minutes) between local time and UTC. The TIMESTAMP WITH TIME ZONE data type has the format:

TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE

where fractional_seconds_precision optionally specifies the number of digits in the fractional part of the SECOND datetime field and can be a number in the range 0 through 9. The default is 6.

The LOCALTIMESTAMP function returns the current date and time in the session time zone. The difference between LOCALTIMESTAMP and CURRENT_TIMESTAMP is that LOCALTIMESTAMP returns a TIMESTAMP value, whereas CURRENT_TIMESTAMP returns a TIMESTAMP WITH TIME ZONE value.

These functions are national language support (NLS)-sensitive—that is, the results will be in the current NLS calendar and datetime formats.

Note: The SYSDATE function returns the current date and time as a DATE data type. You learned how to use the SYSDATE function in the course titled *Oracle Database: SQL Workshop I*.

Comparing Date and Time in a Session's Time Zone

The TIME_ZONE parameter is set to -5:00 and then SELECT statements for each date and time are executed to compare differences.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
ALTER SESSION SET TIME_ZONE = '-5:00';

SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL; 1

SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL; 2

SELECT SESSIONTIMEZONE, LOCALTIMESTAMP FROM DUAL; 3
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Comparing Date and Time in a Session's Time Zone

Results of queries:

Session altered.

SESSIONTIMEZONE	CURRENT_DATE
1 -05:00	28-AUG-2016 18:14:49

1

SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 -05:00	28-AUG-16 06.15.59.648595000 PM -05:00

2

SESSIONTIMEZONE	LOCALTIMESTAMP
1 -05:00	28-AUG-16 06.16.59.026930000 PM

3

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Observe the results of the queries run from the previous slide:

1. The CURRENT_DATE function returns the current date in the session's time zone.
2. The CURRENT_TIMESTAMP function returns the current date and time in the session's time zone as a value of the data type TIMESTAMP WITH TIME ZONE.
3. The LOCALTIMESTAMP function returns the current date and time in the session's time zone.

Note: The code example output may vary depending on when the command is run.

DBTIMEZONE and SESSIONTIMEZONE

- Display the value of the database time zone:

```
SELECT DBTIMEZONE FROM DUAL;
```

DBTIMEZONE
1 +00:00

- Display the value of the session's time zone:

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

SESSIONTIMEZONE
1 -05:00



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

If you are the DBA, you can set the database's default time zone by specifying the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If you omit, the default database time zone is the operating system time zone. Note that the database time zone cannot be changed for a session with an `ALTER SESSION` statement.

The `DBTIMEZONE` function returns the value of the database time zone. The return type is a time zone offset (a character type in the format: '`[+ | -] TZH:TZM`') or a time zone region name, depending on how the user specified the database time zone value in the most recent `CREATE DATABASE` or `ALTER DATABASE` statement. The example in the slide shows that the database time zone is set to "`+00:00`," as the `TIME_ZONE` parameter is in the format:

```
TIME_ZONE = ' [+ | -] hh:mm'
```

The `SESSIONTIMEZONE` function returns the value of the current session's time zone. The return type is a time zone offset (a character type in the format '`[+ | -] TZH:TZM`') or a time zone region name, depending on how the user specified the session time zone value in the most recent `ALTER SESSION` statement. The example in the slide shows that the session time zone is offset to UTC by '`-5:00`' hours.

Observe that the database time zone is different from the current session's time zone.

TIMESTAMP Data Types

Data Type	Fields
TIMESTAMP	Year, Month, Day, Hour, Minute, Second with fractional seconds
TIMESTAMP WITH TIME ZONE	Same as the TIMESTAMP data type; also includes: TIMEZONE_HOUR, and TIMEZONE_MINUTE or TIMEZONE_REGION
TIMESTAMP WITH LOCAL TIME ZONE	Same as the TIMESTAMP data type; also includes a time zone offset in its value



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The TIMESTAMP data type is an extension of the DATE data type.

`TIMESTAMP (fractional_seconds_precision)`

This data type contains the year, month, and day values of date, as well as hour, minute, and second values of time, where `fractional_seconds_precision` is the number of digits in the fractional part of the SECOND datetime field. The accepted values of significant `fractional_seconds_precision` are 0 through 9. The default is 6.

`TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE`

This data type contains all values of TIMESTAMP as well as time zone displacement value.

`TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE`

This data type contains all values of TIMESTAMP, with the following exceptions:

- Data is normalized to the database time zone when it is stored in the database.
- When the data is retrieved, users see the data in the session time zone.

TIMESTAMP Fields

Datetime Field	Valid Values
YEAR	-4712 to 9999 (excluding year 0)
MONTH	01 to 12
DAY	01 to 31
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59.9(N) where 9(N) is precision
TIMEZONE_HOUR	-12 to 14
TIMEZONE_MINUTE	00 to 59



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Difference Between DATE and TIMESTAMP

A

```
-- when hire_date is of  
type DATE  
  
SELECT hire_date  
FROM emp4;
```

HIRE_DATE
1 17-JUN-11
2 21-SEP-09
3 13-JAN-09
4 03-JAN-14
5 21-MAY-15
6 25-JUN-13
7 05-FEB-14
8 07-FEB-15
9 17-AUG-10
10 16-AUG-10

...

ORACLE

B

```
ALTER TABLE emp4  
MODIFY hire_date TIMESTAMP;  
  
SELECT hire_date  
FROM emp4;
```

HIRE_DATE
1 17-JUN-11 12.00.00.000000000 AM
2 21-SEP-09 12.00.00.000000000 AM
3 13-JAN-09 12.00.00.000000000 AM
4 03-JAN-14 12.00.00.000000000 AM
5 21-MAY-15 12.00.00.000000000 AM
6 25-JUN-13 12.00.00.000000000 AM
7 05-FEB-14 12.00.00.000000000 AM
8 07-FEB-15 12.00.00.000000000 AM
9 17-AUG-10 12.00.00.000000000 AM
10 16-AUG-10 12.00.00.000000000 AM

...

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Comparing TIMESTAMP Data Types

```
CREATE TABLE web_orders  
  (order_date TIMESTAMP WITH TIME ZONE,  
   delivery_time TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO web_orders values  
  (current_date, current_timestamp + 2);
```

```
SELECT * FROM web_orders;
```

ORDER_DATE	DELIVERY_TIME
28-AUG-16 06.49.27.000000000 PM -05:00	30-AUG-16 06.49.27.000000000 PM



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

INTERVAL Data Types

- You can use INTERVAL data types to store the difference between two datetime values.
- There are two classes of intervals:
 - Year-month
 - Day-time
- The precision of the interval is:
 - The actual subset of fields that constitutes an interval
 - Specified in the interval qualifier

Data Type	Fields
INTERVAL YEAR TO MONTH	Year, Month
INTERVAL DAY TO SECOND	Days, Hour, Minute, Second with fractional seconds



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

INTERVAL data types are used to store the difference between two datetime values. There are two classes of intervals: year-month intervals and day-time intervals.

A year-month interval is made up of a contiguous subset of fields of YEAR and MONTH, whereas a day-time interval is made up of a contiguous subset of fields consisting of DAY, HOUR, MINUTE, and SECOND. The actual subset of fields that constitute an interval is called the precision of the interval and is specified in the interval qualifier. Because the number of days in a year is calendar-dependent, the year-month interval is NLS-dependent, whereas day-time interval is NLS-independent.

The interval qualifier contains a leading field and may contain a trailing field. In case the trailing field is SECOND, it may also specify the fractional seconds precision, which is the number of digits in the fractional part of the SECOND value. If not specified, the default value for leading field precision is 2 digits and the default value for fractional seconds precision is 6 digits.

INTERVAL YEAR (year_precision) TO MONTH

This data type stores a period of time in years and months, where `year_precision` is the number of digits in the YEAR datetime field. The accepted values are 0 through 9. The default is 6.

INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)

This data type stores a period of time in days, hours, minutes, and seconds, where `day_precision` is the maximum number of digits in the DAY datetime field (accepted values are 0 through 9; the default is 2) and `fractional_seconds_precision` is the number of digits in the fractional part of the SECOND field (accepted values are 0 through 9; the default is 6).

INTERVAL Fields

INTERVAL Field	Valid Values for Interval
YEAR	Any positive or negative integer
MONTH	00 to 11
DAY	Any positive or negative integer
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59.9(N) where 9(N) is precision



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

INTERVAL YEAR TO MONTH: Example

```
CREATE TABLE warranty
(prod_id number, warranty_time INTERVAL YEAR(3) TO MONTH);
INSERT INTO warranty VALUES (123, INTERVAL '8' MONTH);
INSERT INTO warranty VALUES (155, INTERVAL '200' YEAR(3));
INSERT INTO warranty VALUES (678, '200-11');
SELECT * FROM warranty;
```

	PROD_ID	WARRANTY_TIME
1	123	0-8
2	155	200-0
3	678	200-11



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

INTERVAL YEAR TO MONTH stores a period of time by using the YEAR and MONTH datetime fields. Specify INTERVAL YEAR TO MONTH as follows:

INTERVAL YEAR [(year_precision)] TO MONTH

where year_precision is the number of digits in the YEAR datetime field. The default value of year_precision is 2.

Restriction: The leading field must be more significant than the trailing field. For example, INTERVAL '0-1' MONTH TO YEAR is not valid.

Examples:

- INTERVAL '123-2' YEAR(3) TO MONTH
Indicates an interval of 123 years, 2 months
- INTERVAL '123' YEAR(3)
Indicates an interval of 123 years, 0 months
- INTERVAL '300' MONTH(3)
Indicates an interval of 300 months
- INTERVAL '123' YEAR
Returns an error because the default precision is 2, and '123' has 3

The Oracle database supports two interval data types: INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. For interval literals, the system also recognizes other American National Standards Institute (ANSI) interval types, such as INTERVAL '2' YEAR or INTERVAL '10' HOUR. In these cases, each interval is converted to one of the two supported types.

In the example in the slide, a WARRANTY table is created, which contains a warranty_time column that takes the INTERVAL YEAR (3) TO MONTH data type. Different values are inserted into it to indicate years and months for various products. When these rows are retrieved from the table, you see a year value separated from the month value by a (-).

INTERVAL DAY TO SECOND Data Type: Example

```
CREATE TABLE lab  
( exp_id number, test_time INTERVAL DAY(2) TO SECOND );  
  
INSERT INTO lab VALUES (100012, '90 00:00:00');  
INSERT INTO lab VALUES (56098,  
                      INTERVAL '6 03:30:16' DAY TO SECOND);
```

```
SELECT * FROM lab;
```

EXP_ID	TEST_TIME
1	100012 90 0:0:0.0
2	56098 6 3:30:16.0



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

EXTRACT

- Display all employees who were hired after 2007.

```
SELECT last_name, employee_id, hire_date
  FROM employees
 WHERE EXTRACT(YEAR FROM TO_DATE(hire_date, 'DD-MON-RR')) > 2007
 ORDER BY hire_date;
```

- Display the MONTH component from the HIRE_DATE for those employees whose MANAGER_ID is 100.

```
SELECT last_name, hire_date,
       EXTRACT(MONTH FROM HIRE_DATE)
  FROM employees
 WHERE manager_id = 100;
```

LAST_NAME	HIRE_DATE	EXTRACT(MONTHFROMHIRE_DATE)
Kochhar	21-SEP-09	9
De Haan	13-JAN-09	1
Raphaely	07-DEC-10	12
Weiss	18-JUL-12	7
Fripp	10-APR-13	4
...		

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The EXTRACT expression extracts and returns the value of a specified datetime field from a datetime or interval value expression. You can extract any of the components mentioned in the following syntax by using the EXTRACT function. The syntax of the EXTRACT function is:

```
SELECT EXTRACT( { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
                  | TIMEZONE_HOUR
                  | TIMEZONE_MINUTE
                  | TIMEZONE_REGION
                  | TIMEZONE_ABBR } )
  FROM { expr } )
```

When you extract a TIMEZONE_REGION or TIMEZONE_ABBR (abbreviation), the value returned is a string containing the appropriate time zone name or abbreviation. When you extract any of the other values, the value returned is a date in the Gregorian calendar. When extracting from a datetime with a time zone value, the value returned is in UTC.

In the first example in the slide, the EXTRACT function is used to select all employees who were hired after 2007.

In the second example, the EXTRACT function is used to extract the MONTH from the HIRE_DATE column of the EMPLOYEES table for those employees who report to the manager whose EMPLOYEE_ID is 100.

TZ_OFFSET

Display the time zone offset for the 'US/Eastern', 'Canada/Yukon', and 'Europe/London' time zones:

```
SELECT TZ_OFFSET('US/Eastern'),
       TZ_OFFSET('Canada/Yukon'),
       TZ_OFFSET('Europe/London')
  FROM DUAL;
```

	TZ_OFFSET('US/EASTERN')	TZ_OFFSET('CANADA/YUKON')	TZ_OFFSET('EUROPE/LONDON')
1	-04:00	-07:00	+01:00



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `TZ_OFFSET` function returns the time zone offset corresponding to the value entered. The return value is dependent on the time when the statement is executed. For example, if the `TZ_OFFSET` function returns a value `-08:00`, this value indicates that the time zone for which the command was executed is eight hours behind UTC. You can enter a valid time zone name, a time zone offset from UTC (which simply returns itself), or the keyword `SESSIONTIMEZONE` or `DBTIMEZONE`. The syntax of the `TZ_OFFSET` function is:

```
TZ_OFFSET({ 'time_zone_name' | '{ + | - } hh : mi'
           | SESSIONTIMEZONE
           | DBTIMEZONE
         })
```

The Fold Motor Company has its headquarters in Michigan, USA, which is in the US/Eastern time zone. The company president, Mr. Fold, wants to conduct a conference call with the vice president of the Canadian operations and the vice president of the European operations, who are in the Canada/Yukon and Europe/London time zones, respectively. Mr. Fold wants to find out the time in each of these places to make sure that his senior management will be available to attend the meeting. His secretary, Mr. Scott, helps by issuing the queries shown in the example and gets the following results:

- The 'US/Eastern' time zone is four hours behind UTC.
- The 'Canada/Yukon' time zone is seven hours behind UTC.
- The 'Europe/London' time zone is one hour ahead of UTC.

For a listing of valid time zone name values, you can query the V\$TIMEZONE_NAMES dynamic performance view.

```
SELECT * FROM V$TIMEZONE_NAMES;
```

TZNAME	TZABBREV	CON_ID
1 Africa/Abidjan	LMT	0
2 Africa/Abidjan	GMT	0
3 Africa/Accra	LMT	0
4 Africa/Accra	GMT	0
5 Africa/Accra	GHST	0

...

FROM_TZ

Display the TIMESTAMP value '2000-07-12 08:00:00' as a TIMESTAMP WITH TIME ZONE value for the 'Australia/North' time zone region.

```
SELECT FROM_TZ(TIMESTAMP  
  '2000-07-12 08:00:00', 'Australia/North')  
FROM DUAL;
```

```
FROM_TZ(TIMESTAMP'2000-07-1208:00:00','AUSTRALIA/NORTH')  
1 12-JUL-00 08.00.00.000000000 AM AUSTRALIA/NORTH
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `FROM_TZ` function converts a `TIMESTAMP` value to a `TIMESTAMP WITH TIME ZONE` value.

The syntax of the `FROM_TZ` function is as follows:

`FROM_TZ(timestamp_value, time_zone_value)`

where `time_zone_value` is a character string in the format '`TZH:TZM`' or a character expression that returns a string in `TZR` (time zone region) with an optional `TZD` format. `TZD` is an abbreviated time zone string with daylight saving information. `TZR` represents the time zone region in datetime input strings. Examples are '`Australia/North`', '`PST`' for US/Pacific standard time, '`PDT`' for US/Pacific daylight time, and so on.

The example in the slide converts a `TIMESTAMP` value to `TIMESTAMP WITH TIME ZONE`.

Note: To see a listing of valid values for the `TZR` and `TZD` format elements, query the `V$TIMEZONE_NAMES` dynamic performance view.

TO_TIMESTAMP

Display the character string '2016-03-06 11:00:00' as a TIMESTAMP value:

```
SELECT TO_TIMESTAMP ('2016-03-06 11:00:00',
                      'YYYY-MM-DD HH:MI:SS')
FROM DUAL;
```

SQL TO_TIMESTAMP('2016-03-0611:00:00','YYYY-MM-DDHH:MI:SS')
1 06-MAR-16 11.00.00.000000000 AM



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `TO_TIMESTAMP` function converts a string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the TIMESTAMP data type. The syntax of the `TO_TIMESTAMP` function is:

```
TO_TIMESTAMP(char [, fmt [, 'nlsparam' ] ])
```

The optional `fmt` specifies the format of `char`. If you omit `fmt`, the string must be in the default format of the TIMESTAMP data type. The optional `nlsparam` specifies the language in which month and day names, and abbreviations, are returned. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

If you omit `nlsparams`, this function uses the default date language for your session.

The example in the slide converts a character string to a value of TIMESTAMP.

Note: You use the `TO_TIMESTAMP_TZ` function to convert a string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the TIMESTAMP WITH TIME ZONE data type. For more information about this function, see *Oracle Database SQL Language Reference* for Oracle Database 12c.

TO_YMINTERVAL

Display a date that is one year and two months after the hire date for the employees working in the department with the DEPARTMENT_ID 20.

```
SELECT hire_date,
       hire_date + TO_YMINTERVAL('01-02') AS
       HIRE_DATE_YMININTERVAL
  FROM employees
 WHERE department_id = 20;
```

HIRE_DATE	HIRE_DATE_YMININTERVAL
1 17-FEB-12	17-APR-13
2 17-AUG-13	17-OCT-14



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The TO_YMINTERVAL function converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL YEAR TO MONTH data type. The INTERVAL YEAR TO MONTH data type stores a period of time using the YEAR and MONTH datetime fields. The format of INTERVAL YEAR TO MONTH is as follows:

INTERVAL YEAR [(year_precision)] TO MONTH

where year_precision is the number of digits in the YEAR datetime field. The default value of year_precision is 2.

The syntax of the TO_YMINTERVAL function is:

TO_YMINTERVAL (char)

where char is the character string to be converted.

The example in the slide calculates a date that is one year and two months after the hire date for the employees working in the department with the DEPARTMENT_ID 20 of the EMPLOYEES table.

TO_DSINTERVAL

Display a date that is 100 days and 10 hours after the hire date for all the employees.

```
SELECT last_name,
       TO_CHAR(hire_date, 'mm-dd-yy:hh:mi:ss') hire_date,
       TO_CHAR(hire_date +
               TO_DSINTERVAL('100 10:00:00'),
               'mm-dd-yy:hh:mi:ss') hiredate2
  FROM employees;
```

LAST_NAME	HIRE_DATE	HIREDATE2
1 King	06-17-11:12:00:00	09-25-11:10:00:00
2 Kochhar	09-21-09:12:00:00	12-30-09:10:00:00
3 De Haan	01-13-09:12:00:00	04-23-09:10:00:00
4 Hunold	01-03-14:12:00:00	04-13-14:10:00:00
5 Ernst	05-21-15:12:00:00	08-29-15:10:00:00
6 Austin	06-25-13:12:00:00	10-03-13:10:00:00

...

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Daylight Saving Time (DST)

- Start of Daylight Saving:
 - Time jumps from 01:59:59 AM to 03:00:00 AM.
 - Values from 02:00:00 AM to 02:59:59 AM are not valid.
- End of Daylight Saving:
 - Time jumps back from 02:00:00 AM to 01:00:01 AM.
 - Values from 01:00:01 AM to 02:00:00 AM are ambiguous because they are visited twice.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Most western nations advance the clock ahead one hour during the summer months. This period is called daylight saving time (DST). The DST lasts from the start of Daylight Saving to the end of Daylight Saving in most of the US, Mexico, and Canada. The nations of the European Union observe DST, but they call it the summer time period. Europe's summer time period begins a week earlier than its North American counterpart, but ends at the same time.

The Oracle database automatically determines, for any given time zone region, whether the DST is in effect and returns local time values accordingly. The datetime value is sufficient for the Oracle database to determine whether daylight saving time is in effect for a given region in all cases except boundary cases.

A boundary case occurs during the period when the DST goes into or out of effect. For example, in the US/Eastern region, when DST goes into effect, the time changes from 01:59:59 AM to 03:00:00 AM. The one-hour interval between 02:00:00 AM and 02:59:59 AM does not exist. When daylight saving time goes out of effect, the time changes from 02:00:00 AM back to 01:00:01 AM, and the one-hour interval between 01:00:01 AM and 02:00:00 AM is repeated.

ERROR_ON_OVERLAP_TIME

The `ERROR_ON_OVERLAP_TIME` is a session parameter to notify the system to issue an error when it encounters a datetime that occurs in the overlapped period and no time zone abbreviation was specified to distinguish the period.

For example, the DST ends on October 31, at 02:00:01 AM. The overlapped periods are:

- 10/31/2016 01:00:01 AM to 10/31/2016 02:00:00 AM (EDT)
- 10/31/2016 01:00:01 AM to 10/31/2016 02:00:00 AM (EST)

If you input a datetime string that occurs in one of these two periods, you need to specify the time zone abbreviation (for example, EDT or EST) in the input string for the system to determine the period.

Without this time zone abbreviation, the system does the following:

If the `ERROR_ON_OVERLAP_TIME` parameter is `FALSE`, it assumes that the input time is standard time (for example, EST). Otherwise, an error is raised

Quiz



The TIME_ZONE session parameter may be set to:

- a. A relative offset
- b. Database time zone
- c. OS local time zone
- d. A named region



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to use:

- Data types similar to DATE that store fractional seconds and track time zones
- Data types that store the difference between two datetime values
- The following datetime functions:
 - CURRENT_DATE
 - CURRENT_TIMESTAMP
 - LOCALTIMESTAMP
 - DBTIMEZONE
 - SESSIONTIMEZONE
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Practice 10: Overview

This practice focuses on using the datetime functions.



In this practice, you display time zone offsets, CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP. You also set time zones and use the EXTRACT function.



Unauthorized reproduction or distribution prohibited. Copyright © 2019, Oracle and/or its affiliates.

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Oracle Cloud Overview

Lesson Objectives

After completing this lesson, you should be able to do the following:

- Describe the salient features of Oracle Cloud
- Discuss the features of Oracle Database Exadata Express Cloud Service



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

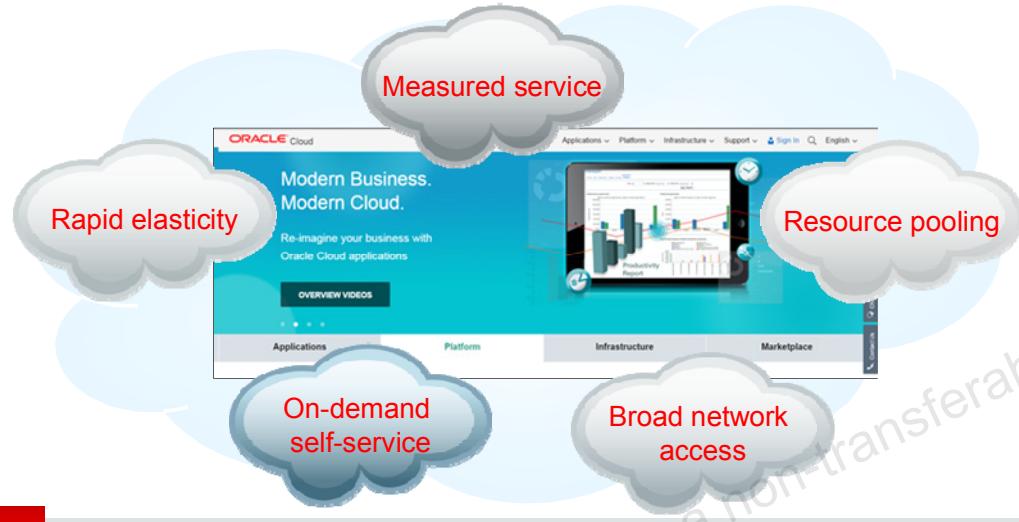
Lesson Agenda

- Overview of Oracle Cloud
- Working with Oracle Database Exadata Express Cloud Service



Introduction to Oracle Cloud

- Any business can now use the enterprise cloud provided by Oracle.
- You can access the Oracle Cloud from cloud.oracle.com.



The Oracle Cloud is an enterprise cloud for business. Oracle Cloud services are built on Oracle Exalogic Elastic Cloud and Oracle Exadata Database Machine, together offering a platform that delivers extreme performance and scalability.

The top two benefits of cloud computing are speed and cost.

As a result, the applications and databases deployed in the Oracle Cloud are portable and you can easily move them to or from a private cloud or on-premise environment.

- You can request and get the cloud services provisioned through a self-service interface.
- You can either use an integrated development and deployment platform to rapidly extend and create new services.

Using Oracle Cloud services, you can benefit from the following five essential characteristics:

- **On-demand self-service:** You can provision, monitor, and manage cloud on your own.
- **Resource pooling:** You can share resources and maintain a level of abstraction between consumers and services.
- **Rapid elasticity:** You can quickly scale up or down as needed.
- **Measured service:** You pay for what you use with either internal chargeback (private cloud) or external billing (public cloud).
- **Broad network access:** You can access the cloud services through a browser on any networked device.

Oracle Cloud Services

Oracle Cloud provides three types of services:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SaaS generally refers to applications that are delivered to end users over the Internet. Oracle CRM On Demand is an example of a SaaS offering that provides both multitenant as well as single-tenant options, depending on the customer's preferences.

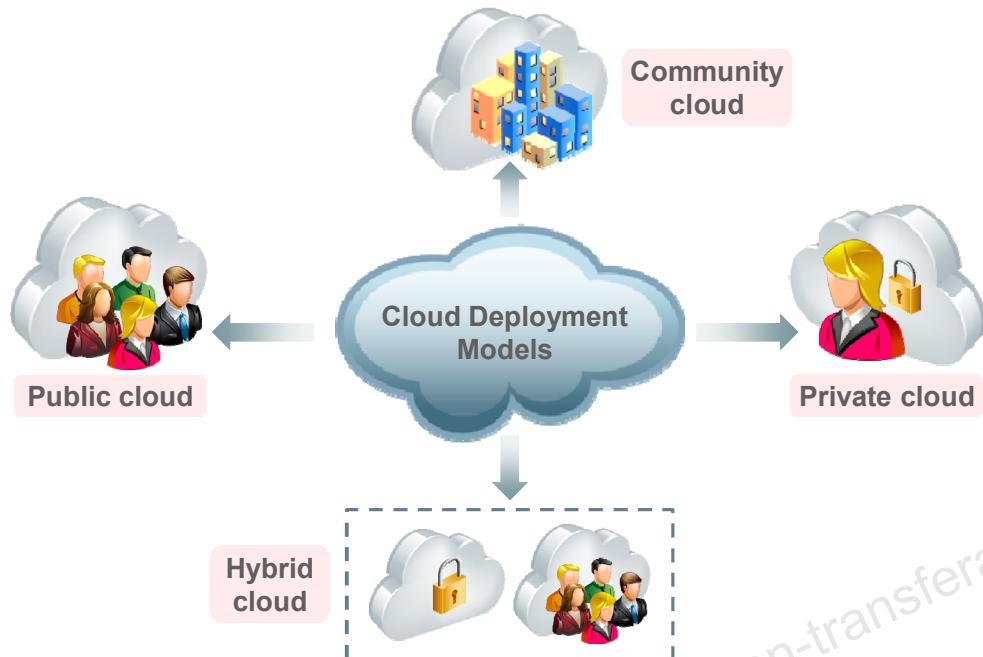
PaaS generally refers to an application development and deployment platform that is delivered as a service to developers, enabling them to quickly build and deploy a SaaS application to end users. The platform typically includes databases, middleware, and development tools, all delivered as a service via the Internet.

IaaS refers to computing hardware (servers, storage, and network) delivered as a service. This service typically includes the associated software as well as operating systems, virtualization, clustering, and so on. Examples of IaaS in the public cloud include Amazon's Elastic Compute Cloud (EC2) and Simple Storage Service (S3).

The Oracle Cloud Database is built as a PaaS model. It provides on-demand access to database services in a self-service, scalable, and metered manner. You can deploy a database within a virtual machine in an IaaS platform.

You can rapidly deploy Oracle Cloud Database on Oracle Exadata, which is a pre-integrated and optimized hardware platform that supports both online transaction processing (OLTP) and Data Warehouse workloads.

Cloud Deployment Models



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- **Private cloud:** A single organization uses a private cloud, which it typically controls, manages, and hosts in private data centers. However, the organization can also outsource hosting and operation to a third-party service provider. Amazon's Virtual Private Cloud is an example of a private cloud in an external provider setting.
- **Public cloud:** Multiple organizations (tenants) use a private cloud on a shared basis. This private cloud is hosted and managed by a third-party service provider. For example: Amazon's Elastic Compute Cloud (EC2), IBM's Blue Cloud, Sun Cloud, and Google AppEngine
- **Community cloud:** A group of related organizations, who want to make use of a common cloud computing environment, uses the community cloud. It is managed by the participating organizations or by a third-party managed service provider. It is hosted internally or externally. For example, a community might consist of the different branches of the military, all the universities in a given region, or all the suppliers to a large manufacturer.
- **Hybrid cloud:** A single organization that wants to adopt both private and public clouds for a single application uses the hybrid cloud. A third model, the hybrid cloud, is maintained by both internal and external providers. For example, an organization might use a public cloud service, such as Amazon Simple Storage Service (Amazon S3), for archived data but continue to maintain in-house (private cloud) storage for operational customer data.

Lesson Agenda

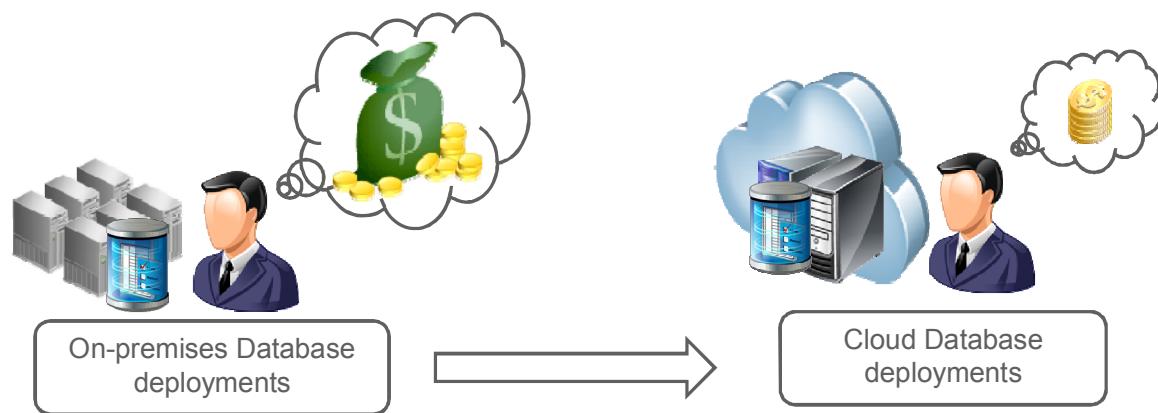
- Overview of Oracle Cloud
- Working with Oracle Database Exadata Express Cloud Service



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Evolving from On-premises to Exadata Express



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cloud deployments provide end users and enterprises with different capabilities to store and process data. They enable users to have high performance and huge computing resources at a lower price as compared to traditional on-premises deployments.

Exadata Express is a powerful database machine, extended as a cloud service. End users can use it for Oracle 12c database deployments. It delivers a complete database experience for developers and enterprises.

Exadata express being a cloud deployment provides high scalability, performance and availability to its users.

It is fully managed database, therefore you need not worry about patching, upgrading or other DBA tasks.

What is in Exadata Express?

- A fully managed database service
- Provides powerful yet elastic database cloud service for developers
- Provides on-demand access to a shared pool of database resources
- Comes with built-in tools for rapid application development
 - APEX for web application development
 - Compatibility with clients such as SQL Developer, SQLcl



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exadata Express is a fully managed database service where end users need not worry about upgrades to the database and other components of the service. All enhancements are automatically managed by the cloud service.

Being a cloud deployment, Exadata Express, allows end users to scale their data virtually to unlimited size.

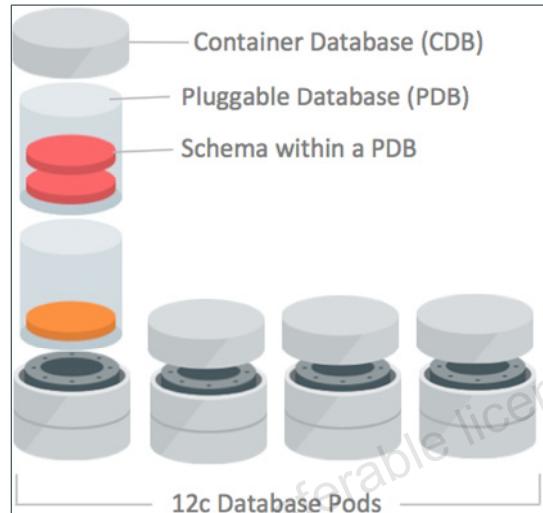
Dynamic provisioning of resources allows users to access huge amount of compute and storage resources in no time.

For developers, it provides built-in application development tool – APEX. APEX(Application Express) is a rapid web application development tool for Oracle database. Developers with minimal development experience can develop and deploy professional applications through web browser using APEX.

Oracle makes a variety of database client drivers and tools available for use with Oracle Database Exadata Express Cloud Service. You can use Exadata Express with Oracle SQL Developer, an IDE used for SQL, PL/SQL development and Oracle SQLcl, an enhanced command line interface.

Exadata Express for Users

- Oracle manages the service as multiple Container databases(CDBs), also known as database pods
- Each CDB can accommodate upto 1000 Pluggable databases(PDBs).
- Each user is provisioned with a PDB on subscribing to the service, where the user can create several schemas.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exadata Express is ideal for production applications that consist of small to medium sized data as well as developers, testers, evaluators and other users who are looking for a full Oracle Database experience at an affordable entry-level price. It is a fully managed database service, is organized into Container databases(CDBs). These container databases are also known as database pods.

Each container database in turn can contain several Pluggable databases(PDBs). When a user subscribes to the Exadata Service, a pluggable database is provisioned. Within the PDB, the user can create several schemas. However, PDB Services are constrained by CPU, storage and memory.

Exadata Express for Developers

- Developers can connect with a wide range of data sources for their applications
 - JSON Document Storage
 - Document Style data access
 - Oracle Rest Data Services



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

JSON Document Storage - Oracle Database in Exadata Express provides direct storage, access and management of JSON documents. See JSON Support in Oracle Database New Features Guide 12c Release 2 (12.2).

Document-Style Data Access - Oracle Database in Exadata Express gives you the ability to store and access data as schema-less documents and collections using the Simple Oracle Document Access (SODA) API. See Working with JSON and Other Data Using SODA in Using Oracle Database Exadata Express Cloud Service.

Oracle REST Data Services 3 - Exadata Express includes the newest Oracle REST Data Services (ORDS). With ORDS 3, it's easy to develop modern RESTful interfaces for relational data and now JSON documents stored in Oracle Database.

Getting Started with Exadata Express

1. Purchase a subscription.
2. Activate and verify the service.
3. Verify activation.
4. Learn about users and roles.
5. Create accounts for your users and assign them appropriate privileges and roles.
6. Set the password for the database user authorized to perform administrative tasks for your service (PDB_ADMIN).

Note: You can refer to Using Oracle Database Exadata Express Cloud Service (<https://docs.oracle.com/cloud/latest/exadataexpress-cloud/CSDBP/toc.htm>) for details on the subscription process.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The steps in the slide provide you a brief overview of how to get started with Exadata Express. In the following slides, you will learn in detail how to perform the aforementioned steps.

Oracle Exadata Express Cloud Service

You can refer to Working with Oracle Database Exadata Express Cloud Service (<http://oukc.oracle.com/public/redir.html?type=player&offid=1984115860>) to gain an introduction to the service and its features.



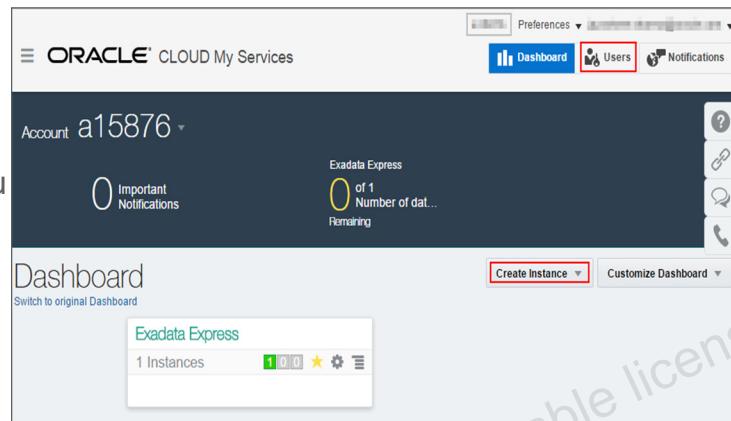
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In this video, you gain an introduction to Oracle Exadata Express Cloud Service and its features. You will take a tour of its service console and also learn about the different database clients such as Oracle SQL Developer, SQL CL, SQL Workshop and SQL * Plus that can be used to connect to Oracle Exadata Express Cloud Service.

Note: This demonstration has audio, which cannot be played inside OU Classroom. However participants can access this link on open internet and review at their convenience.

Getting Started with Exadata Express

- On signing into the service, you get access to the dashboard.
- Dashboard allows you to create database instances and users.
- The number of instances you create is limited by the amount of resources you have access to.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

After successful subscription to the service, you can login to your account and access the dashboard. Based on the type of subscription you can create instances.

The instances would appear on the dashboard. You can see an instance created in the image . To manage the instance, click on the instance.

Managing Exadata

The screenshot shows the Oracle Cloud Service Instances dashboard. On the left, there is a list of service instances, with the 'exa4' instance selected. The 'Open Service Console' button is highlighted with a red box. An arrow points from this button to the 'Service Console: exa4' window on the right. The 'Service Console' window contains several sections: 'Web Access' (with links to 'Go to SQL Workshop', 'Define REST Data Services', and 'Develop with App Builder'), 'Client Access' (with links to 'Download Client Credentials', 'Disable Client Access', and 'Install Productivity Apps'), 'Administration' (with links to 'Create Database Schema', 'Set Administrator Password', and 'Create Document Store'), and 'Download Tools' (with links to 'Download Drivers' and 'Manage Application Express').

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

On clicking the instance on the dashboard, you see various details about the dashboard. You can access the services by clicking on 'Open Service Console'.

The service console provides you access to tools for Web Access, Client Access and Administration.

Service Console

- Service Console is the interface to use and manage the Exadata service
- It provides three different perspectives of the instance
 - Web Access
 - Client Access
 - Administration

The screenshot shows the Oracle Service Console interface for an Exadata system named 'exa4'. The interface is divided into three main sections:

- Web Access:** Develop database and web apps using Oracle Application Express (APEX). Includes links to 'Go to SQL Workshop', 'Define REST Data Services', and 'Install Productivity Apps'.
- Client Access:** Enable database client access, then connect using drivers and tools. Includes links to 'Download Client Credentials', 'Disable Client Access', and 'Download Tools'.
- Administration:** Manage your cloud database. Includes links to 'Create Database Schema', 'Set Administrator Password', and 'Manage Application Express'.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Web access provides utilities which enable you to develop database and web applications using Oracle Application Express(APEX).

Database clients can connect to Exadata Express service using SQL *Net Access. Some examples of supported database clients are SQLcl, SQL Developer, SQL *Plus, JDBC Thin client,ODP.NET, OCI and Instant Client. Client Access in the Service Console allows you to configure with the client you use.

Administration in the Service Console provides for performing administration tasks such as create new database schemas for database objects, set or reset administration password, create a schema-less documents and collections interface, and use administrative options to manage Oracle Application Express.

Web Access through Service Console

To execute SQL and PL/SQL

For quick development of applications

To define RESTful services on the database

Reuse apps from pre-built APEX applications

Web Access

Go to SQL Workshop

Run SQL commands, execute SQL scripts and browse database objects

Define REST Data Services

Create and manage RESTful web service interfaces to your database

Develop with App Builder

Declaratively develop and deploy data-driven apps

Install Productivity Apps

Browse and install productivity apps

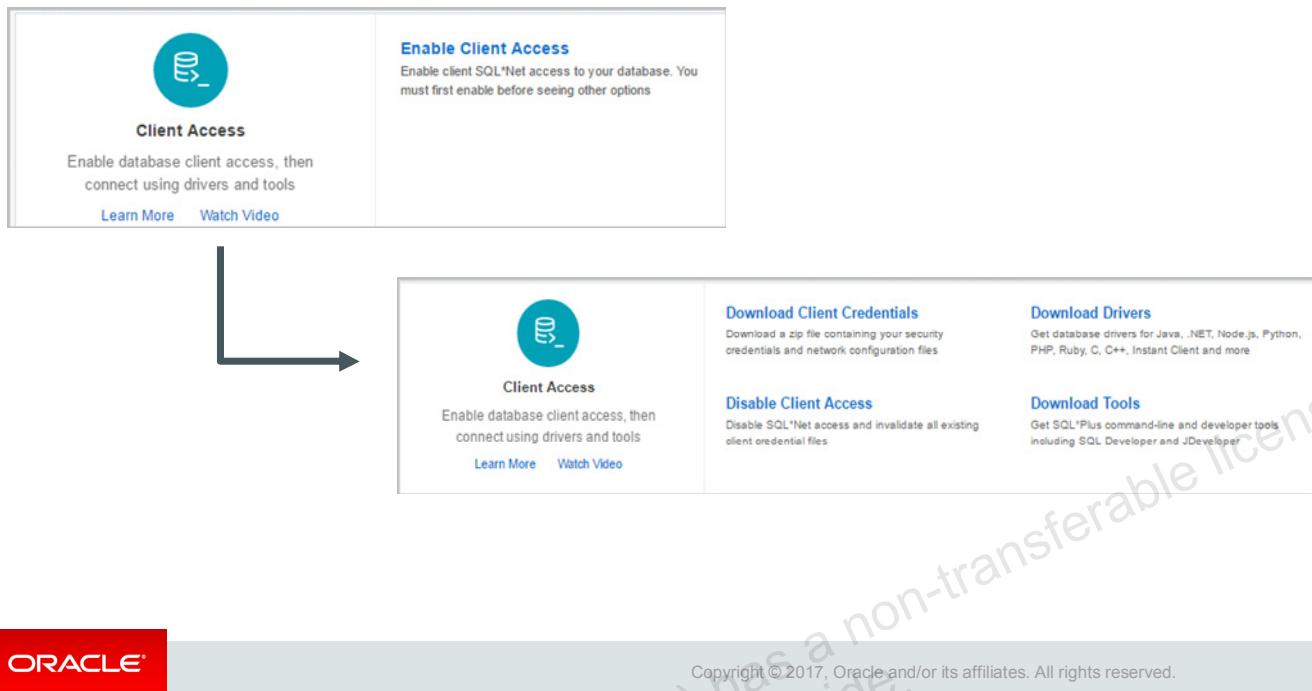
Learn More Watch Video

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Option	Description
Go to SQL Workshop	Allows you go directly to browser-based SQL Workshop, where you can run SQL statements, execute scripts and explore database objects.
Develop with App Builder	Quickly declaratively develop database and websheet applications. You can import files such as database applications and plug-ins. There is a dashboard showing metrics about your applications and workspace utilities to manage defaults, themes, metadata, exports, and more.
Define REST Data Services	Directly access the page to define and manage RESTful web services that view and manipulate data objects within your database.
Install Productivity Apps	Install from a gallery of pre-built Oracle Application Express Productivity Apps.

Client Access Configuration through Service Console



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You have to enable client access to allow SQL*Net Access to your service. Using SQL *Net Access software you can connect the Exadata instance to different clients. This option is only available when client access has not yet been enabled. Once you enable the client access, four options appear in the console:

Option	Description
Download Client Credentials	Download client credentials needed for clients to access your service.
Download Drivers	Go directly to the Oracle Technology Network page to download and install database drivers including for Java, Instant Client, C, C++, Microsoft .NET, Node.js, Python, PHP, Ruby, and more.
Disable Client Access	Use this option to disallow SQL*Net access to your service. This option is only available when client access has been enabled.
Download tools	Go directly to the Oracle Technology Network page to download and install tools such as SQL*Plus, SQLcl, command-line and integrated development environments such as Oracle SQL Developer, JDeveloper, Oracle JET, and more

Database Administration through Service Console

The screenshot shows the Oracle Database Service Console. On the left, there's a navigation bar with a gear icon labeled "Administration" and a sub-section "Manage your cloud database". Below it are "Learn More" and "Watch Video" links. The main content area has four sections:

- Create Database Schema**: Create a new schema for database objects.
- Set Administrator Password**: Set or reset your database's privileged user (PDB_ADMIN) account password.
- Create Document Store**: Enable or disable a schema-less document-style interface, with JSON storage and access.
- Manage Application Express**: Use Application Express (APEX) administrative options.

Callout boxes provide additional context:

- An arrow points from the "Create Database Schema" section to a callout box: "Create a new schema for database objects".
- An arrow points from the "Create Document Store" section to a callout box: "To create a document store using a schema".
- An arrow points from the "Set Administrator Password" section to a callout box: "Set or reset password for admin".
- An arrow points from the "Manage Application Express" section to a callout box: "To manage tasks such as archiving APEX schemas and association among APEX schema".

At the bottom left is the ORACLE logo, and at the bottom right is the copyright notice: "Copyright © 2017, Oracle and/or its affiliates. All rights reserved."

You can perform various administration tasks through 'Administration' in the service console.

Option	Description
Create Database Schema	Create a new schema for database objects. Schema is the set of database objects, such as tables and views that belong to that user account.
Create Document Store	This option enables you to create a document store, using either an existing schema or new schema, and to enable SODA for REST, which enables REST-based operations on the schema using Oracle's SODA for REST API. It also enables SODA for Java, which is Oracle's SODA for Java API for use with Java programs.
Set Administrator Password	Use this option to set the password for the PDB_ADMIN database user that is authorized to perform administrative tasks.
Manage Application Express	Options here allow you to enable application archiving to archive your Oracle Application Express applications to database tables, manage the association between schemas and Oracle Application Express, and manage messages and set preferences for the workspace.

SQL Workshop

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Web Access
Develop database and web apps using Oracle Application Express (APEX)
[Learn More](#) [Watch Video](#)

Go to SQL Workshop
Run SQL commands, execute SQL scripts and browse database objects

Develop with App Builder
Declaratively develop and deploy data-driven apps

Define REST Data Services
Create and manage RESTful web service interfaces to your database

Install Productivity Apps
Browse and install productivity apps



1

Clicking on SQL workshop will lead you to APEX interface

ORACLE Application Express Application Builder SQL Workshop Team Development Packaged Apps

Object Browser SQL Commands SQL Scripts Utilities RESTful Services

2

To run SQL or PL/SQL you can use the SQL commands utility

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL Workshop

You can run SQL statements in the editor.

The screenshot shows the Oracle Application Express SQL Workshop interface. At the top, there are tabs for Application Builder, SQL Workshop (which is selected), Team Development, and Packaged Apps. Below the tabs, a toolbar includes a magnifying glass icon, a help icon, and a user profile icon. The main area is titled "SQL Commands" and shows a schema dropdown set to "BZJNMKSSA". A "Rows" dropdown is set to 10, and buttons for "Clear Command" and "Find Tables" are visible. A "Save" button and a "Run" button are at the bottom right. The SQL command entered is "SELECT * FROM emp;". The results tab is selected, displaying a table with the following data:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	-	11/17/1981	5000	-	10
7698	BLAKE	MANAGER	7839	05/01/1981	2850	-	30
7782	CLARK	MANAGER	7839	06/09/1981	2450	-	10
7566	JONES	MANAGER	7839	04/02/1981	2975	-	20
7788	SCOTT	ANALYST	7566	12/09/1982	3000	-	20
7902	FORD	ANALYST	7566	12/03/1981	3000	-	20

Below the table, there are tabs for Explain, Describe, Saved SQL, and History. The Oracle logo is in the bottom left, and a copyright notice is in the bottom right: "Copyright © 2017, Oracle and/or its affiliates. All rights reserved."

Connecting through Database Clients

You can connect to Exadata Express through various database clients.

Some of the database clients include:

- SQL*Plus
- SQLcl
- SQL Developer
- .Net and Visual Studio
- JDBC Thin Client



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You must first configure database clients and Oracle Database Exadata Express Cloud Service to communicate with each other.

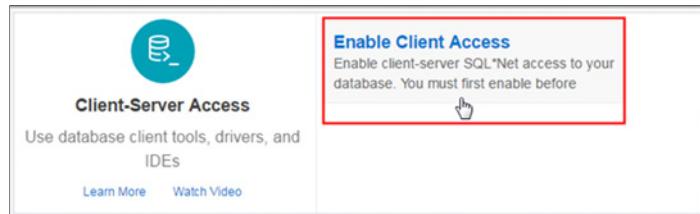
Prerequisite tasks for database client connectivity require you to:

- Enable SQL*Net access to your service.
- Download client credentials.
- Follow set-up instructions for the specific database client you want to connect with.

In the following topics, you will learn how to enable SQL*Net access, download client credentials and make connections using Oracle SQL Developer and SQLcl.

Enabling SQL*Net Access for Client Applications

Enable SQL*Net Access in the Service Console to obtain the various Database Client options.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

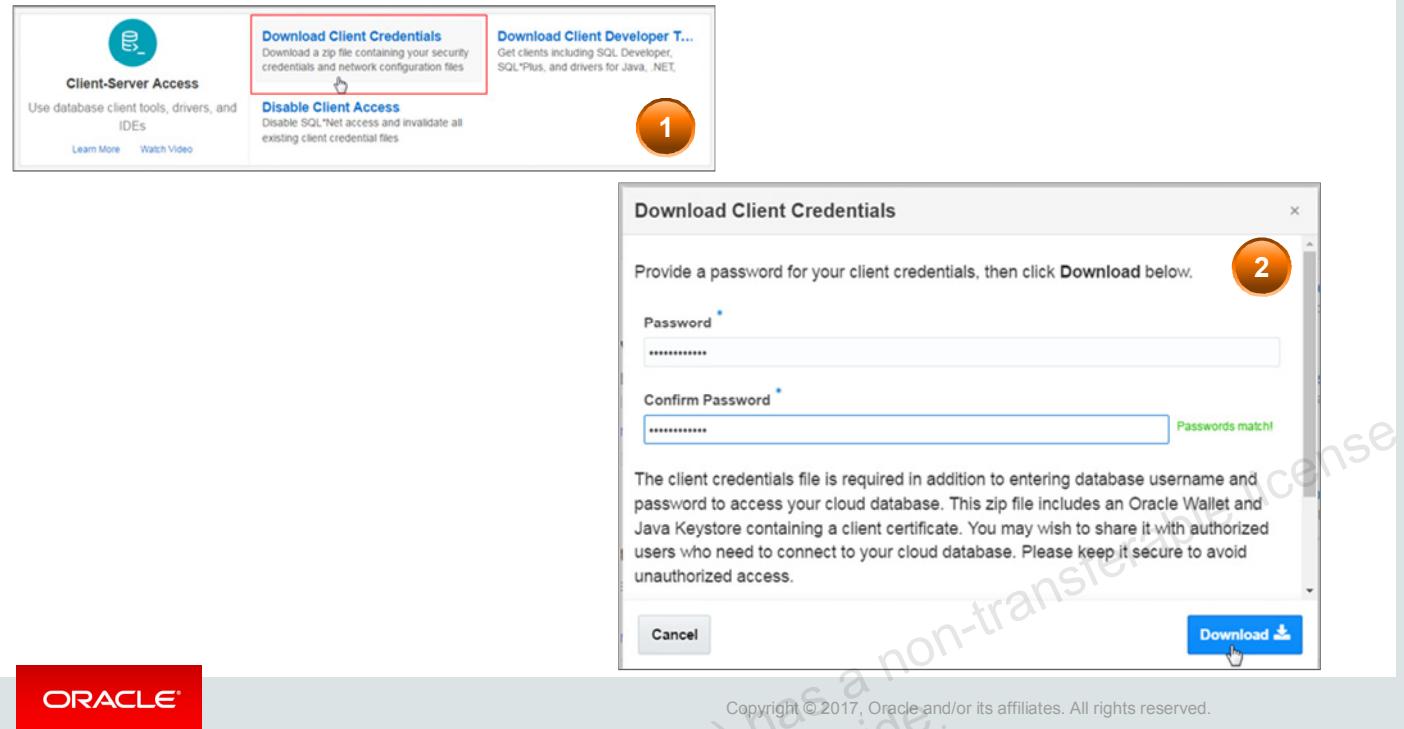
You can connect to your Exadata Express cloud service from diverse database clients over SQL*Net also called as Oracle Database Net Services. Some examples of supported clients include SQL*Plus, SQLcl, SQL Developer, JDBC Thin, ODP.NET, OCI, and Instant Client.

Database drivers for all popular programming and scripting languages such as Python, PHP, Node.js, C/C++, Ruby and Perl are supported. SQL*Net access has to be enabled as a prerequisite for all clients and drivers connecting over SQL*Net.

The Service Administrator must do the following to enable SQL*Net:

- Navigate to the Service Console for **Exadata Express** and open the service console.
- Click **Enable Client Access**.
- Download the client credentials.
- Now, depending on the client-side application and driver being used, you need to configure the application connection string for that application.

Downloading Client Credentials



You can easily download the zip files for client credentials from service console. Its contents include Oracle Wallet and Java Keystore as well as essential client configuration files. While downloading the zip file, you are prompted to enter a password.

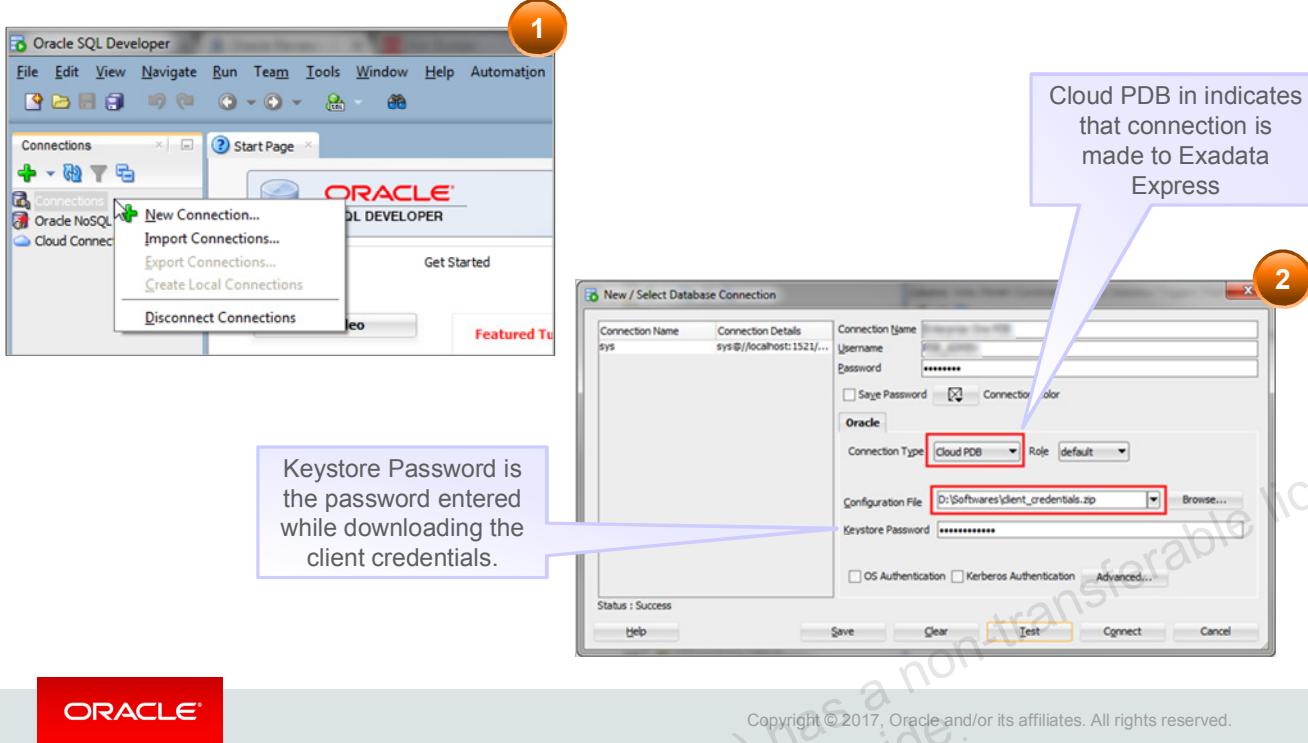
Note that client credential zip files should be carefully managed. Please remember to keep the file secure to avoid unauthorized database access. If you believe the security of this file has been compromised, then immediately disable client access using the cloud service console.

The steps to download the client credentials are as follows.

1. Navigate to Exadata Express and open the service console.
2. Click Download Client Credentials to download a zip file containing your security credentials and network configuration files.
3. Enter a password to create a password-protected Oracle Wallet and Java Keystore files for the service.
4. Click Download and save the downloaded zip file to a secure location that is accessible by your database client(s).

In the following topics, you will learn how to use these credentials to connect to the cloud database.

Connecting Oracle SQL Developer



In order to connect to Oracle Database Exadata Express cloud service, you need to download and install Oracle SQL Developer Release 4.1.5 or later. You should have also downloaded the Client Credentials from Oracle Exadata Express service console. You should then configure an Oracle Cloud connection in the Oracle SQL Developer.

The connection can be created as follows:

1. Run Oracle SQL Developer locally.
2. Under Connections, right click Connections and select **New Connection**.
3. Enter the following details:
 - Connection Name – Enter a name for this cloud connection.
 - Username – Enter username required to sign into Exadata Express.
 - Password – Enter password required to sign into Exadata Express.
 - Connection type – Select **Cloud PDB**.
 - Configuration File - Click Browse and select the **Client Credentials** zip file that you previously downloaded from the Exadata Express service console.
 - Keystore Password – Enter the password provided while downloading the Client Credentials from the Exadata Express service console.
4. Click Test. If the status is Success, click Connect.

If you have connected successfully, the tables and other objects from Exadata Express display under the new connection.

Connecting Oracle SQLcl

```
D:\PDB Service\SQL CL\sqlcl-no-jre-latest\sqlcl\bin>sql /nolog  
SQLcl: Release 4.2.0.16.160.2007 RC on Thu Sep 08 12:18:07 2016  
Copyright (c) 1982, 2016, Oracle. All rights reserved.  
SQL>
```

1

```
SQL> set cloudconfig client_credentials.zip  
Wallet Password: *****  
Using temp directory:C:\Users\APOTHU~1.ORA\AppData\Local\Temp\  
oracle_cloud_config6707346342028726502
```

2

```
SQL> conn pdb_admin/welcome1@dbaccess  
Connected.  
SQL>
```

3



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned about:

- The salient features of Oracle Cloud
- Oracle Database Exadata Express Cloud Service



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Table Descriptions

ORACLE®

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Schema Description

Overall Description

The Oracle Database sample schemas portray a sample company that operates worldwide to fill orders for several different products. The company has three divisions:

- **Human Resources (HR):** Tracks information about the employees and facilities
- **Order Entry:** Tracks product inventories and sales through various channels
- **Sales History:** Tracks business statistics to facilitate business decisions

Each of these divisions is represented by a schema. In this course, you have access to the objects in all the schemas. However, the emphasis of the examples, demonstrations, and practices is on the HR schema.

All scripts necessary to create the sample schemas reside in the \$ORACLE_HOME/demo/schema/ folder.

HR

This is the schema that is used in this course. In the HR records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

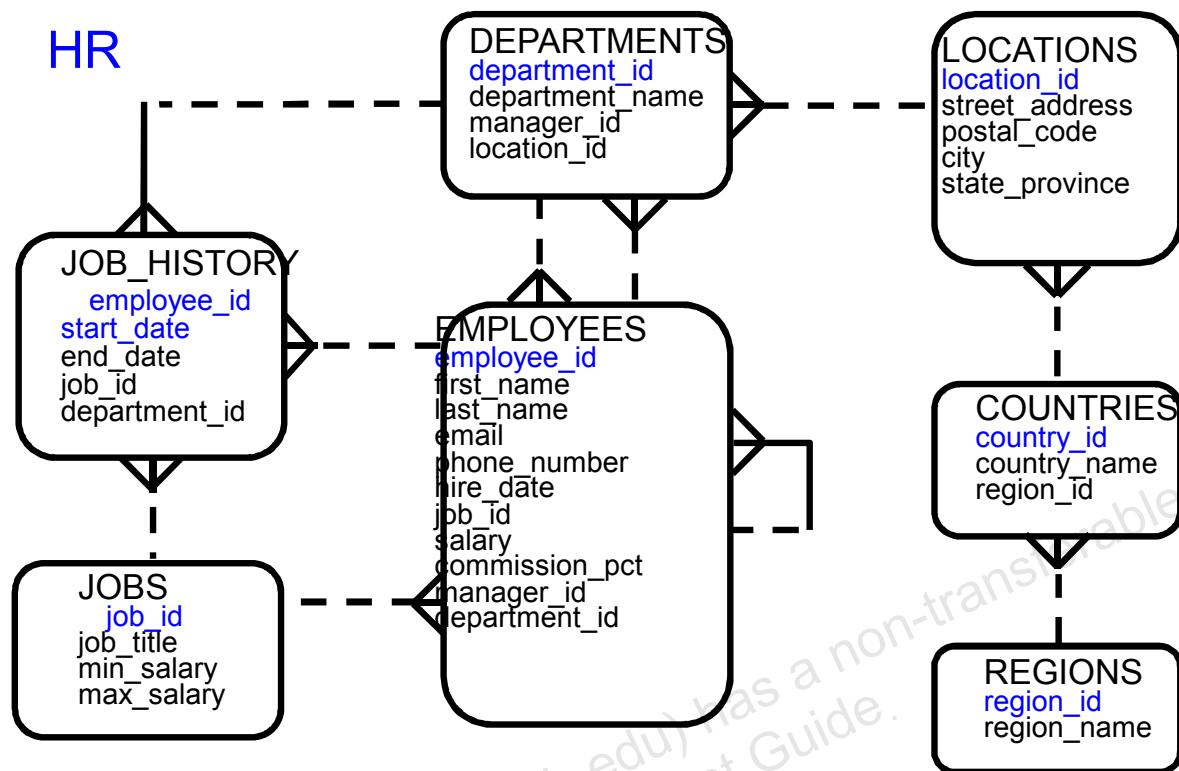
The company also tracks information about jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range for the job. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration the employee was working for, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department, and each department is identified either by a unique department number or a short name. Each department is associated with one location, and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is located geographically.

HR Entity Relationship Diagram

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.



HR Table Descriptions

DESCRIBE countries

Name	Null	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT * FROM countries

#	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

DESCRIBE departments

Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SELECT * FROM departments

#	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

DESCRIBE employees

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM employees

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-11	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-09	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-09	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-14	AC_MGR	12008	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-15	IT_PROG	6000	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-15	IT_PROG	4200	(null)	103	60
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-15	ST_MAN	5800	(null)	100	50
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-11	ST_CLERK	3500	(null)	124	50
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-13	ST_CLERK	3100	(null)	124	50
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-14	ST_CLERK	2600	(null)	124	50
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-14	ST_CLERK	2500	(null)	124	50
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-16	SA_MAN	10500	0.2	100	80
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-12	SA REP	11000	0.3	149	80
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-14	SA REP	8600	0.2	149	80
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-15	SA REP	7000	0.15	149	(null)
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-11	AD_ASST	4400	(null)	101	10
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-12	MK MAN	13000	(null)	100	20
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-13	MK REP	6000	(null)	201	20
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-10	AC_MGR	12008	(null)	101	110
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-10	AC_ACCOUNT	8300	(null)	205	80

DESCRIBE job_history

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM job_history

#	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-09	24-JUL-14	IT_PROG	60
2	101	21-SEP-09	27-OCT-16	AC_ACCOUNT	110
3	101	28-OCT-09	15-MAR-13	AC_MGR	110
4	201	17-FEB-12	19-DEC-15	MK_REP	20
5	114	24-MAR-14	31-DEC-15	ST_CLERK	50
6	122	01-JAN-15	31-DEC-15	ST_CLERK	50
7	200	17-SEP-95	17-JUN-09	AD_ASST	90
8	176	24-MAR-14	31-DEC-14	SA_REP	80
9	176	01-JAN-15	31-DEC-15	SA_MAN	80
10	200	01-JUL-10	31-DEC-14	AC_ACCOUNT	90

DESCRIBE jobs

Name	Null	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT * FROM jobs

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1 AD_PRES	President	20080	40000
2 AD_VP	Administration Vice President	15000	30000
3 AD_ASST	Administration Assistant	3000	6000
4 AC_MGR	Accounting Manager	8200	16000
5 AC_ACCOUNT	Public Accountant	4200	9000
6 SA_MAN	Sales Manager	10000	20080
7 SA_REP	Sales Representative	6000	12008
8 ST_MAN	Stock Manager	5500	8500
9 ST_CLERK	Stock Clerk	2008	5000
10 IT_PROG	Programmer	4000	10000
11 MK_MAN	Marketing Manager	9000	15000
12 MK_REP	Marketing Representative	4000	9000

DESCRIBE locations

Name	Null	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1400 2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500 2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700 2004 Charade Rd	98199	Seattle	Washington	US
4	1800 460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500 Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

```
DESCRIBE regions
```

Name	Null	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions
```

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Copyright©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports
- Browse the Data Modeling options in SQL Developer



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

What is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using standard Oracle database authentication.



SQL Developer

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, which is the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

You can connect to any target Oracle database schema by using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

SQL Developer is the interface to administer the Oracle Application Express Listener. The new interface enables you to specify global settings and multiple database settings with different database connections for the Application Express Listener. SQL Developer provides the option to drag and drop objects by table or column name onto the worksheet. It provides improved DB Diff comparison options, GRANT statement support in the SQL editor, and DB Doc reporting. Additionally, SQL Developer includes support for Oracle Database 12c features.

Specifications of SQL Developer

- Is shipped along with Oracle Database 12c Release 1
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle SQL Developer is shipped along with Oracle Database 12c Release 1 by default. SQL Developer is developed in Java, leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

The default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver and, therefore, no Oracle Home is required. SQL Developer does not require an installer and you need to simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1 and, later, and all Oracle database editions, including Express Edition.

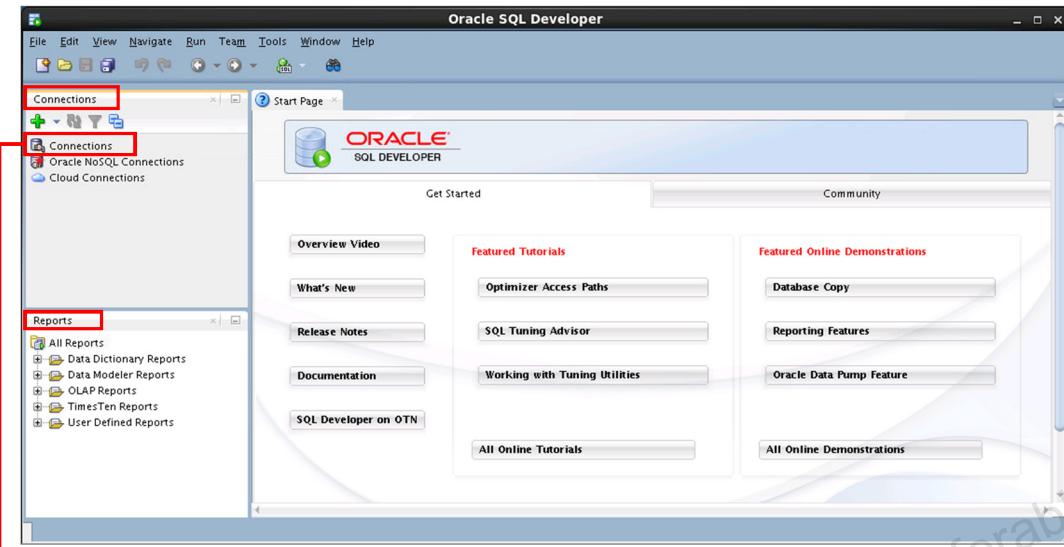
Note: For Oracle Database 12c Release 1, you have to download and install SQL Developer. SQL Developer is freely downloadable from the following link:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

For instructions on how to install SQL Developer, see the website at:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 3.2 Interface



You must define a connection to start using SQL Developer for running SQL queries on a database schema.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The SQL Developer interface contains two main navigation tabs:

- **Connections:** By using this tab, you can browse database objects and users to which you have access.
- **Reports:** Identified by the Reports icon, this tab enables you to run predefined reports or create and add your own reports.

General Navigation and Use

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about selected objects. You can customize many aspects of the appearance and behavior of SQL Developer by setting preferences.

Note: You need to define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures and functions.

Menus

The following menus contain standard entries, plus entries for features that are specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options
- **Versioning:** Provides integrated support for the following versioning and source control systems—Concurrent Versions System (CVS) and Subversion
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet. It also contains options related to migrating third-party databases to Oracle.

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging.

Creating a Database Connection

- You must have at least one database connection to use SQL Developer.
- You can create and test connections for:
 - Multiple databases
 - Multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an Extensible Markup Language (XML) file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.



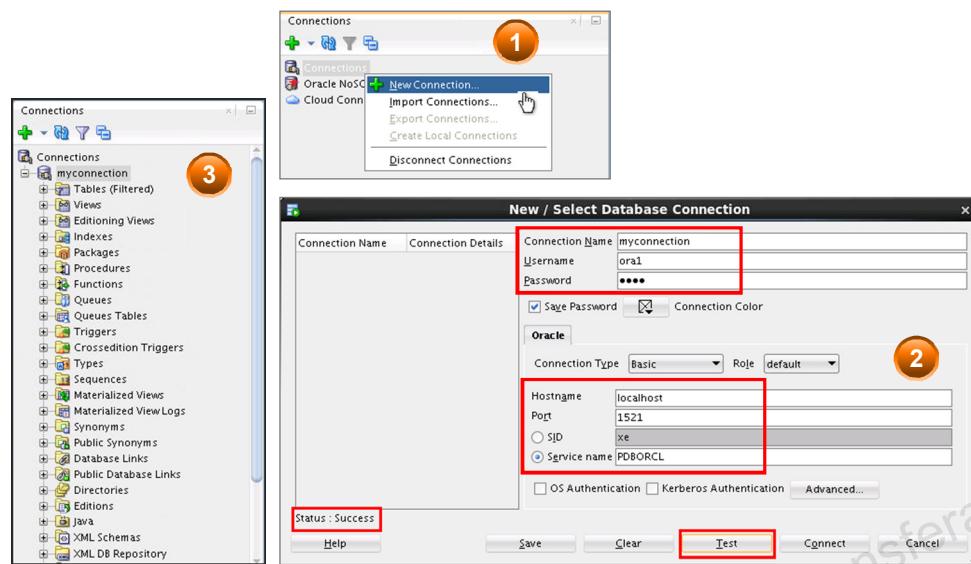
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Creating a Database Connection

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New / Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 - a. From the Role drop-down list, you can select either *default* or SYSDBA. (You choose SYSDBA for the `sys` user or any user with database administrator privileges.)
 - b. You can select the connection type as:
Basic: In this type, enter host name and SID for the database that you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.
TNS: You can select any one of the database aliases imported from the `tnsnames.ora` file.
LDAP: You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.
Advanced: You can define a custom JDBC URL to connect to the database.
Local/Bequeath: If the client and database exist on the same computer, a client connection can be passed directly to a dedicated server process without going through the listener.
 - c. Click Test to ensure that the connection has been set correctly.
 - d. Click Connect.

If you select the Save Password check box, the password is saved to an XML file. So, after you close the SQL Developer connection and open it again, you are not prompted for the password.

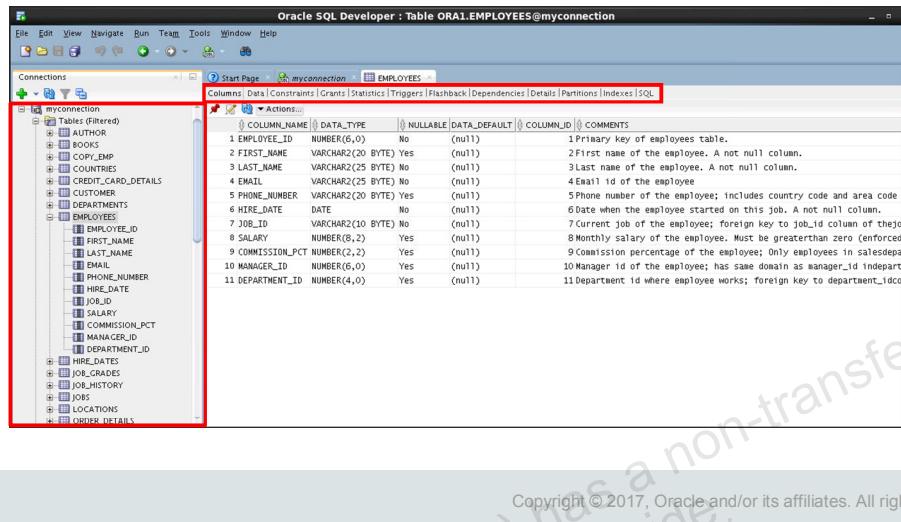
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions (dependencies, details, statistics, and so on).

Note: From the same New>Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

After you create a database connection, you can use the Connections Navigator to browse through many objects in a database schema, including Tables, Views, Indexes, Packages, Procedures, Triggers, and Types.

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about the selected objects. You can customize many aspects of the appearance of SQL Developer by setting preferences.

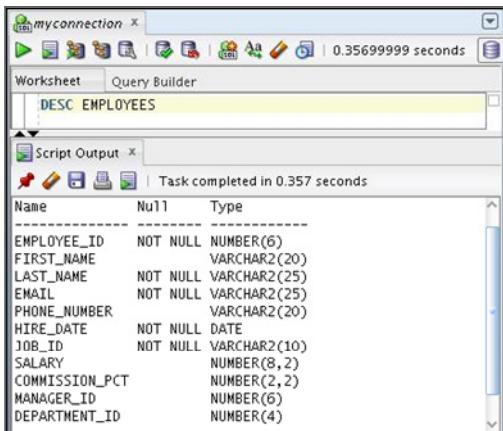
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following steps:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. Using the Data tab, you can view the table data and also enter new rows, update data, and commit these changes to the database.

Displaying the Table Structure

Use the DESCRIBE command to display the structure of a table:



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a window titled "myconnection x". Below it, a "Worksheet" tab is active, showing the command "DESC EMPLOYEES". To the right of the worksheet is a "Script Output" tab which displays the results of the describe command. The output shows the following table structure:

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

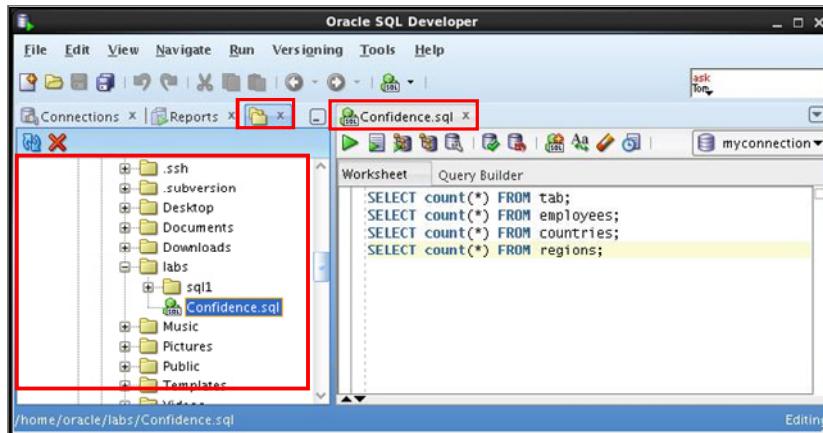


ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Browsing Files

Use the File Navigator to explore the file system and open system files.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

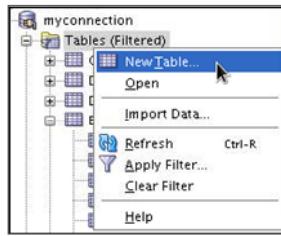
Browsing Database Objects

You can use the File Navigator to browse and open system files.

- To view the File Navigator, click the View tab and select Files, or select View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL Worksheet area.

Creating a Schema Object

- SQL Developer supports the creation of any schema object by:
 - Executing a SQL statement in SQL Worksheet
 - Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the DDL for adjustments such as creating a new object or editing an existing schema object.



ORACLE

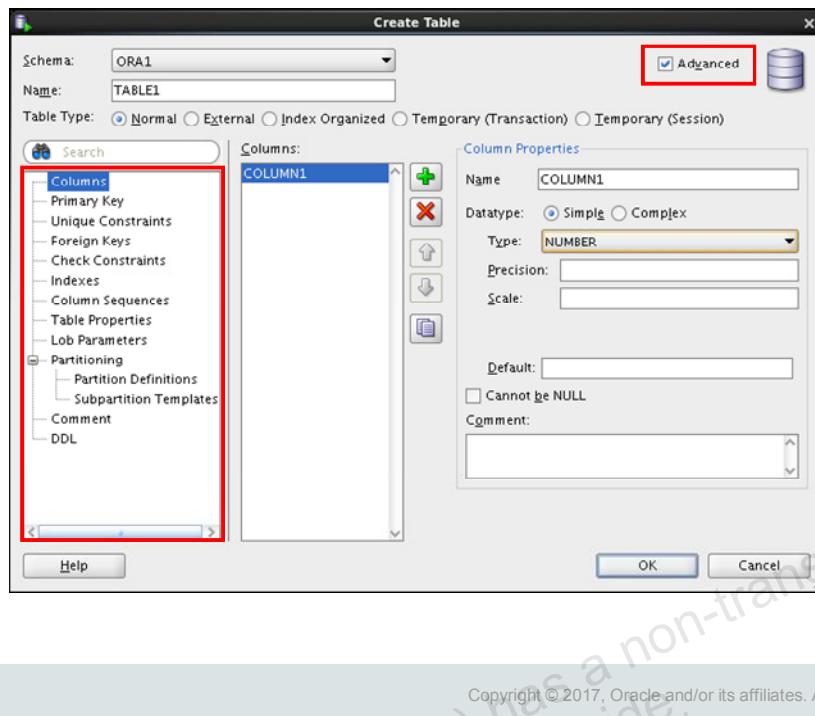
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects by using the context menus. When created, you can edit objects using an edit dialog box or one of the many context-sensitive menus.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example



In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

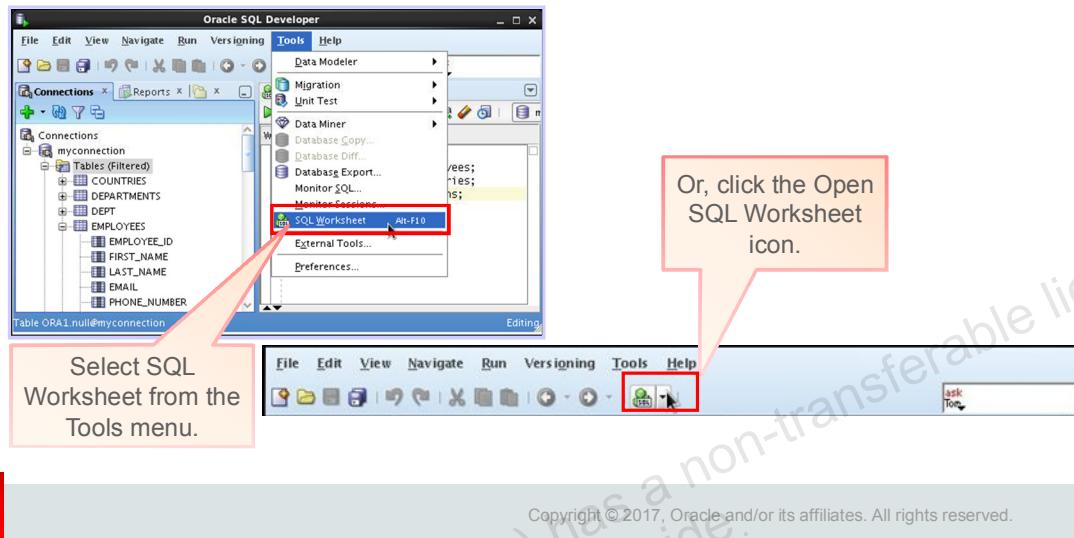
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click Tables and select Create TABLE.
2. In the Create Table dialog box, select Advanced.
3. Specify the column information.
4. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Using SQL Worksheet

- Use SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by SQL Worksheet are ignored and not passed to the database.

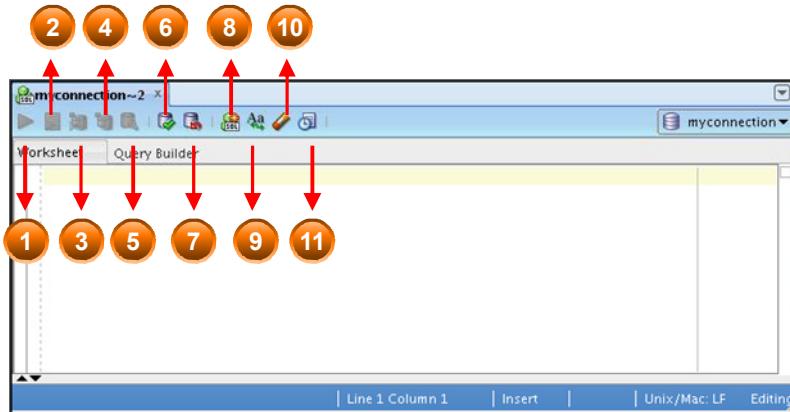
You can specify the actions that can be processed by the database connection associated with the worksheet, such as:

- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by using one of the following:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Using SQL Worksheet



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

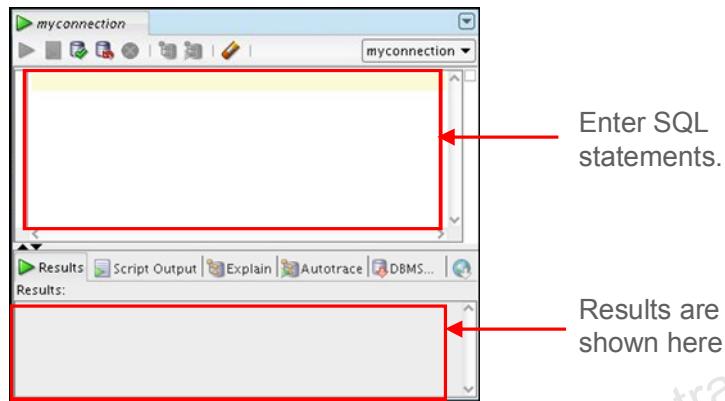
You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of the SQL statements that you have executed. You can use SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Run Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all the statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Autotrace:** Generates trace information for the statement
4. **Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
5. **SQL Tuning Advisory:** Analyzes high-volume SQL statements and offers tuning recommendations
6. **Commit:** Writes any changes to the database and ends the transaction
7. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction

8. **Unshared SQL Worksheet:** Creates a separate unshared SQL Worksheet for a connection
9. **To Upper/Lower/InitCap:** Changes the selected text to uppercase, lowercase, or initcap, respectively
10. **Clear:** Erases the statement or statements in the Enter SQL Statement box
11. **SQL History:** Displays a dialog box with information about the SQL statements that you have executed

Using SQL Worksheet

- Use SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



ORACLE

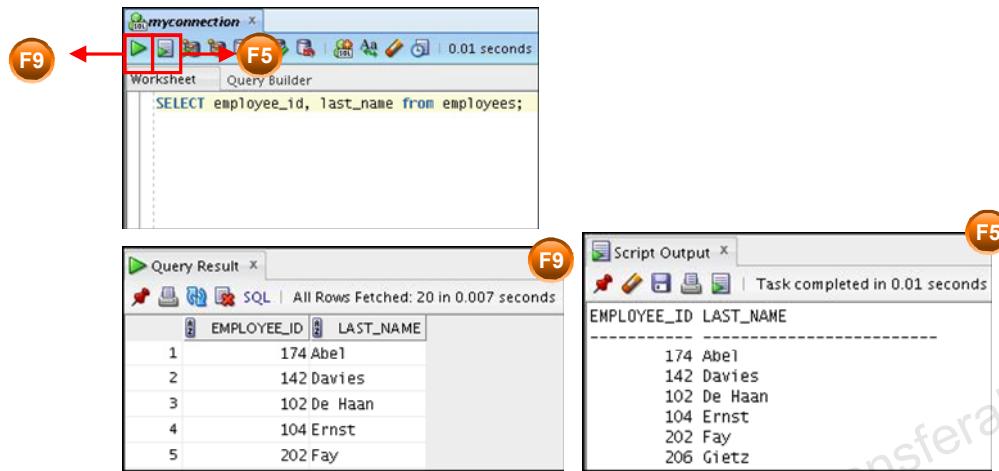
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from SQL Worksheet to the Oracle database. The SQL*Plus commands that are used in SQL Developer must be interpreted by SQL Worksheet before being passed to the database.

SQL Worksheet currently supports a number of SQL*Plus commands. Commands that are not supported by SQL Worksheet are ignored and not sent to the Oracle database. Through SQL Worksheet, you can execute the SQL statements and some of the SQL*Plus commands.

Executing SQL Statements

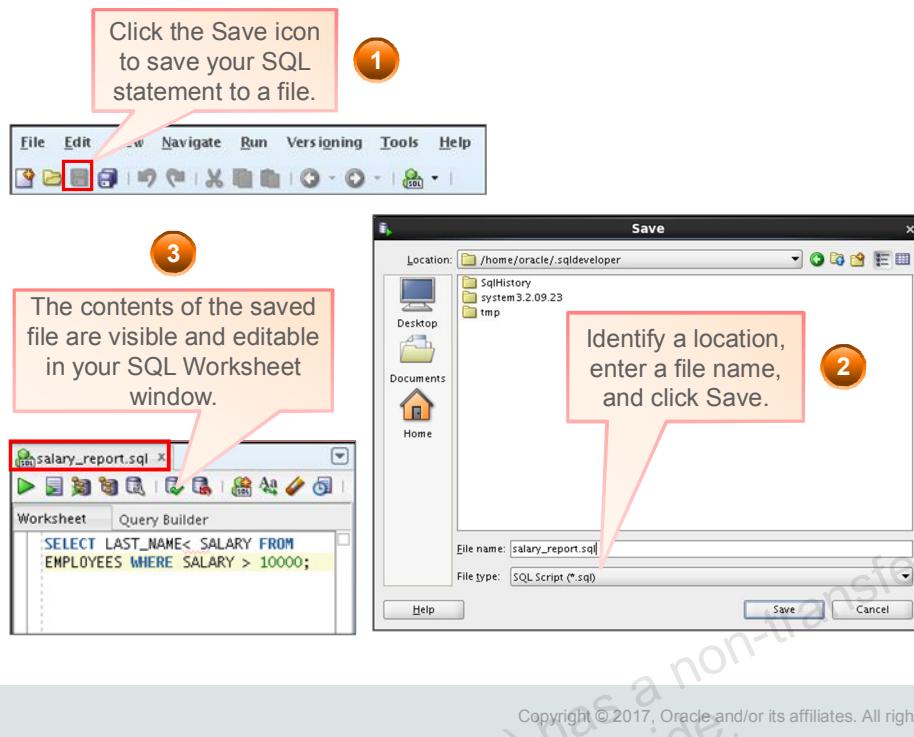
Use the Enter SQL Statement box to enter single or multiple SQL statements.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Saving SQL Scripts



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can save your SQL statements from SQL Worksheet to a text file. To save the contents of the Enter SQL Statement box, perform the following steps:

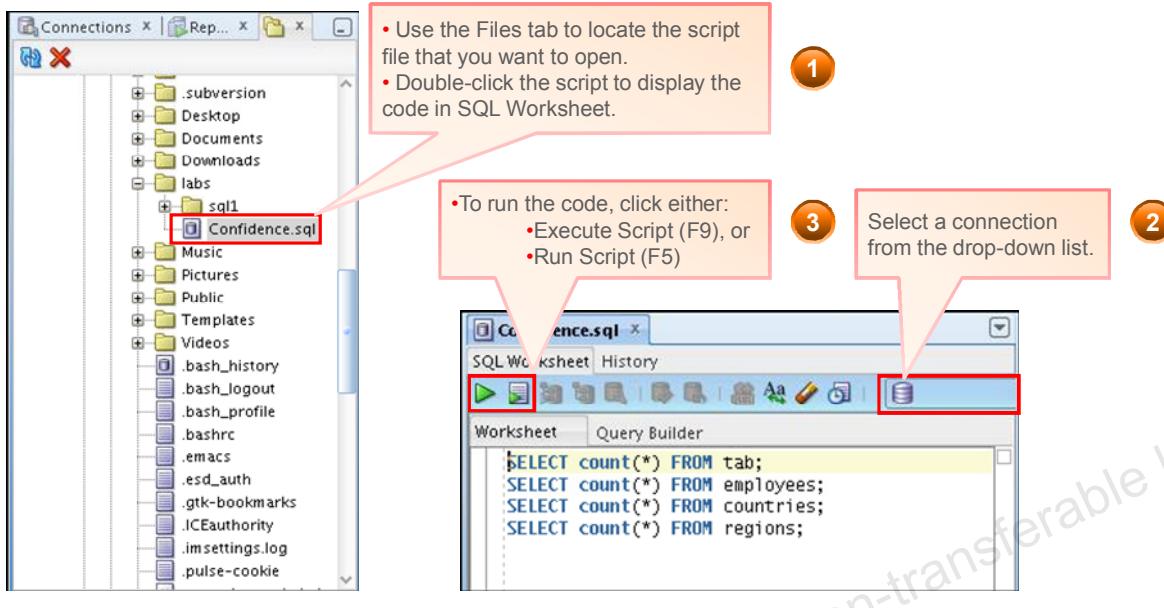
1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file is displayed as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the "Select default path to look for scripts" field.

Executing Saved Script Files: Method 1



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

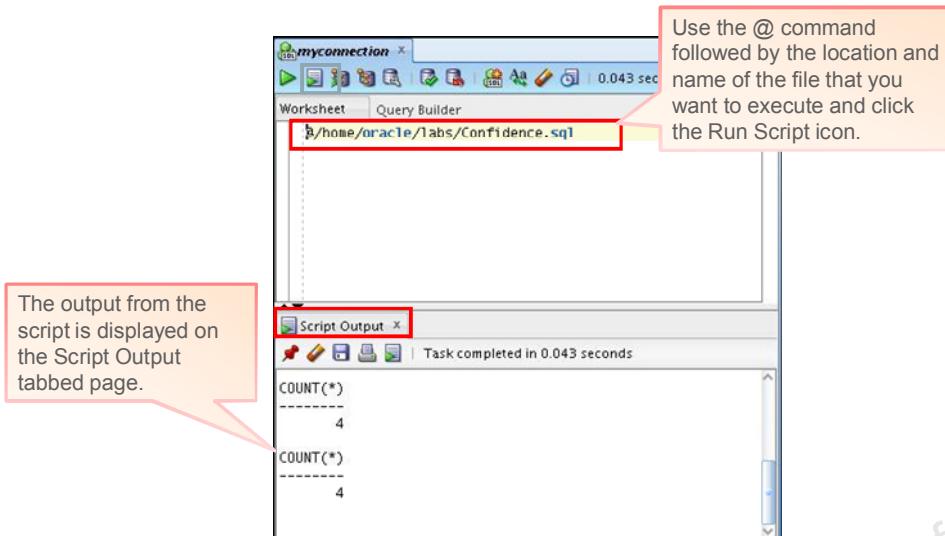
To open a script file and display the code in the SQL Worksheet area, perform the following steps:

1. In the files navigator, select (or navigate to) the script file that you want to open.
2. Double-click the file to open it. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Alternatively, you can also do the following:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Executing Saved Script Files: Method 2



ORACLE

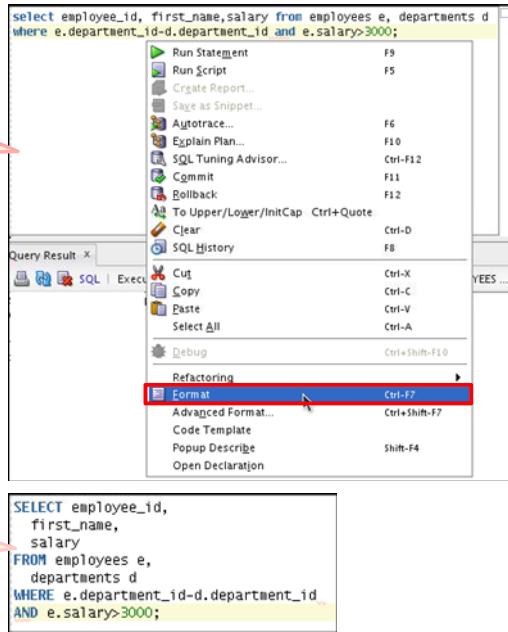
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To run a saved SQL script, perform the following steps:

1. Use the @ command followed by the location and the name of the file that you want to run in the Enter SQL Statement window.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The File Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

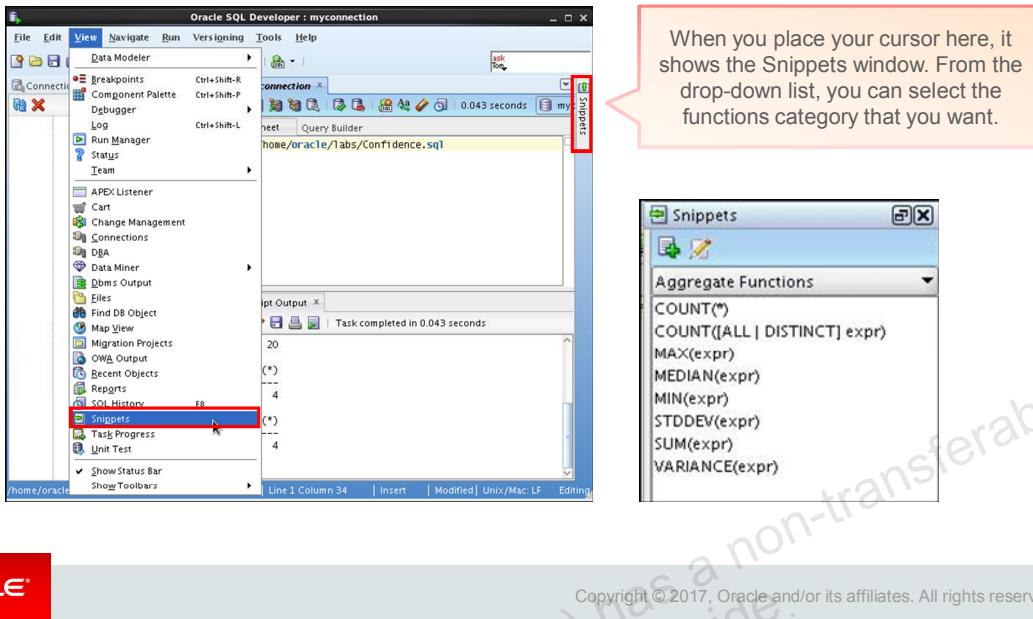
You may want to format the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting the SQL code.

To format the SQL code, right-click in the statement area and select Format.

In the example in the slide, before formatting, the SQL code has the keywords not capitalized and the statement not properly indented. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Using Snippets

Snippets are code fragments that may be just syntax or examples.

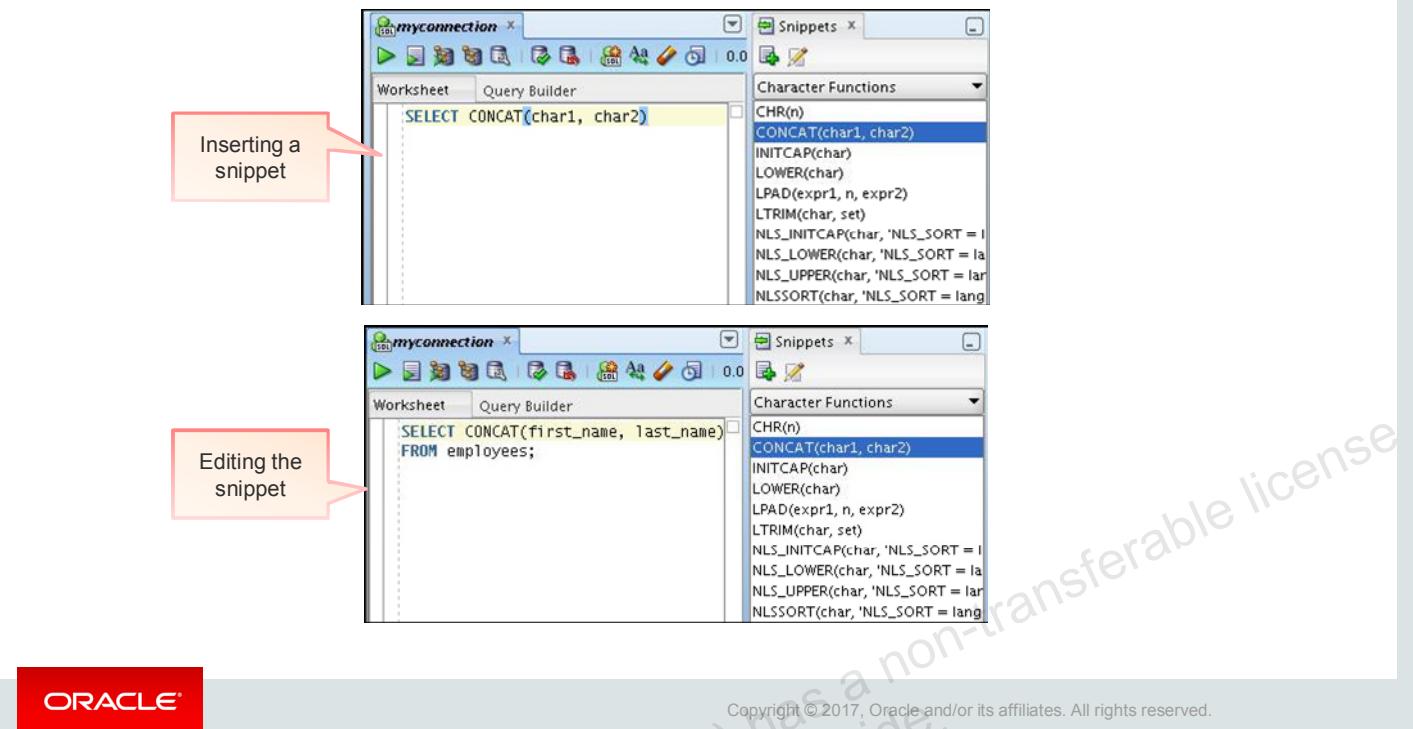


You may want to use certain code fragments when you use SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has a feature called Snippets. Snippets are code fragments such as SQL functions, optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets to the Editor window.

To display Snippets, select View > Snippets.

The Snippets window is displayed on the right. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

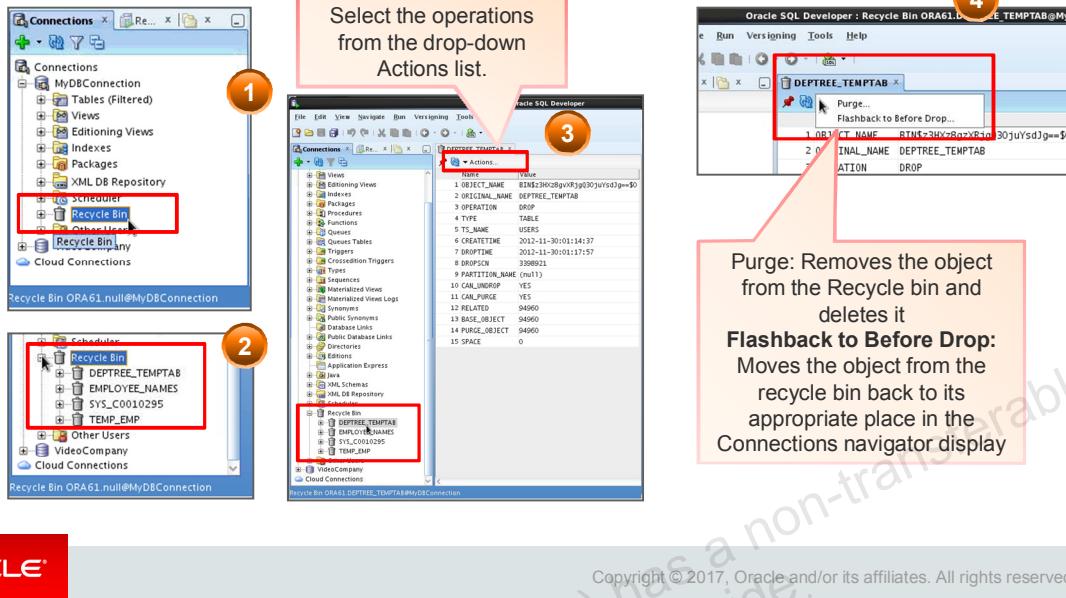
To insert a Snippet into your code in SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window to the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT (char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Using the Recycle Bin

The recycle bin holds objects that have been dropped.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

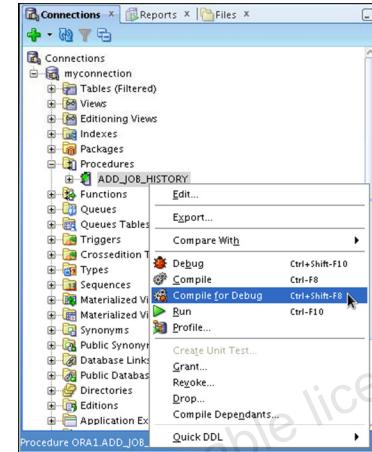
The recycle bin is a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

To use the recycle bin, perform the following steps:

1. In the Connections navigator, select (or navigate to) the recycle bin.
2. Expand Recycle Bin and click the object name. The object details are displayed in the SQL Worksheet area.
3. Click the Actions drop-down list and select the operation that you want to perform on the object.

Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform Step Into and Step Over tasks.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

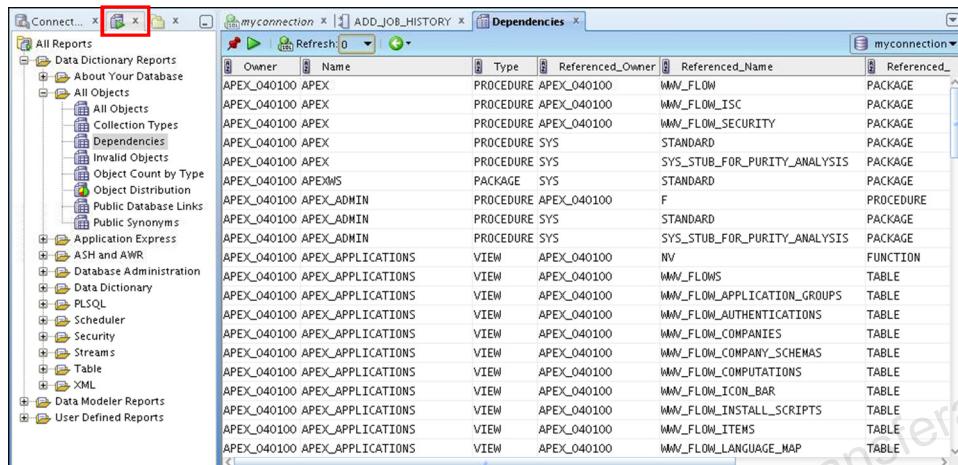
In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to the next statement.
- **Step to End of Method** goes to the last statement of the current method.
- **Pause** halts execution, but does not exit, thus allowing you to resume execution.
- **Terminate** halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection** removes invalid objects from the cache in favor of more frequently accessed and more valid objects.

These options are also available as icons on the Debugging tab of the output window.

Database Reporting

SQL Developer provides a number of predefined reports about the database and its objects.



The screenshot shows the Oracle SQL Developer interface. The left sidebar has a tree view under 'All Reports' with categories like 'About Your Database', 'Data Dictionary Reports', 'All Objects', 'Collection Types', 'Dependencies', etc. The 'Dependencies' node is expanded, showing a list of database objects. The main pane displays a table titled 'Dependencies' with columns: Owner, Name, Type, Referenced_Owner, Referenced_Name, and Referenced_Type. The table lists numerous objects from the APEX_040100 schema, such as procedures, functions, views, and tables, along with their referenced objects like MM_FLOW, MM_FLOW_ISC, etc. The top bar shows tabs for 'myconnection', 'ADD_JOB_HISTORY', and 'Dependencies'. The 'Dependencies' tab is active, indicated by a red box around its title. The bottom right corner of the interface contains the ORACLE logo and the copyright notice: 'Copyright © 2017, Oracle and/or its affiliates. All rights reserved.'

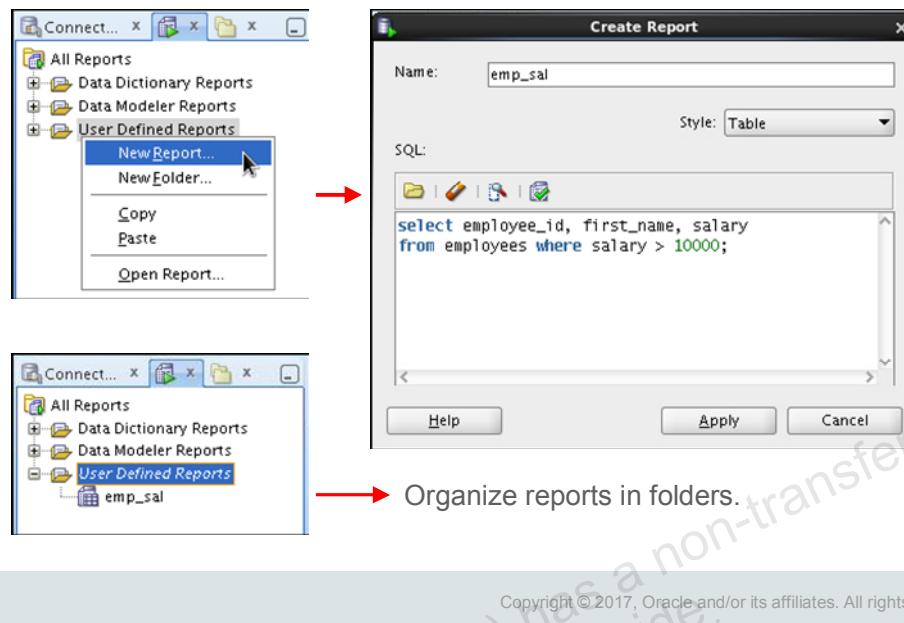
SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab on the left of the window. Individual reports are displayed in tabbed panes on the right of the window; for each report, you can select (by using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following steps:

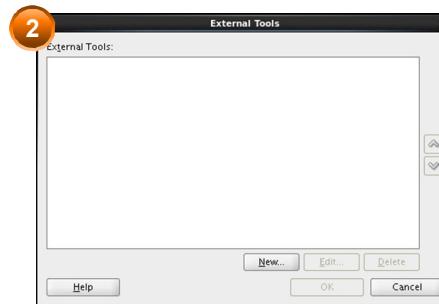
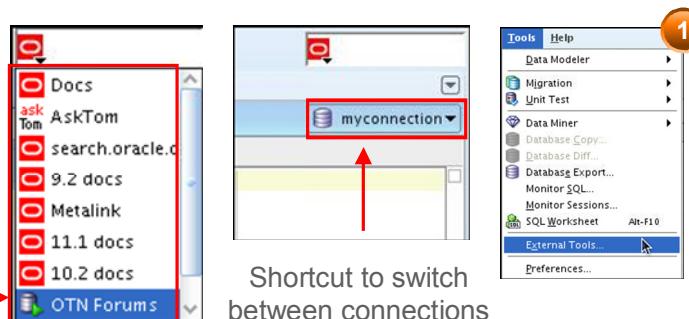
1. Right-click the User Defined Reports node under Reports and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

In the example in the slide, the report name is specified as `emp_sal`. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

You can organize user-defined reports in folders and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` in the directory for user-specific information.

Search Engines and External Tools

Links to popular search engines and discussion forums



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

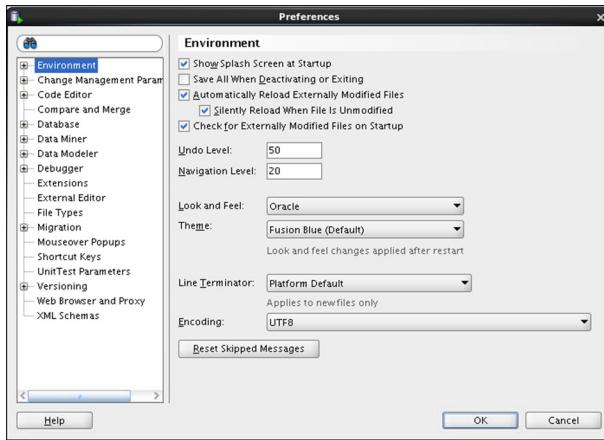
To enhance the productivity of developers, SQL Developer has added quick links to popular search engines and discussion forums such as AskTom, Google, and so on. Also, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

You can add external tools to the existing list or even delete shortcuts to the tools that you do not use frequently. To do so, perform the following steps:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



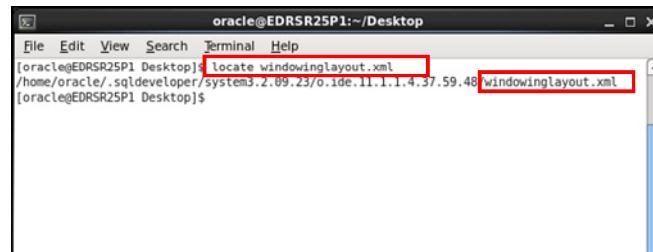
You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your needs. To modify SQL Developer preferences, select Tools, and then Preferences.

The preferences are grouped into the following categories:

- Environment
- Change Management Parameter
- Code Editors
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types

- Migration
- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

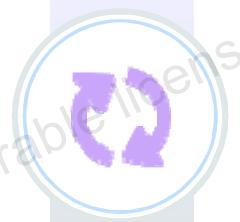
Resetting the SQL Developer Layout



```
oracle@EDRSR2SP1:~/Desktop
File Edit View Search Terminal Help
[oracle@EDRSR2SP1 Desktop] locate windowinglayout.xml
/home/oracle/.sqldeveloper/system3.2.09.23/o.ide.11.1.1.4.37.59.48>windowinglayout.xml
[oracle@EDRSR2SP1 Desktop]$
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

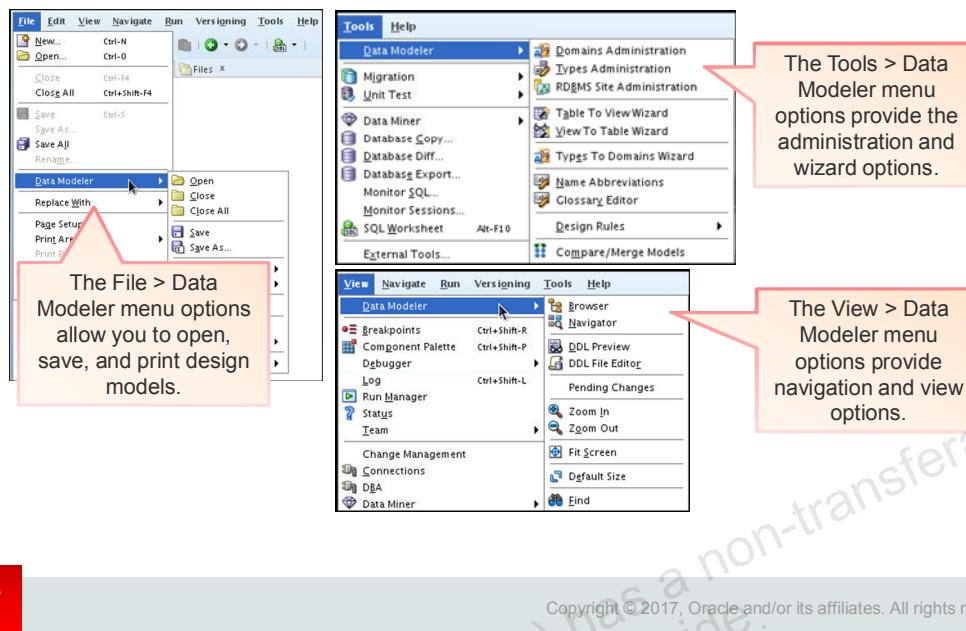


While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit SQL Developer.
2. Open a terminal window and use the locate command to find the location of windowinglayout.xml.
3. Go to the directory that has windowinglayout.xml and delete it.
4. Restart SQL Developer.

Data Modeler in SQL Developer

SQL Developer includes an integrated version of SQL Developer Data Modeler.



Using the integrated version of the SQL Developer Data Modeler, you can:

- Create, open, import, and save a database design
- Create, modify, and delete Data Modeler objects

To display Data Modeler in a pane, click Tools, and then Data Modeler. The Data Modeler menu under Tools includes additional commands, for example, that enable you to specify design rules and preferences.

Summary

In this appendix, you should have learned how to use SQL Developer to:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the Data Modeling options in SQL Developer



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Using SQL*Plus

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license
to use this Student Guide.

Objectives

After completing this appendix, you should be able to do the following:

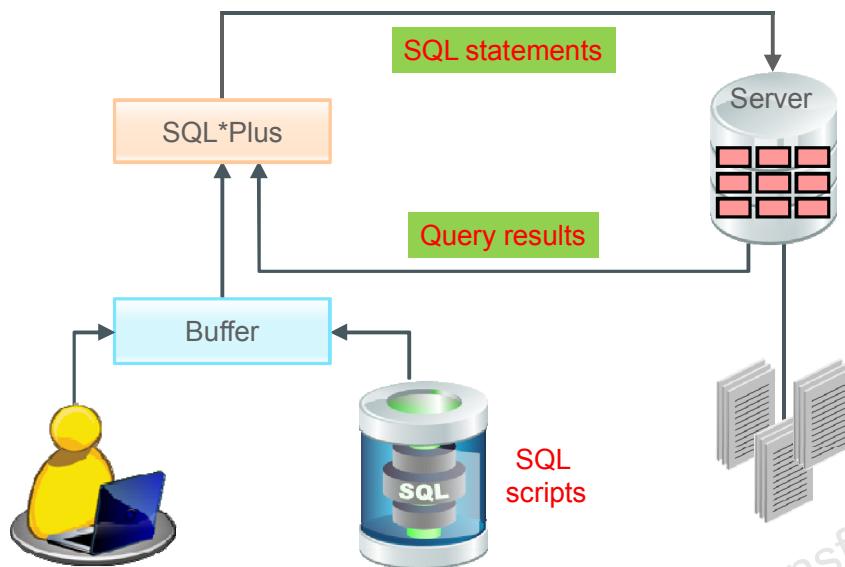
- Log in to SQL*Plus
- Edit SQL commands
- Format the output by using SQL*Plus commands
- Interact with script files



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL and SQL*Plus Interaction



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Features of SQL*Plus

- Accepts ad hoc entry of statements
- Accepts SQL input from files
- Provides a line editor for modifying SQL statements
- Controls environmental settings
- Formats query results into basic reports
- Accesses local and remote databases

SQL Statements Versus SQL*Plus Commands

SQL

- A language
- ANSI-standard
- Keywords cannot be abbreviated.
- Statements manipulate data and table definitions in the database.



SQL*Plus

- An environment
- Oracle-proprietary
- Keywords can be abbreviated.
- Commands do not allow manipulation of values in the database.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI)-standard SQL	Is the Oracle-proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (-) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

SQL*Plus: Overview

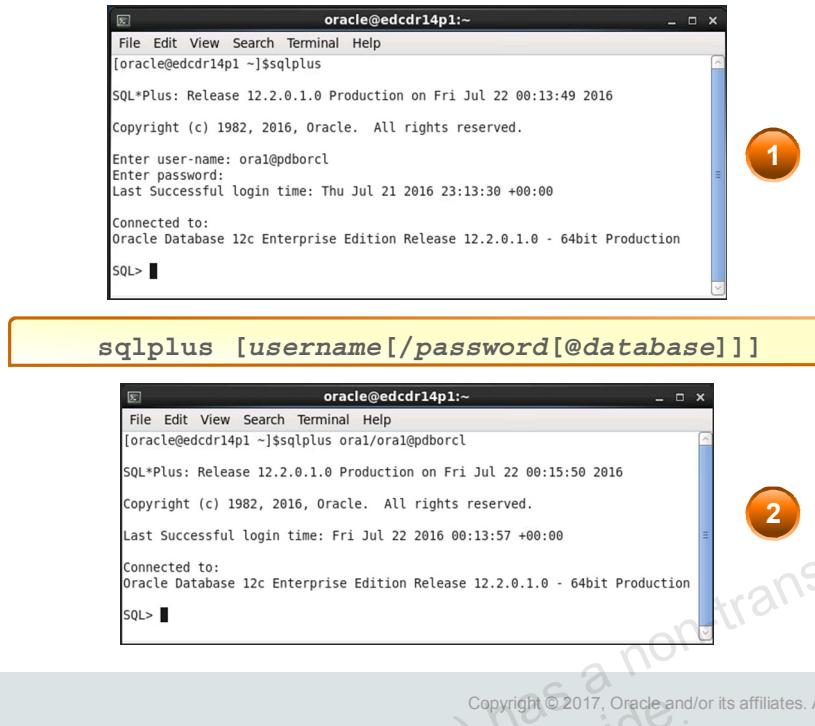
- Log in to SQL*Plus.
- Describe the table structure.
- Edit your SQL statement.
- Execute SQL from SQL*Plus.
- Save SQL statements to files and append SQL statements to files.
- Execute saved files.
- Load commands from the file to buffer to edit.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Logging in to SQL*Plus



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

How you invoke SQL*Plus depends on the type of operating system that you are running Oracle Database on.

To log in from a Linux environment, perform the following steps:

1. Right-click your Linux desktop and select terminal.
2. Enter the `sqlplus` command shown in the slide.
3. Enter the username, password, and database name.

In the syntax:

<code>username</code>	Your database username
<code>password</code>	Your database password (Your password is visible if you enter it here.)
<code>@database</code>	The database connect string

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the password prompt.

Displaying the Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table:

```
DESC [RIBE] tablename
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In SQL*Plus, you can display the structure of a table by using the DESCRIBE command. The result of the command is a display of column names and data types, as well as an indication if a column must contain data.

In the syntax:

tablename The name of any existing table, view, or synonym that is accessible to the user

To describe the DEPARTMENTS table, use the following command:

```
SQL> DESCRIBE DEPARTMENTS
```

Name	Null	Type
DEPARTMENT_ID		NOT NULL NUMBER(4)
DEPARTMENT_NAME		NOT NULL VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Displaying the Table Structure

```
DESCRIBE departments
```

Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER (4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2 (30)
MANAGER_ID		NUMBER (6)
LOCATION_ID		NUMBER (4)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL*Plus Editing Commands

- A [PPEND] *text*
- C [HANGE] / *old* / *new*
- C [HANGE] / *text* /
- CL [EAR] BUFF [ER]
- DEL
- DEL *n*
- DEL *m n*



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Guidelines

- If you press Enter before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing Enter twice. The SQL prompt appears.

Command	Description
A [PPEND] <i>text</i>	Adds <i>text</i> to the end of the current line
C [HANGE] / <i>old</i> / <i>new</i>	Changes <i>old</i> text to <i>new</i> in the current line
C [HANGE] / <i>text</i> /	Deletes <i>text</i> from the current line
CL [EAR] BUFF [ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL <i>n</i>	Deletes line <i>n</i>
DEL <i>m n</i>	Deletes lines <i>m</i> to <i>n</i> inclusive

SQL*Plus Editing Commands

- I [NPUT]
- I [NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- O *text*



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can enter only one SQL*Plus command for each SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).

Command	Description
I [NPUT]	Inserts an indefinite number of lines
I [NPUT] <i>text</i>	Inserts a line consisting of <i>text</i>
L[IST]	Lists all lines in the SQL buffer
L[IST] <i>n</i>	Lists one line (specified by <i>n</i>)
L[IST] <i>m n</i>	Lists a range of lines (<i>m</i> to <i>n</i>) inclusive
R[UN]	Displays and runs the current SQL statement in the buffer
<i>n</i>	Specifies the line to make the current line
<i>n text</i>	Replaces line <i>n</i> with <i>text</i>
O <i>text</i>	Inserts a line before line 1

Using LIST, n, and APPEND

```
LIST
 1  SELECT last_name
 2* FROM employees
```

```
1
1* SELECT last_name
```

```
A , job_id
1* SELECT last_name, job_id
```

```
LIST
 1  SELECT last_name, job_id
 2* FROM employees
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- Use the L [IST] command to display the contents of the SQL buffer. The asterisk (*) beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number (n) of the line that you want to edit. The new current line is displayed.
- Use the A [PPEND] command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the LIST command.

Note: Many SQL*Plus commands, including LIST and APPEND, can be abbreviated to just their first letter. LIST can be abbreviated to L and APPEND to A.

Using the CHANGE Command

```
LIST  
1* SELECT * from employees
```

```
c/employees/departments  
1* SELECT * from departments
```

```
LIST  
1* SELECT * from departments
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- Use L [IST] to display the contents of the buffer.
- Use the C [HANGE] command to alter the contents of the current line in the SQL buffer. In this case, replace the employees table with the departments table. The new current line is displayed.
- Use the L [IST] command to verify the new contents of the buffer.

SQL*Plus File Commands

- `SAVE filename`
- `GET filename`
- `START filename`
- `@ filename`
- `EDIT filename`
- `SPOOL filename`
- `EXIT`



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
<code>SAV[E] filename [.ext]</code> [REP[LACE] APP[END]]	Saves the current contents of the SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
<code>GET filename [.ext]</code>	Writes the contents of a previously saved file to the SQL buffer. The default extension for the file name is .sql.
<code>STA[RT] filename [.ext]</code>	Runs a previously saved command file
<code>@ filename</code>	Runs a previously saved command file (same as START)
<code>ED[IT]</code>	Invokes the editor and saves the buffer contents to a file named afiedt.buf
<code>ED[IT] [filename [.ext]]</code>	Invokes the editor to edit the contents of a saved file
<code>SPO[OL] [filename [.ext]] OFF OUT</code>	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the printer.
<code>EXIT</code>	Quits SQL*Plus

Using the SAVE and START Commands

LIST

```
1  SELECT last_name, manager_id, department_id  
2* FROM employees
```

SAVE my_query

Created file my_query

START my_query

LAST_NAME

MANAGER_ID DEPARTMENT_ID

King 90

Kochhar 100 90

...

107 rows selected.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SAVE

Use the **SAVE** command to store the current contents of the buffer in a file. Thus, you can store frequently used scripts for use in the future.

START

Use the **START** command to run a script in SQL*Plus. You can also, alternatively, use the symbol @ to run a script.

```
@my_query
```

SERVEROUTPUT Command

- Use the SET SERVEROUT [PUT] command to control whether to display the output of stored procedures or PL/SQL blocks in SQL*Plus.
- The DBMS_OUTPUT line length limit is increased from 255 bytes to 32767 bytes.
- The default size is now unlimited.
- Resources are not pre-allocated when SERVEROUTPUT is set.
- Because there is no performance penalty, use UNLIMITED unless you want to conserve physical memory.

```
SET SERVEROUT [PUT] {ON | OFF} [SIZE {n | UNL [IMITED]}]
[FOR [MAT] {WRA [PPED] | WOR [D_WRAPPED] | TRU [NCATED]}]
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Most of the PL/SQL programs perform input and output through SQL statements, to store data in database tables or query those tables. All other PL/SQL input/output is done through APIs that interact with other programs. For example, the DBMS_OUTPUT package has procedures, such as PUT_LINE. To see the result outside of PL/SQL, you require another program, such as SQL*Plus, to read and display the data passed to DBMS_OUTPUT.

SQL*Plus does not display DBMS_OUTPUT data unless you first issue the SQL*Plus command SET SERVEROUTPUT ON as follows:

```
SET SERVEROUTPUT ON
```

Notes

- SIZE sets the number of bytes of the output that can be buffered within the Oracle Database server. The default is UNLIMITED. n cannot be less than 2000 or greater than 1,000,000.
- For additional information about SERVEROUTPUT, see *Oracle Database PL/SQL User's Guide and Reference 12c*.

Using the SQL*Plus SPOOL Command

```
SPO[OL] [file_name[.ext]] [CRE[ATE] | REP[LACE] |  
APP[END]] | OFF | OUT]
```

Option	Description
file_name [. ext]	Spools output to the specified file name
CRE [ATE]	Creates a new file with the name specified
REP [LACE]	Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file.
APP [END]	Adds the contents of the buffer to the end of the file that you specify
OFF	Stops spooling
OUT	Stops spooling and sends the file to your computer's standard (default) printer



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The SPOOL command stores query results in a file or optionally sends the file to a printer. The SPOOL command has been enhanced. You can now append to, or replace an existing file, where previously you could use SPOOL only to create (and replace) a file. REPLACE is the default.

To spool the output generated by commands in a script without displaying the output on screen, use SET TERMOUT OFF. SET TERMOUT OFF does not affect the output from commands that run interactively.

You must use quotation marks around file names that contain white space. To create a valid HTML file using SPOOL APPEND commands, you must use PROMPT or a similar command to create the HTML page header and footer. The SPOOL APPEND command does not parse HTML tags. SET SQLPLUSCOMPAT [IBILITY] to 9.2 or earlier to disable the CREATE, APPEND, and SAVE parameters.

Using the AUTOTRACE Command

- It displays a report after the successful execution of SQL DML statements such as SELECT, INSERT, UPDATE, and DELETE.
- The report can now include execution statistics and the query execution path.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]  
[STATISTICS]
```

```
SET AUTOTRACE ON  
-- The AUTOTRACE report includes both the optimizer  
-- execution path and the SQL statement execution  
-- statistics
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this appendix, you should have learned how to use SQL*Plus as an environment to:

- Execute SQL statements
- Edit SQL statements
- Format the output
- Interact with script files



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

ORACLE

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- Execute a basic SELECT statement
- Create, alter, and drop a table using data definition language (DDL) statements
- Insert, update, and delete rows from one or more tables using data manipulation language (DML) statements
- Commit, roll back, and create savepoints by using transaction control statements
- Perform join operations on one or more tables



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Basic SELECT Statement

- Use the SELECT statement to:
 - Identify the columns to be displayed
 - Retrieve data from one or more tables, object tables, views, object views, or materialized views
- A SELECT statement is also known as a query because it queries a database.
- Syntax:

```
SELECT { * | [DISTINCT] column|expression [alias],... }  
FROM    table;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In its simplest form, a SELECT statement must include the following:

- A SELECT clause, which specifies the columns to be displayed
- A FROM clause, which identifies the table containing the columns that are listed in the SELECT clause

In the syntax:

SELECT	Is a list of one or more columns
*	Selects all columns
DISTINCT	Suppresses duplicates
column / expression	Selects the named column or the expression
alias	Gives different headings to the selected columns
FROM table	Specifies the table containing the columns

Note: Throughout this course, the words keyword, clause, and statement are used as follows:

- A keyword refers to an individual SQL (for example, SELECT and FROM are keywords).
- A clause is a part of a SQL statement (for example, SELECT employee_id, last_name).
- A statement is a combination of two or more clauses (for example, SELECT * FROM employees).

SELECT Statement

- Select all columns:

```
SELECT *  
FROM job_history;
```

#	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA REP	80
9	176	01-JAN-07	31-DEC-07	SA MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Select specific columns:

```
SELECT manager_id, job_id  
FROM employees;
```

#	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2	100	AD_VP
3	100	AD_VP
4	102	IT_PROG
5	103	IT_PROG
6	103	IT_PROG
7	100	ST_MAN
8	124	ST_CLERK
9	124	ST_CLERK
10	124	ST_CLERK



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can display all columns of data in a table by following the `SELECT` keyword with an asterisk (*) or by listing all the column names after the `SELECT` keyword. The first example in the slide displays all the rows from the `job_history` table. Specific columns of the table can be displayed by specifying the column names, separated by commas. The second example in the slide displays the `manager_id` and `job_id` columns from the `employees` table.

In the `SELECT` clause, specify the columns in the order in which you want them to appear in the output. For example, the following SQL statement displays the `location_id` column before displaying the `department_id` column:

```
SELECT location_id, department_id FROM departments;
```

Note: You can enter your SQL statement in SQL Worksheet and click the Run Statement icon or press F9 to execute a statement in SQL Developer. The output displayed on the Results tabbed page appears as shown in the slide.

WHERE Clause

- Use the optional WHERE clause to:
 - Filter rows in a query
 - Produce a subset of rows
- Syntax:

```
SELECT * FROM table  
[WHERE condition];
```

- Example:

```
SELECT location_id from departments  
WHERE department_name = 'Marketing';
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORDER BY Clause

- Use the optional ORDER BY clause to specify the row order.
- Syntax:

```
SELECT * FROM table  
[WHERE condition]  
[ORDER BY {<column>|<position>} [ASC|DESC] [, ...] ];
```

- Example:

```
SELECT last_name, department_id, salary  
FROM employees  
ORDER BY department_id ASC, salary DESC;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

GROUP BY Clause

- Use the optional GROUP BY clause to group columns with matching values into subsets.
- Each group has no two rows with the same value for the grouping column or columns.
- Syntax:

```
SELECT <column1, column2, ... column_n>
  FROM table
  [WHERE condition]
  [GROUP BY <column> [, ...] ]
  [ORDER BY <column> [, ...] ] ;
```

- Example:

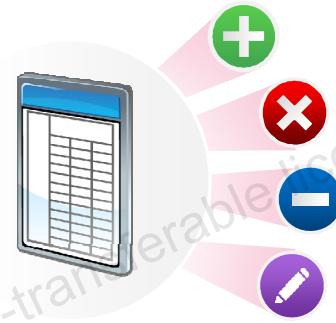
```
SELECT department_id, MIN(salary), MAX (salary)
  FROM employees
 GROUP BY department_id ;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Data Definition Language

- DDL statements are used to define, structurally change, and drop schema objects.
- The commonly used DDL statements are:
 - CREATE TABLE, ALTER TABLE, and DROP TABLE
 - GRANT, REVOKE
 - TRUNCATE



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

DDL statements enable you to alter the attributes of an object without altering the applications that access the object. You can also use DDL statements to alter the structure of objects while database users are performing work in the database. These statements are most frequently used to:

- Create, alter, and drop schema objects and other database structures, including the database itself and database users
- Delete all the data in schema objects without removing the structure of these objects
- Grant and revoke privileges and roles

Oracle Database implicitly commits the current transaction before and after every DDL statement.

CREATE TABLE Statement

- Use the CREATE TABLE statement to create a table in the database.
- Syntax:

```
CREATE TABLE tablename (
  {column-definition | Table-level constraint}
  [ , {column-definition | Table-level constraint} ] * )
```

- Example:

```
CREATE TABLE teach_dept (
  department_id NUMBER(3) PRIMARY KEY,
  department_name VARCHAR2(10));
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To create a table, you must have the CREATE TABLE privilege and a storage area in which to create objects.

The table owner and the database owner automatically gain the following privileges on the table after it is created:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

The table owner and the database owner can grant the preceding privileges to other users.

ALTER TABLE Statement

- Use the ALTER TABLE statement to modify the definition of an existing table in the database.
- Example1:

```
ALTER TABLE teach_dept  
ADD location_id NUMBER NOT NULL;
```

- Example 2:

```
ALTER TABLE teach_dept  
MODIFY department_name VARCHAR2(30) NOT NULL;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The ALTER TABLE statement allows you to make changes to an existing table.

You can:

- Add a column to a table
- Add a constraint to a table
- Modify an existing column definition
- Drop a column from a table
- Drop an existing constraint from a table
- Increase the width of the VARCHAR and CHAR columns
- Change a table to have read-only status

Example 1 in the slide adds a new column called location_id to the teach_dept table.

Example 2 updates the existing department_name column from VARCHAR2 (10) to VARCHAR2 (30), and adds a NOT NULL constraint to it.

DROP TABLE Statement

- The DROP TABLE statement removes the table and all its data from the database.
- Example:

```
DROP TABLE teach_dept;
```

- DROP TABLE with the PURGE clause drops the table and releases the space that is associated with it.

```
DROP TABLE teach_dept PURGE;
```

- The CASCADE CONSTRAINTS clause drops all referential integrity constraints from the table.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The DROP TABLE statement allows you to remove a table and its contents from the database, and pushes it to the recycle bin. Dropping a table invalidates dependent objects and removes object privileges on the table.

Use the PURGE clause along with the DROP TABLE statement to release back to the tablespace the space allocated for the table. You cannot roll back a DROP TABLE statement with the PURGE clause, nor can you recover the table if you have dropped it with the PURGE clause.

The CASCADE CONSTRAINTS clause allows you to drop the reference to the primary key and unique keys in the dropped table.

GRANT Statement

- The GRANT statement assigns the privilege to perform the following operations:
 - Insert or delete data.
 - Create a foreign key reference to the named table or to a subset of columns from a table.
 - Select data, a view, or a subset of columns from a table.
 - Create a trigger on a table.
 - Execute a specified function or procedure.
- Example:

```
GRANT SELECT any table to PUBLIC;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the GRANT statement to:

- Assign privileges to a specific user or role, or to all users, to perform actions on database objects
- Grant a role to a user, to PUBLIC, or to another role

Before you issue a GRANT statement, check that the `derby.database.sql` authorization property is set to True. This property enables SQL Authorization mode. You can grant privileges on an object if you are the owner of the database.

You can grant privileges to all users by using the PUBLIC keyword. When PUBLIC is specified, the privileges or roles affect all current and future users.

Privilege Types

Assign the following privileges by using the GRANT statement:

- ALL PRIVILEGES
- DELETE
- INSERT
- REFERENCES
- SELECT
- UPDATE



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

REVOKE Statement

- Use the REVOKE statement to remove privileges from a user to perform actions on database objects.
- Revoke a system privilege from a user:

```
REVOKE DROP ANY TABLE  
  FROM hr;
```

- Revoke a role from a user:

```
REVOKE dw_manager  
  FROM sh;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The REVOKE statement removes privileges from a specific user (or users) or role to perform actions on database objects. It performs the following operations:

- Revokes a role from a user, from PUBLIC, or from another role
- Revokes privileges for an object if you are the owner of the object or the database owner

Note: To revoke a role or system privilege, you must have been granted the privilege with the ADMIN OPTION.

TRUNCATE TABLE Statement

- Use the TRUNCATE TABLE statement to remove all the rows from a table.
- Example:

```
TRUNCATE TABLE employees_demo;
```

- By default, Oracle Database performs the following tasks:
 - De-allocates space used by the removed rows
 - Sets the NEXT storage parameter to the size of the last extent removed from the segment by the truncation process



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Data Manipulation Language

- DML statements query or manipulate data in the existing schema objects.
- A DML statement is executed when:
 - New rows are added to a table by using the `INSERT` statement
 - Existing rows in a table are modified by using the `UPDATE` statement
 - Existing rows are deleted from a table by using the `DELETE` statement
- A transaction consists of a collection of DML statements that form a logical unit of work.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

DML statements enable you to query or change the contents of an existing schema object. These statements are most frequently used to:

- Add new rows of data to a table or view by specifying a list of column values or using a subquery to select and manipulate existing data
- Change column values in the existing rows of a table or view
- Remove rows from tables or views

A collection of DML statements that forms a logical unit of work is called a transaction. Unlike DDL statements, DML statements do not implicitly commit the current transaction.

INSERT Statement

- Use the `INSERT` statement to add new rows to a table.
- Syntax:

```
INSERT INTO table [(column [, column...])]  
VALUES (value [, value...]);
```

- Example:

```
INSERT INTO departments  
VALUES (200, 'Development', 104, 1400);  
1 rows inserted.
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `INSERT` statement adds rows to a table. Make sure to insert a new row containing values for each column and to list the values in the default order of the columns in the table. Optionally, you can also list the columns in the `INSERT` statement.

Example:

```
INSERT INTO job_history (employee_id, start_date, end_date,  
job_id)  
VALUES (120, '25-JUL-06', '12-FEB-08', 'AC_ACCOUNT');
```

The syntax discussed in the slide allows you to insert a single row at a time. The `VALUES` keyword assigns the values of expressions to the corresponding columns in the column list.

UPDATE Statement Syntax

- Use the UPDATE statement to modify the existing rows in a table.
- Update more than one row at a time (if required).

```
UPDATE    table  
SET        column = value [, column = value, ...]  
[WHERE     condition];
```

- Example:

```
UPDATE    copy_emp  
SET  
[22 rows updated]
```

- Specify SET *column_name*= NULL to update a column value to NULL.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The UPDATE statement modifies the existing values in a table. Confirm the update operation by querying the table to display the updated rows. You can modify a specific row or rows by specifying the WHERE clause.

Example:

```
UPDATE employees  
SET      salary = 17500  
WHERE   employee_id = 102;
```

In general, use the primary key column in the WHERE clause to identify the row to update. For example, to update a specific row in the employees table, use employee_id to identify the row instead of employee_name, because more than one employee may have the same name.

Note: Typically, the condition keyword is composed of column names, expressions, constants, subqueries, and comparison operators.

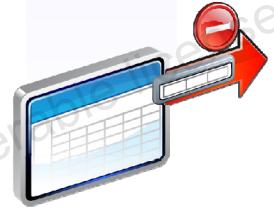
DELETE Statement

- Use the DELETE statement to delete the existing rows from a table.
- Syntax:

```
DELETE [FROM] table  
[WHERE condition];
```

- Write the DELETE statement by using the WHERE clause to delete specific rows from a table.

```
DELETE FROM departments  
WHERE department_name = 'Finance';  
1 rows deleted
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The DELETE statement removes existing rows from a table. You must use the WHERE clause to delete a specific row or rows from a table based on the condition. The condition identifies the rows to be deleted. It may contain column names, expressions, constants, subqueries, and comparison operators.

The first example deletes the finance department from the departments table. You can confirm the delete operation by using the SELECT statement to query the table.

```
SELECT *  
FROM departments  
WHERE department_name = 'Finance';
```

If you omit the WHERE clause, all rows in the table are deleted.

Example:

```
DELETE FROM copy_emp;
```

The preceding example deletes all the rows from the copy_emp table.

Transaction Control Statements

- Transaction control statements are used to manage the changes made by DML statements.
- The DML statements are grouped into transactions.
- Transaction control statements include:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

COMMIT Statement

- Use the COMMIT statement to:
 - Permanently save the changes made to the database during the current transaction
 - Erase all savepoints in the transaction
 - Release transaction locks
- Example:

```
INSERT INTO departments
VALUES      (201, 'Engineering', 106, 1400);
COMMIT;
1 rows inserted.
committed.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The COMMIT statement ends the current transaction by making all the pending data changes permanent. It releases all row and table locks, and erases any savepoints that you may have marked since the last commit or rollback. The changes made using the COMMIT statement are visible to all users.

Oracle recommends that you explicitly end every transaction in your application programs with a COMMIT or ROLLBACK statement, including the last transaction, before disconnecting from Oracle Database. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction is automatically rolled back.

Note: Oracle Database issues an implicit COMMIT before and after any DDL statement.

ROLLBACK Statement

- Use the ROLLBACK statement to undo changes made to the database during the current transaction.
- Use the TO SAVEPOINT clause to undo a part of the transaction after the savepoint.
- Example:

```
UPDATE      employees
SET         salary = 7000
WHERE        last_name = 'Ernst';
SAVEPOINT   Ernst_sal;

UPDATE      employees
SET         salary = 12000
WHERE        last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ernst_sal;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SAVEPOINT Statement

- Use the SAVEPOINT statement to name and mark the current point in the processing of a transaction.
- Specify a name to each savepoint.
- Use distinct savepoint names within a transaction to avoid overriding.
- Syntax:

```
SAVEPOINT savepoint;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Joins

Use a join to query data from more than one table:

```
SELECT table1.column, table2.column  
FROM   table1, table2  
WHERE  table1.column1 = table2.column2;
```

- Write the join condition in the WHERE clause.
- Prefix the column name with the table name when the same column name appears in more than one table.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When data from more than one table in the database is required, a *join* condition is used. Rows in one table can be joined to rows in another table according to common values that exist in the corresponding columns (usually primary and foreign key columns).

To display data from two or more related tables, write a simple join condition in the WHERE clause.

In the syntax:

table1.column
table1.column1 =
table2.column2

Denotes the table and column from which data is retrieved
Is the condition that joins (or relates) the tables together

Guidelines

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join n tables together, you need a minimum of $n-1$ join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

Types of Joins

- Natural join
- Equijoin
- Nonequijoin
- Outer join
- Self-join
- Cross join



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use table aliases instead of full table name prefixes.
- Table aliases give a table a shorter name.
 - This keeps SQL code smaller and uses less memory.
- Use column aliases to distinguish columns that have identical names but reside in different tables.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column in the SELECT list could be from either the DEPARTMENTS table or the EMPLOYEES table. Therefore, it is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using a table prefix improves performance because you tell the Oracle server exactly where to find the columns.

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. Therefore, you can use table aliases, instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, thereby using less memory.

The table name is specified in full, followed by a space, and then the table alias. For example, the EMPLOYEES table can be given an alias of e, and the DEPARTMENTS table an alias of d.

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the `FROM` clause, that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.
- A table alias is valid only for the current `SELECT` statement.

Natural Join

- The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
- It selects rows from tables that have the same names and data values of columns.
- Example:

```
SELECT country_id, location_id, country_name, city
FROM countries NATURAL JOIN locations;
```

COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1 US	1400	United States of America	Southlake
2 US	1500	United States of America	San Francisco
3 US	1700	United States of America	Seattle
4 CA	1800	Canada	Toronto
5 UK	2500	United Kingdom	Oxford

ORACLE

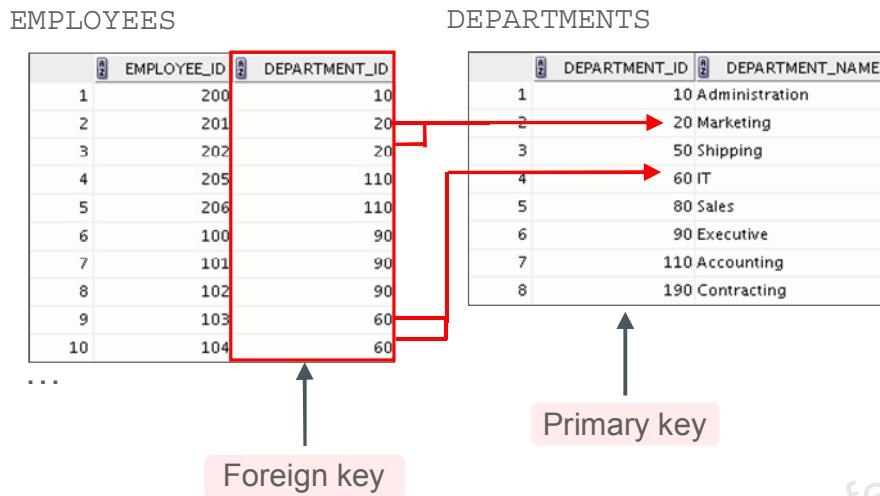
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the NATURAL JOIN keywords.

Note: The join can happen only on those columns that have the same names and data types in both tables. If the columns have the same name but different data types, the NATURAL JOIN syntax causes an error.

In the example in the slide, the COUNTRIES table is joined to the LOCATIONS table by the COUNTRY_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Equijoins



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

An equijoin is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. To determine an employee's department name, you compare the values in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin; that is, values in the DEPARTMENT_ID column in both the tables must be equal. Often, this type of join involves primary and foreign key complements.

Note: Equijoins are also called simple joins.

Retrieving Records with Equijoins

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200 Whalen	10	10	1700
2	201 Hartstein	20	20	1800
3	202 Fay	20	20	1800
4	144 Vargas	50	50	1500
5	143 Matos	50	50	1500
6	142 Davies	50	50	1500
7	141 Raji	50	50	1500
8	124 Mourgos	50	50	1500
9	103 Hunold	60	60	1400
10	104 Ernst	60	60	1400
11	107 Lorentz	60	60	1400
...				



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide:

- **The SELECT clause specifies the column names to retrieve:**
 - Employee last name, employee ID, and department ID, which are columns in the EMPLOYEES table
 - Department ID and location ID, which are columns in the DEPARTMENTS table
- **The FROM clause specifies the two tables that the database must access:**
 - EMPLOYEES table
 - DEPARTMENTS table
- **The WHERE clause specifies how the tables are to be joined:**

e.department_id = d.department_id

Because the DEPARTMENT_ID column is common to both tables, it must be prefixed with the table alias to avoid ambiguity. Other columns that are not present in both the tables need not be qualified by a table alias, but it is recommended for better performance.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a “_1” to differentiate between the two DEPARTMENT_IDS.

Additional Search Conditions Using the AND and WHERE Operators

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In addition to the join, you may have criteria for your WHERE clause to restrict the rows in consideration for one or more tables in the join. The example in the slide performs a join on the DEPARTMENTS and LOCATIONS tables and, in addition, displays only those departments with ID equal to 20 or 50. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions.

Both queries produce the same output.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON    e.salary
      BETWEEN j.lowest_sal AND j.highest_sal;
```

#	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Fay	6000	C

...

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates a nonequijoin to evaluate an employee's salary grade. The salary must be between any pair of low and high salary ranges.

It is important to note that all employees appear only once when this query is executed. There are two reasons for this:

- None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and the high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits that are provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

Note: Other conditions (such as `<=` and `>=`) can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using the `BETWEEN` condition. The Oracle server translates the `BETWEEN` condition to a pair of `AND` conditions. Therefore, using `BETWEEN` has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the example in the slide for performance reasons, not because of possible ambiguity.

Retrieving Records by Using the USING Clause

- You can use the USING clause to match only one column when more than one column matches.
- You cannot specify this clause with a NATURAL join.
- Do not qualify the column name with a table name or table alias.
- Example:

```
SELECT country_id, country_name, location_id, city
FROM   countries JOIN locations
USING (country_id) ;
```

#	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the COUNTRY_ID columns in the COUNTRIES and LOCATIONS tables are joined and thus the LOCATION_ID of the location where an employee works is shown.

Retrieving Records by Using the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the `ON` clause to specify arbitrary conditions or specify columns to join.
- The `ON` clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name, j.department_id,  
FROM   employees e JOIN job_history j  
ON     (e.employee_id = j.employee_id);
```

#	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101 Kochhar		110
2	101 Kochhar		110
3	102 De Haan		60
4	176 Taylor		80
5	176 Taylor		80
6	200 Whalen		90
7	200 Whalen		90
8	201 Hartstein		20



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Left Outer Join

- A join between two tables that returns all matched rows as well as the unmatched rows from the left table is called a LEFT OUTER JOIN.
- Example:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c [LEFT OUTER JOIN] locations l
ON    (c.country_id = l.country_id) ;
```

COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1 CA	Canada	1800	Toronto
2 DE	Germany	(null)	(null)
3 UK	United Kingdom	2500	Oxford
4 US	United States of America	1400	Southlake
5 US	United States of America	1500	South San Francisco
6 US	United States of America	1700	Seattle

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Right Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the right table is called a **RIGHT OUTER JOIN**.
- Example:

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Davies	50	Shipping
...		
Higgins	110	Accounting
Gietz	110	Accounting
(null)	190	Contracting



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Full Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from both tables is called a FULL OUTER JOIN.
- Example:

```
SELECT e.last_name, d.department_id, d.manager_id,
       d.department_name
  FROM employees e FULL OUTER JOIN departments d
 WHERE (e.manager_id = d.manager_id) ;
```

LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1 King	(null)	(null)	(null)
2 Kochhar	90	100	Executive
3 De Haan	90	100	Executive
4 Hunold	(null)	(null)	(null)
...			
19 Higgins	(null)	(null)	(null)
20 Gietz	110	205	Accounting
21 (null)	190	(null)	Contracting
22 (null)	10	200	Administration



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This query retrieves all the rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all the rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Self-Join: Example

```
SELECT worker.last_name || ' works for '
    || manager.last_name
  FROM employees worker JOIN employees manager
 WHERE worker.manager_id = manager.employee_id
  ORDER BY worker.last_name;
```

#	WORKER.LAST_NAME "WORKSFOR' MANAGER.LAST_NAME
1	Abel works for Zlotkey
2	Davies works for Mourgos
3	De Haan works for King
4	Ernst works for Hunold
5	Fay works for Hartstein
6	Gietz works for Higgins
7	Grant works for Zlotkey
8	Hartstein works for King
9	Higgins works for Kochhar

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cross Join

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- Example:

```
SELECT department_name, city  
FROM department CROSS JOIN location;
```

#	DEPARTMENT_NAME	CITY
1	Administration	Oxford
2	Administration	Seattle
3	Administration	South San Francisco
4	Administration	Southlake
5	Administration	Toronto
6	Marketing	Oxford
7	Marketing	Seattle
8	Marketing	South San Francisco
9	Marketing	Southlake
10	Marketing	Toronto

...

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this appendix, you should have learned how to use:

- The SELECT statement to retrieve rows from one or more tables
- DDL statements to alter the structure of objects
- DML statements to manipulate data in the existing schema objects
- Transaction control statements to manage the changes made by DML statements
- Joins to display data from multiple tables



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Copyright©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to use:

- The ROLLUP operation to produce subtotal values
- The CUBE operation to produce cross-tabulation values
- The GROUPING function to identify the row values created by ROLLUP or CUBE
- GROUPING SETS to produce a single result set



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Review of Group Functions

- Group functions operate on sets of rows to give one result per group.

```
SELECT      [column,] group_function(column) . . .
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Example:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct), MAX(hire_date)
  FROM employees
 WHERE job_id LIKE 'SA%';
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Review of the GROUP BY Clause

- Syntax:

```
SELECT      [column,] group_function(column) . . .
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Example:

```
SELECT      department_id, job_id, SUM(salary),
            COUNT(employee_id)
  FROM        employees
 GROUP BY   department_id, job_id;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Review of the HAVING Clause

- Use the HAVING clause to specify which groups are to be displayed.
- You further restrict the groups on the basis of a limiting condition.

```
SELECT      [column,] group_function(column) ...
FROM        table
[WHERE       condition]
[GROUP BY   group_by_expression]
[HAVING     having_expression]
[ORDER BY   column];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

GROUP BY with ROLLUP and CUBE Operators

- Use ROLLUP or CUBE with GROUP BY to produce superaggregate rows by cross-referencing columns.
- ROLLUP grouping produces a result set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a result set containing the rows from ROLLUP and cross-tabulation rows.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ROLLUP Operator

- ROLLUP is an extension to the GROUP BY clause.
- Use the ROLLUP operation to produce cumulative aggregates, such as subtotals.

```
SELECT      [column, lgroup_function(column)]. . .
FROM        table
[WHERE       condition]
[GROUP BY   ROLLUP group_by_expression]
[HAVING     having_expression];
[ORDER BY   column];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ROLLUP Operator: Example

```
SELECT department_id, job_id, SUM(salary)
  FROM employees
 WHERE department_id < 60
 GROUP BY ROLLUP(department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	440
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

1

2

3



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide:

- Total salaries for every job ID within a department for those departments whose department ID is less than 60 are displayed by the GROUP BY clause
- The ROLLUP operator displays:
 - The total salary for each department whose department ID is less than 60
 - The total salary for all departments whose department ID is less than 60, irrespective of the job IDs

In this example, 1 indicates a group totaled by both DEPARTMENT_ID and JOB_ID, 2 indicates a group totaled only by DEPARTMENT_ID, and 3 indicates the grand total.

The ROLLUP operator creates subtotals that roll up from the most detailed level to a grand total, following the grouping list specified in the GROUP BY clause. First, it calculates the standard aggregate values for the groups specified in the GROUP BY clause (in the example, the sum of salaries grouped on each job within a department). Then it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. (In the example, the sum of salaries for each department is calculated, followed by the sum of salaries for all departments.)

- Given n expressions in the ROLLUP operator of the GROUP BY clause, the operation results in $n + 1$ (in this case, $2 + 1 = 3$) groupings.
- Rows based on the values of the first n expressions are called rows or regular rows, and the others are called superaggregate rows.

CUBE Operator

- CUBE is an extension to the GROUP BY clause.
- You can use the CUBE operator to produce cross-tabulation values with a single SELECT statement.

```
SELECT      [column, ] group_function(column)...
FROM        table
[WHERE      condition]
[GROUP BY   CUBE group_by_expression]
[HAVING    having_expression]
[ORDER BY   column];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

CUBE Operator: Example

```
SELECT      department_id, job_id, SUM(salary)
FROM        employees
WHERE       department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	(null) (null)		211200
2	(null) HR_REP		6500
3	(null) MK_MAN		13000
4	(null) MK_REP		6000
5	(null) PU_MAN		11000
6	(null) ST_MAN		36400
7	(null) AD_ASST		4400
8	(null) PU_CLERK		13900
9	(null) SH_CLERK		64300
10	(null) ST_CLERK		55700
11	10 (null)		4400
12	10 AD_ASST		4400
13	20 (null)		19000
14	20 MK_MAN		13000
15	20 MK_REP		6000
16	30 (null)		24900



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The output of the SELECT statement in the example can be interpreted as follows:

- The total salary for every job within a department (for departments with a department ID of lower than 60)
- The total salary for each department with a department ID of lower than 60
- The total salary for each job irrespective of the department
- The total salary for those departments with a department ID of lower than 60, irrespective of the job title

In this example, 1 indicates the grand total, 2 indicates the rows totaled by JOB_ID alone, 3 indicates some of the rows totaled by DEPARTMENT_ID and JOB_ID, and 4 indicates some of the rows totaled by DEPARTMENT_ID alone.

The CUBE operator has also performed the ROLLUP operation to display the subtotals for those departments whose department ID is less than 60 and the total salary for those departments whose department ID is less than 60, irrespective of the job titles. Further, the CUBE operator displays the total salary for every job irrespective of the department.

Note: Similar to the ROLLUP operator, producing subtotals in n dimensions (that is, n columns in the GROUP BY clause) without a CUBE operator requires that 2^n SELECT statements be linked with UNION ALL. Thus, a report with three dimensions requires $2^3 = 8$ SELECT statements to be linked with UNION ALL.

GROUPING Function

The GROUPING function:

- Is used with either the CUBE or ROLLUP operator
- Is used to find the groups forming the subtotal in a row
- Is used to differentiate stored NULL values from NULL values created by ROLLUP or CUBE
- Returns 0 or 1

```
SELECT      [column,] group_function(column) . . .
              GROUPING(expr)
FROM        table
[WHERE      condition]
[GROUP BY  [ROLLUP] [CUBE] group_by_expression]
[HAVING    having_expression]
[ORDER BY  column];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The GROUPING function can be used with either the CUBE or the ROLLUP operator to help you understand how a summary value has been obtained.

It uses a single column as its argument. The *expr* in the GROUPING function must match one of the expressions in the GROUP BY clause. The function returns a value of 0 or 1.

The values returned by the GROUPING function are useful to:

- Determine the level of aggregation of a given subtotal (that is, the group or groups on which the subtotal is based)
- Identify whether a NULL value in the expression column of a row of the result set indicates:
 - A NULL value from the base table (stored NULL value)
 - A NULL value created by ROLLUP or CUBE (as a result of a group function on that expression)

A value of 0 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has been used to calculate the aggregate value.
- The NULL value in the expression column is a stored NULL value.

A value of 1 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has not been used to calculate the aggregate value.
- The NULL value in the expression column is created by ROLLUP or CUBE as a result of grouping.

GROUPING Function: Example

```
SELECT department_id DEPTID, job_id JOB,
       SUM(salary),
       GROUPING(department_id) GRP_DEPT,
       GROUPING(job_id) GRP_JOB
  FROM employees
 WHERE department_id < 50
 GROUP BY ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

GROUPING SETS

- The GROUPING SETS syntax is used to define multiple groupings in the same query.
- All groupings specified in the GROUPING SETS clause are computed and the results of individual groupings are combined with a UNION ALL operation.
- Grouping set efficiency:
 - Only one pass over the base table is required.
 - There is no need to write complex UNION statements.
 - The more elements GROUPING SETS has, the greater is the performance benefit.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

GROUPING SETS is a further extension of the GROUP BY clause that you can use to specify multiple groupings of data. Doing so facilitates efficient aggregation, which in turn facilitates analysis of data across multiple dimensions.

A single SELECT statement can now be written by using GROUPING SETS to specify various groupings (which can also include ROLLUP or CUBE operators), rather than multiple SELECT statements combined by UNION ALL operators.

For example:

```
SELECT department_id, job_id, manager_id, AVG(salary)
  FROM employees
 GROUP BY
 GROUPING SETS
 ((department_id, job_id, manager_id),
 (department_id, manager_id), (job_id, manager_id));
```

This statement calculates aggregates over three groupings:

(department_id, job_id, manager_id), (department_id, manager_id), and
(job_id, manager_id)

Without this feature, multiple queries combined together with UNION ALL are required to obtain the output of the preceding SELECT statement. A multiquery approach is inefficient because it requires multiple scans of the same data.

Compare the previous example with the following alternative:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

This statement computes all the 8 ($2 * 2 * 2$) groupings, though only the (department_id, job_id, manager_id), (department_id, manager_id), and (job_id, manager_id) groups are of interest to you.

Another alternative is the following statement:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

This statement requires three scans of the base table, which makes it inefficient.

CUBE and ROLLUP can be thought of as grouping sets with very specific semantics and results. The following equivalencies show this fact:

CUBE(a, b, c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP(a, b, c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a), ())

GROUPING SETS: Example

```
SELECT department_id, job_id,  
       manager_id, AVG(salary)  
FROM employees  
GROUP BY GROUPING SETS  
((department_id,job_id), (job_id,manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4	...	ST_CLERK	121	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FL_MGR	(null)	12000

2

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The query in the slide calculates aggregates over two groupings. The table is divided into the following groups:

- Department ID, Job ID
- Job ID, Manager ID

The average salaries for each of these groups are calculated. The result set displays the average salary for each of the two groups.

In the output, the group marked as 1 can be interpreted as the following:

- The average salary of all employees with the SH_CLERK job ID under manager 122 is \$ 3,200.
- The average salary of all employees with the AC_MGR job ID under manager 101 is \$ 12,000, and so on.

The group marked as 2 in the output is interpreted as the following:

- The average salary of all employees with the AC_MGR job ID in department 110 is \$ 12,000.
- The average salary of all employees with the AD_PRES job ID in department 90 is \$ 24,000, and so on.

The example in the slide can also be written as:

```
SELECT department_id, job_id, NULL as manager_id,  
       AVG(salary) as AVGSAL  
  FROM employees  
 GROUP BY department_id, job_id  
UNION ALL  
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL  
  FROM employees  
 GROUP BY job_id, manager_id;
```

In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query will need two scans of the base table EMPLOYEES. This could be very inefficient. Therefore, the usage of the GROUPING SETS statement is recommended.

Composite Columns

- A composite column is a collection of columns that are treated as a unit.
`ROLLUP (a, (b, c), d)`
- Use parentheses within the GROUP BY clause to group columns, so that they are treated as a unit while computing ROLLUP or CUBE operations.
- When used with ROLLUP or CUBE, composite columns would require skipping aggregation across certain levels.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A composite column is a collection of columns that are treated as a unit during the computation of groupings. You specify the columns in parentheses as in the following statement: `ROLLUP (a, (b, c), d)`

Here, `(b, c)` forms a composite column and is treated as a unit. In general, composite columns are useful in ROLLUP, CUBE, and GROUPING SETS. For example, in CUBE or ROLLUP, composite columns would require skipping aggregation across certain levels.

That is, `GROUP BY ROLLUP(a, (b, c))` is equivalent to:

`GROUP BY a, b, c UNION ALL`

`GROUP BY a UNION ALL`

`GROUP BY ()`

Here, `(b, c)` is treated as a unit and ROLLUP is not applied across `(b, c)`. It is as though you have an alias—for example, `z` as an alias for `(b, c)`, and the GROUP BY expression reduces to: `GROUP BY ROLLUP(a, z)`.

Note: `GROUP BY ()` is typically a SELECT statement with NULL values for the columns `a` and `b` and only the aggregate function. It is generally used for generating grand totals.

```
SELECT    NULL, NULL, aggregate_col  
FROM      <table_name>  
GROUP BY  () ;
```

Compare this with the normal ROLLUP .

For example:

```
GROUP BY ROLLUP(a, b, c)
```

This would be equivalent to:

```
GROUP BY a, b, c UNION ALL
```

```
GROUP BY a, b UNION ALL
```

```
GROUP BY a UNION ALL
```

```
GROUP BY ()
```

Similarly:

```
GROUP BY CUBE((a, b), c)
```

This would be equivalent to:

```
GROUP BY a, b, c UNION ALL
```

```
GROUP BY a, b UNION ALL
```

```
GROUP BY c UNION ALL
```

```
GROUP BY ()
```

The following table shows the GROUPING SETS specification and the equivalent GROUP BY specification.

GROUPING SETS Statements	Equivalent GROUP BY Statements
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a, ROLLUP(b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

Composite Columns: Example

```
SELECT department_id, job_id, manager_id,
       SUM(salary)
  FROM employees
 GROUP BY ROLLUP( department_id, (job_id, manager_id));
```

The diagram illustrates the four levels of ROLLUP grouping:

- Level 1 (arrow 1):** Groups by department_id. It shows two rows: one for department 10 (AD_ASST) and one for department 20 (MK_MAN).
- Level 2 (arrow 2):** Groups by (job_id, manager_id). It shows three rows: one for (SA_REP), one for (AD_ASST), and one for (MK_MAN).
- Level 3 (arrow 3):** Groups by (department_id, job_id, manager_id). It shows two rows: one for (10, SA_REP) and one for (20, MK_MAN).
- Level 4 (arrow 4):** Grand total (all data).

#	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null) SA_REP	149	7000	
2	(null) (null)	(null)	7000	
3	10 AD_ASST	101	4400	
4	10 (null)	(null)	4400	
5	20 MK_MAN	100	13000	
6	20 MK_REP	201	6000	
7	20 (null)	(null)	19000	

#	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
40	100 FLMGR	101	12000	
41	100 FLACCOUNT	108	39600	
42	100 (null)	(null)	51600	
43	110 AC_MGR	101	12000	
44	110 AC_ACCOUNT	205	8300	
45	110 (null)	(null)	20300	
46	(null) (null)	(null)	691400	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Consider the example:

```
SELECT department_id, job_id, manager_id, SUM(salary)
  FROM employees
 GROUP BY ROLLUP( department_id, job_id, manager_id);
```

This query results in the Oracle server computing the following groupings:

- (job_id, manager_id)
- (department_id, job_id, manager_id)
- (department_id)
- Grand total

If you are interested only in specific groups, you cannot limit the calculation to those groupings without using composite columns. With composite columns, this is possible by treating JOB_ID and MANAGER_ID columns as a single unit while rolling up. Columns enclosed in parentheses are treated as a unit while computing ROLLUP and CUBE. This is illustrated in the example in the slide.

By enclosing the JOB_ID and MANAGER_ID columns in parentheses, you indicate to the Oracle server to treat JOB_ID and MANAGER_ID as a single unit, that is, as a composite column.

The example in the slide computes the following groupings:

- department_id, job_id, manager_id)
- department_id)
- ()

The example in the slide displays the following:

- Total salary for every job and manager (labeled 1)
- Total salary for every department, job, and manager (labeled 2)
- Total salary for every department (labeled 3)
- Grand total (labeled 4)

The example in the slide can also be written as:

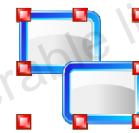
```
SELECT      department_id, job_id, manager_id, SUM(salary)
FROM        employees
GROUP       BY department_id,job_id, manager_id
UNION      ALL
SELECT      department_id, TO_CHAR(NULL) ,TO_NUMBER(NULL) ,
SUM(salary)
FROM        employees
GROUP BY    department_id
UNION ALL
SELECT    TO_NUMBER(NULL), TO_CHAR(NULL) ,TO_NUMBER(NULL) ,  SUM(salary)
FROM      employees
GROUP BY ();
```

In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query would need three scans of the base table, EMPLOYEES. This could be very inefficient. Therefore, the use of composite columns is recommended.

Concatenated Groupings

- Concatenated groupings offer a concise way to generate useful combinations of groupings.
- To specify concatenated grouping sets, you separate multiple grouping sets, ROLLUP, and CUBE operations with commas, so that the Oracle server combines them into a single GROUP BY clause.
- The result is a cross-product of groupings from each GROUPING SET.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Concatenated Groupings: Example

```
SELECT department_id, job_id, manager_id,
       SUM(salary)
  FROM employees
 GROUP BY department_id,
           ROLLUP(job_id),
           CUBE(manager_id);
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null) SA_REP	149	7000
2	10 AD_ASST	101	4400
3	20 MK_MAN	100	13000
4	20 MK_REP	201	6000

The diagram illustrates the execution steps of the query:

- Step 1:** The initial result set shows groups by department_id. Row 1 (dept 1, null job) is highlighted. An arrow labeled "1" points to the first row.
- Step 2:** The result set is shown after applying ROLLUP for job_id. Row 1 (dept 1, null job) is highlighted. An arrow labeled "2" points to the first row.
- Step 3:** The final result set is shown after applying CUBE for manager_id. Row 1 (dept 1, null job, null manager) is highlighted. An arrow labeled "3" points to the first row.

1	90 AD_VP	100	34000
	90 AD_PRES	(null)	24000
	(null) SA_REP	(null)	7000
	10 AD_ASST	(null)	4400
2	110 (null)	101	12000
	110 (null)	205	8300
	110 (null)	(null)	20300



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide results in the following groupings:

- (department_id, job_id,) (1)
- (department_id, manager_id) (2)
- (department_id) (3)

The total salary for each of these groups is calculated.

The following is another example of a concatenated grouping.

```
SELECT department_id, job_id, manager_id, SUM(salary) total
  FROM employees
 WHERE department_id<60
 GROUP BY GROUPING SETS(department_id),
          GROUPING SETS (job_id, manager_id);
```

Summary

In this appendix, you should have learned how to use the:

- ROLLUP operation to produce subtotal values
- CUBE operation to produce cross-tabulation values
- GROUPING function to identify the row values created by ROLLUP or CUBE
- GROUPING SETS syntax to define multiple groupings in the same query
- GROUP BY clause to combine expressions in various ways:
 - Composite columns
 - Concatenated grouping sets



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- ROLLUP and CUBE are extensions of the GROUP BY clause.
- ROLLUP is used to display subtotal and grand total values.
- CUBE is used to display cross-tabulation values.
- The GROUPING function enables you to determine whether a row is an aggregate produced by a CUBE or ROLLUP operator.
- With the GROUPING SETS syntax, you can define multiple groupings in the same query. GROUP BY computes all the groupings specified and combines them with UNION ALL.
- Within the GROUP BY clause, you can combine expressions in various ways:
 - To specify composite columns, you group columns within parentheses so that the Oracle server treats them as a unit while computing ROLLUP or CUBE operations.
 - To specify concatenated grouping sets, you separate multiple grouping sets, ROLLUP, and CUBE operations with commas so that the Oracle server combines them into a single GROUP BY clause. The result is a cross-product of groupings from each grouping set.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Hierarchical Retrieval

ORACLE

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Sample Data from the EMPLOYEES Table

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100	King	AD_PRES	(null)
2	101	Kochhar	AD_VP	100
3	102	De Haan	AD_VP	100
4	103	Hunold	IT_PROG	102
5	104	Ernst	IT_PROG	103
6	107	Lorentz	IT_PROG	103

...

16	200	Whalen	AD_ASST	101
17	201	Hartstein	MK_MAN	100
18	202	Fay	MK_REP	201
19	205	Higgins	AC_MGR	101
20	206	Gietz	AC_ACCOUNT	205



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

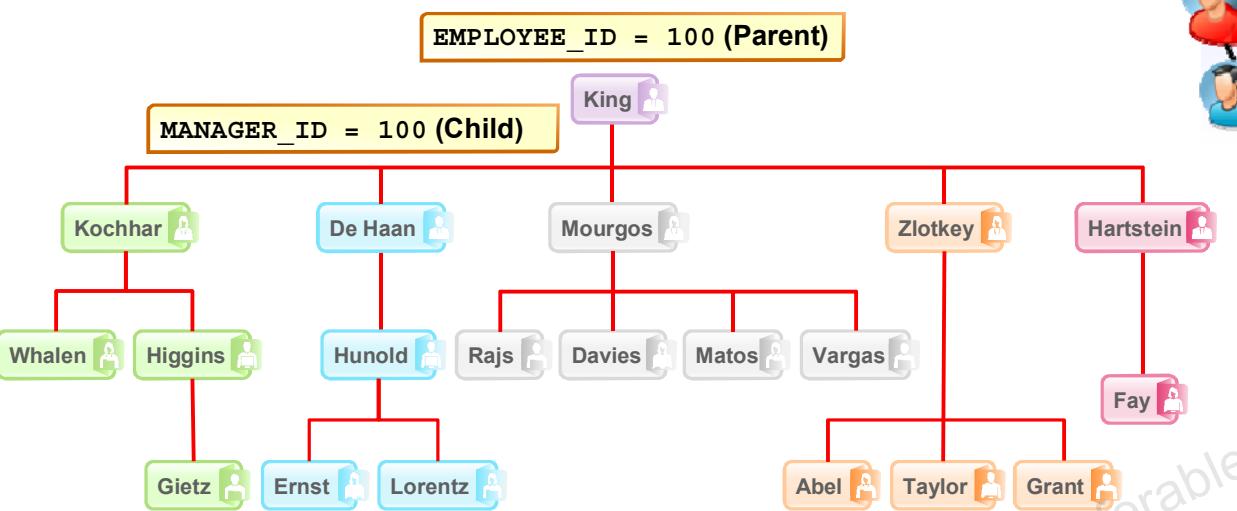
Using hierarchical queries, you can retrieve data based on a natural hierarchical relationship between the rows in a table. A relational database does not store records in a hierarchical way. However, where a hierarchical relationship exists between the rows of a single table, a process called *tree walking* enables the hierarchy to be constructed. A hierarchical query is a method of reporting, with the branches of a tree in a specific order.

Imagine a family tree with the eldest members of the family found close to the base or trunk of the tree and the youngest members representing branches of the tree. Branches can have their own branches, and so on.

A hierarchical query is possible when a relationship exists between rows in a table. For example, in the slide, you see that Kochhar, De Haan, and Hartstein report to MANAGER_ID 100, which is King's EMPLOYEE_ID.

Note: Hierarchical trees are used in various fields such as human genealogy (family trees), livestock (breeding purposes), corporate management (management hierarchies), manufacturing (product assembly), evolutionary research (species development), and scientific research.

Natural Tree Structure



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The EMPLOYEES table has a tree structure representing the management reporting line. The hierarchy can be created by looking at the relationship between equivalent values in the EMPLOYEE_ID and MANAGER_ID columns. This relationship can be exploited by joining the table to itself. The MANAGER_ID column contains the employee number of the employee's manager.

The parent/child relationship of a tree structure enables you to control:

- The direction in which the hierarchy is walked
- The starting point inside the hierarchy

Note: The slide displays an inverted tree structure of the management hierarchy of the employees in the EMPLOYEES table.

Hierarchical Queries

```
SELECT [LEVEL], column, expr...
  FROM table
 [WHERE condition(s)]
 [START WITH condition(s)]
 [CONNECT BY PRIOR condition(s)] ;
```

condition:

```
expr comparison_operator expr
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Keywords and Clauses

Hierarchical queries can be identified by the presence of the CONNECT BY and START WITH clauses.

In the syntax:

SELECT	Is the standard SELECT clause
LEVEL	For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on.
FROM <i>table</i>	Specifies the table, view, or snapshot containing the columns. You can select from only one table.
WHERE	Restricts the rows returned by the query without affecting other rows of the hierarchy
<i>condition</i>	Is a comparison with expressions
START WITH	Specifies the root rows of the hierarchy (where to start). This clause is required for a true hierarchical query.
CONNECT BY	Specifies the columns in which the relationship between parent and
PRIOR	child PRIOR rows exist. This clause is required for a hierarchical query.

Walking the Tree

Starting Point

- Specifies the condition that must be met
- Accepts any valid condition

```
START WITH column1 = value
```

Using the EMPLOYEES table, start with the employee whose last name is Kochhar.

```
...START WITH last_name = 'Kochhar'
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The row or rows to be used as the root of the tree are determined by the START WITH clause. The START WITH clause can contain any valid condition.

Examples

Using the EMPLOYEES table, start with King, the president of the company.

```
... START WITH manager_id IS NULL
```

Using the EMPLOYEES table, start with employee Kochhar. A START WITH condition can contain a subquery.

```
... START WITH employee_id = (SELECT employee_id  
                           FROM   employees  
                           WHERE  last_name = 'Kochhar')
```

If the START WITH clause is omitted, the tree walk is started with all the rows in the table as root rows.

Note: The CONNECT BY and START WITH clauses are not American National Standards Institute (ANSI) SQL standard.

Walking the Tree

```
CONNECT BY PRIOR column1 = column2
```

Walk from the top down, using the EMPLOYEES table.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Direction

Top down	→ Column1 = Parent Key Column2 = Child Key
Bottom up	→ Column1 = Child Key Column2 = Parent Key



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The direction of the query is determined by the CONNECT BY PRIOR column placement. For top-down, the PRIOR operator refers to the parent row. For bottom-up, the PRIOR operator refers to the child row. To find the child rows of a parent row, the Oracle server evaluates the PRIOR expression for the parent row and the other expressions for each row in the table. Rows for which the condition is true are the child rows of the parent. The Oracle server always selects child rows by evaluating the CONNECT BY condition with respect to a current parent row.

Examples

Walk from the top down using the EMPLOYEES table. Define a hierarchical relationship in which the EMPLOYEE_ID value of the parent row is equal to the MANAGER_ID value of the child row:

```
... CONNECT BY PRIOR employee_id = manager_id
```

Walk from the bottom up using the EMPLOYEES table:

```
... CONNECT BY PRIOR manager_id = employee_id
```

The PRIOR operator does not necessarily need to be coded immediately following CONNECT BY. Thus, the following CONNECT BY PRIOR clause gives the same result as the one in the preceding example:

```
... CONNECT BY employee_id = PRIOR manager_id
```

Note: The CONNECT BY clause cannot contain a subquery.

Walking the Tree: From the Bottom Up

```
SELECT employee_id, last_name, job_id, manager_id  
FROM   employees  
START  WITH employee_id = 101  
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Walking the Tree: From the Top Down

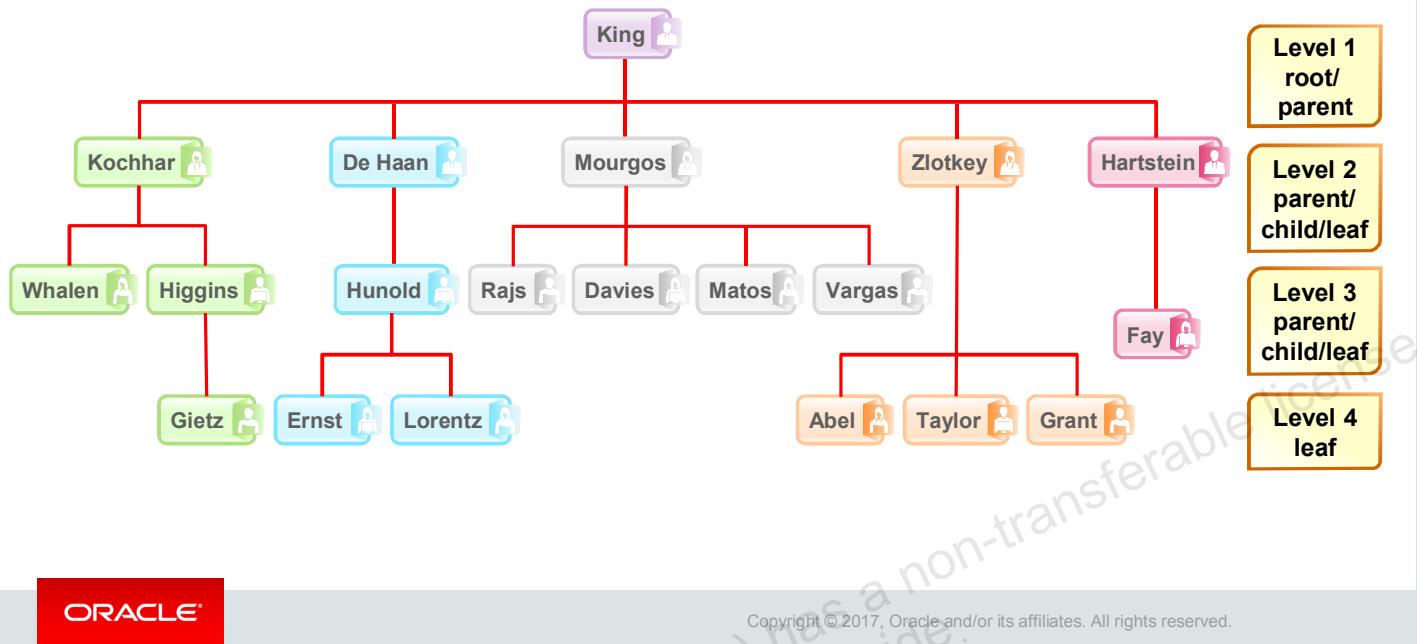
```
SELECT last_name||' reports to'||  
PRIOR last_name "Walk Top Down"  
FROM employees  
START WITH last_name = 'King'  
CONNECT BY PRIOR employee_id = manager_id ;
```

```
Walk Top Down  
1 King reports to  
2 King reports to  
3 Kochhar reports to King  
4 Greenberg reports to Kochhar  
5 Faviet reports to Greenberg  
...  
105 Grant reports to Zlotkey  
106 Johnson reports to Zlotkey  
107 Hartstein reports to King  
108 Fay reports to Hartstein
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Ranking Rows with the LEVEL Pseudocolumn



You can explicitly show the rank or level of a row in the hierarchy by using the LEVEL pseudocolumn. This will make your report more readable. The forks where one or more branches split away from a larger branch are called nodes, and the very end of a branch is called a leaf or leaf node. The graphic in the slide shows the nodes of the inverted tree with their LEVEL values. For example, employee Higgins is a parent and a child, whereas employee Davies is a child and a leaf.

LEVEL Pseudocolumn

Value	Level for Top Down	Level for Bottom up
1	A root node	A root node
2	A child of a root node	The parent of a root node
3	A child of a child, and so on	A parent of a parent, and so on

In the slide, King is the root or parent (LEVEL = 1). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgins, and Hunold are children and also parents (LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant, and Fay are children and leaves (LEVEL = 3 and LEVEL = 4).

Note: A *root node* is the highest node within an inverted tree. A *child node* is any nonroot node. A *parent node* is any node that has children. A *leaf node* is any node without children. The number of levels returned by a hierarchical query may be limited by available user memory.

Formatting Hierarchical Reports Using LEVEL and LPAD

Create a report displaying company management levels, beginning with the highest level and indenting each of the following levels.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
      AS org_chart
  FROM employees
 START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The nodes in a tree are assigned level numbers from the root. Use the LPAD function in conjunction with the LEVEL pseudocolumn to display a hierarchical report as an indented tree.

In the example in the slide:

- LPAD(*char1, n [, char2]*) returns *char1*, left-padded to length *n* with the sequence of characters in *char2*. The argument *n* is the total length of the return value as it is displayed on your terminal screen.
- LPAD(last_name, LENGTH(last_name) + (LEVEL*2) - 2, '_') defines the display format
- *char1* is the LAST_NAME, *n* the total length of the return value, is length of the LAST_NAME + (LEVEL*2) - 2, and *char2* is '_'

That is, this tells SQL to take the LAST_NAME and left-pad it with the '_' character until the length of the resultant string is equal to the value determined by LENGTH(last_name) + (LEVEL*2) - 2.

For King, LEVEL = 1. Therefore, $(2 * 1) - 2 = 2 - 2 = 0$. So King does not get padded with any '_' character and is displayed in column 1.

For Kochhar, LEVEL = 2. Therefore, $(2 * 2) - 2 = 4 - 2 = 2$. So Kochhar gets padded with 2 ' _ ' characters and is displayed indented.

The rest of the records in the EMPLOYEES table are displayed similarly.

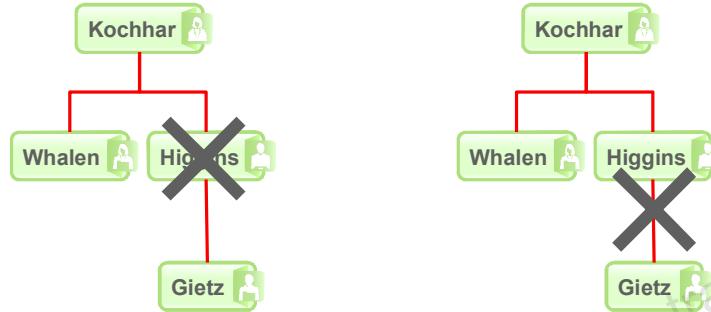
ORG_CHART	
1	King
2	_Kochhar
3	_Greenberg
4	_Faviet
5	_Chen
6	_Sciarra
7	_Urman
8	_Popp
9	_Whalen
10	_Mawris
11	_Baer
12	_Higgins
13	_Gietz
14	_De Haan
15	_Hunold
16	_Ernst
17	_Austin

Pruning Branches

Use the WHERE clause
to eliminate a node.

Use the CONNECT BY clause
to eliminate a branch.

```
WHERE last_name != 'Higgins'CONNECT BY PRIOR  
      employee_id = manager_id  
      AND last_name != 'Higgins'
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the WHERE and CONNECT BY clauses to prune the tree (that is, to control which nodes or rows are displayed). The predicate you use acts as a Boolean condition.

Examples

Starting at the root, walk from the top down, and eliminate employee Higgins in the result, but process the child rows.

```
SELECT department_id, employee_id, last_name, job_id, salary  
FROM employees  
WHERE last_name != 'Higgins'  
START WITH manager_id IS NULL  
CONNECT BY PRIOR employee_id = manager_id;
```

Starting at the root, walk from the top down, and eliminate employee Higgins and all child rows.

```
SELECT department_id, employee_id, last_name, job_id, salary  
FROM employees  
START WITH manager_id IS NULL  
CONNECT BY PRIOR employee_id = manager_id  
AND last_name != 'Higgins';
```

Summary

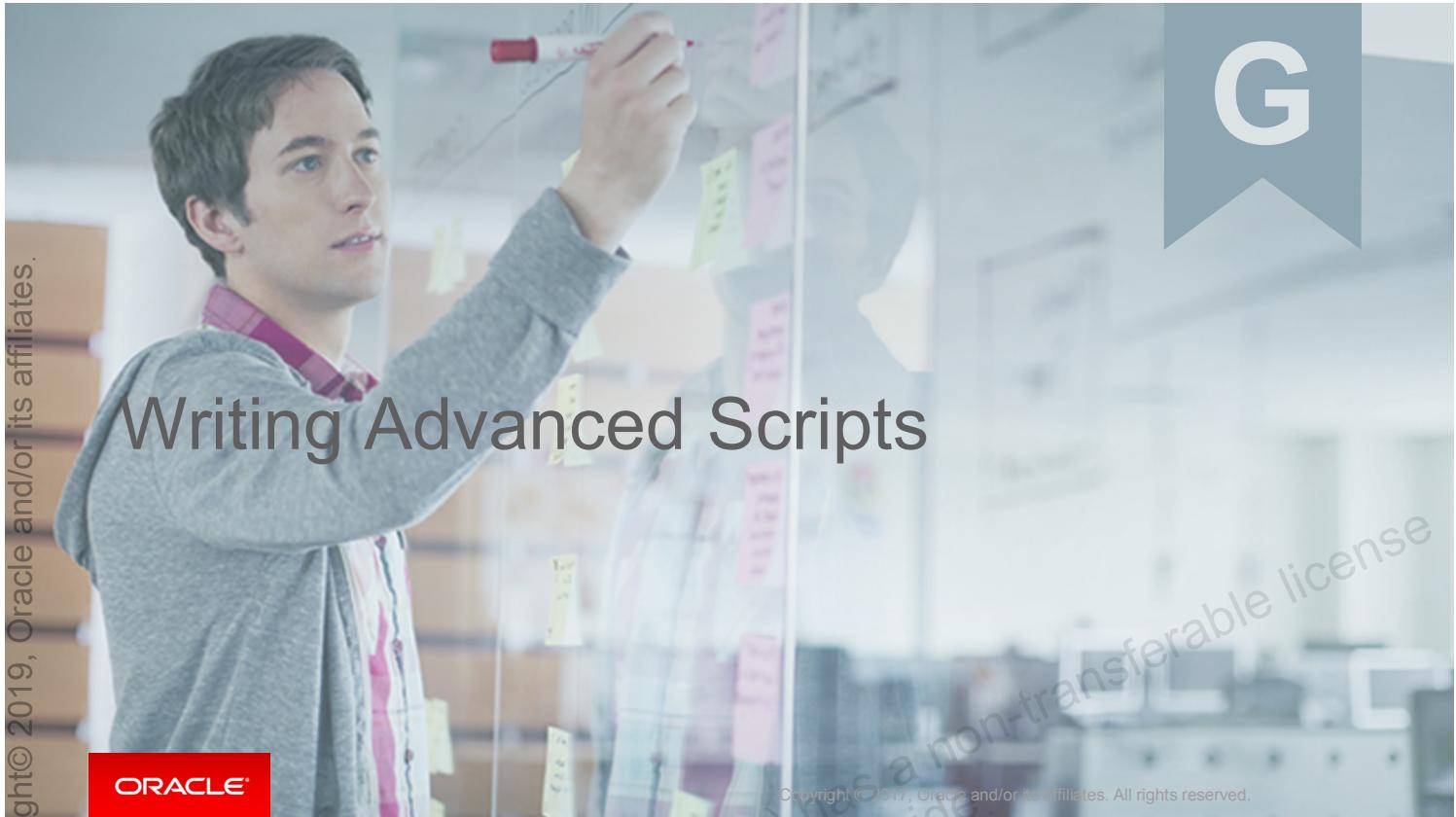
In this appendix, you should have learned how to:

- Use hierarchical queries to view a hierarchical relationship between rows in a table
- Specify the direction and starting point of the query
- Eliminate nodes or branches by pruning



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Copyright©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- Describe the type of problems that are solved by using SQL to generate SQL
- Create a basic SQL script
- Capture the output in a file
- Dump the contents of a table to a file
- Generate a dynamic predicate

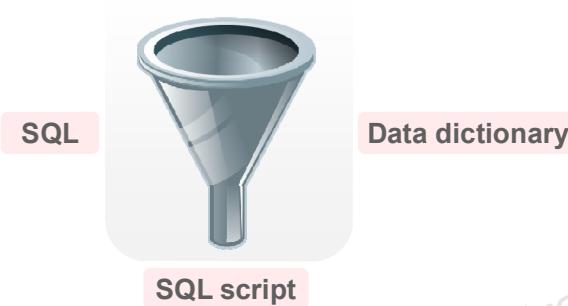


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using SQL to Generate SQL

- SQL can be used to generate scripts in SQL.
- The data dictionary is:
 - A collection of tables and views that contain database information
 - Created and maintained by the Oracle server



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL can be a powerful tool to generate other SQL statements. In most cases, this involves writing a script file. You can use SQL-from-SQL to:

- Avoid repetitive coding
- Access information from the data dictionary
- Drop or re-create database objects
- Generate dynamic predicates that contain run-time parameters

The examples used in this appendix involve selecting information from the data dictionary. The data dictionary is a collection of tables and views that contain information about the database. This collection is created and maintained by the Oracle server. All data dictionary tables are owned by the SYS user. Information stored in the data dictionary includes names of Oracle server users, privileges granted to users, database object names, table constraints, and audit information. There are four categories of data dictionary views. Each category has a distinct prefix that reflects its intended use.

Prefix	Description
USER_	Contains details of objects owned by the user
ALL_	Contains details of objects to which the user has been granted access rights, in addition to objects owned by the user
DBA_	Contains details of users with DBA privileges to access any object in the database
V\$	Stores information about database server performance and locking; available only to the DBA

Creating a Basic Script

```
SELECT 'CREATE TABLE ' || table_name ||
       '_test' || 'AS SELECT * FROM ' ||
       || table_name || ' WHERE 1=2;' ||
       AS "Create Table Script"
FROM user_tables;
```

```
Create Table Script
1 CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2 CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3 CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4 CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5 CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6 CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide produces a report with CREATE TABLE statements from every table you own. Each CREATE TABLE statement produced in the report includes the syntax to create a table using the table name with a suffix of _test and having only the structure of the corresponding existing table. The old table name is obtained from the TABLE_NAME column of the data dictionary view USER_TABLES.

The next step is to enhance the report to automate the process.

Note: You can query the data dictionary tables to view various database objects that you own. The data dictionary views frequently used include:

- USER_TABLES: Displays description of the user's own tables
- USER_OBJECTS: Displays all the objects owned by the user
- USER_TAB_PRIVS_MADE: Displays all grants on objects owned by the user
- USER_COL_PRIVS_MADE: Displays all grants on columns of objects owned by the user

Controlling the Environment

```
SET ECHO OFF  
SET FEEDBACK OFF  
SET PAGESIZE 0
```

SQL statement

```
SET FEEDBACK ON  
SET PAGESIZE 24  
SET ECHO ON
```

Set system variables to appropriate values.

Set system variables back to the default value.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To execute the SQL statements that are generated, you must capture them in a file that can then be run. You must also plan to clean up the output that is generated and make sure that you suppress elements such as headings, feedback messages, top titles, and so on. In SQL Developer, you can save these statements to a script. To save the contents of the Enter SQL Statement box, click the Save icon or select Save from the File menu. Alternatively, you can right-click in the Enter SQL Statement box and select the Save File option from the drop-down menu.

Note: Some of the SQL*Plus statements are not supported by SQL Worksheet. For the complete list of SQL*Plus statements that are supported, and not supported by SQL Worksheet, refer to the topic titled *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* in SQL Developer Online Help.

The Complete Picture

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';' 
FROM user_objects
WHERE object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The output of the command in the slide is saved into a file called `dropem.sql` in SQL Developer. To save the output into a file in SQL Developer, you use the Save File option under the Script Output pane. The `dropem.sql` file contains the following data. This file can now be started from SQL Developer by locating the script file, loading it, and executing it.

A Z	'DROPTABLE' OBJECT_NAME ';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

Dumping the Contents of a Table to a File

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0

SELECT
  'INSERT INTO departments_test VALUES
  (' || department_id || ', "' || department_name || 
   '', '' || location_id || '')';
  AS "Insert Statements Script"
FROM  departments
/

SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Sometimes, it is useful to have the values for the rows of a table in a text file in the format of an INSERT INTO VALUES statement. This script can be run to populate the table in case the table has been dropped accidentally.

The example in the slide produces INSERT statements for the DEPARTMENTS_TEST table, captured in the data.sql file using the Save File option in SQL Developer.

The contents of the data.sql script file are as follows:

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

Dumping the Contents of a Table to a File

Source	Result
' '' X '' '	' X '
' '' '' '	' '
' '' '' department_name '' '' '	' Administration '
' '' , '' '' '	' , '
' '') ; '	') ; '



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Generating a Dynamic Predicate

```
COLUMN my_col NEW_VALUE dyn_where_clause  
  
SELECT DECODE(''||deptno', null,  
DECODE (''||hiredate', null, ' ',  
'WHERE hire_date=TO_DATE(''||'||deptno'||',''DD-MON-YYYY'))',  
DECODE (''||hiredate', null,  
'WHERE department_id = '||''||deptno',  
'WHERE department_id = '||''||deptno'||'  
' AND hire_date = TO_DATE(''||'||deptno'||',''DD-MON-YYYY'))')  
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide generates a `SELECT` statement that retrieves data of all employees in a department who were hired on a specific day. The script generates the `WHERE` clause dynamically.

Note: After the user variable is in place, you must use the `UNDEFINE` command to delete it.

The first `SELECT` statement prompts you to enter the department number. If you do not enter any department number, the department number is treated as null by the `DECODE` function and the user is then prompted for the hire date. If you do not enter any hire date, the hire date is treated as null by the `DECODE` function and the dynamic `WHERE` clause that is generated is also a null, which causes the second `SELECT` statement to retrieve all the rows from the `EMPLOYEES` table.

Note: The `NEW_V[ALUE]` variable specifies a variable to hold a column value. You can reference the variable in `TTITLE` commands. Use `NEW_VALUE` to display column values or the date in the top title. You must include the column in a `BREAK` command with the `SKIP PAGE` action. The variable name cannot contain a pound sign (#). `NEW_VALUE` is useful for master/detail reports in which there is a new master record for each page.

Note: Here, the hire date must be entered in the DD-MON-YYYY format.

The SELECT statement in the slide can be interpreted as follows:

```

IF    (<<deptno>> is not entered) THEN
    IF  (<<hiredate>> is not entered)  THEN
        return empty string
    ELSE
        return the string 'WHERE hire_date =
TO_DATE(''<<hiredate>>'', 'DD-MON-YYYY')'
    ELSE
        IF (<<hiredate>> is not entered)  THEN
            return the string 'WHERE department_id =
<<deptno>> entered'
        ELSE
            return the string 'WHERE department_id =
<<deptno>> entered
                                AND hire_date =
TO_DATE(''<<hiredate>>'', 'DD-MON-YYYY') '
        END IF
    
```

The returned string becomes the value of the DYN_WHERE_CLAUSE variable, which will be used in the second SELECT statement.

Note: Use SQL*Plus for these examples.

When the first example in the slide is executed, the user is prompted for the values for DEPTNO and HIREDATE:

Enter the values of DEPTNO and HIREDATE: 10 and 17-SEP-2007

The following value for MY_COL is generated:

<input type="text"/> MY_COL 1 WHERE department_id = 10 AND hire_date = TO_DATE('17-SEP-2007', 'DD-MON-YYYY')

When the second example in the slide is executed, the following output is generated:

LAST_NAME ----- Whalen

Summary

In this appendix, you should have learned how to:

- Create a basic SQL script
- Capture the output in a file
- Dump the contents of a table to a file
- Generate a dynamic predicate



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.



Oracle Database Architectural Components

ORACLE

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this appendix, you should be able to:

- List the major database architectural components
- Describe the background processes
- Explain the memory structures
- Correlate the logical and physical storage structures



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database Architecture: Overview

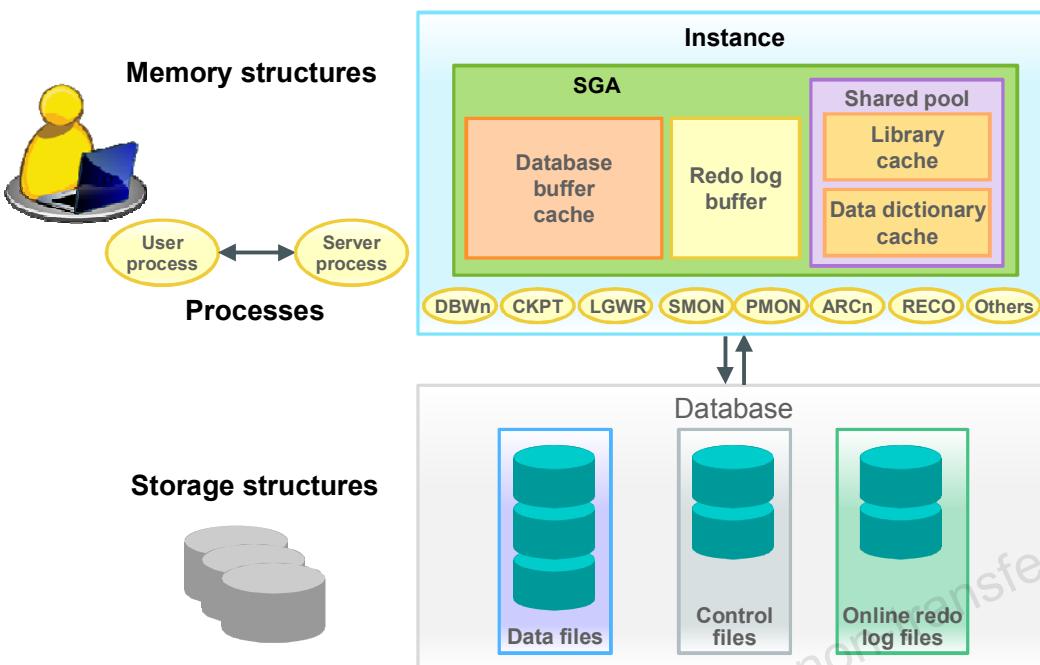
The Oracle Relational Database Management System (RDBMS) is a database management system that provides an open, comprehensive, integrated approach to information management.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database Server Structures



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

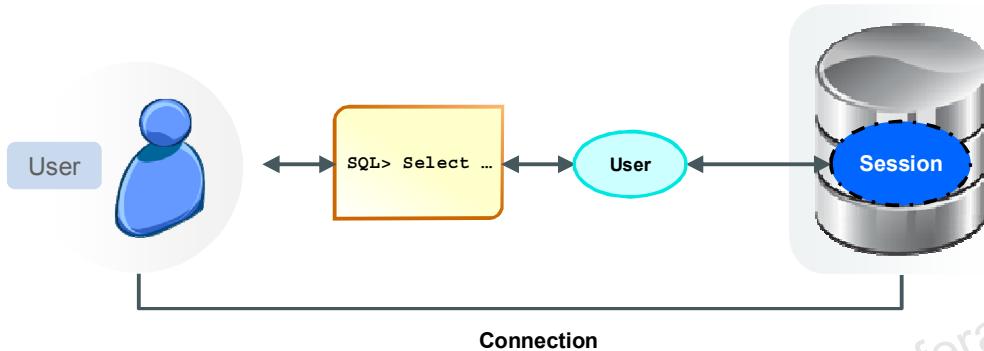
Oracle Database consists of two main components—the instance and the database.

- The instance consists of the System Global Area (SGA), which is a collection of memory structures, and the background processes that perform tasks within the database. Every time an instance is started, the SGA is allocated and the background processes are started.
- The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures. The physical storage structures include:
 - The control files where the database configuration is stored
 - The redo log files that have information required for database recovery
 - The data files where all data is stored

An Oracle instance uses memory structures and processes to manage and access the database storage structures. All memory structures exist in the main memory of the computers that constitute the database server. Processes are jobs that work in the memory of these computers. A process is defined as a “thread of control” or a mechanism in an operating system that can run a series of steps.

Connecting to the Database

- Connection: A communication pathway between a user process and a database instance
- Session: A specific connection of a user to a database instance through a user process



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To access information in the database, the user needs to connect to the database using a tool (such as SQL*Plus). After the user establishes connection, a session is created for the user. Connection and session are closely related to user process, but are very different in meaning.

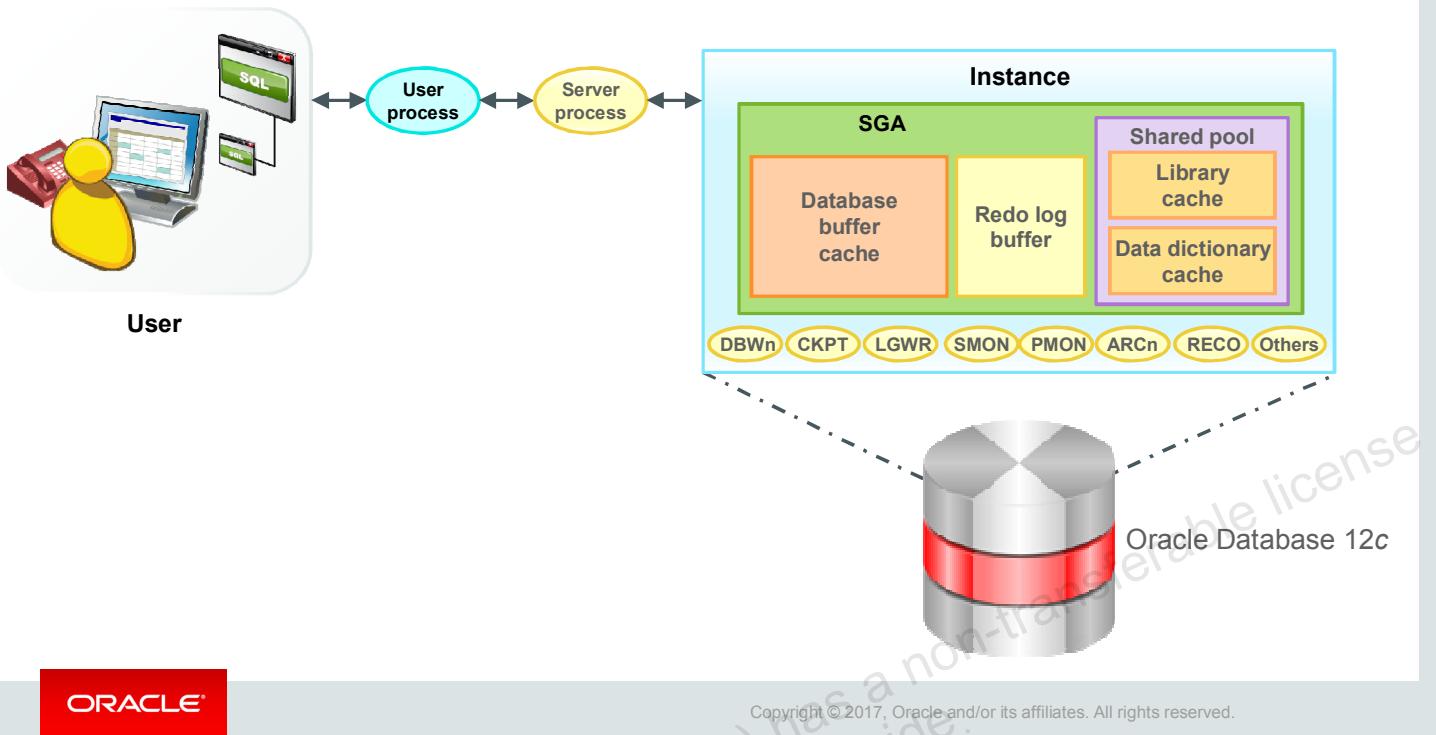
A connection is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established by using available interprocess communication mechanisms or network software (when different computers run the database application and Oracle Database, and communicate through a network).

A session represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

In the case of a dedicated connection, the session is serviced by a permanent dedicated process. In the case of a shared connection, the session is serviced by an available server process selected from a pool, either by the middle tier or by Oracle shared server architecture.

Multiple sessions can be created and exist concurrently for a single Oracle Database user using the same username, but through different applications, or multiple invocations of the same application.

Interacting with an Oracle Database



ORACLE®

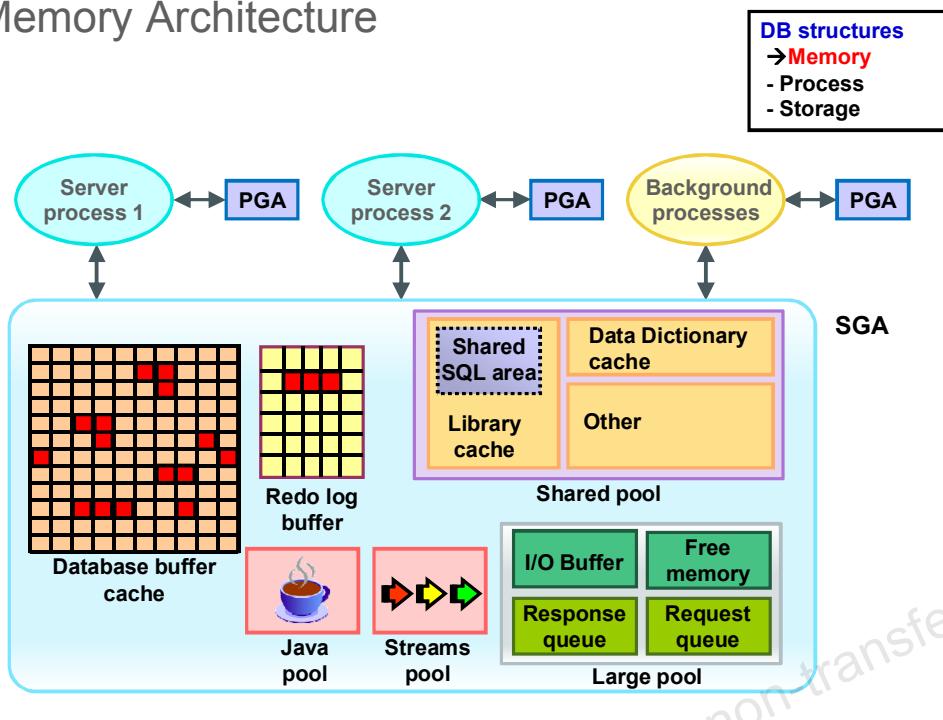
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The following example describes Oracle Database operations at the most basic level. It illustrates an Oracle Database configuration where the user and the associated server process are on separate computers, connected through a network.

1. An instance has started on a node where Oracle Database is installed, often called the host or database server.
2. A user starts an application spawning a user process. The application attempts to establish a connection to the server. (The connection may be local, client server, or a three-tier connection from a middle tier.)
3. The server runs a listener that has the appropriate Oracle Net Services handler. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a DML-type SQL statement and commits the transaction. For example, the user changes the address of a customer in a table and commits the change.
5. The server process receives the statement and checks the shared pool (an SGA component) for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data, and the existing shared SQL area is used to process the statement. If not, a new shared SQL area is allocated for the statement, so it can be parsed and processed.

6. The server process retrieves any necessary data values, either from the actual data file (in which the table is stored) or from those cached in the SGA.
7. The server process modifies data in the SGA. Because the transaction is committed, the log writer process (LGWR) immediately records the transaction in the redo log file. The database writer (DBW n) process writes modified blocks permanently to disk when doing so is efficient.
8. If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Oracle Memory Architecture



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data shared among users, and private data areas for each connected user.

Two basic memory structures are associated with an instance:

- The SGA is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- The Program Global Areas (PGA) are memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

- **Database buffer cache:** Caches blocks of data retrieved from the database
- **Redo Log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Shared pool:** Caches various constructs that can be shared among users
- **Large pool:** Is an optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and input/output (I/O) server processes
- **Java pool:** Is used for all session-specific Java code and data within the Java Virtual Machine (JVM)
- **Streams pool:** Is used by Oracle Streams to store information required by capture and apply

When you start the instance by using Enterprise Manager or SQL*Plus, the amount of memory allocated for the SGA is displayed.

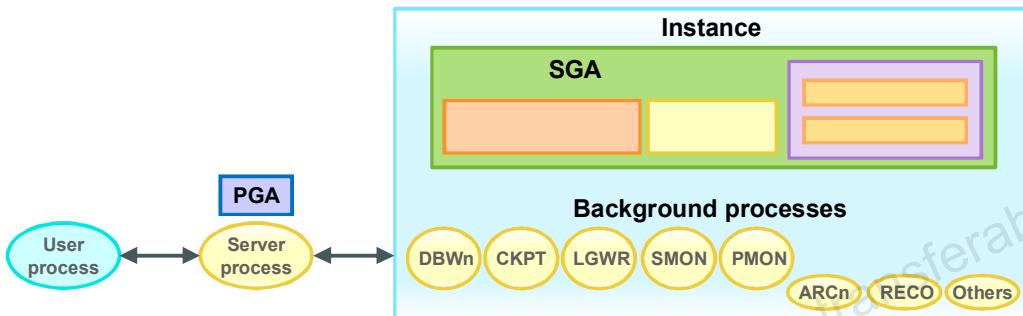
With the dynamic SGA infrastructure, the size of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool changes without shutting down the instance.

Oracle Database uses initialization parameters to create and configure memory structures. For example, the `SGA_TARGET` parameter specifies the total size of the SGA components. If you set `SGA_TARGET` to 0, Automatic Shared Memory Management is disabled.

Process Architecture

DB structures
- Memory
→ **Process**
- Storage

- User process:
 - Is started when a database user or a batch process connects to Oracle Database
- Database processes:
 - Server process: Connects to the Oracle instance and is started when a user establishes a session
 - Background processes: Are started when an Oracle instance is started



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The processes in an Oracle Database server can be categorized into two major groups:

- User processes that run the application or Oracle tool code
- Oracle Database processes that run the Oracle database server code. These include server processes and background processes.

When a user runs an application program or an Oracle tool such as SQL*Plus, Oracle Database creates a user process to run the user's application. Oracle Database also creates a server process to execute the commands issued by the user process. In addition, the Oracle server also has a set of background processes for an instance that interact with each other and with the operating system to manage the memory structures and asynchronously perform I/O to write data to disk, and perform other required tasks.

The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

- With a dedicated server, for each user, the database application is run by a different process (a user process) than the one that runs the Oracle server code (a dedicated server process).
- A shared server eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. In some situations when the application and Oracle Database operate on the same computer, it is possible to combine the user process and the corresponding server process into a single process to reduce system overhead. However, when the application and Oracle Database operate on different computers, a user process always communicates with Oracle Database through a separate server process.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA.
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called background processes. An Oracle Database instance can have many background processes.

The following background processes are required for a successful startup of the database instance:

- Database writer (DBW n)
- Log writer (LGWR)
- Checkpoint (CKPT)
- System monitor (SMON)
- Process monitor (PMON)

The following background processes are a few examples of optional background processes that can be started if required:

- Recoverer (RECO)
- Job queue
- Archiver (ARC n)
- Queue monitor (QM n)

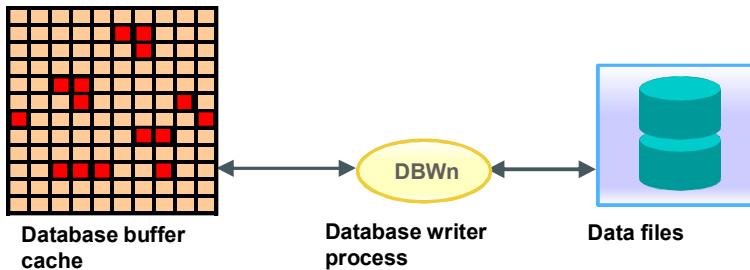
Other background processes may be found in more advanced configurations such as Real Application Clusters (RAC). See the V\$BGPROCESS view for more information about the background processes.

On many operating systems, background processes are created automatically when an instance is started.

Database Writer Process

Writes modified (dirty) buffers in the database buffer cache to disk:

- Asynchronously while performing other processing
- Periodically to advance the checkpoint



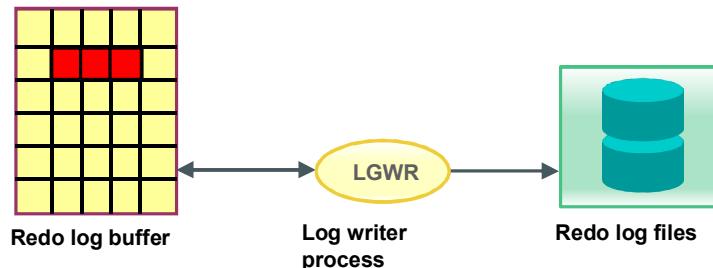
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The database writer (DBW n) process writes the contents of buffers to data files. The DBW n processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBW n processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked “dirty” and is added to the LRUW list of dirty buffers that is kept in system change number (SCN) order, thereby matching the order of Redo corresponding to these changed buffers that is written to the Redo logs. When the number of available buffers in the buffer cache falls below an internal threshold such that server processes find it difficult to obtain available buffers, DBW n writes dirty buffers to the data files in the order that they were modified by following the order of the LRUW list.

Log Writer Process

- Writes the redo log buffer to a redo log file on disk
- The Log Writer (LGWR) writes:
 - When a process commits a transaction
 - When the redo log buffer is one-third full
 - Before a DBW n process writes modified buffers to disk



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The log writer (LGWR) process is responsible for redo log buffer management by writing the redo log buffer entries to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. The LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

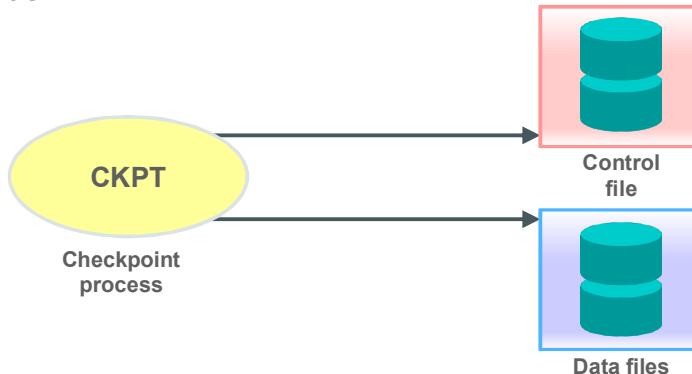
LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

- When a user process commits a transaction
- When the redo log buffer is one-third full
- Before a DBW n process writes modified buffers to disk, if necessary

Checkpoint Process

Records checkpoint information in:

- The control file
- Each data file header



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

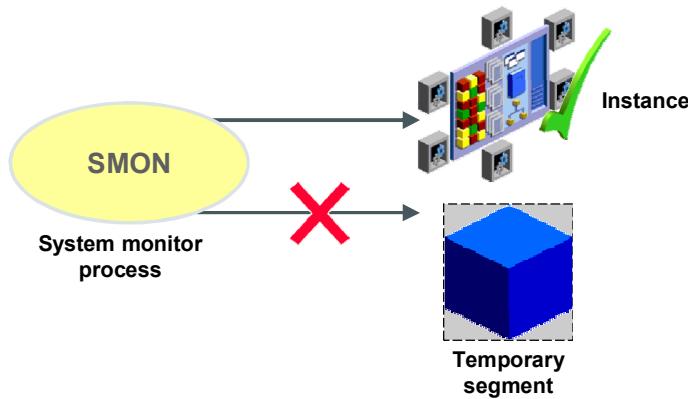
A checkpoint is a data structure that defines an SCN in the redo thread of a database. Checkpoints are recorded in the control file and each data file header, and are a crucial element of recovery.

When a checkpoint occurs, Oracle Database must update the headers of all data files to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBW n always performs that work. The SCNs recorded in the file headers guarantee that all the changes made to database blocks before that SCN have been written to disk.

The statistic DBWR checkpoints displayed by the SYSTEM_STATISTICS monitor in Oracle Enterprise Manager indicate the number of checkpoint requests completed.

System Monitor Process

- Performs recovery at instance startup
- Cleans up unused temporary segments

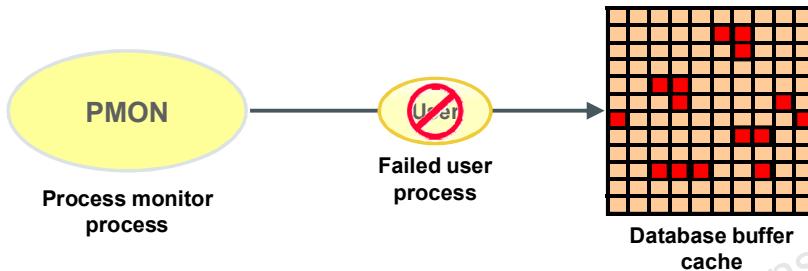


ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Process Monitor Process

- Performs process recovery when a user process fails:
 - Cleans up the database buffer cache
 - Frees resources used by the user process
- Monitors sessions for idle session timeout
- Dynamically registers database services with listeners



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

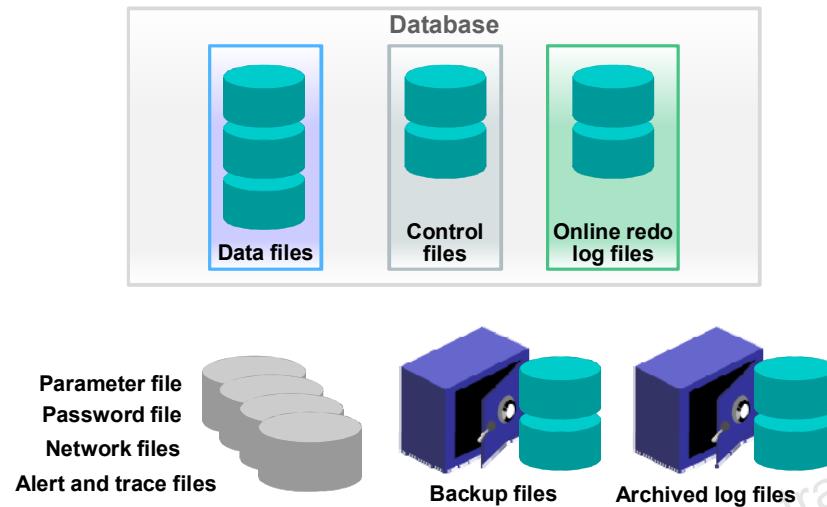
The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Oracle Database Storage Architecture

DB structures
- Memory
- Process
→ Storage



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The files that constitute an Oracle database are organized into the following:

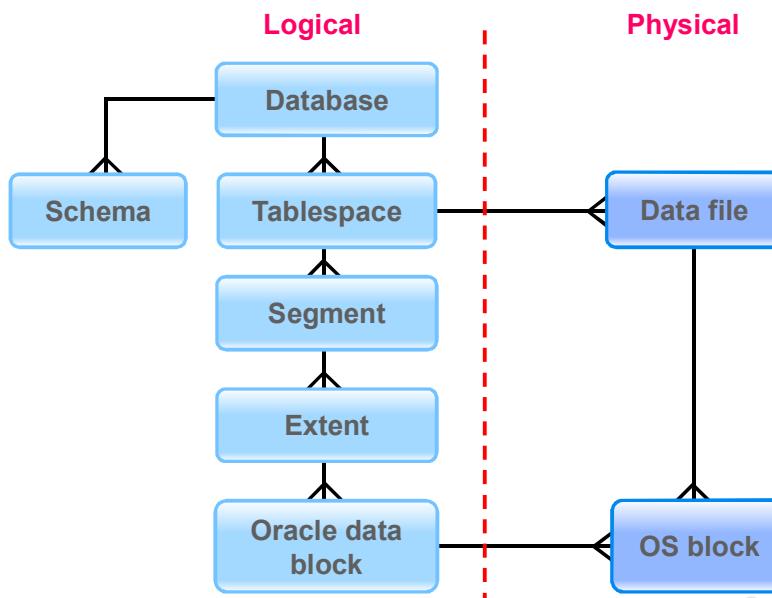
- **Control files:** Contain data about the database itself (that is, physical database structure information). These files are critical to the database. Without them, you cannot open data files to access the data within the database.
- **Data files:** Contain the user or application data of the database, as well as metadata and the data dictionary
- **Online redo log files:** Allow for instance recovery of the database. If the database server crashes and does not lose any data files, the instance can recover the database with the information in these files.

The following additional files are important to the successful running of the database:

- **Backup files:** Are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. Using these files and a backup of the database, you can recover a lost data file. That is, archive logs enable the recovery of restored data files.
- **Parameter file:** Is used to define how the instance is configured when it starts up
- **Password file:** Allows sysdba/sysoper/sysasm to connect remotely to the database and perform administrative tasks

- **Network files:** Are used for starting the database listener and store information required for user connections
- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services.
- **Alert log files:** These are special trace entries. The alert log of a database is a chronological log of messages and errors. Each instance has one alert log file. Oracle recommends that you review this alert log periodically.

Logical and Physical Database Structures



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

An Oracle database has logical and physical storage structures.

Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group all of an application's objects to simplify some administrative operations. You may have a tablespace for application data and an additional one for application indexes.

Databases, Tablespaces, and Data Files

The relationship among databases, tablespaces, and data files is illustrated in the slide. Each database is logically divided into one or more tablespaces. One or more data files are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. If it is a TEMPORARY tablespace, instead of a data file, the tablespace has a temporary file.

Schemas

A schema is a collection of database objects that are owned by a database user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. In general, schema objects include everything that your application creates in the database.

Data Blocks

At the finest level of granularity, an Oracle database's data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on the disk. A data block size is specified for each tablespace when it is created. A database uses and allocates free database space in Oracle data blocks.

Extents

The next level of logical database space is called an extent. An extent is a specific number of contiguous data blocks (obtained in a single allocation) that are used to store specific type of information.

Segments

The level of logical database storage above an extent is called a segment. A segment is a set of extents allocated for a certain logical structure. For example, the different types of segments include:

- **Data segments:** Each nonclustered, non-indexed-organized table has a data segment with the exception of external tables, global temporary tables, and partitioned tables, where each table has one or more segments. All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
- **Index segments:** Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One `UNDO` tablespace is created per database instance that contains numerous undo segments to temporarily store *undo* information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, to roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle Database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the instance for future use. Specify a default temporary tablespace for every user or a default temporary tablespace, which is used databasewide.

The Oracle Database dynamically allocates space. When the existing extents of a segment are full, additional extents are added. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on the disk.

Processing a SQL Statement

- Connect to an instance using:
 - The user process
 - The server process
- The Oracle server components that are used depend on the type of SQL statement:
 - Queries return rows.
 - Data manipulation language (DML) statements log changes.
 - Commit ensures transaction recovery.
- Some Oracle server components do not participate in SQL statement processing.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Processing a Query

- Parse:
 - Search for an identical statement.
 - Check the syntax, object names, and privileges.
 - Lock the objects used during parse.
 - Create and store the execution plan.
- Execute: Identify the rows selected.
- Fetch: Return the rows to the user process.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Queries are different from other types of SQL statements because, if successful, they return data as results. Other statements simply return success or failure, whereas a query can return one row or thousands of rows.

There are three main stages in the processing of a query:

- Parse
- Execute
- Fetch

During the parse stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.

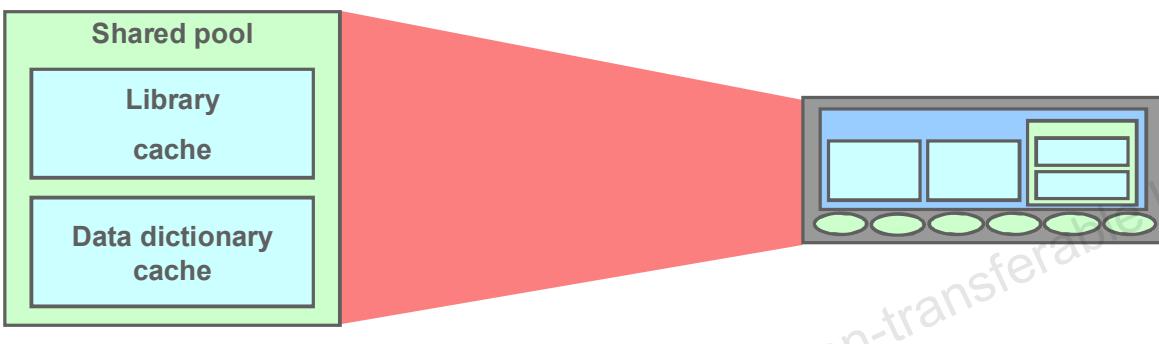
During parse, the server process performs the following functions:

- Searches for an existing copy of the SQL statement in the shared pool
- Validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

The execute stage executes the statement using the best optimizer approach and the fetch stage retrieves the rows back to the user.

Shared Pool

- The library cache contains the SQL statement text, parsed code, and execution plan.
- The data dictionary cache contains table, column, and other object definitions and privileges.
- The shared pool is sized by `SHARED_POOL_SIZE`.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

During the parse stage, the server process uses the area in the SGA known as the shared pool to compile the SQL statement. The shared pool has two primary components:

- Library cache
- Data dictionary cache

Library Cache

The library cache stores information about the most recently used SQL statements in a memory structure called a shared SQL area. The shared SQL area contains:

- The text of the SQL statement
- The parse tree, which is a compiled version of the statement
- The execution plan, with steps to be taken when executing the statement

The optimizer is the function in the Oracle server that determines the optimal execution plan.

If a SQL statement is reexecuted and a shared SQL area already contains the execution plan for the statement, the server process does not need to parse the statement. The library cache improves the performance of applications that reuse SQL statements by reducing parse time and memory requirements. If the SQL statement is not reused, it is eventually aged out of the library cache.

Data Dictionary Cache

The data dictionary cache, also known as the dictionary cache or row cache, is a collection of the most recently used definitions in the database. It includes information about database files, tables, indexes, columns, users, privileges, and other database objects.

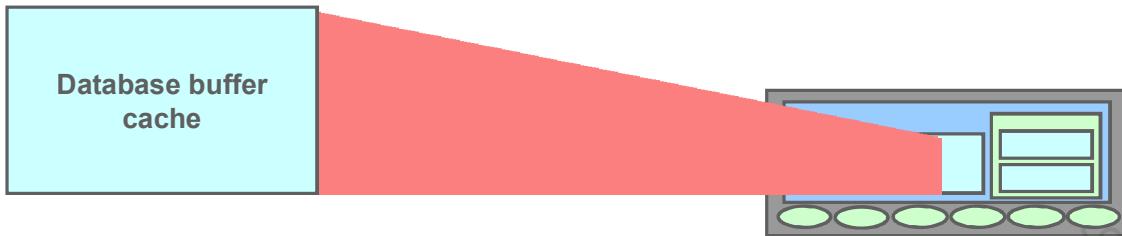
During the parse phase, the server process looks for the information in the dictionary cache to resolve the object names specified in the SQL statement and to validate the access privileges. If necessary, the server process initiates the loading of this information from the data files.

Sizing the Shared Pool

The size of the shared pool is specified by the `SHARED_POOL_SIZE` initialization parameter.

Database Buffer Cache

- The database buffer cache stores the most recently used blocks.
- The size of a buffer is based on `DB_BLOCK_SIZE`.
- The number of buffers is defined by `DB_BLOCK_BUFFERS`.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When a query is processed, the server process looks in the database buffer cache for any blocks it needs. If the block is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the buffer cache. Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads. The Oracle server uses a least recently used algorithm to age out buffers that have not been accessed recently to make room for new blocks in the buffer cache.

Sizing the Database Buffer Cache

The size of each buffer in the buffer cache is equal to the size of an Oracle block, and it is specified by the `DB_BLOCK_SIZE` parameter. The number of buffers is equal to the value of the `DB_BLOCK_BUFFERS` parameter.

Program Global Area (PGA)

- Is not shared
- Is writable only by the server process
- Contains:
 - Sort area
 - Session information
 - Cursor state
 - Stack space



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

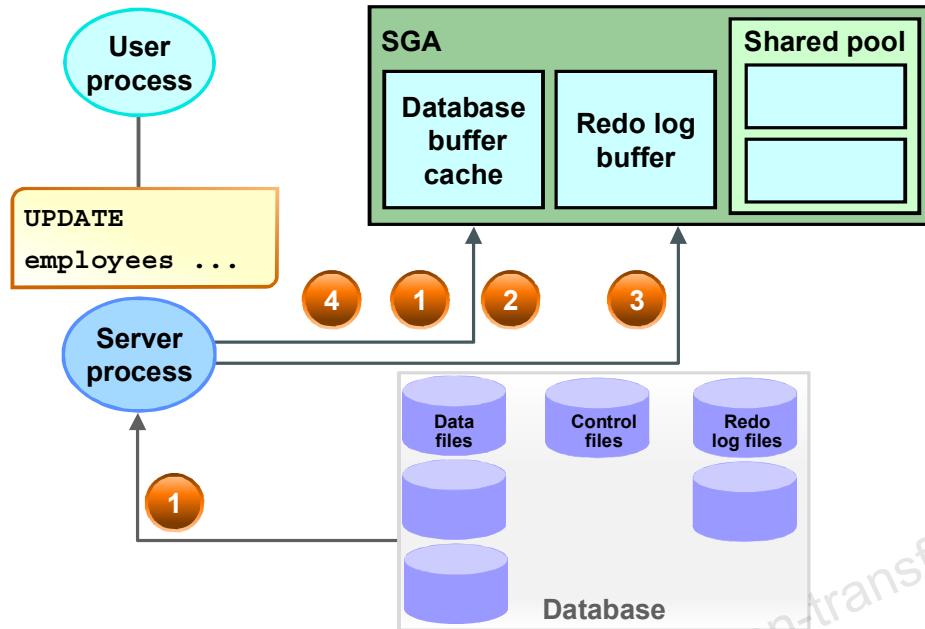
A Program Global Area (PGA) is a memory region that contains data and control information for a server process. It is a nonshared memory created by Oracle when a server process is started. Access to it is exclusive to that server process, and is read and written only by the Oracle server code acting on behalf of it. The PGA memory allocated by each server process attached to an Oracle instance is referred to as the aggregated PGA memory allocated by the instance.

In a dedicated server configuration, the PGA of the server includes the following components:

- **Sort area:** Is used for any sorts that may be required to process the SQL statement
- **Session information:** Includes user privileges and performance statistics for the session
- **Cursor state:** Indicates the stage in the processing of the SQL statements that are currently used by the session
- **Stack space:** Contains other session variables

The PGA is allocated when a process is created and deallocated when the process is terminated.

Processing a DML Statement



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query.
- Execute requires additional processing to make data changes.

DML Execute Phase

To execute a DML statement:

- If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache.
- The server process places locks on the rows that are to be modified.
- In the redo log buffer, the server process records the changes to be made to the rollback and data blocks.
- The rollback block changes record the values of the data before it is modified. The rollback block is used to store the “before image” of the data, so that the DML statements can be rolled back if necessary.
- The data block changes record the new values of the data.

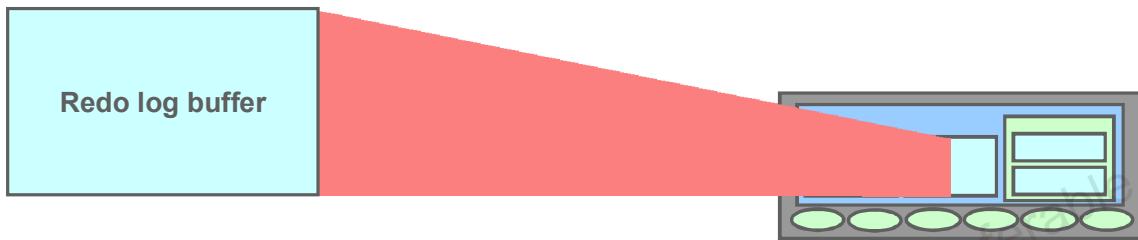
The server process records the “before image” to the rollback block and updates the data block. Both of these changes are done in the database buffer cache. Any changed blocks in the buffer cache are marked as dirty buffers (that is, buffers that are not the same as the corresponding blocks on the disk).

The processing of a `DELETE` or `INSERT` command uses similar steps. The “before image” for a `DELETE` contains the column values in the deleted row, and the “before image” of an `INSERT` contains the row location information.

Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA can also lose these changes.

Redo Log Buffer

- Has its size defined by `LOG_BUFFER`
- Records changes made through the instance
- Is used sequentially
- Is a circular buffer



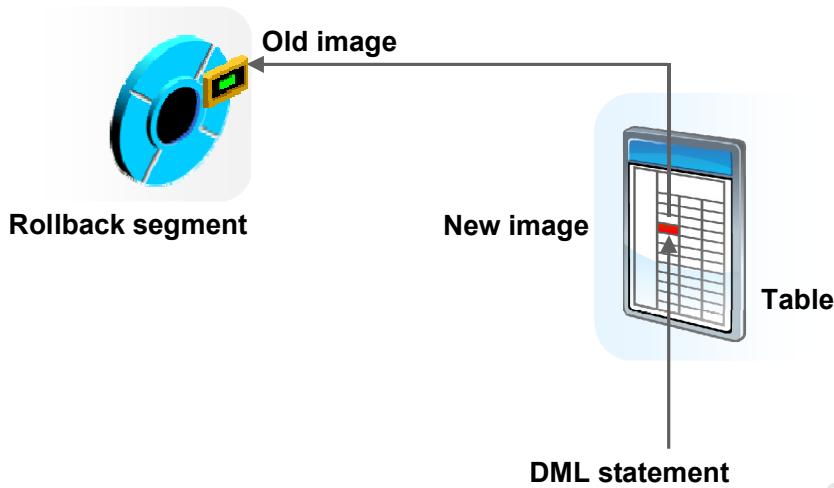
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The server process records most of the changes made to data file blocks in the redo log buffer, which is a part of the SGA. The redo log buffer has the following characteristics:

- Its size in bytes is defined by the `LOG_BUFFER` parameter.
- It records the block that is changed, the location of the change, and the new value in a redo entry. A redo entry makes no distinction between the types of block that is changed; it only records which bytes are changed in the block.
- The redo log buffer is used sequentially, and changes made by one transaction may be interleaved with changes made by other transactions.
- It is a circular buffer that is reused after it is filled, but only after all the old redo entries are recorded in the redo log files.

Rollback Segment



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

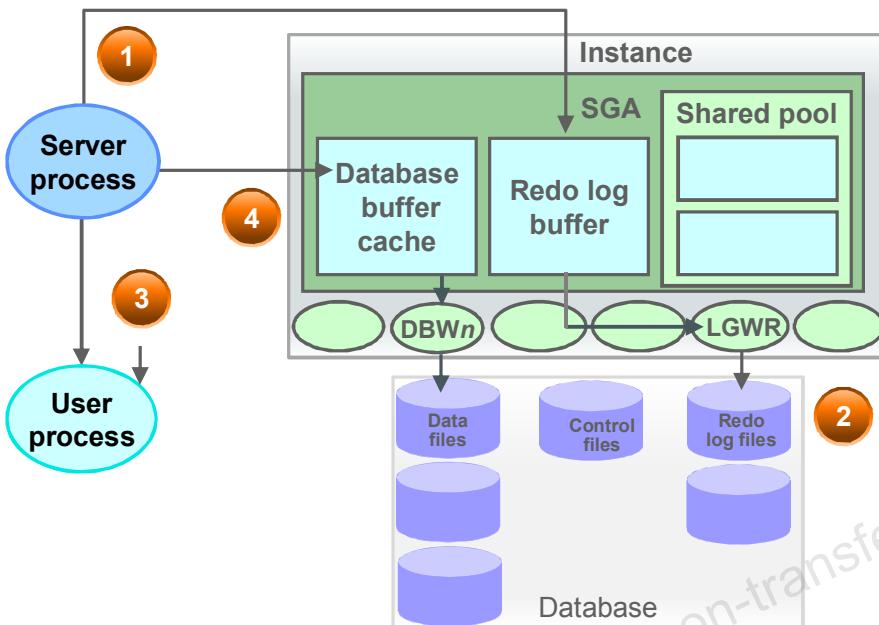
Before making a change, the server process saves the old data value in a rollback segment. This “before image” is used to:

- Undo the changes if the transaction is rolled back
- Provide read consistency by ensuring that other transactions do not see uncommitted changes made by the DML statement
- Recover the database to a consistent state in case of failures

Rollback segments, such as tables and indexes, exist in data files, and rollback blocks are brought into the database buffer cache as required. Rollback segments are created by the DBA.

Changes to rollback segments are recorded in the redo log buffer.

COMMIT Processing



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The Oracle server uses a fast COMMIT mechanism that guarantees that the committed changes can be recovered in case of instance failure.

System Change Number

Whenever a transaction commits, the Oracle server assigns a commit SCN to the transaction. The SCN is monotonically incremented and is unique within the database. It is used by the Oracle server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables the Oracle server to perform consistency checks without depending on the date and time of the operating system.

Steps in Processing COMMITS

When a COMMIT is issued, the following steps are performed:

1. The server process places a commit record, along with the SCN, in the redo log buffer.
2. LGWR performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, the Oracle server can guarantee that the changes will not be lost even if there is an instance failure.

3. The user is informed that the COMMIT is complete.
4. The server process records information to indicate that the transaction is complete and that resource locks can be released.

Flushing of the dirty buffers to the data file is performed independently by DBW0 and can occur either before or after the commit.

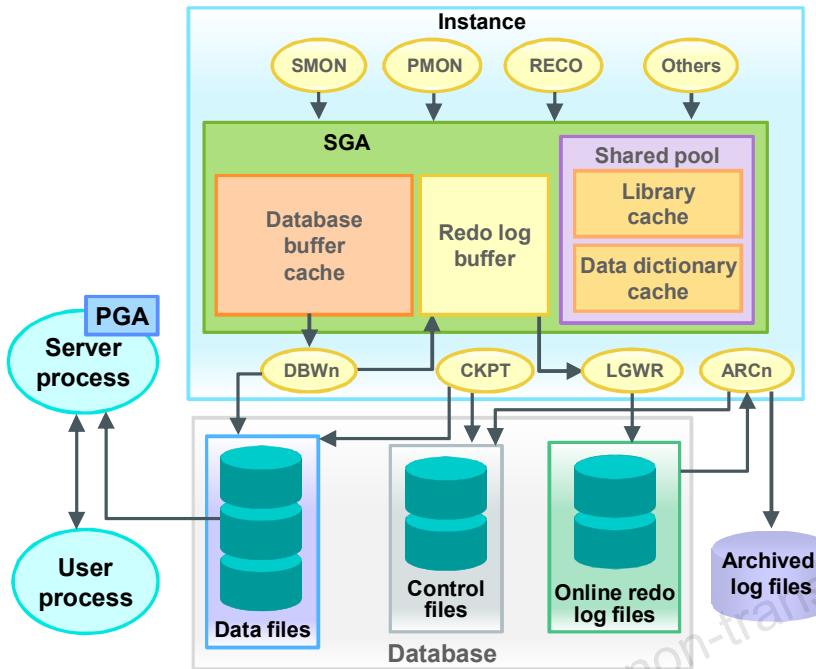
Advantages of the Fast COMMIT

The fast COMMIT mechanism ensures data recovery by writing changes to the redo log buffer instead of the data files. It has the following advantages:

- Sequential writes to the log files are faster than writing to different blocks in the data file.
- Only the minimal information that is necessary to record changes is written to the log files; writing to the data files would require whole blocks of data to be written.
- If multiple transactions request to commit at the same time, the instance piggybacks redo log records into a single write.
- Unless the redo log buffer is particularly full, only one synchronous write is required per transaction. If piggybacking occurs, there can be less than one synchronous write per transaction.
- Because the redo log buffer may be flushed before the COMMIT, the size of the transaction does not affect the amount of time needed for an actual COMMIT operation.

Note: Rolling back a transaction does not trigger LGWR to write to disk. The Oracle server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

Summary of the Oracle Database Architecture



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

An Oracle database comprises an instance and its associated database:

- An instance comprises the SGA and the background processes
 - **SGA:** Database buffer cache, redo log buffer, shared pool, and so on
 - **Background processes:** SMON, PMON, DBW n , CKPT, LGWR, and so on
- A database comprises storage structures:
 - **Logical:** Tablespaces, schemas, segments, extents, and Oracle block
 - **Physical:** Data files, control files, redo log files

When a user accesses the Oracle database through an application, a server process communicates with the instance on behalf of the user process.

Summary

In this appendix, you should have learned how to:

- List the major database architectural components
- Describe the background processes
- Explain the memory structures
- Correlate the logical and physical storage structures



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

Regular Expression Support

ORACLE®

Copyright ©2017, Oracle and/or its affiliates. All rights reserved.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license
to use this Student Guide.

Objectives

After completing this appendix, you should be able to:

- List the benefits of using regular expressions
- Use regular expressions to search for, match, and replace strings



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

What Are Regular Expressions?

- You use regular expressions to search for (and manipulate) simple and complex patterns in string data by using standard syntax conventions.
- You use a set of SQL functions and conditions to search for and manipulate strings in SQL and PL/SQL.
- You specify a regular expression by using:
 - Metacharacters, which are operators that specify the search algorithms
 - Literals, which are the characters for which you are searching



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Benefits of Using Regular Expressions

Regular expressions enable you to implement complex match logic in the database with the following benefits:

- By centralizing match logic in Oracle Database, you avoid intensive string processing of SQL result sets by middle-tier applications.
- Using server-side regular expressions to enforce constraints, you eliminate the need to code data validation logic on the client.
- The built-in SQL and PL/SQL regular expression functions and conditions make string manipulations more powerful and easier than in previous releases of Oracle Database.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the Regular Expressions Functions and Conditions in SQL and PL/SQL

Function or Condition Name	Description
REGEXP_LIKE	Is similar to the LIKE operator, but performs regular expression matching instead of simple pattern matching (condition)
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_INSTR	Searches a string for a regular expression pattern and returns the position where the match is found
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and extracts the matched substring
REGEXP_COUNT	Returns the number of times a pattern match is found in an input sting



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database provides a set of SQL functions that you use to search and manipulate strings by using regular expressions. You use these functions on a text literal, bind variable, or any column that holds character data, such as CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, and VARCHAR2 (but not LONG). A regular expression must be enclosed within single quotation marks. This ensures that the entire expression is interpreted by the SQL function and can improve the readability of your code.

- REGEXP_LIKE: This condition searches a character column for a pattern. Use this condition in the WHERE clause of a query to return rows matching the regular expression that you specify.
- REGEXP_REPLACE: This function searches for a pattern in a character column and replaces each occurrence of that pattern with the pattern that you specify.
- REGEXP_INSTR: This function searches a string for a given occurrence of a regular expression pattern. You specify which occurrence you want to find and the start position to search from. This function returns an integer indicating the position in the string where the match is found.
- REGEXP_SUBSTR: This function returns the actual substring matching the regular expression pattern that you specify.
- REGEXP_COUNT: This function returns the number of times a pattern match is found in the input string.

What are Metacharacters?

- Metacharacters are special characters that have a special meaning such as a wildcard, a repeating character, a nonmatching character, or a range of characters.
- You can use several predefined metacharacter symbols in the pattern matching.
- For example, the `^ (f | ht) tps? : $` regular expression searches for the following from the beginning of the string:
 - The literals `f` or `ht`
 - The `t` literal
 - The `p` literal, optionally followed by the `s` literal
 - The colon “`:`” literal at the end of the string



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The regular expression in the slide matches the `http:`, `https:`, `ftp:`, and `ftps:` strings.

Note: For a complete list of the regular expressions' metacharacters, see the *Oracle Database Advanced Application Developer's Guide* for Oracle Database 12c.

Using Metacharacters with Regular Expressions

Syntax	Description
.	Matches any character in the supported character set, except NULL
+	Matches one or more occurrences
?	Matches zero or one occurrence
*	Matches zero or more occurrences of the preceding subexpression
{m}	Matches exactly m occurrences of the preceding expression
{m, }	Matches at least m occurrences of the preceding subexpression
{m, n}	Matches at least m , but not more than n , occurrences of the preceding subexpression
[...]	Matches any single character in the list within the brackets
	Matches one of the alternatives
(...)	Treats the enclosed expression within the parentheses as a unit. The subexpression can be a string of literals or a complex expression containing operators.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using Metacharacters with Regular Expressions

Syntax	Description
^	Matches the beginning of a string
\$	Matches the end of a string
\	Treats the subsequent metacharacter in the expression as a literal
\n	Matches the <i>n</i> th (1–9) preceding subexpression of whatever is grouped within parentheses. The parentheses cause an expression to be remembered; a backreference refers to it.
\d	A digit character
[:class:]	Matches any character belonging to the specified POSIX character class
[^:class:]	Matches any single character <i>not</i> in the list within the brackets



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Beginning/end of line anchor, “ ^ ” and “ \$ ”: ^def matches def in the string defghi but does not match def in abcdef. def\$ matches def in the string abcdef but does not match def in the string defghi.

Escape character “ \ ”: \+ searches for a +. It matches the plus character in the string abc+def, but does not match Abcdef.

Backreference, “ \n ” : (abc | def) xy\1 matches the strings abcxyabc and defxydef, but does not match abcxydef or abcxy. A backreference enables you to search for a repeated string without knowing the actual string ahead of time. For example, the expression ^ (.*) \1\$ matches a line consisting of two adjacent instances of the same string.

Digit character, “ \d ”: The expression ^\ [\d{3}] \ \d{3} - \d{4} \$ matches [650] 555-1212 but does not match 650-555-1212.

Character class, “ [:class:] ”: [[:upper:]]+ searches for one or more consecutive uppercase characters. This matches DEF in the string abcDEFghi but does not match the string abcdefghi.

Nonmatching character list (or class), “ [^...] ”: [^abc] matches the character d in the string abcdef, but not a, b, or c.

Regular Expressions Functions and Conditions: Syntax

```
REGEXP_LIKE  (source_char, pattern [,match_option])
```

```
REGEXP_INSTR  (source_char, pattern [, position  
[, occurrence [, return_option  
[, match_option [, subexpr]]]])
```

```
REGEXP_SUBSTR (source_char, pattern [, position  
[, occurrence [, match_option  
[, subexpr]]]])
```

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence  
[, match_option]]]])
```

```
REGEXP_COUNT (source_char, pattern [, position  
[, occurrence [, match_option]]])
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The syntax for the regular expressions functions and conditions is as follows:

- **source_char**: A character expression that serves as the search value
- **pattern**: A regular expression, a text literal
- **occurrence**: A positive integer indicating which occurrence of pattern in source_char Oracle Server should search for. The default is 1.
- **position**: A positive integer indicating the character of source_char where Oracle Server should begin the search. The default is 1.
- **return_option**:
 - 0: Returns the position of the first character of the occurrence (default)
 - 1: Returns the position of the character following the occurrence
- **Replacestr**: Character string replacing pattern
- **match_parameter**:
 - “c”: Uses case-sensitive matching (default)
 - “i”: Uses non-case-sensitive matching
 - “n”: Allows match-any-character operator
 - “m”: Treats source string as multiple lines
- **subexpr**: Fragment of pattern enclosed in parentheses. You learn more about subexpressions later in this appendix.

Performing a Basic Search by Using the REGEXP_LIKE Condition

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Replacing Patterns by Using the REGEXP_REPLACE Function

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence [, match_option]]]])
```

```
SELECT last_name, REGEXP_REPLACE(phone_number, '\.', '-')  
AS phone  
FROM employees;
```

Original

#	LAST_NAME	PHONE
1	O'Connell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

Partial results

#	LAST_NAME	PHONE
1	O'Connell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Finding Patterns by Using the REGEXP_INSTR Function

```
REGEXP_INSTR  (source_char, pattern [, position [, occurrence [,  
return_option [, match_option]]]])
```

```
SELECT street_address,  
REGEXP_INSTR(street_address,'[:alpha:]') AS  
    First_Alpha_Position  
FROM locations;
```

STREET_ADDRESS	FIRST_ALPHA_POSITION
1 1297 Via Cola di Rie	6
2 93091 Calle della Testa	7
3 2017 Shinjuku-ku	6
4 9450 Kamiya-cho	6



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Extracting Substrings by Using the REGEXP_SUBSTR Function

```
REGEXP_SUBSTR (source_char, pattern [, position  
[, occurrence [, match_option]]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') AS Road  
FROM locations;
```

ROAD
1 Via
2 Calle
3 (null)
4 (null)
5 Jabberwocky



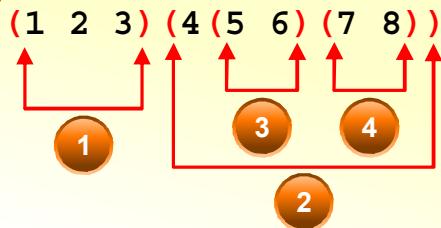
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Subexpressions

Examine this expression:

(1 2 3) (4 (5 6) (7 8))

The subexpressions are:



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database provides regular expression support parameters to access a subexpression. In the slide example, a string of digits is shown. The parentheses identify the subexpressions within the string of digits. Reading from left to right, and from outer parentheses to the inner parentheses, the subexpressions in the string of digits are:

1. 123
2. 45678
3. 56
4. 78

You can search for any of those subexpressions with the `REGEXP_INSTR` and `REGEXP_SUBSTR` functions.

Using Subexpressions with Regular Expression Support

```
SELECT
  REGEXP_INSTR
  (1 ('0123456789',           -- source char or search value
  2 '(123) (4(56) (78))',    -- regular expression patterns
  3 1,                      -- position to start searching
  4 1,                      -- occurrence
  5 0,                      -- return option
  6 'i',                    -- match option (case insensitive)
  7 1)                     -- subexpression on which to search
    "Position"
  FROM dual;
```

Position
1
2



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Why Access the *n*th Subexpression?

- A more realistic use: DNA sequencing
- You may need to find a specific subpattern that identifies a protein needed for immunity in mouse DNA.

```
SELECT
  REGEXP_INSTR ('ccacacctttccctccactcgttccacgttaaaagggtccctccatccccatgcccccttacc
  ctgcagggttagagtaggttagaaaccaggagagactccaagctccatctgtggaggggtccatcctggctgcagagagaggagaa
  tttggcccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcattttcaccctgc
  ccagcaggacactgcagcacccaaagggtttccaggagtaggggtccctcaagaggcttgggtctgtggccacatcctgga
  atttttcaagtgtatgtcacagccctgaggcatgtagggcgtgggatgcgcctgtctgtctctctctgaaccctg
  aacccttggctaccccagagcacttagagccag',
  '(gtc(tcac)(aaag))',
  1, 1, 0, 'i',
  1) "Position"
FROM dual;
```

Position
195



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In life sciences, you may need to extract the offsets of subexpression matches from a DNA sequence for further processing. For example, you may need to find a specific protein sequence, such as the begin offset for the DNA sequence preceded by `gtc` and followed by `tcac` followed by `aaag`. To accomplish this goal, you can use the `REGEXP_INSTR` function, which returns the position where a match is found.

In the slide example, the position of the first subexpression (`gtc`) is returned. `gtc` appears starting in position 195 of the DNA string.

If you modify the slide example to search for the second subexpression (`tcac`), the query results in the following output. `tcac` appears starting in position 198 of the DNA string.

Position
198

If you modify the slide example to search for the third subexpression (`aaag`), the query results in the following output. `aaag` appears starting in position 202 of the DNA string.

Position
202

REGEXP_SUBSTR: Example

```
SELECT
  REGEXP_SUBSTR
  (
    1 ('acgctgcactgca', -- source char or search value
    2 'acg(.* )gca',   -- regular expression pattern
    3 1,                -- position to start searching
    4 1,                -- occurrence
    5 'i',              -- match option (case insensitive)
    6 1)                -- sub-expression
  "Value"
FROM dual;
```

Value
1 ctgcact



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the REGEXP_COUNT Function

```
REGEXP_COUNT (source_char, pattern [, position  
[, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT( 'ccacctttccctccactcctcacgttccactgtaaagcgccctccatccccatgcccccttaccctgcag  
ggtagagtaggctgaaaaccagagagctccaagctccatctgtggagaggtgcacatcctggctgcagagagaggagaatttgc  
caccacccttagtccacaagccctgagttcatagcattttgagtttaccctgcccagcaggacactgcagcacccaaagggtccagg  
cctcaagaggcttgggctgtggccacatcctggattttcaagttgatggcacagccctgaggcatgtagggcgtgggatgcgtctg  
ccttcctgaaaccctgtggctaccctgagacttagagccag',  
'gtc') AS Count  
FROM dual;
```

R	COUNT
1	4



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The REGEXP_COUNT function evaluates strings by using characters as defined by the input character set. It returns an integer indicating the number of occurrences of the pattern. If no match is found, the function returns 0.

In the slide example, the number of occurrences for a DNA substring is determined by using the REGEXP_COUNT function.

The following example shows that the number of times the pattern 123 occurs in the string 123123123123 is three times. The search starts from the second position of the string.

```
SELECT REGEXP_COUNT  
( '123123123123', -- source char or search value  
  '123',           -- regular expression pattern  
  2,               -- position where the search should start  
  'i')             -- match option (case insensitive)  
    As Count  
FROM dual;
```

R	COUNT
1	3

Regular Expressions and Check Constraints: Examples

```
ALTER TABLE emp8
ADD CONSTRAINT email_addr
CHECK (REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
(500,'Christian','Patel','ChrisP2creme.com',
1234567890,'12-Jan-2004','HR REP',2000,null,102,40);
```

```
Error starting at line 1 in command:
INSERT INTO emp8 VALUES
(500,'Christian','Patel',
'ChrisP2creme.com', 1234567890,
'12-Jan-2004', 'HR REP', 2000, null, 102, 40)
Error report:
SQL Error: ORA-02290: check constraint (TEACH_B.EMAIL_ADDR) violated
02290. 00000 - "check constraint (%s,%s) violated"
*Cause:  The values being inserted do not satisfy the named check
*Action:  do not insert values that violate the constraint.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Regular expressions can also be used in CHECK constraints. In this example, a CHECK constraint is added on the EMAIL column of the EMPLOYEES table. This ensures that only strings containing an “@” symbol are accepted. The constraint is tested. The CHECK constraint is violated because the email address does not contain the required symbol. The NOVALIDATE clause ensures that existing data is not checked.

For the slide example, the emp8 table is created by using the following code:

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

Note: The slide example is executed by using the Execute Statement option in SQL Developer. The output format differs if you use the Run Script option.

Quiz



With the use of regular expressions in SQL and PL/SQL, you can:

- a. Avoid intensive string processing of SQL result sets by middle-tier applications
- b. Avoid data validation logic on the client
- c. Enforce constraints on the server



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this appendix, you should have learned how to use regular expressions to search for, match, and replace strings.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license
to use this Student Guide.