

# **Oracle Database 10g: SQL Fundamentals I**

**Student Guide • Volume 2**

D17108GC30

Edition 3.0

January 2009

D57871

**ORACLE®**

**Authors**

Salome Clement  
Chaitanya Koratamaddi  
Nancy Greenberg

**Technical Contributors  
and Reviewers**

Wayne Abbott  
Christian Bauwens  
Claire Bennett  
Perry Benson  
Brian Boxx  
Zarko Cesljas  
Dairy Chan  
Laszlo Czinkoczki  
Joel Goodman  
Matthew Gregory  
Sushma Jagannath  
Yash Jain  
Angelika Krupp  
Isabelle Marchand  
Malika Marghadi  
Valli Pataballa  
Narayanan Radhakrishnan  
Bryan Roberts  
Helen Robertson  
Lata Shivaprasad  
John Soltani  
James Spiller  
Priya Vennapusa

**Editors**

Arijit Ghosh  
Raj Kumar

**Graphic Designer**

Rajiv Chandrabhanu

**Publisher**

Giri Venugopal

**Copyright © 2009, Oracle. 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 is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

## Contents

### Preface

#### I Introduction

Lesson Objectives	I-2
Goals of the Course	I-3
Oracle10g	I-4
Oracle Database 10g	I-6
Oracle Application Server 10g	I-7
Oracle Enterprise Manager 10g Grid Control	I-8
Relational and Object Relational Database Management Systems	I-9
Oracle Internet Platform	I-10
System Development Life Cycle	I-11
Data Storage on Different Media	I-13
Relational Database Concept	I-14
Definition of a Relational Database	I-15
Data Models	I-16
Entity Relationship Model	I-17
Entity Relationship Modeling Conventions	I-19
Relating Multiple Tables	I-21
Relational Database Terminology	I-23
Relational Database Properties	I-25
Communicating with an RDBMS Using SQL	I-26
Oracle's Relational Database Management System	I-27
SQL Statements	I-28
Tables Used in the Course	I-29
Summary	I-30

#### 1 Retrieving Data Using the SQL `SELECT` Statement

Objectives	1-2
Capabilities of SQL <code>SELECT</code> Statements	1-3
Basic <code>SELECT</code> Statement	1-4
Selecting All Columns	1-5
Selecting Specific Columns	1-6
Writing SQL Statements	1-7
Column Heading Defaults	1-8

Arithmetic Expressions	1-9
Using Arithmetic Operators	1-10
Operator Precedence	1-11
Defining a Null Value	1-12
Null Values in Arithmetic Expressions	1-13
Defining a Column Alias	1-14
Using Column Aliases	1-15
Concatenation Operator	1-16
Literal Character Strings	1-17
Using Literal Character Strings	1-18
Alternative Quote (q) Operator	1-19
Duplicate Rows	1-20
Development Environments for SQL	1-21
What Is Oracle SQL Developer?	1-22
Oracle SQL Developer Interface	1-23
Creating a Database Connection	1-24
Browsing Database Objects	1-27
Using the SQL Worksheet	1-28
Executing SQL Statements	1-31
Formatting the SQL Code	1-32
Saving SQL Statements	1-33
Running Script Files	1-34
Displaying the Table Structure	1-35
Using the DESCRIBE Command	1-36
Summary	1-37
Practice 1: Overview	1-38

## 2 Restricting and Sorting Data

Objectives	2-2
Limiting Rows Using a Selection	2-3
Limiting the Rows That Are Selected	2-4
Using the WHERE Clause	2-5
Character Strings and Dates	2-6
Comparison Conditions	2-7
Using Comparison Conditions	2-8
Using the BETWEEN Condition	2-9
Using the IN Condition	2-10
Using the LIKE Condition	2-11
Using the NULL Conditions	2-13
Logical Conditions	2-14

Using the <code>AND</code> Operator	2-15
Using the <code>OR</code> Operator	2-16
Using the <code>NOT</code> Operator	2-17
Rules of Precedence	2-18
Using the <code>ORDER BY</code> Clause	2-20
Sorting	2-21
Substitution Variables	2-22
Using the <code>&amp;</code> Substitution Variable	2-24
Character and Date Values with Substitution Variables	2-26
Specifying Column Names, Expressions, and Text	2-27
Using the <code>&amp;&amp;</code> Substitution Variable	2-28
Using the <code>DEFINE</code> Command	2-29
Using the <code>VERIFY</code> Command	2-30
Summary	2-31
Practice 2: Overview	2-32

### **3 Using Single-Row Functions to Customize Output**

Objectives	3-2
SQL Functions	3-3
Two Types of SQL Functions	3-4
Single-Row Functions	3-5
Character Functions	3-7
Case-Manipulation Functions	3-9
Using Case-Manipulation Functions	3-10
Character-Manipulation Functions	3-11
Using the Character-Manipulation Functions	3-12
Number Functions	3-13
Using the <code>ROUND</code> Function	3-14
Using the <code>TRUNC</code> Function	3-15
Using the <code>MOD</code> Function	3-16
Working with Dates	3-17
Arithmetic with Dates	3-20
Using Arithmetic Operators with Dates	3-21
Date Functions	3-22
Using Date Functions	3-23
Practice 3: Overview of Part 1	3-25
Conversion Functions	3-26
Implicit Data Type Conversion	3-27
Explicit Data Type Conversion	3-29
Using the <code>TO_CHAR</code> Function with Dates	3-32

Elements of the Date Format Model	3-33
Using the TO_CHAR Function with Dates	3-37
Using the TO_CHAR Function with Numbers	3-38
Using the TO_NUMBER and TO_DATE Functions	3-41
RR Date Format	3-43
RR Date Format: Example	3-44
Nesting Functions	3-45
General Functions	3-47
NVL Function	3-48
Using the NVL Function	3-49
Using the NVL2 Function	3-50
Using the NULLIF Function	3-51
Using the COALESCE Function	3-52
Conditional Expressions	3-54
CASE Expression	3-55
Using the CASE Expression	3-56
DECODE Function	3-57
Using the DECODE Function	3-58
Summary	3-60
Practice 3: Overview of Part 2	3-61

#### **4 Reporting Aggregated Data Using the Group Functions**

Objectives	4-2
What Are Group Functions?	4-3
Types of Group Functions	4-4
Group Functions: Syntax	4-5
Using the AVG and SUM Functions	4-6
Using the MIN and MAX Functions	4-7
Using the COUNT Function	4-8
Using the DISTINCT Keyword	4-9
Group Functions and Null Values	4-10
Creating Groups of Data	4-11
Creating Groups of Data: GROUP BY Clause Syntax	4-12
Using the GROUP BY Clause	4-13
Grouping by More Than One Column	4-15
Using the GROUP BY Clause on Multiple Columns	4-16
Illegal Queries Using Group Functions	4-17
Restricting Group Results	4-19
Restricting Group Results with the HAVING Clause	4-20

Using the `HAVING` Clause 4-21

Nesting Group Functions 4-23

Summary 4-24

Practice 4: Overview 4-25

## **5 Displaying Data from Multiple Tables**

Objectives 5-2

Obtaining Data from Multiple Tables 5-3

Types of Joins 5-4

Joining Tables Using SQL:1999 Syntax 5-5

Creating Natural Joins 5-6

Retrieving Records with Natural Joins 5-7

Creating Joins with the `USING` Clause 5-8

Joining Column Names 5-9

Retrieving Records with the `USING` Clause 5-10

Qualifying Ambiguous Column Names 5-11

Using Table Aliases 5-12

Creating Joins with the `ON` Clause 5-13

Retrieving Records with the `ON` Clause 5-14

Self-Joins Using the `ON` Clause 5-15

Applying Additional Conditions to a Join 5-17

Creating Three-Way Joins with the `ON` Clause 5-18

Nonequijoins 5-19

Retrieving Records with Nonequijoins 5-20

Outer Joins 5-21

`INNER` Versus `OUTER` Joins 5-22

`LEFT OUTER JOIN` 5-23

`RIGHT OUTER JOIN` 5-24

`FULL OUTER JOIN` 5-25

Cartesian Products 5-26

Generating a Cartesian Product 5-27

Creating `Cross Joins` 5-28

Summary 5-29

Practice 5: Overview 5-30

## **6 Using Subqueries to Solve Queries**

Objectives 6-2

Using a Subquery to Solve a Problem 6-3

Subquery Syntax 6-4

Using a Subquery 6-5

Guidelines for Using Subqueries	6-6
Types of Subqueries	6-7
Single-Row Subqueries	6-8
Executing Single-Row Subqueries	6-9
Using Group Functions in a Subquery	6-10
The <code>HAVING</code> Clause with Subqueries	6-11
What Is Wrong with This Statement?	6-12
Will This Statement Return Rows?	6-13
Multiple-Row Subqueries	6-14
Using the <code>ANY</code> Operator in Multiple-Row Subqueries	6-15
Using the <code>ALL</code> Operator in Multiple-Row Subqueries	6-16
Null Values in a Subquery	6-17
Summary	6-19
Practice 6: Overview	6-20

## 7 Using the Set Operators

Objectives	7-2
Set Operators	7-3
Tables Used in This Lesson	7-4
<code>UNION</code> Operator	7-8
Using the <code>UNION</code> Operator	7-9
<code>UNION ALL</code> Operator	7-11
Using the <code>UNION ALL</code> Operator	7-12
<code>INTERSECT</code> Operator	7-13
Using the <code>INTERSECT</code> Operator	7-14
<code>MINUS</code> Operator	7-15
Set Operator Guidelines	7-17
The Oracle Server and Set Operators	7-18
Matching the <code>SELECT</code> Statements	7-19
Matching the <code>SELECT</code> Statement: Example	7-20
Controlling the Order of Rows	7-21
Summary	7-22
Practice 7: Overview	7-23

## 8 Manipulating Data

Objectives	8-2
Data Manipulation Language	8-3
Adding a New Row to a Table	8-4
<code>INSERT</code> Statement Syntax	8-5
Inserting New Rows	8-6



Inserting Rows with Null Values	8-7
Inserting Special Values	8-8
Inserting Specific Date Values	8-9
Creating a Script	8-10
Copying Rows from Another Table	8-11
Changing Data in a Table	8-12
UPDATE Statement Syntax	8-13
Updating Rows in a Table	8-14
Updating Two Columns with a Subquery	8-15
Updating Rows Based on Another Table	8-16
Removing a Row from a Table	8-17
DELETE Statement	8-18
Deleting Rows from a Table	8-19
Deleting Rows Based on Another Table	8-20
TRUNCATE Statement	8-21
Using a Subquery in an INSERT Statement	8-22
Database Transactions	8-24
Advantages of COMMIT and ROLLBACK Statements	8-26
Controlling Transactions	8-27
Rolling Back Changes to a Marker	8-28
Implicit Transaction Processing	8-29
State of the Data Before COMMIT or ROLLBACK	8-31
State of the Data After COMMIT	8-32
Committing Data	8-33
State of the Data After ROLLBACK	8-34
Statement-Level Rollback	8-36
Read Consistency	8-37
Implementation of Read Consistency	8-38
Summary	8-39
Practice 8: Overview	8-40

## **9 Using DDL Statements to Create and Manage Tables**

Objectives	9-2
Database Objects	9-3
Naming Rules	9-4
CREATE TABLE Statement	9-5
Referencing Another User's Tables	9-6
DEFAULT Option	9-7
Creating Tables	9-8
Data Types	9-9

Datetime Data Types	9-11
Including Constraints	9-17
Constraint Guidelines	9-18
Defining Constraints	9-19
NOT NULL Constraint	9-21
UNIQUE Constraint	9-22
PRIMARY KEY Constraint	9-24
FOREIGN KEY Constraint	9-25
FOREIGN KEY Constraint: Keywords	9-27
CHECK Constraint	9-28
CREATE TABLE: Example	9-29
Violating Constraints	9-30
Creating a Table by Using a Subquery	9-32
ALTER TABLE Statement	9-34
Dropping a Table	9-35
Summary	9-36
Practice 9: Overview	9-37

## **10 Creating Other Schema Objects**

Objectives	10-2
Database Objects	10-3
What Is a View?	10-4
Advantages of Views	10-5
Simple Views and Complex Views	10-6
Creating a View	10-7
Retrieving Data from a View	10-10
Modifying a View	10-11
Creating a Complex View	10-12
Rules for Performing DML Operations on a View	10-13
Using the WITH CHECK OPTION Clause	10-16
Denying DML Operations	10-18
Removing a View	10-20
Practice 10: Overview of Part 1	10-21
Sequences	10-22
CREATE SEQUENCE Statement: Syntax	10-24
Creating a Sequence	10-25
NEXTVAL and CURRVAL Pseudocolumns	10-26
Using a Sequence	10-28
Caching Sequence Values	10-29
Modifying a Sequence	10-30

Guidelines for Modifying a Sequence	10-31
Indexes	10-33
How Are Indexes Created?	10-35
Creating an Index	10-36
Index Creation Guidelines	10-37
Removing an Index	10-38
Synonyms	10-39
Creating and Removing Synonyms	10-41
Summary	10-42
Practice 10: Overview of Part 2	10-43

## **11 Managing Objects with Data Dictionary Views**

Objectives	11-2
The Data Dictionary	11-3
Data Dictionary Structure	11-4
How to Use the Dictionary Views	11-6
USER_OBJECTS and ALL_OBJECTS Views	11-7
USER_OBJECTS View	11-8
Table Information	11-9
Column Information	11-10
Constraint Information	11-12
View Information	11-15
Sequence Information	11-16
Synonym Information	11-18
Adding Comments to a Table	11-19
Summary	11-20
Practice 11: Overview	11-21

### **A Practice Solutions**

### **B Table Descriptions and Data**

### **C Oracle Join Syntax**

### **D Using SQL\*Plus**

### **E Using SQL Developer**

### **Index**

**Additional Practices**

**Additional Practices: Table Descriptions and Data**

**Additional Practices: Solutions**

## Using DDL Statements to Create and Manage Tables

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

In this lesson, you are introduced to the data definition language (DDL) statements. You are taught the basics of how to create simple tables, alter them, and remove them. The data types available in DDL are shown, and schema concepts are introduced. Constraints are tied into this lesson. Exception messages that are generated from violating constraints during DML are shown and explained.

## Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Database Objects

An Oracle Database can contain multiple data structures. Each structure should be outlined in the database design so that it can be created during the build stage of database development.

- **Table:** Stores data
- **View:** Subset of data from one or more tables
- **Sequence:** Generates numeric values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives alternative names to objects

### Oracle Table Structures

- Tables can be created at any time, even while users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- Table structure can be modified online.

**Note:** More database objects are available but are not covered in this course.

## Naming Rules

Table names and column names:

- Must begin with a letter
- Must be 1–30 characters long
- Must contain only A–Z, a–z, 0–9, \_, \$, and #
- Must not duplicate the name of another object owned by the same user
- Must not be an Oracle server–reserved word

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Naming Rules

You name database tables and columns according to the standard rules for naming any Oracle Database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, \_ (underscore), \$, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server–reserved word.

### Naming Guidelines

Use descriptive names for tables and other database objects.

**Note:** Names are case-insensitive. For example, EMPLOYEES is treated as the same name as eMPLOYEES or eMpLOYEES.

For more information, see “Object Names and Qualifiers” in the *Oracle Database SQL Reference*.



## CREATE TABLE Statement

- You must have:
  - The CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.] table
              (column datatype [DEFAULT expr] [, ...]);
```

- You specify the:
  - Table name
  - Column name, column data type, and column size



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE TABLE Statement

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements, which are a subset of SQL statements used to create, modify, or remove Oracle Database structures. These statements have an immediate effect on the database, and they also record information in the data dictionary.

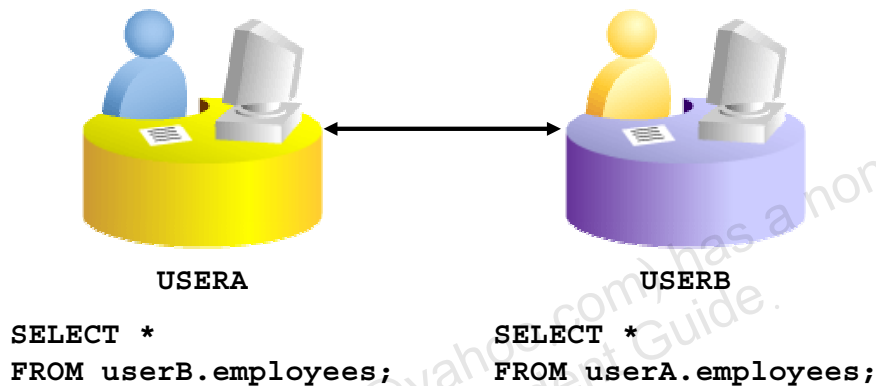
To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator uses data control language statements to grant privileges to users (DCL statements are covered in a later lesson).

In the syntax:

<i>schema</i>	Is the same as the owner's name
<i>table</i>	is the name of the table
DEFAULT <i>expr</i>	Specifies a default value if a value is omitted in the INSERT statement
<i>column</i>	Is the name of the column
<i>datatype</i>	Is the column's data type and length

## Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Referencing Another User's Tables

A *schema* is a collection of objects. Schema objects are the logical structures that directly refer to the data in a database. Schema objects include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.

If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named USERA and USERB, and both have an EMPLOYEES table, then if USERA wants to access the EMPLOYEES table that belongs to USERB, he or she must prefix the table name with the schema name:

```
SELECT *
FROM   userb.employees;
```

If USERB wants to access the EMPLOYEES table that is owned by USERA, he must prefix the table name with the schema name:

```
SELECT *
FROM   usera.employees;
```

## DEFAULT Option

- Specify a default value for a column during an insert.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates
  (id          NUMBER(8),
   hire_date DATE DEFAULT SYSDATE);
CREATE TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DEFAULT Option

When you define a table, you can specify that a column be given a default value by using the DEFAULT option. This option prevents null values from entering the columns if a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

**Note:** CURRVAL and NEXTVAL are explained later in this lesson.

## Creating Tables

- Create the table.

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc        VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
```

CREATE TABLE succeeded.

- Confirm table creation.

```
DESCRIBE dept
```

Name	Null	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE
4 rows selected		

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating Tables

The example in the slide creates the DEPT table, with four columns: DEPTNO, DNAME, LOC, and CREATE\_DATE. The CREATE\_DATE column has a default value. If a value is not provided for an INSERT statement, the system date is automatically inserted.

It further confirms the creation of the table by issuing the DESCRIBE command.

Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

## Data Types

Data Type	Description
VARCHAR2 ( <i>size</i> )	Variable-length character data
CHAR ( <i>size</i> )	Fixed-length character data
NUMBER ( <i>p</i> , <i>s</i> )	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Data Types

When you identify a column for a table, you need to provide a data type for the column. There are several data types available:

Data Type	Description
VARCHAR2 ( <i>size</i> )	Variable-length character data (A maximum <i>size</i> must be specified: minimum <i>size</i> is 1; maximum <i>size</i> is 4,000.)
CHAR [( <i>size</i> )]	Fixed-length character data of length <i>size</i> bytes (Default and minimum <i>size</i> is 1; maximum <i>size</i> is 2,000.)
NUMBER [( <i>p</i> , <i>s</i> )]	Number having precision <i>p</i> and scale <i>s</i> (The precision is the total number of decimal digits, and the scale is the number of digits to the right of the decimal point; the precision can range from 1 to 38, and the scale can range from –84 to 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)

**Data Types (continued)**

Data Type	Description
RAW ( <i>size</i> )	Raw binary data of length <i>size</i> (A maximum <i>size</i> must be specified: maximum <i>size</i> is 2,000.)
LONG RAW	Raw binary data of variable length (up to 2 GB)
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

**Guidelines**

- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

## Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Other Datetime Data Types

Data Type	Description
TIMESTAMP	Enables the time to be stored as a date with fractional seconds. There are several variations of this data type.
INTERVAL YEAR TO MONTH	Enables time to be stored as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month.
INTERVAL DAY TO SECOND	Enables time to be stored as an interval of days, hours, minutes, and seconds. Used to represent the precise difference between two datetime values.

**Note:** These datetime data types are available with Oracle9i and later releases. For detailed information about the datetime data types, see the topics “TIMESTAMP Datatype,” “INTERVAL YEAR TO MONTH Datatype,” and “INTERVAL DAY TO SECOND Datatype” in the *Oracle SQL Reference*.

## Datetime Data Types

- The `TIMESTAMP` data type is an extension of the `DATE` data type.
- It stores the year, month, and day of the `DATE` data type plus hour, minute, and second values as well as the fractional second value.
- You can optionally specify the time zone.

```
TIMESTAMP[(fractional_seconds_precision)]
```

```
TIMESTAMP[(fractional_seconds_precision)]  
WITH TIME ZONE
```

```
TIMESTAMP[(fractional_seconds_precision)]  
WITH LOCAL TIME ZONE
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### TIMESTAMP Data Type

The `TIMESTAMP` data type is an extension of the `DATE` data type. It stores the year, month, and day of the `DATE` data type plus hour, minute, and second values. This data type is used for storing precise time values.

The `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 to 9. The default is 6.

#### Example

In this example, a table is created named `NEW_EMPLOYEES`, with a column `START_DATE` that has a data type of `TIMESTAMP`:

```
CREATE TABLE new_employees  
(employee_id NUMBER,  
 first_name VARCHAR2(15),  
 last_name VARCHAR2(15),  
 ...  
 start_date TIMESTAMP(7),  
 ...);
```

Suppose that two rows are inserted in the `NEW_EMPLOYEES` table. The displayed output shows the differences. (A `DATE` data type defaults to display the `DD-MON-RR` format.):



**TIMESTAMP Data Type (continued)**

```

SELECT start_date
FROM   new_employees;

17-JUN-03 12.00.00.000000 AM
21-SEP-03 12.00.00.000000 AM

```

**TIMESTAMP WITH TIME ZONE Data Type**

TIMESTAMP WITH TIME ZONE is a variant of TIMESTAMP that includes a time-zone displacement in its value. The time-zone displacement is the difference (in hours and minutes) between local time and UTC (Universal Time Coordinate, formerly known as Greenwich Mean Time). This data type is used for collecting and evaluating date information across geographic regions.

For example,

```
TIMESTAMP '2003-04-15 8:00:00 -8:00'
```

is the same as

```
TIMESTAMP '2003-04-15 11:00:00 -5:00'
```

That is, 8:00 a.m. Pacific Standard Time is the same as 11:00 a.m. Eastern Standard Time.

This can also be specified as follows:

```
TIMESTAMP '2003-04-15 8:00:00 US/Pacific'
```

**TIMESTAMP WITH LOCAL TIME ZONE Data Type**

TIMESTAMP WITH LOCAL TIME ZONE is another variant of TIMESTAMP that includes a time-zone displacement in its value. It differs from TIMESTAMP WITH TIME ZONE in that data stored in the database is normalized to the database time zone, and the time-zone displacement is not stored as part of the column data. When users retrieve the data, it is returned in the users' local session time zone. The time-zone displacement is the difference (in hours and minutes) between local time and UTC.

Unlike TIMESTAMP WITH TIME ZONE, you can specify columns of type TIMESTAMP WITH LOCAL TIME ZONE as part of a primary or unique key, as in the following example:

```

CREATE TABLE time_example
  (order_date TIMESTAMP WITH LOCAL TIME ZONE);

INSERT INTO time_example VALUES('15-JAN-04 09:34:28 AM');

SELECT *
FROM   time_example;

ORDER_DATE
-----
15-JAN-04 09.34.28.000000 AM

```

The TIMESTAMP WITH LOCAL TIME ZONE type is appropriate for two-tier applications in which you want to display dates and times using the time zone of the client system.

## Datetime Data Types

- The INTERVAL YEAR TO MONTH data type stores a period of time using the YEAR and MONTH datetime fields:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

- The INTERVAL DAY TO SECOND data type stores a period of time in terms of days, hours, minutes, and seconds:

```
INTERVAL DAY [(day_precision)]  
TO SECOND [(fractional_seconds_precision)]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INTERVAL YEAR TO MONTH Data Type

INTERVAL YEAR TO MONTH stores a period of time using the YEAR and MONTH datetime fields.

Use INTERVAL YEAR TO MONTH to represent the difference between two datetime values, where the only significant portions are the year and month. For example, you might use this value to set a reminder for a date that is 120 months in the future, or check whether 6 months have elapsed since a particular date.

In the syntax:

`year_precision`

is the number of digits in the YEAR datetime field.

The default value of `year_precision` is 2.

#### 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 digits

**INTERVAL YEAR TO MONTH Data Type (continued)**

```

CREATE TABLE time_example2
(loan_duration INTERVAL YEAR (3) TO MONTH);

INSERT INTO time_example2 (loan_duration)
VALUES (INTERVAL '120' MONTH(3));

SELECT TO_CHAR(sysdate+loan_duration, 'dd-mon-yyyy')
FROM   time_example2;           --today's date is 11-11-2008

```

	TO_CHAR(SYSDATE+LOAN_DURATION,'DD-MON-YYYY')
1	11-nov-2018

**INTERVAL DAY TO SECOND Data Type**

INTERVAL DAY TO SECOND stores a period of time in terms of days, hours, minutes, and seconds.

Use INTERVAL DAY TO SECOND to represent the precise difference between two datetime values. For example, you might use this value to set a reminder for a time that is 36 hours in the future, or to record the time between the start and end of a race. To represent long spans of time, including multiple years, with high precision, you can use a large value for the days portion.

In the syntax:

day\_precision

Is the number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2.

fractional\_seconds\_precision

Is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.

**Examples**


- INTERVAL '4 5:12:10.222' DAY TO SECOND(3)  
Indicates 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second.
- INTERVAL '180' DAY(3)  
Indicates 180 days.
- INTERVAL '4 5:12:10.222' DAY TO SECOND(3)  
Indicates 4 days, 5 hours, 12 minutes, 10 seconds, and 222 thousandths of a second
- INTERVAL '4 5:12' DAY TO MINUTE  
Indicates 4 days, 5 hours, and 12 minutes
- INTERVAL '400 5' DAY(3) TO HOUR  
Indicates 400 days and 5 hours.
- INTERVAL '11:12:10.2222222' HOUR TO SECOND(7)  
Indicates 11 hours, 12 minutes, and 10.2222222 seconds.

**INTERVAL DAY TO SECOND Data Type (continued)****Example**

```
CREATE TABLE time_example3
(day_duration INTERVAL DAY (3) TO SECOND);
```

```
INSERT INTO time_example3 (day_duration)
VALUES (INTERVAL '180' DAY(3));
```

```
SELECT sysdate + day_duration "Half Year"
FROM   time_example3;           --today's date is 11-11-2008
```

	 Half Year
1	10-MAY-09
2	10-MAY-09

## Including Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables
- Provide rules for Oracle tools, such as Oracle Developer

### Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

## Constraint Guidelines

- You can name a constraint, or the Oracle server generates a name with the `SYS_Cn` format.
- Create a constraint at either of the following times:
  - At the same time as the table is created
  - After the table has been created
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Constraint Guidelines

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where *n* is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the table has been created.

For more information, see “Constraints” in *Oracle Database SQL Reference*.

## Defining Constraints

- Syntax:

```
CREATE TABLE [schema.] table
  (column datatype [DEFAULT expr]
   [column_constraint],
   ...
   [table_constraint] [, ...] );
```

- Column-level constraint:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint:

```
column, ...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Defining Constraints

The slide gives the syntax for defining constraints when creating a table. You can create the constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses.

NOT NULL constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

<i>schema</i>	Is the same as the owner's name
<i>table</i>	Is the name of the table
DEFAULT <i>expr</i>	Specifies a default value to use if a value is omitted in the INSERT statement
<i>column</i>	Is the name of the column
<i>datatype</i>	Is the column's data type and length
<i>column_constraint</i>	Is an integrity constraint as part of the column definition
<i>table_constraint</i>	Is an integrity constraint as part of the table definition

## Defining Constraints

- Column-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

1

- Table-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Defining Constraints (continued)

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also temporarily disabled.

Both slide examples create a primary key constraint on the EMPLOYEE\_ID column of the EMPLOYEES table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint.

More details about the primary key constraint are provided later in this lesson.



## NOT NULL Constraint

Ensures that null values are not permitted for the column:

	EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
1	178	Grant	KCRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	(null)
2	206	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	110
3	205	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	110
4	100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
5	102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90

...  
↑  
**NOT NULL constraint**  
(No row can contain  
a null value for  
this column.)

↑  
**NOT NULL  
constraint**

↑  
**Absence of NOT NULL  
constraint**  
(Any row can contain  
a null value for this  
column.)

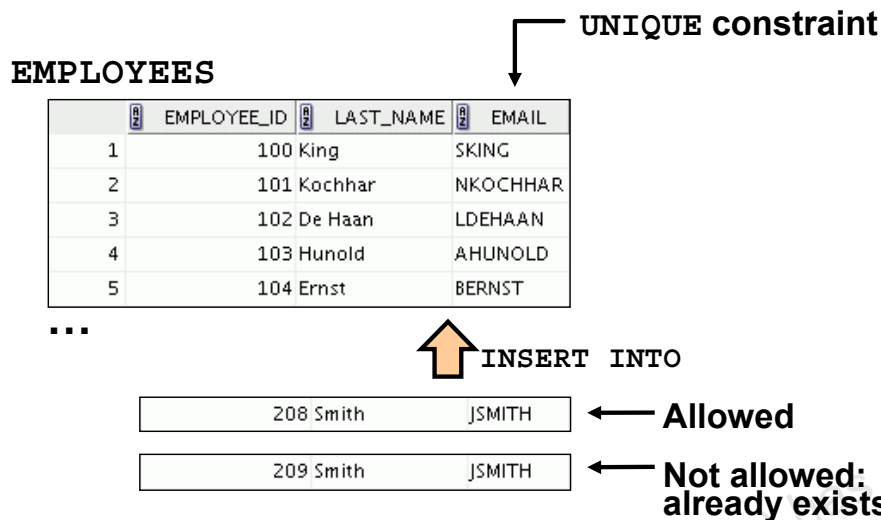
ORACLE

Copyright © 2009, Oracle. All rights reserved.

### NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level.

## UNIQUE Constraint



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### UNIQUE Constraint

A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.

UNIQUE constraints enable the input of nulls unless you also define NOT NULL constraints for the same columns. In fact, any number of rows can include nulls for columns without NOT NULL constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint.

**Note:** Because of the search mechanism for UNIQUE constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite UNIQUE key constraint.

## UNIQUE Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees (  
    employee_id      NUMBER(6) ,  
    last_name        VARCHAR2(25) NOT NULL ,  
    email            VARCHAR2(25) ,  
    salary           NUMBER(8,2) ,  
    commission_pct   NUMBER(2,2) ,  
    hire_date        DATE NOT NULL ,  
    ...  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### UNIQUE Constraint (continued)

UNIQUE constraints can be defined at the column level or table level. A composite unique key is created by using the table-level definition.

The example in the slide applies the UNIQUE constraint to the EMAIL column of the EMPLOYEES table. The name of the constraint is EMP\_EMAIL\_UK.

**Note:** The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

## PRIMARY KEY Constraint

### DEPARTMENTS

PRIMARY KEY

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	(null)	(null)
4	40	Human Resources	(null)	2500
5	50	Shipping	124	1500

Not allowed  
(null value)



INSERT INTO

(null)	Public Accounting	(null)	1400
50	Finance	124	1500

Not allowed  
(50 already exists)

ORACLE

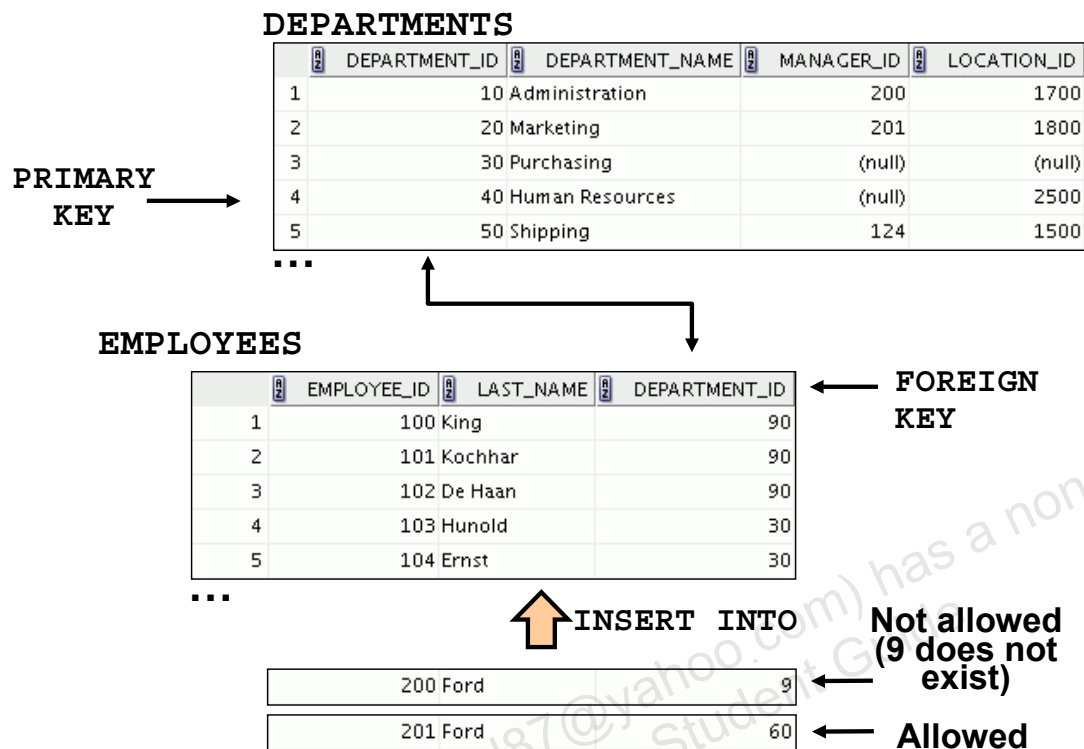
Copyright © 2009, Oracle. All rights reserved.

### PRIMARY KEY Constraint

A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or set of columns that uniquely identifies each row in a table. This constraint enforces uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

**Note:** Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

## FOREIGN KEY Constraint



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT\_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT\_ID column of the DEPARTMENTS table (the referenced or parent table).

#### Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

## FOREIGN KEY Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees (
  employee_id      NUMBER(6),
  last_name        VARCHAR2(25) NOT NULL,
  email            VARCHAR2(25),
  salary           NUMBER(8,2),
  commission_pct   NUMBER(2,2),
  hire_date        DATE NOT NULL,
  ...
  department_id    NUMBER(4),
  CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
    REFERENCES departments(department_id),
  CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOREIGN KEY Constraint (continued)

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT\_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP\_DEPTID\_FK.

The foreign key can also be defined at the column level, provided the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear. For example:

```
CREATE TABLE employees
(
  ...
  department_id NUMBER(4) CONSTRAINT emp_deptid_fk
    REFERENCES departments(department_id),
  ...
)
```

## FOREIGN KEY Constraint: Keywords

- FOREIGN KEY: Defines the column in the child table at the table-constraint level
- REFERENCES: Identifies the table and column in the parent table
- ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted
- ON DELETE SET NULL: Converts dependent foreign key values to null

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOREIGN KEY Constraint: Keywords

The foreign key is defined in the child table, and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- FOREIGN KEY is used to define the column in the child table at the table-constraint level.
- REFERENCES identifies the table and column in the parent table.
- ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table are also deleted.
- ON DELETE SET NULL converts foreign key values to null when the parent value is removed.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the ON DELETE CASCADE or the ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table.

## CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

```
..., salary NUMBER(2)
      CONSTRAINT emp_salary_min
      CHECK (salary > 0),...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CHECK Constraint

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as query conditions, with the following exceptions:

- References to the CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
- Calls to SYSDATE, UID, USER, and USERENV functions
- Queries that refer to other values in other rows

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

```
CREATE TABLE employees
(
  ...
  salary NUMBER(8,2) CONSTRAINT emp_salary_min
  CHECK (salary > 0),
  ...
)
```



## CREATE TABLE: Example

```
CREATE TABLE employees
( employee_id      NUMBER(6)
  CONSTRAINT emp_employee_id PRIMARY KEY
, first_name       VARCHAR2(20)
, last_name        VARCHAR2(25)
  CONSTRAINT emp_last_name_nn NOT NULL
, email            VARCHAR2(25)
  CONSTRAINT emp_email_nn    NOT NULL
  CONSTRAINT emp_email_uk    UNIQUE
, phone_number     VARCHAR2(20)
, hire_date        DATE
  CONSTRAINT emp_hire_date_nn NOT NULL
, job_id           VARCHAR2(10)
  CONSTRAINT emp_job_nn      NOT NULL
, salary           NUMBER(8,2)
  CONSTRAINT emp_salary_ck   CHECK (salary>0)
, commission_pct   NUMBER(2,2)
, manager_id       NUMBER(6)
, department_id    NUMBER(4)
  CONSTRAINT emp_dept_fk     REFERENCES
    departments (department_id));
```

ORACLE

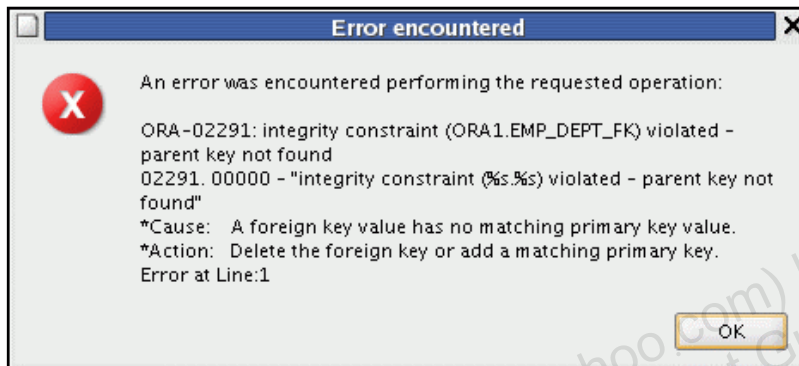
Copyright © 2009, Oracle. All rights reserved.

### CREATE TABLE: Example

The example shows the statement used to create the EMPLOYEES table in the HR schema.

## Violating Constraints

```
UPDATE employees  
SET    department_id = 55  
WHERE  department_id = 110;
```



Department 55 does not exist.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Integrity Constraint Error

When you have constraints in place on columns, an error is returned to you if you try to violate the constraint rule.

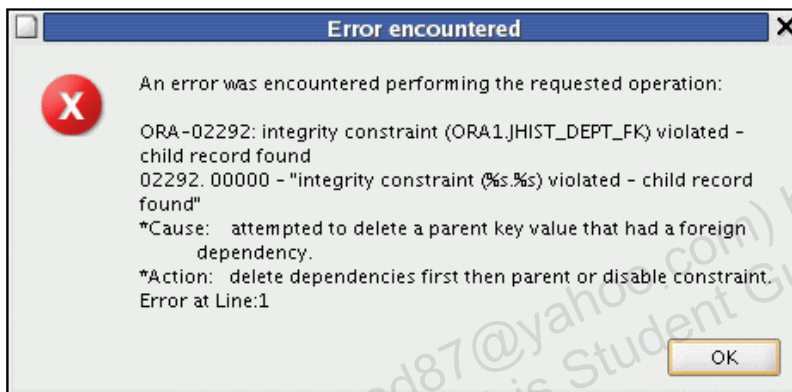
For example, if you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

In the example in the slide, department 55 does not exist in the parent table, DEPARTMENTS, and so you receive the *parent key* violation ORA-02291.

## Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE      department_id = 60;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Integrity Constraint Error (continued)

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department 60 from the DEPARTMENTS table, but it results in an error because that department number is used as a foreign key in the EMPLOYEES table. If the parent record that you attempt to delete has child records, then you receive the *child record found* violation ORA-02292.

The following statement works because there are no employees in department 70:

```
DELETE FROM departments
WHERE      department_id = 70;
```

1 row deleted.

## Creating a Table by Using a Subquery

- Create a table and insert rows by combining the `CREATE TABLE` statement and the `AS` subquery option.

```
CREATE TABLE table
      [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Table from Rows in Another Table

A second method for creating a table is to apply the `AS subquery` clause, which both creates the table and inserts rows returned from the subquery.

In the syntax:

*table* is the name of the table

*column* is the name of the column, default value, and integrity constraint

*subquery* is the `SELECT` statement that defines the set of rows to be inserted into the new table

### Guidelines

- The table is created with the specified column names, and the rows retrieved by the `SELECT` statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery `SELECT` list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the `NOT NULL` constraint are passed to the new table. The other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

## Creating a Table by Using a Subquery

```
CREATE TABLE dept80
AS
SELECT  employee_id, last_name,
        salary*12 ANNSAL,
        hire_date
FROM    employees
WHERE   department id = 80;
CREATE TABLE Succeeded.
```

```
DESCRIBE dept80
```

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE
4 rows selected		

ORACLE

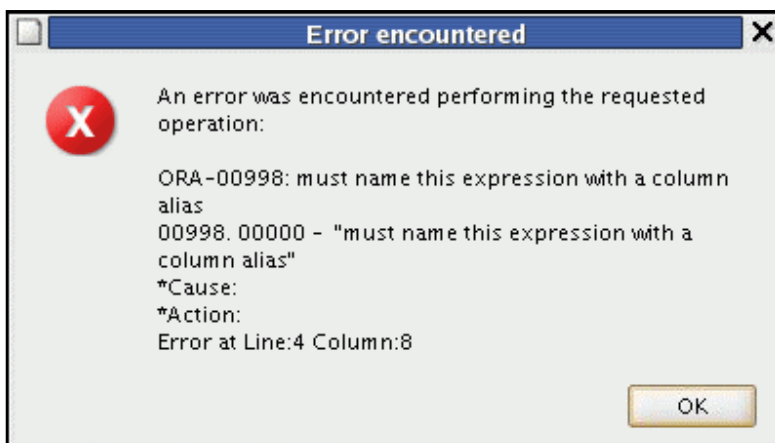
Copyright © 2009, Oracle. All rights reserved.

### Creating a Table from Rows in Another Table (continued)

The slide example creates a table named DEPT80, which contains details of all the employees working in department 80. Notice that the data for the DEPT80 table comes from the EMPLOYEES table.

You can verify the existence of a database table and check column definitions by using the DESCRIBE command.

Be sure to provide a column alias when selecting an expression. The expression SALARY\*12 is given the alias ANNSAL. Without the alias, the following error is generated:



## ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE Statement

After you create a table, you may need to change the table structure for any of the following reasons:

- You omitted a column.
- Your column definition needs to be changed.
- You need to remove columns.

You can do this by using the ALTER TABLE statement. For information about the ALTER TABLE statement, see the *Oracle Database 10g SQL Fundamentals II* course.

## Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You *cannot* roll back the DROP TABLE statement.

```
DROP TABLE dept80;  
DROP TABLE succeeded.
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Dropping a Table

The DROP TABLE statement removes the definition of an Oracle table. When you drop a table, the database loses all the data in the table and all the indexes associated with it.

#### Syntax

```
DROP TABLE table
```

In the syntax, *table* is the name of the table.

#### Guidelines

- All data is deleted from the table.
- Any views and synonyms remain but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the DROP ANY TABLE privilege can remove a table.

**Note:** The DROP TABLE statement, once executed, is irreversible. The Oracle server does not question the action when you issue the DROP TABLE statement. If you own that table or have a high-level privilege, then the table is immediately removed. As with all DDL statements, DROP TABLE is committed automatically.

## Summary

In this lesson, you should have learned how to:

- Use the `CREATE TABLE` statement to create a table and include constraints
- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to do the following:

### **CREATE TABLE**

- Use the `CREATE TABLE` statement to create a table and include constraints.
- Create a table based on another table by using a subquery.

### **DROP TABLE**

- Remove rows and a table structure.
- Once executed, this statement cannot be rolled back.



## Practice 9: Overview

This practice covers the following topics:

- Creating new tables
- Creating a new table by using the `CREATE TABLE AS` syntax
- Verifying that tables exist
- Dropping tables

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Practice 9: Overview

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table is added to the database. Create the syntax in the command file, and then execute the command file to create the table.

## Practice 9

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab\_09\_01.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

<b>Column Name</b>	ID	NAME
<b>Key Type</b>	Primary key	
<b>Nulls/Unique</b>		
<b>FK Table</b>		
<b>FK Column</b>		
<b>Data type</b>	NUMBER	VARCHAR2
<b>Length</b>	7	25

Name	Null	Type
-----	-----	-----
ID	NOT NULL	NUMBER(7)
NAME		VARCHAR2(25)
2 rows selected		

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.
3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab\_09\_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

<b>Column Name</b>	ID	LAST_NAME	FIRST_NAME	DEPT_ID
<b>Key Type</b>				
<b>Nulls/Unique</b>				
<b>FK Table</b>				DEPT
<b>FK Column</b>				ID
<b>Data type</b>	NUMBER	VARCHAR2	VARCHAR2	NUMBER
<b>Length</b>	7	25	25	7

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

**Practice 9 (continued)**

4. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPARTMENT\_ID columns. Name the columns in your new table ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPT\_ID, respectively.
5. Drop the EMP table.

VINOD BUSANAGA (b\_vinod87@yahoo.com) has a non-transferable  
license to use this Student Guide.

# 10

## Creating Other Schema Objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- Create simple and complex views
- Retrieve data from views
- Create, maintain, and use sequences
- Create and maintain indexes
- Create private and public synonyms

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

In this lesson, you are introduced to the view, sequence, synonym, and index objects. You are taught the basics of creating and using views, sequences, and indexes.

## Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Database Objects

There are several other objects in a database in addition to tables. In this lesson, you learn about views, sequences, indexes, and synonyms.

With views, you can present and hide data from tables.

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of some queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

# What Is a View?

## EMPLOYEES table

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
4	103	Alexander	Hunold					9000
5	104	Bruce	Ernst					6000
6	107	Diana						4200
7	124	Kevin						5800
8	141	Trenna						3500
9	142	Curtis						3100
10	143	Randall						2600
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
20	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

ORACLE

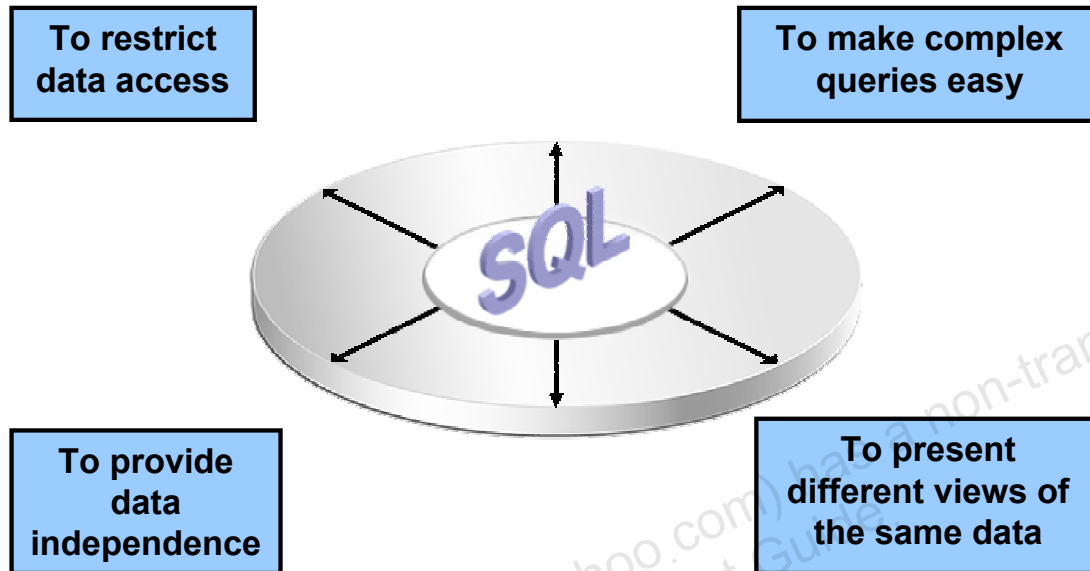
Copyright © 2009, Oracle. All rights reserved.

## What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a `SELECT` statement in the data dictionary.



## Advantages of Views



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Advantages of Views

- Views restrict access to the data because the view can display selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

For more information, see “CREATE VIEW” in the *Oracle SQL Reference*.

## Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Simple Views and Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (INSERT, UPDATE, and DELETE) operations.

- A simple view is one that:
  - Derives data from only one table
  - Contains no functions or groups of data
  - Can perform DML operations through the view
- A complex view is one that:
  - Derives data from many tables
  - Contains functions or groups of data
  - Does not always allow DML operations through the view

## Creating a View

- You embed a subquery in the CREATE VIEW statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
  AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- The subquery can contain complex SELECT syntax.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View

You can create a view by embedding a subquery in the CREATE VIEW statement.

In the syntax:

OR REPLACE	Re-creates the view if it already exists
FORCE	Creates the view regardless of whether or not the base tables exist
NOFORCE	Creates the view only if the base tables exist (This is the default.)
<i>view</i>	Is the name of the view
<i>alias</i>	Specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
<i>subquery</i>	Is a complete SELECT statement (You can use aliases for the columns in the SELECT list.)
WITH CHECK OPTION	Specifies that only those rows that are accessible to the view can be inserted or updated
<i>constraint</i>	Is the name assigned to the CHECK OPTION constraint
WITH READ ONLY	ensures that no DML operations can be performed on this view

## Creating a View

- Create the EMPVU80 view, which contains details of employees in department 80:

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;
CREATE VIEW succeeded.
```

- Describe the structure of the view by using the DESCRIBE command:

```
DESCRIBE empvu80
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View (continued)

The example in the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the DESCRIBE command.

Name	Null	Type
-----	-----	-----
EMPLOYEE_ID	NOT NULL	NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
3 rows selected		

### Guidelines for Creating a View

- The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.
- If you do not specify a constraint name for a view created with the WITH CHECK OPTION, the system assigns a default name in the format SYS\_Cn.
- You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it or regranteeing object privileges previously granted on it.

## Creating a View

- Create a view by using column aliases in the subquery:

```
CREATE VIEW    salvu50
AS SELECT     employee_id ID_NUMBER, last_name NAME,
              salary*12 ANN_SALARY
FROM          employees
WHERE         department_id = 50;
CREATE VIEW succeeded.
```

- Select the columns from this view by the given alias names:

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View (continued)

You can control the column names by including column aliases in the subquery.

The example in the slide creates a view containing the employee number (EMPLOYEE\_ID) with the alias ID\_NUMBER, name (LAST\_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN\_SALARY for every employee in department 50.

As an alternative, you can use an alias after the CREATE statement and before the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery.

```
CREATE OR REPLACE VIEW    salvu50 (ID_NUMBER, NAME, ANN_SALARY)
AS SELECT     employee_id, last_name, salary*12
FROM          employees
WHERE         department_id = 50;
CREATE VIEW succeeded.
```

## Retrieving Data from a View

```
SELECT *  
FROM salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.

## Modifying a View

- Modify the EMPVU80 view by using a CREATE OR REPLACE VIEW clause. Add an alias for each column name:

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
FROM      employees
WHERE     department_id = 80;
CREATE VIEW succeeded.
```

- Column aliases in the CREATE OR REPLACE VIEW clause are listed in the same order as the columns in the subquery.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Modifying a View



With the OR REPLACE option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranteeing object privileges.

**Note:** When assigning column aliases in the CREATE OR REPLACE VIEW clause, remember that the aliases are listed in the same order as the columns in the subquery.





## Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views. 
- You cannot remove a row if the view contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Performing DML Operations on a View

You can perform DML operations on data through a view if those operations follow certain rules.

You can remove a row from a view unless it contains any of the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

## Rules for Performing DML Operations on a View

You cannot modify data in a view if it contains:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Performing DML Operations on a View (continued)

You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions (for example, `SALARY * 12`).

## Rules for Performing DML Operations on a View

You cannot add data through a view if the view includes:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions
- `NOT NULL` columns in the base tables that are not selected by the view

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Performing DML Operations on a View (continued)

You can add data through a view unless it contains any of the items listed in the slide. You cannot add data to a view if the view contains `NOT NULL` columns without default values in the base table. All required values must be present in the view. Remember that you are adding values directly to the underlying table *through* the view.

For more information, see “`CREATE VIEW`” in the *Oracle SQL Reference*.

## Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
   FROM        employees
   WHERE       department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
CREATE VIEW succeeded.
```

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the WITH CHECK OPTION Clause

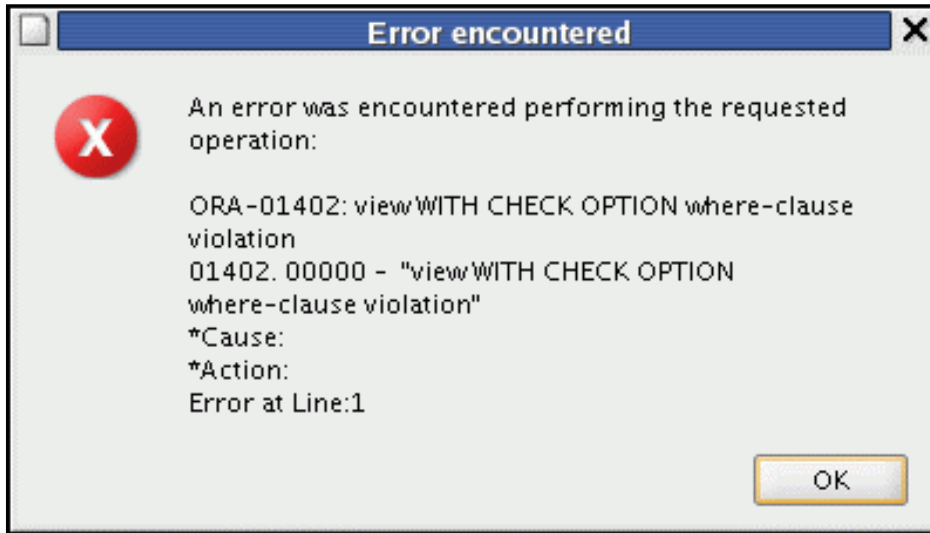
It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select, and therefore it enables integrity constraints and data validation checks to be enforced on data being inserted or updated.

**Note:** The WITH CHECK OPTION clause relates to inserts and updates on the view only. It has no effect on deletes.

If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```

**Using the WITH CHECK OPTION Clause (continued)**

**Note:** No rows are updated because if the department number were to change to 10, the view would no longer be able to see that employee. With the WITH CHECK OPTION clause, therefore, the view can see only employees in department 20 and does not allow the department number for those employees to be changed through the view.

## Denying DML Operations

- You can ensure that no DML operations occur by adding the `WITH READ ONLY` option to your view definition.
- Any attempt to perform a DML operation on any row in the view results in an Oracle server error.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the `WITH READ ONLY` option. The example in the next slide modifies the `EMPVU10` view to prevent any DML operations on the view.

## Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
  (employee_number, employee_name, job_title)
AS SELECT      employee_id, last_name, job_id
  FROM        employees
  WHERE       department_id = 10
  WITH READ ONLY ;
CREATE VIEW succeeded.
```

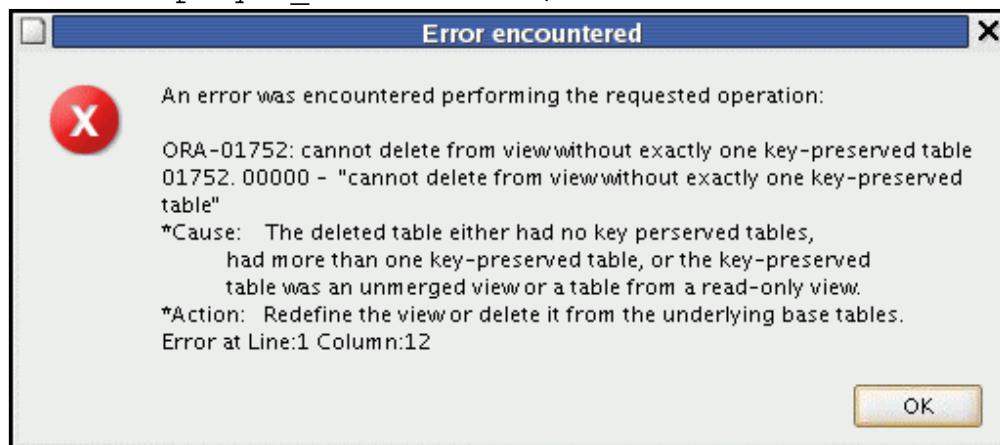
ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Denying DML Operations (continued)

Any attempt to remove a row from a view with a read-only constraint results in an error:

```
DELETE FROM empvu10
WHERE employee_number = 200;
```



Any attempt to insert a row or modify a row using the view with a read-only constraint results in an Oracle server error:

```
01733: virtual column not allowed here.
```

## Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;  
DROP VIEW empvu80 succeeded.
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Removing a View

You use the DROP VIEW statement to remove a view. The statement removes the view definition from the database. Dropping views has no effect on the tables on which the view was based. Views or other applications based on deleted views become invalid. Only the creator or a user with the DROP ANY VIEW privilege can remove a view.

In the syntax:

*view*            is the name of the view



## Practice 10: Overview of Part 1

This practice covers the following topics:

- Creating a simple view
- Creating a complex view
- Creating a view with a check constraint
- Attempting to modify data in the view
- Removing views

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Practice 10: Overview of Part 1

Part 1 of this lesson's practice provides you with a variety of exercises in creating, using, and removing views.

Complete questions 1–6 at the end of this lesson.

## Sequences

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

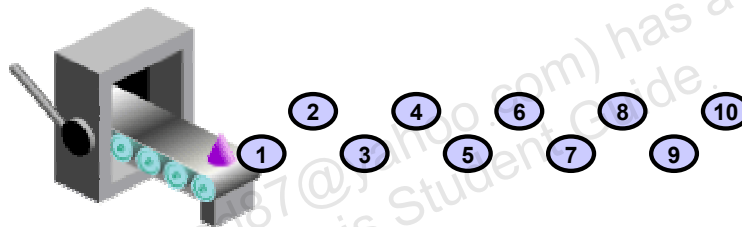
### Sequences

A sequence is a database object that creates integer values. You can create sequences and then use them to generate numbers.

# Sequences

A sequence:

- Can automatically generate unique numbers
- Is a sharable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Sequences (continued)

A sequence is a user-created database object that can be shared by multiple users to generate integers.

You can define a sequence to generate unique values or to recycle and use the same numbers again.

A typical usage for sequences is to create a primary key value, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence-generating routine.

Sequence numbers are stored and generated independently of tables. Therefore, the same sequence can be used for multiple tables.

## CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Sequence

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

<i>sequence</i>	Is the name of the sequence generator
INCREMENT BY <i>n</i>	Specifies the interval between sequence numbers, where <i>n</i> is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH <i>n</i>	Specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
MAXVALUE <i>n</i>	Specifies the maximum value the sequence can generate
NOMAXVALUE	Specifies a maximum value of $10^{27}$ for an ascending sequence and $-1$ for a descending sequence (This is the default option.)
MINVALUE <i>n</i>	Specifies the minimum sequence value
NOMINVALUE	Specifies a minimum value of 1 for an ascending sequence and $-(10^{26})$ for a descending sequence (This is the default option.)

## Creating a Sequence

- Create a sequence named `DEPT_DEPTID_SEQ` to be used for the primary key of the `DEPARTMENTS` table.
- Do not use the `CYCLE` option.

```
CREATE SEQUENCE dept_deptid_seq
      INCREMENT BY 10
      START WITH 120
      MAXVALUE 9999
      NOCACHE
      NOCYCLE;
CREATE SEQUENCE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Sequence (continued)

`CYCLE` | `NOCYCLE`

Specifies whether the sequence continues to generate values after reaching its maximum or minimum value (`NOCYCLE` is the default option.)

`CACHE n` | `NOCACHE`

Specifies how many values the Oracle server preallocates and keeps in memory (By default, the Oracle server caches 20 values.)

The example in the slide creates a sequence named `DEPT_DEPTID_SEQ` to be used for the `DEPARTMENT_ID` column of the `DEPARTMENTS` table. The sequence starts at 120, does not allow caching, and does not cycle.

Do not use the `CYCLE` option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

For more information, see “`CREATE SEQUENCE`” in the *Oracle SQL Reference*.

**Note:** The sequence is not tied to a table. Generally, you should name the sequence after its intended use. However, the sequence can be used anywhere, regardless of its name.

## NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### NEXTVAL and CURRVAL Pseudocolumns

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When you reference *sequence*.CURRVAL, the last value returned to that user's process is displayed.

**NEXTVAL and CURRVAL Pseudocolumns (continued)****Rules for Using NEXTVAL and CURRVAL**

You can use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

For more information, see “Pseudocolumns” and “CREATE SEQUENCE” in *Oracle SQL Reference*.

## Using a Sequence

- Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
                        department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);

1 row created.
```

- View the current value for the DEPT\_DEPTID\_SEQ sequence:

```
SELECT dept_deptid_seq.CURRVAL
FROM    dual;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using a Sequence

The example in the slide inserts a new department in the DEPARTMENTS table. It uses the DEPT\_DEPTID\_SEQ sequence to generate a new department number as follows.

You can view the current value of the sequence:

```
SELECT dept_deptid_seq.CURRVAL
FROM    dual;
```

	CURRVAL
1	120

Suppose that you now want to hire employees to staff the new department. The INSERT statement to be executed for all new employees can include the following code:

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq.CURRVAL, ...);
```

**Note:** The preceding example assumes that a sequence called EMPLOYEE\_SEQ has already been created to generate new employee numbers.



## Caching Sequence Values

- Caching sequence values in memory gives faster access to those values.
- Gaps in sequence values can occur when:
  - A rollback occurs
  - The system crashes
  - A sequence is used in another table

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Caching Sequence Values

You can cache sequences in memory to provide faster access to those sequence values. The cache is populated the first time you refer to the sequence. Each request for the next sequence value is retrieved from the cached sequence. After the last sequence value is used, the next request for the sequence pulls another cache of sequences into memory.

#### Gaps in the Sequence

Although sequence generators issue sequential numbers without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in memory, those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. If you do so, each table can contain gaps in the sequential numbers.

## Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 999999
      NOCACHE
      NOCYCLE;
ALTER SEQUENCE dept_deptid_seq succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Modifying a Sequence

If you reach the MAXVALUE limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the MAXVALUE. To continue to use the sequence, you can modify it by using the ALTER SEQUENCE statement.

#### Syntax

```
ALTER SEQUENCE sequence
      [INCREMENT BY n]
      [{MAXVALUE n | NOMAXVALUE}]
      [{MINVALUE n | NOMINVALUE}]
      [{CYCLE | NOCYCLE}]
      [{CACHE n | NOCACHE}];
```

In the syntax, *sequence* is the name of the sequence generator.

For more information, see “ALTER SEQUENCE” in the *Oracle SQL Reference*.

## Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the DROP statement:

```
DROP SEQUENCE dept_deptid_seq;
DROP SEQUENCE dept_deptid_seq succeeded.
```

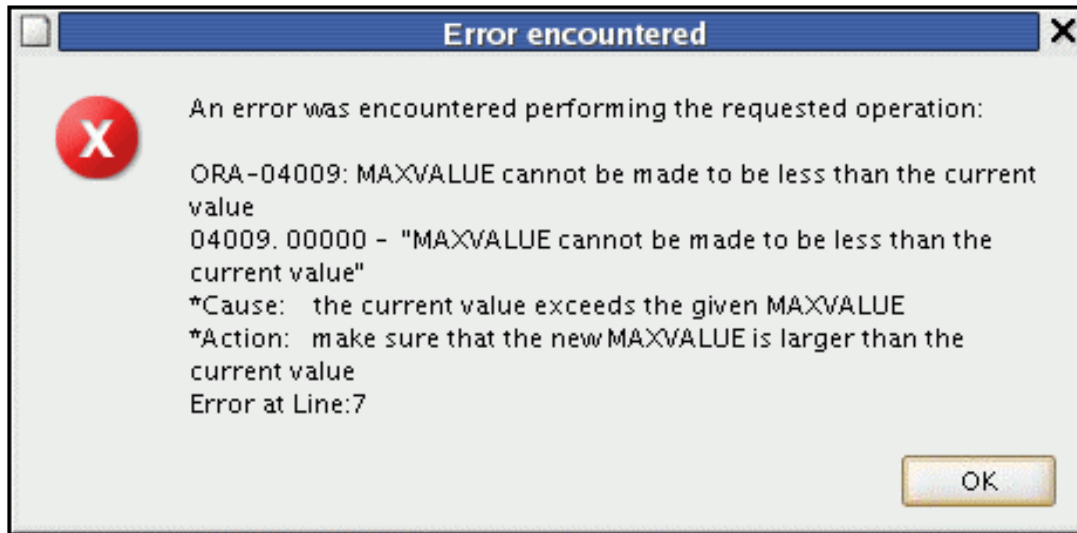
ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence to modify it. You must be the owner or have the DROP ANY SEQUENCE privilege to remove it.
- Only future sequence numbers are affected by the ALTER SEQUENCE statement.
- The START WITH option cannot be changed using ALTER SEQUENCE. The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed. For example, a new MAXVALUE that is less than the current sequence number cannot be imposed.

```
ALTER SEQUENCE dept_deptid_seq
    INCREMENT BY 20
    MAXVALUE 90
    NOCACHE
    NOCYCLE;
```

**Guidelines for Modifying a Sequence (continued)**

## Indexes

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

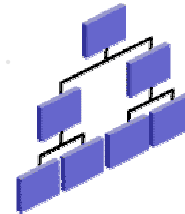
### Indexes

Indexes are database objects that you can create to improve the performance of some queries. Indexes can also be created automatically by the server when you create a primary key or unique constraint.

# Indexes

An index:

- Is a schema object
- Can be used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table that it indexes
- Is used and maintained automatically by the Oracle server



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Indexes (continued)

An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. After an index is created, no direct activity is required by the user.

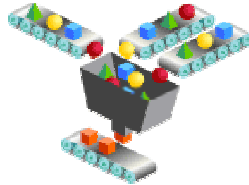
Indexes are logically and physically independent of the table that they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

**Note:** When you drop a table, corresponding indexes are also dropped.

For more information, see “Schema Objects: Indexes” in *Database Concepts*.

## How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.



- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Types of Indexes

Two types of indexes can be created.

**Unique index:** The Oracle server automatically creates this index when you define a column in a table to have a `PRIMARY KEY` or a `UNIQUE` key constraint. The name of the index is the name that is given to the constraint.

**Nonunique index:** This is an index that a user can create. For example, you can create a `FOREIGN KEY` column index for a join in a query to improve retrieval speed.

**Note:** You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

## Creating an Index

- Create an index on one or more columns:

```
CREATE INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST\_NAME column in the EMPLOYEES table:

```
CREATE INDEX emp_last_name_idx
ON      employees(last_name);
CREATE INDEX succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating an Index

Create an index on one or more columns by issuing the CREATE INDEX statement.

In the syntax:

<i>index</i>	Is the name of the index
<i>table</i>	Is the name of the table
<i>column</i>	Is the name of the column in the table to be indexed

For more information, see “CREATE INDEX” in *Oracle SQL Reference*.



## Index Creation Guidelines

Create an index when:	
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a WHERE clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
Do not create an index when:	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### More Is Not Always Better

Having more indexes on a table does not produce faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes that you have associated with a table, the more effort the Oracle server must make to update all the indexes after a DML operation.

### When to Create an Index

Therefore, you should create indexes only if:

- The column contains a wide range of values
- The column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or join condition
- The table is large and most queries are expected to retrieve less than 2% to 4% of the rows

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. A unique index is then created automatically.

## Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `UPPER_LAST_NAME_IDX` index from the data dictionary:

```
DROP INDEX emp_last_name_idx;  
DROP INDEX emp_last_name_idx succeeded.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Removing an Index

You cannot modify indexes. To change an index, you must drop it and then re-create it.

Remove an index definition from the data dictionary by issuing the `DROP INDEX` statement. To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

In the syntax, *index* is the name of the index.

**Note:** If you drop a table, indexes and constraints are automatically dropped but views and sequences remain.

## Synonyms

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Synonyms

Synonyms are database objects that enable you to call a table by another name. You can create synonyms to give an alternative name to a table.

## Synonyms

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

- Create an easier reference to a table that is owned by another user
- Shorten lengthy object names

```
CREATE [PUBLIC] SYNONYM synonym
FOR      object;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Synonym for an Object

To refer to a table that is owned by another user, you need to prefix the table name with the name of the user who created it, followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

<code>PUBLIC</code>	Creates a synonym that is accessible to all users
<code><i>synonym</i></code>	Is the name of the synonym to be created
<code><i>object</i></code>	Identifies the object for which the synonym is created

#### Guidelines

- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects that are owned by the same user.

For more information, see “CREATE SYNONYM” in the *Oracle SQL Reference*.

## Creating and Removing Synonyms

- Create a shortened name for the DEPT\_SUM\_VU view:

```
CREATE SYNONYM d_sum
FOR dept_sum_vu;
CREATE SYNONYM succeeded.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;
DROP SYNONYM succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Synonym

The slide example creates a synonym for the DEPT\_SUM\_VU view for quicker reference.

The database administrator can create a public synonym that is accessible to all users. The following example creates a public synonym named DEPT for Alice's DEPARTMENTS table:

```
CREATE PUBLIC SYNONYM dept
FOR alice.departments;
CREATE SYNONYM succeeded.
```

### Removing a Synonym

To remove a synonym, use the DROP SYNONYM statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM dept;
DROP SYNONYM succeeded.
```

For more information, see "DROP SYNONYM" in the *Oracle SQL Reference*.

## Summary

In this lesson, you should have learned how to:

- Create, use, and remove views
- Automatically generate sequence numbers by using a sequence generator
- Create indexes to improve query retrieval speed
- Use synonyms to provide alternative names for objects

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

In this lesson, you should have learned about database objects such as views, sequences, indexes, and synonyms.

## Practice 10: Overview of Part 2

This practice covers the following topics:

- Creating sequences
- Using sequences
- Creating nonunique indexes
- Creating synonyms

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Practice 10: Overview of Part 2

Part 2 of this lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym.

Complete questions 7–10 at the end of this lesson.

**Practice 10****Part 1**

1. The staff in the HR department want to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES\_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to be EMPLOYEE.
2. Confirm that the view works. Display the contents of the EMPLOYEES\_VU view.

	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110
6	100	King	90
7	101	Kochhar	90
8	102	De Haan	90
9	103	Hunold	30
10	104	Ernst	30
11	107	Lorentz	30
12	124	Mourgos	50
13	141	Rajs	50
14	142	Davies	50
15	143	Matos	50
16	144	Vargas	50
17	149	Zlotkey	80
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

3. Using your EMPLOYEES\_VU view, write a query for the HR department to display all employee names and department numbers.

	EMPLOYEE	DEPARTMENT_ID
1	Whalen	10
2	Hartstein	20

...

19	Taylor	80
20	Grant	(null)



**Practice 10 (continued)**

4. Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. You are asked to label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.
5. Display the structure and contents of the DEPT50 view.

Name	Null	Type
EMPNO	NOT NULL	NUMBER(6)
EMPLOYEE	NOT NULL	VARCHAR2(25)
DEPTNO		NUMBER(4)

	EMPNO	EMPLOYEE	DEPTNO
1	124	Mourgos	50
2	141	Rajs	50
3	142	Davies	50
4	143	Matos	50
5	144	Vargas	50

6. Test your view. Attempt to reassign Matos to department 80.

**Practice 10 (continued)****Part 2**

7. You need a sequence that can be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT\_ID\_SEQ.
8. To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab\_10\_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.
9. Create a nonunique index on the NAME column in the DEPT table.
10. Create a synonym for your EMPLOYEES table. Call it EMP.

# 11

## Managing Objects with Data Dictionary Views

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- Use the data dictionary views to research data on your objects
- Query various data dictionary views

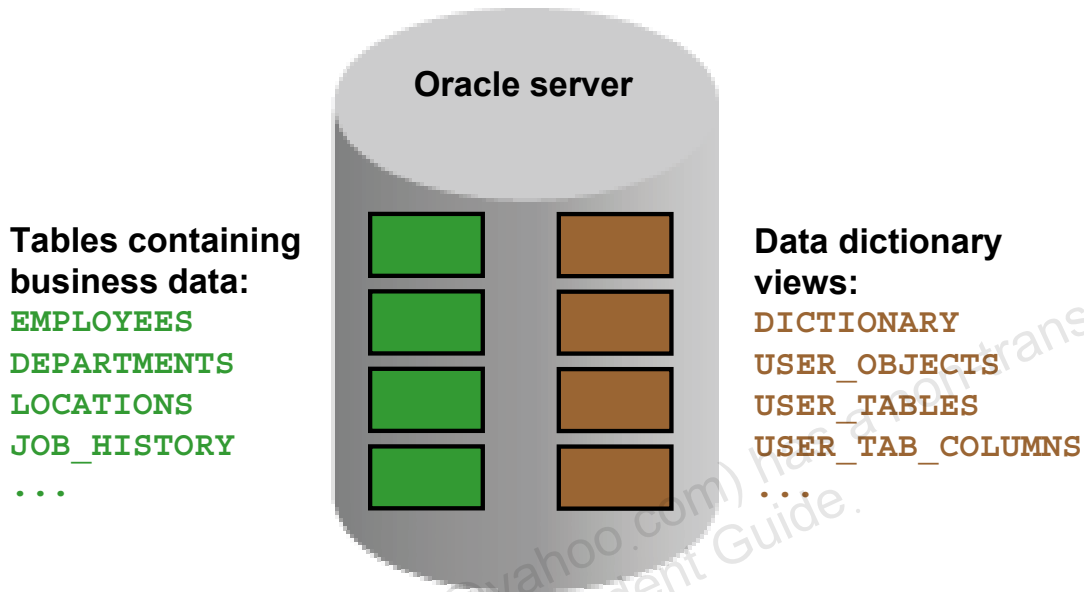
**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

In this lesson, you are introduced to the data dictionary views. You learn that the dictionary views can be used to retrieve metadata and create reports about your schema objects.

## Data Dictionary



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Data Dictionary

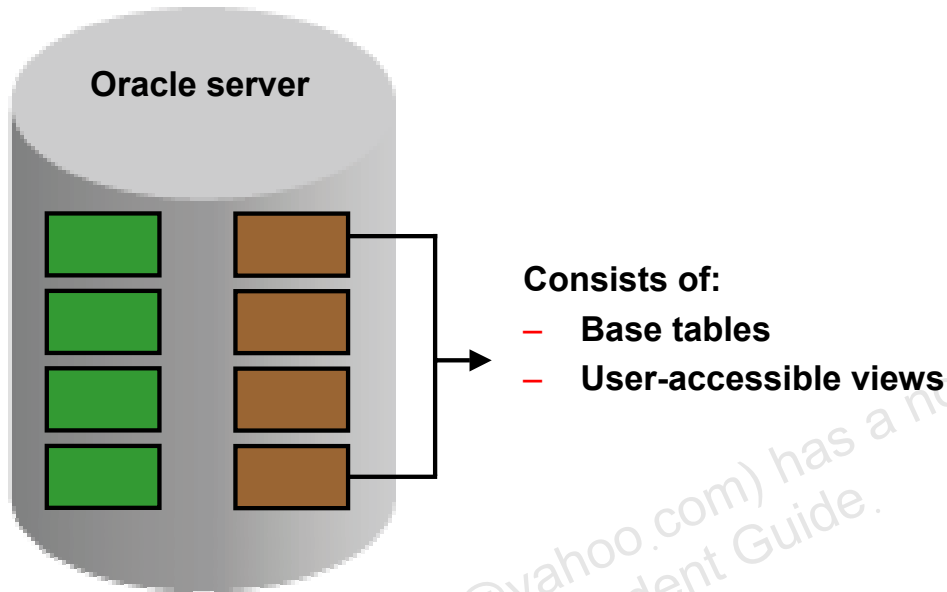
User tables are tables created by the user and contain business data, such as **EMPLOYEES**. There is another collection of tables and views in the Oracle Database known as the data dictionary. This collection is created and maintained by the Oracle server and contains information about the database. The *data dictionary* is structured in tables and views, just like other database data. Not only is the data dictionary central to every Oracle Database, but it is also an important tool for all users, from end users to application designers and database administrators.

You use SQL statements to access the data dictionary. Because the data dictionary is read-only, you can issue only queries against its tables and views.

You can query the dictionary views that are based on the dictionary tables to find information such as:

- Definitions of all schema objects in the database (tables, views, indexes, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- Default values for columns
- Integrity constraint information
- Names of Oracle users
- Privileges and roles that each user has been granted
- Other general database information

## Data Dictionary Structure



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Data Dictionary Structure

Underlying base tables store information about the associated database. Only the Oracle server should write to and read these tables. You rarely access them directly.

There are several views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information (such as user or table names) using joins and WHERE clauses to simplify the information. Most users are given access to the views rather than the base tables.

The Oracle user SYS owns all base tables and user-accessible views of the data dictionary. No Oracle user should *ever* alter (UPDATE, DELETE, or INSERT) any rows or schema objects contained in the SYS schema, because such activity can compromise data integrity.

## Data Dictionary Structure

View naming convention:

View Prefix	Purpose
USER	User's view (what is in your schema; what you own)
ALL	Expanded user's view (what you can access)
DBA	Database administrator's view (what is in everyone's schemas)
V\$	Performance-related data

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Data Dictionary Structure (continued)

The data dictionary consists of sets of views. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes. For example, there is a view named `USER_OBJECTS`, another named `ALL_OBJECTS`, and a third named `DBA_OBJECTS`.

These three views contain similar information about objects in the database, except that the scope is different. `USER_OBJECTS` contains information about objects that you own or created. `ALL_OBJECTS` contains information about all objects to which you have access. `DBA_OBJECTS` contains information on all objects that are owned by all users. For views that are prefixed with `ALL` or `DBA`, there is usually an additional column in the view named `OWNER` to identify who owns the object.

There is also a set of views that is prefixed with `v$`. These views are dynamic in nature and hold information about performance. Dynamic performance tables are not true tables, and they should not be accessed by most users. However, database administrators can query and create views on the tables and grant access to those views to other users. This course does not go into details about these views.

## How to Use the Dictionary Views

Start with `DICTIONARY`. It contains the names and descriptions of the dictionary tables and views.

```
DESCRIBE DICTIONARY
```

Name	Null	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

	TABLE_NAME	COMMENTS
1	USER_OBJECTS	Objects owned by the user

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### How to Use the Dictionary Views

To familiarize yourself with the dictionary views, you can use the dictionary view named `DICTIONARY`. It contains the name and short description of each dictionary view to which you have access.

You can write queries to search for information on a particular view name, or you can search the `COMMENTS` column for a word or phrase. In the example shown, the `DICTIONARY` view is described. It has two columns. The `SELECT` statement retrieves information about the dictionary view named `USER_OBJECTS`. The `USER_OBJECTS` view contains information about all the objects that you own.

You can write queries to search the `COMMENTS` column for a word or phrase. For example, the following query returns the names of all views that you are permitted to access in which the `COMMENTS` column contains the word *columns*:

```
SELECT table_name
FROM   dictionary
WHERE  LOWER(comments) LIKE '%columns';
```

**Note:** The names in the data dictionary are uppercase.



## USER\_OBJECTS and ALL\_OBJECTS Views

- Use the USER\_OBJECTS view to:
  - See all of the objects that are owned by you
  - Obtain a listing of all object names and types in your schema, plus the following information:
    - Date created
    - Date of last modification
    - Status (valid or invalid)
- Use the ALL\_OBJECTS view to see all objects to which you have access

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_OBJECTS View

You can query the USER\_OBJECTS view to see the names and types of all the objects in your schema. There are several columns in this view:

- **OBJECT\_NAME:** Name of the object
- **OBJECT\_ID:** Dictionary object number of the object
- **OBJECT\_TYPE:** Type of object (such as TABLE, VIEW, INDEX, SEQUENCE)
- **CREATED:** Timestamp for the creation of the object
- **LAST\_DDL\_TIME:** Timestamp for the last modification of the object resulting from a DDL command
- **STATUS:** Status of the object (VALID, INVALID, or N/A)
- **GENERATED:** Was the name of this object system-generated? (Y | N)

**Note:** This is not a complete listing of the columns. For a complete listing, see “USER\_OBJECTS” in the *Oracle Database Reference*.

You can also query the ALL\_OBJECTS view to see a listing of all objects to which you have access.

## USER\_OBJECTS View

```
SELECT object_name, object_type, created, status
FROM   user_objects
ORDER BY object_type;
```

	OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
1	REG_ID_PK	INDEX	29-OCT-08	VALID
2	DEPT_NAME_IDX	INDEX	11-NOV-08	VALID
3	DEPARTMENT_ID_PK	INDEX	11-NOV-08	VALID
4	LOC_COUNTRY_IX	INDEX	29-OCT-08	VALID
5	LOC_STATE_PROVINCE_IX	INDEX	29-OCT-08	VALID
6	LOC_CITY_IX	INDEX	29-OCT-08	VALID
7	JHIST_DEPARTMENT_IX	INDEX	29-OCT-08	VALID
8	JHIST_EMPLOYEE_IX	INDEX	29-OCT-08	VALID

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_OBJECTS View (continued)

The example shows the names, types, dates of creation, and status of all objects that are owned by this user.

The OBJECT\_TYPE column holds the values of either TABLE, VIEW, SEQUENCE, INDEX, PROCEDURE, FUNCTION, PACKAGE, or TRIGGER.

The STATUS column holds a value of VALID, INVALID, or N/A. While tables are always valid, the views, procedures, functions, packages, and triggers may be invalid.

### The CAT View

For a simplified query and output, you can query the CAT view. This view contains only two columns: TABLE\_NAME and TABLE\_TYPE. It provides the names of all your INDEX, TABLE, CLUSTER, VIEW, SYNONYM, SEQUENCE, or UNDEFINED objects.

## Table Information

### USER\_TABLES:

```
DESCRIBE user_tables
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

...

```
SELECT table_name
FROM user_tables;
```

TABLE_NAME
1 REGIONS
2 LOCATIONS
3 DEPARTMENTS
4 JOBS
5 EMPLOYEES

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_TABLES View

You can use the USER\_TABLES view to obtain the names of all of your tables. The USER\_TABLES view contains information about your tables. In addition to providing the table name, it contains detailed information on the storage.

The TABS view is a synonym of the USER\_TABLES view. You can query it to see a listing of tables that you own:

```
SELECT table_name
FROM tabs;
```

**Note:** For a complete listing of the columns in the USER\_TABLES view, see “USER\_TABLES” in the *Oracle Database Reference*.

You can also query the ALL\_TABLES view to see a listing of all tables to which you have access.

## Column Information

### USER\_TAB\_COLUMNS:

```
DESCRIBE user_tab_columns
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)
COLUMN_ID		NUMBER
DEFAULT_LENGTH		NUMBER
DATA_DEFAULT		LONG()

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Column Information

You can query the USER\_TAB\_COLUMNS view to find detailed information about the columns in your tables. While the USER\_TABLES view provides information on your table names and storage, detailed column information is found in the USER\_TAB\_COLUMNS view.

This view contains information such as:

- Column names
- Column data types
- Length of data types
- Precision and scale for NUMBER columns
- Whether nulls are allowed (Is there a NOT NULL constraint on the column?)
- Default value

**Note:** For a complete listing and description of the columns in the USER\_TAB\_COLUMNS view, see “USER\_TAB\_COLUMNS” in the *Oracle Database Reference*.

## Column Information

```
SELECT column_name, data_type, data_length,
       data_precision, data_scale, nullable
FROM   user_tab_columns
WHERE  table_name = 'EMPLOYEES';
```

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NULLABLE
1	EMPLOYEE_ID	NUMBER	22	6	0	N
2	FIRST_NAME	VARCHAR2	20	(null)	(null)	Y
3	LAST_NAME	VARCHAR2	25	(null)	(null)	N
4	EMAIL	VARCHAR2	25	(null)	(null)	N
5	PHONE_NUMBER	VARCHAR2	20	(null)	(null)	Y
6	HIRE_DATE	DATE	7	(null)	(null)	N
7	JOB_ID	VARCHAR2	10	(null)	(null)	N
8	SALARY	NUMBER	22	8	2	Y
9	COMMISSION_PCT	NUMBER	22	2	2	Y
10	MANAGER_ID	NUMBER	22	6	0	Y
11	DEPARTMENT_ID	NUMBER	22	4	0	Y

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Column Information (continued)

By querying the USER\_TAB\_COLUMNS table, you can find details about your columns such as the names, data types, data type lengths, null constraints, and default value for a column.

The example shown displays the columns, data types, data lengths, and null constraints for the EMPLOYEES table. Note that this information is similar to the output from the DESCRIBE command.

## Constraint Information

- `USER_CONSTRAINTS` describes the constraint definitions on your tables.
- `USER_CONS_COLUMNS` describes columns that are owned by you and that are specified in constraints.

```
DESCRIBE user_constraints
```

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG()
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)
DEFERRABLE		VARCHAR2(14)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Constraint Information

You can find out the names of your constraints, the type of constraint, the table name to which the constraint applies, the condition for check constraints, foreign key constraint information, deletion rule for foreign key constraints, the status, and many other types of information about your constraints.

**Note:** For a complete listing and description of the columns in the `USER_CONSTRAINTS` view, see “`USER_CONSTRAINTS`” in the *Oracle Database Reference*.

## Constraint Information

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1	EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL	(null)	(null)	ENABLED
2	EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL	(null)	(null)	ENABLED
3	EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL	(null)	(null)	ENABLED
4	EMP_JOB_NN	C	"JOB_ID" IS NOT NULL	(null)	(null)	ENABLED
5	EMP_SALARY_MIN	C	salary > 0	(null)	(null)	ENABLED
6	EMP_EMAIL_UK	U	(null)	(null)	(null)	ENABLED
7	EMP_EMP_ID_PK	P	(null)	(null)	(null)	ENABLED
8	EMP_DEPT_FK	R	(null)	DEPT_ID_PK	NO ACTION	ENABLED
9	EMP_JOB_FK	R	(null)	JOB_ID_PK	NO ACTION	ENABLED
10	EMP_MANAGER_FK	R	(null)	EMP_EMP_ID_PK	NO ACTION	ENABLED

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_CONSTRAINTS: Example

In the example shown, the USER\_CONSTRAINTS view is queried to find the names, types, check conditions, name of the unique constraint that the foreign key references, deletion rule for a foreign key, and status for constraints on the EMPLOYEES table.

The CONSTRAINT\_TYPE can be:

- C (check constraint on a table)
- P (primary key)
- U (unique key)
- R (referential integrity)
- V (with check option, on a view)
- O (with read-only, on a view)

The DELETE\_RULE can be:

- **CASCADE:** If the parent record is deleted, the child records are deleted too.
- **NO ACTION:** A parent record can be deleted only if no child records exist.

The STATUS can be:

- **ENABLED:** Constraint is active.
- **DISABLED:** Constraint is made not active.

## Constraint Information

```
DESCRIBE user_cons_columns
```

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	COLUMN_NAME
1	EMP_EMAIL_UK	EMAIL
2	EMP_SALARY_MIN	SALARY
3	EMP_JOB_NN	JOB_ID
4	EMP_HIRE_DATE_NN	HIRE_DATE
5	EMP_EMAIL_NN	EMAIL

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Querying USER\_CONS\_COLUMNS

To find the names of the columns to which a constraint applies, query the USER\_CONS\_COLUMNS dictionary view. This view tells you the name of the owner of a constraint, the name of the constraint, the table that the constraint is on, the names of the columns with the constraint, and the original position of column or attribute in the definition of the object.

**Note:** A constraint may apply to more than one column.

You can also write a join between the USER\_CONSTRAINTS and USER\_CONS\_COLUMNS to create customized output from both tables.



## View Information

1

```
DESCRIBE user_views
```

Name	Null	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

2

```
SELECT DISTINCT view_name FROM user_views;
```

VIEW_NAME
1 EMP_DETAILS_VIEW

3

```
SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';
```

TEXT
1 SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Views in the Data Dictionary

After your view is created, you can query the data dictionary view called `USER_VIEWS` to see the name of the view and the view definition. The text of the `SELECT` statement that constitutes your view is stored in a `LONG` column. The `LENGTH` column is the number of characters in the `SELECT` statement. By default, when you select from a `LONG` column, only the first 80 characters of the column's value are displayed. To see more than 80 characters in SQL\*Plus, use the command `SET LONG`:

```
SET LONG 1000
```

In the examples in the slide:

1. The `USER_VIEWS` columns are displayed. Note that this is a partial listing.
2. The names of your views are retrieved.
3. The `SELECT` statement for the `EMP_DETAILS_VIEW` is displayed from the dictionary.

### Data Access Using Views

When you access data using a view, the Oracle server performs the following operations:

- It retrieves the view definition from the data dictionary table `USER_VIEWS`.
- It checks access privileges for the view base table.
- It converts the view query into an equivalent operation on the underlying base table or tables. In other words, data is retrieved from, or an update is made to, the base tables.

## Sequence Information

```
DESCRIBE user_sequences
```

Name	Null	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### USER\_SEQUENCES View

The USER\_SEQUENCES view describes all sequences that are owned by you. When you create the sequence, you specify criteria that are stored in the USER\_SEQUENCES view. The columns in this view are:

- **SEQUENCE\_NAME:** Name of the sequence
- **MIN\_VALUE:** Minimum value of the sequence
- **MAX\_VALUE:** Maximum value of the sequence
- **INCREMENT\_BY:** Value by which sequence is incremented
- **CYCLE\_FLAG:** Does sequence wrap around on reaching limit?
- **ORDER\_FLAG:** Are sequence numbers generated in order?
- **CACHE\_SIZE:** Number of sequence numbers to cache
- **LAST\_NUMBER:** Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. This number is likely to be greater than the last sequence number that was used.

- Verify your sequence values in the `USER_SEQUENCES` data dictionary table.

[illegible]

- The `LAST_NUMBER` column displays the next available sequence number if `NOCACHE` is specified.

Copyright © 2009, Oracle. All rights reserved.

After creating your sequence, it is documented in the data dictionary. Because a sequence is a database object, you can identify it in the `USER OBJECTS` data dictionary table.

## Viewing the Next Available Sequence Value Without Incrementing It

If the sequence was created with NOCACHE, it is possible to view the next available sequence value without incrementing it by querying the USER\_SEQUENCES table.

## Synonym Information

```
DESCRIBE user_synonyms
```

Name	Null	Type
SYNONYM_NAME	NOT NULL	VARCHAR2(30)
TABLE_OWNER		VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
DB_LINK		VARCHAR2(128)

```
SELECT *
FROM user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1 EMP		ORA1	EMPLOYEES	(null)



Copyright © 2009, Oracle. All rights reserved.

### USER\_SYNONYMS View

The USER\_SYNONYMS dictionary view describes private synonyms (synonyms that are owned by you).

You can query this view to find your synonyms. You can query ALL\_SYNONYMS to find out the name of all of the synonyms that are available to you and the objects on which these synonyms apply.

The columns in this view are:

- **SYNONYM\_NAME:** Name of the synonym
- **TABLE\_OWNER:** Owner of the object that is referenced by the synonym
- **TABLE\_NAME:** Name of the table or view that is referenced by the synonym
- **DB\_LINK:** Name of the database link reference (if any)

## Adding Comments to a Table

- You can add comments to a table or column by using the COMMENT statement:

```
COMMENT ON TABLE employees
IS 'Employee Information';
COMMENT ON succeeded.
```

- Comments can be viewed through the data dictionary views:
  - ALL\_COL\_COMMENTS
  - USER\_COL\_COMMENTS
  - ALL\_TAB\_COMMENTS
  - USER\_TAB\_COMMENTS

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Adding Comments to a Table

You can add a comment of up to 4,000 bytes about a column, table, view, or snapshot by using the COMMENT statement. The comment is stored in the data dictionary and can be viewed in one of the following data dictionary views in the COMMENTS column:

- ALL\_COL\_COMMENTS
- USER\_COL\_COMMENTS
- ALL\_TAB\_COMMENTS
- USER\_TAB\_COMMENTS

#### Syntax

```
COMMENT ON TABLE table | COLUMN table.column
IS 'text';
```

In the syntax:

*table*        Is the name of the table  
*column*      Is the name of the column in a table  
*text*        Is the text of the comment

You can drop a comment from the database by setting it to empty string (' '):

```
COMMENT ON TABLE employees IS ' ';
```

## Summary

In this lesson, you should have learned how to find information about your objects by using the following dictionary views:

- DICTIONARY
- USER\_OBJECTS
- USER\_TABLES
- USER\_TAB\_COLUMNS
- USER\_CONSTRAINTS
- USER\_CONS\_COLUMNS
- USER\_VIEWS
- USER\_SEQUENCES
- USER\_TAB\_SYNONYMS

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

In this lesson, you learned about some of the dictionary views that are available to you. You can use these dictionary views to find information about your tables, constraints, views, sequences, and synonyms.

## Practice 11: Overview

This practice covers the following topics:

- Querying the dictionary views for table and column information
- Querying the dictionary views for constraint information
- Querying the dictionary views for view information
- Querying the dictionary views for sequence information
- Querying the dictionary views for synonym information
- Adding a comment to a table and querying the dictionary views for comment information

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Practice 11: Overview

In this practice, you query the dictionary views to find information about the objects in your schema.

## Practice 11

- For a specified table, create a script that reports the column names, data types, and lengths of the data types, as well as whether nulls are allowed. Prompt the user to enter the table name. Give appropriate aliases to the DATA\_PRECISION and DATA\_SCALE columns. Save this script in a file named lab\_11\_01.sql.

For example, if the user enters DEPARTMENTS, the following output results:

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	PRECISION	SCALE	NULLABLE
1	DEPARTMENT_ID	NUMBER	22	4	0	N
2	DEPARTMENT_NAME	VARCHAR2	30	(null)	(null)	N
3	MANAGER_ID	NUMBER	22	6	0	Y
4	LOCATION_ID	NUMBER	22	4	0	Y

- Create a script that reports the column name, constraint name, constraint type, search condition, and status for a specified table. You must join the USER\_CONSTRAINTS and USER\_CONS\_COLUMNS tables to obtain all this information. Prompt the user to enter the table name. Save the script in a file named lab\_11\_02.sql.

For example, if the user enters DEPARTMENTS, the following output results:

	COLUMN_NAME	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	STATUS
1	DEPARTMENT_NAME	DEPT_NAME_NN	C	"DEPARTMENT_NAME" IS NOT NULL	ENABLED
2	DEPARTMENT_ID	DEPT_ID_PK	P	(null)	ENABLED
3	LOCATION_ID	DEPT_LOC_FK	R	(null)	ENABLED
4	MANAGER_ID	DEPT_MGR_FK	R	(null)	ENABLED

- Add a comment to the DEPARTMENTS table. Then query the USER\_TAB\_COMMENTS view to verify that the comment is present.

	COMMENTS
1	Company department information including name, code, and location.

- Find the names of all synonyms that are in your schema.

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	EMP	ORA1	EMPLOYEES	(null)



## Practice 11 (continued)

5. You need to determine the names and definitions of all the views in your schema. Create a report that retrieves the view information: the view name and text from the `USER_VIEWS` data dictionary view.

**Note:** Another view already exists. The EMP\_DETAILS\_VIEW was created as part of your schema. Also, if you completed practice 10, you see the DEPT50 view.

**Note:** To see more contents of a LONG column, use the command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

	VIEW_NAME	TEXT
1	EMP_DETAILS_VIEW	SELECT e.employee_id, e.job_id, e.ma...
2	EMPLOYEES_VU	SELECT employee_id, last_name emplo...

6. Find the names of your sequences. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script `lab_11_06.sql`. Run the statement in your script.

[illegible]

VINOD BUSANAGA (b\_vinod87@yahoo.com) has a non-transferable  
license to use this Student Guide.

---

# A Practice Solutions

---

## Practice 1: Solutions

### Part 1

Test your knowledge:

1. The following SELECT statement executes successfully:

```
SELECT last_name, job_id, salary AS Sal
FROM employees;
```

**True/False**

2. The following SELECT statement executes successfully:

```
SELECT *
FROM job_grades;
```

**True/False**

3. There are four coding errors in this statement. Can you identify them?

```
SELECT      employee_id, last_name
sal x 12    ANNUAL SALARY
FROM        employees;
```

- **The EMPLOYEES table does not contain a column called sal. The column is called SALARY.**
- **The multiplication operator is \*, not x, as shown in line 2.**
- **The ANNUAL SALARY alias cannot include spaces. The alias should read ANNUAL\_SALARY or should be enclosed in double quotation marks.**
- **A comma is missing after the LAST\_NAME column.**

### Part 2

*Note the following location for the lab files:*

`\home\oracle\labs\SQL\labs`

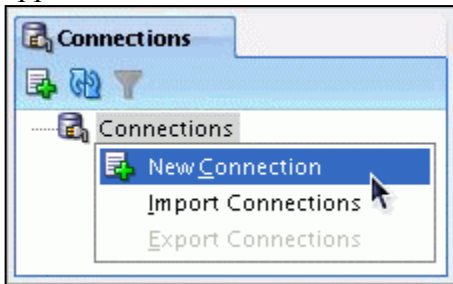
*If you are asked to save any lab files, save them at this location.*

To start Oracle SQL Developer, double-click the SQL Developer desktop icon.

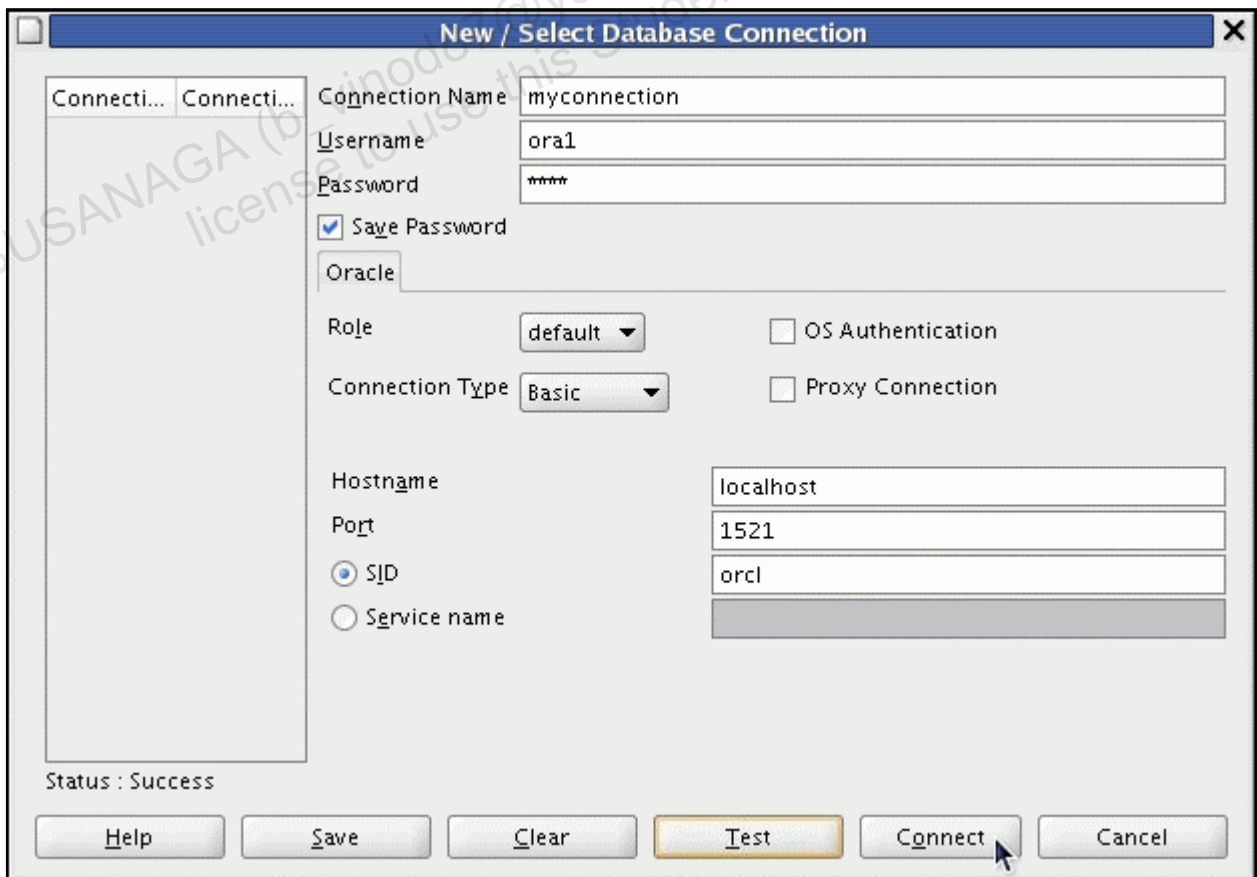
Before you begin with the practices, you need a database connection to be able to connect to the database and issue SQL queries.

**Practice 1: Solutions (continued)**

4. To create a new database connection in the Connections Navigator, right-click Connections. Select New Connection from the menu. The New/Select Database Connection dialog box appears.



5. Create a database connection using the following information:
- Connection Name: myconnection
  - Username: ora1
  - Password: ora1
  - Hostname: localhost
  - Port: 1521
  - SID: ORCL
  - Ensure that you select the Save Password check box.



**Practice 1: Solutions (continued)**

You have been hired as a SQL programmer for Acme Corporation. Your first task is to create some reports based on data from the Human Resources tables.

6. Your first task is to determine the structure of the DEPARTMENTS table and its contents.

```
DESCRIBE departments

SELECT *
FROM departments;
```

7. You need to determine the structure of the EMPLOYEES table.

```
DESCRIBE employees
```

The HR department wants a query to display the last name, job code, hire date, and employee number for each employee, with the employee number appearing first. Provide an alias STARTDATE for the HIRE\_DATE column. Save your SQL statement to a file named lab\_01\_07.sql so that you can dispatch this file to the HR department.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM employees;
```

8. Test your query in the lab\_01\_07.sql file to ensure that it runs correctly.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM employees;
```

9. The HR department needs a query to display all unique job codes from the EMPLOYEES table.

```
SELECT DISTINCT job_id
FROM employees;
```

**Part 3**

If you have time, complete the following exercises:

10. The HR department wants more descriptive column headings for its report on employees. Copy the statement from lab\_01\_07.sql to the SQL Developer text box. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Then run your query again.

```
SELECT employee_id "Emp #", last_name "Employee",
       job_id "Job", hire_date "Hire Date"
FROM employees;
```

**Practice 1: Solutions (continued)**

11. The HR department has requested a report of all employees and their job IDs. Display the last name concatenated with the job ID (separated by a comma and space) and name the column Employee and Title.

```
SELECT last_name || ', ' || job_id "Employee and Title"
FROM   employees;
```

If you want an extra challenge, complete the following exercise:

12. To familiarize yourself with the data in the EMPLOYEES table, create a query to display all the data from the EMPLOYEES table. Separate each column output with a comma. Name the column THE\_OUTPUT.

```
SELECT employee_id || ', ' || first_name || ', ' || last_name
       || ', ' || email || ', ' || phone_number || ', ' || job_id
       || ', ' || manager_id || ', ' || hire_date || ', ' || salary
       || ', ' || commission_pct || ', ' || department_id
       THE_OUTPUT
FROM   employees;
```

## Practice 2: Solutions

The HR department needs your assistance with creating some queries.

1. Because of budget issues, the HR department needs a report that displays the last name and salary of employees earning more than \$12,000. Place your SQL statement in a text file named `lab_02_01.sql`. Run your query.

```
SELECT last_name, salary
FROM employees
WHERE salary > 12000;
```

2. Create a report that displays the last name and department number for employee number 176.

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 176;
```

3. The HR departments needs to find high-salary and low-salary employees. Modify `lab_02_01.sql` to display the last name and salary for all employees whose salary is not in the \$5,000–\$12,000 range. Place your SQL statement in a text file named `lab_02_03.sql`.

```
SELECT last_name, salary
FROM employees
WHERE salary NOT BETWEEN 5000 AND 12000;
```

4. Create a report to display the last name, job ID, and start date for the employees whose last names are Matos and Taylor. Order the query in ascending order by start date.

```
SELECT last_name, job_id, hire_date
FROM employees
WHERE last_name IN ('Matos', 'Taylor')
ORDER BY hire_date;
```

5. Display the last name and department number of all employees in departments 20 or 50 in ascending alphabetical order by name.

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN (20, 50)
ORDER BY last_name ASC;
```



**Practice 2: Solutions (continued)**

6. Modify lab\_02\_03.sql to list the last name and salary of employees who earn between \$5,000 and \$12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Resave lab\_02\_03.sql as lab\_02\_06.sql. Run the statement in lab\_02\_06.sql.

```
SELECT    last_name "Employee", salary "Monthly Salary"
FROM      employees
WHERE     salary BETWEEN 5000 AND 12000
AND       department_id IN (20, 50);
```

7. The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.

```
SELECT    last_name, hire_date
FROM      employees
WHERE     hire_date LIKE '%94';
```

8. Create a report to display the last name and job title of all employees who do not have a manager.

```
SELECT    last_name, job_id
FROM      employees
WHERE     manager_id IS NULL;
```

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

```
SELECT    last_name, salary, commission_pct
FROM      employees
WHERE     commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

10. Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. (You can use the query created in practice exercise 1 and modify it.) Save this query to a file named lab\_02\_10.sql.

```
SELECT    last_name, salary
FROM      employees
WHERE     salary > &sal_amt;
```

**Practice 2: Solutions (continued)**

11. The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary, and department for that manager's employees. The HR department wants the ability to sort the report on a selected column. You can test the data with the following values:

manager ID = 103, sorted by employee last name

manager ID = 201, sorted by salary

manager ID = 124, sorted by employee ID

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE manager_id = &mgr_num
ORDER BY &order_col;
```

If you have time, complete the following exercises:

12. Display all employee last names in which the third letter of the name is *a*.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '__a%';
```

13. Display the last names of all employees who have both an *a* and an *e* in their last names.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%a%'
AND last_name LIKE '%e%';
```

If you want an extra challenge, complete the following exercises:

14. Display the last name, job, and salary for all employees whose job is either that of a sales representative or a stock clerk, and whose salary is not equal to \$2,500, \$3,500, or \$7,000.

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id IN ('SA_REP', 'ST_CLERK')
AND salary NOT IN (2500, 3500, 7000);
```

15. Modify lab\_02\_06.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab\_02\_06.sql as lab\_02\_15.sql. Rerun the statement in lab\_02\_15.sql.

```
SELECT last_name "Employee", salary "Monthly Salary",
commission_pct
FROM employees
WHERE commission_pct = .20;
```

**Practice 3: Solutions**

1. Write a query to display the current date. Label the column Date.

```
SELECT sysdate "Date"
FROM dual;
```

2. The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary. Place your SQL statement in a text file named lab\_03\_02.sql.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary"
FROM employees;
```

3. Run your query in the lab\_03\_02.sql file.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary"
FROM employees;
```

4. Modify your lab\_03\_02.sql query to add a column that subtracts the old salary from the new salary. Label the column Increase. Save the contents of the file as lab\_03\_04.sql. Run the revised query.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary",
       ROUND(salary * 1.155, 0) - salary "Increase"
FROM employees;
```

5. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees' last names.

```
SELECT INITCAP(last_name) "Name",
       LENGTH(last_name) "Length"
FROM employees
WHERE last_name LIKE 'J%'
OR last_name LIKE 'M%'
OR last_name LIKE 'A%'
ORDER BY last_name ;
```

**Practice 3: Solutions (continued)**

Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, the output should show all employees whose last name starts with the letter *H*.

```
SELECT  INITCAP(last_name) "Name",
        LENGTH(last_name) "Length"
FROM    employees
WHERE   last_name LIKE '&start_letter%'
ORDER BY last_name;
```

6. The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS\_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

**Note:** Your results will differ.

```
SELECT last_name, ROUND(MONTHS_BETWEEN(
        SYSDATE, hire_date)) MONTHS_WORKED
FROM    employees
ORDER BY months_worked;
```

7. Create a report that produces the following for each employee:  
 <employee last name> earns <salary> monthly but wants <3 times salary>.

Label the column Dream Salaries.

```
SELECT  last_name || ' earns '
        || TO_CHAR(salary, 'fm$99,999.00')
        || ' monthly but wants '
        || TO_CHAR(salary * 3, 'fm$99,999.00')
        || '. ' "Dream Salaries"
FROM    employees;
```

If you have time, complete the following exercises:

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the “\$” symbol. Label the column SALARY.

```
SELECT last_name,
        LPAD(salary, 15, '$') SALARY
FROM    employees;
```

**Practice 3: Solutions (continued)**

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

```
SELECT last_name, hire_date,
       TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY'),
               'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
FROM   employees;
```

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') DAY
FROM   employees
ORDER BY TO_CHAR(hire_date - 1, 'd');
```

If you want an extra challenge, complete the following exercises:

11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, show "No Commission." Label the column COMM.

```
SELECT last_name,
       NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM   employees;
```

12. Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES\_AND\_THEIR\_SALARIES.

```
SELECT rpad(last_name, 8) || ' ' ||
       rpad(' ', salary/1000+1, '*')
       EMPLOYEES_AND_THEIR_SALARIES
FROM   employees
ORDER BY salary DESC;
```

**Practice 3: Solutions (continued)**

13. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB\_ID, using the following data:

<i>Job</i>	<i>Grade</i>
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

```
SELECT job_id, decode (job_id,
                        'ST_CLERK', 'E',
                        'SA_REP',   'D',
                        'IT_PROG',  'C',
                        'ST_MAN',   'B',
                        'AD_PRES',  'A',
                        '0') GRADE
FROM employees;
```

14. Rewrite the statement in the preceding exercise using the CASE syntax.

```
SELECT job_id, CASE job_id
                WHEN 'ST_CLERK' THEN 'E'
                WHEN 'SA_REP'   THEN 'D'
                WHEN 'IT_PROG'  THEN 'C'
                WHEN 'ST_MAN'   THEN 'B'
                WHEN 'AD_PRES'  THEN 'A'
                ELSE '0' END GRADE
FROM employees;
```

**Practice 4: Solutions**

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

**True/False**

2. Group functions include nulls in calculations.

**True/False**

3. The WHERE clause restricts rows before inclusion in a group calculation.

**True/False**

The HR department needs the following reports:

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named lab\_04\_04.sql.

```
SELECT ROUND(MAX(salary),0) "Maximum",
       ROUND(MIN(salary),0) "Minimum",
       ROUND(SUM(salary),0) "Sum",
       ROUND(AVG(salary),0) "Average"
FROM   employees;
```

5. Modify the query in lab\_04\_04.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab\_04\_04.sql as lab\_04\_05.sql. Run the statement in lab\_04\_05.sql.

```
SELECT job_id, ROUND(MAX(salary),0) "Maximum",
       ROUND(MIN(salary),0) "Minimum",
       ROUND(SUM(salary),0) "Sum",
       ROUND(AVG(salary),0) "Average"
FROM   employees
GROUP BY job_id;
```

6. Write a query to display the number of people with the same job.

```
SELECT job_id, COUNT(*)
FROM   employees
GROUP BY job_id;
```

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named lab\_04\_06.sql.

```
SELECT job_id, COUNT(*)
FROM   employees
WHERE  job_id = '&job_title'
GROUP BY job_id;
```

**Practice 4: Solutions (continued)**

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER\_ID column to determine the number of managers.*

```
SELECT COUNT(DISTINCT manager_id) "Number of Managers"
FROM employees;
```

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SELECT MAX(salary) - MIN(salary) DIFFERENCE
FROM employees;
```

If you have time, complete the following exercises:

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is \$6,000 or less. Sort the output in descending order of salary.

```
SELECT manager_id, MIN(salary)
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

If you want an extra challenge, complete the following exercises:

10. Create a query that displays the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT COUNT(*) total,
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1995, 1, 0)) "1995",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1996, 1, 0)) "1996",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1997, 1, 0)) "1997",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1998, 1, 0)) "1998"
FROM employees;
```



**Practice 4: Solutions (continued)**

11. Create a matrix query to display the job, the salary for that job based on the department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT    job_id "Job",
          SUM(DECODE(department_id , 20, salary)) "Dept 20",
          SUM(DECODE(department_id , 50, salary)) "Dept 50",
          SUM(DECODE(department_id , 80, salary)) "Dept 80",
          SUM(DECODE(department_id , 90, salary)) "Dept 90",
          SUM(salary) "Total"
FROM      employees
GROUP BY  job_id;
```

**Practice 5: Solutions**

1. Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output. Use a NATURAL JOIN to produce the results.

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

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

```
SELECT last_name, department_id, department_name
FROM   employees
JOIN   departments
USING (department_id);
```

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id, d.department_name
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   locations l
ON     (d.location_id = l.location_id)
WHERE  LOWER(l.city) = 'toronto';
```

4. Create a report to display the last name and employee number of employees along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab\_05\_04.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w join employees m
ON     (w.manager_id = m.employee_id);
```

5. Modify lab\_05\_04.sql to display all employees, including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab\_05\_05.sql. Run the query in lab\_05\_05.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w
LEFT   OUTER JOIN employees m
ON     (w.manager_id = m.employee_id);
```

**Practice 5: Solutions (continued)**

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_05_06.sql`.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the `JOB_GRADES` table, first show the structure of the `JOB_GRADES` table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON     (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

9. The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_05_09.sql`.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON     (w.manager_id = m.employee_id)
WHERE  w.hire_date < m.hire_date;
```

**Practice 6: Solutions**

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name the user supplies (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```

UNDEFINE Enter_name

SELECT last_name, hire_date
FROM   employees
WHERE  department_id = (SELECT department_id
                        FROM   employees
                        WHERE  last_name = '&Enter_name')
AND    last_name <> '&Enter_name';

```

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in ascending order by salary.

```

SELECT employee_id, last_name, salary
FROM   employees
WHERE  salary > (SELECT AVG(salary)
                 FROM   employees)
ORDER BY salary;

```

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named lab\_06\_03.sql. Run your query.

```

SELECT employee_id, last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM   employees
                        WHERE  last_name like '%u%');

```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```

SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM   departments
                        WHERE  location_id = 1700);

```

Modify the query so that the user is prompted for a location ID. Save this to a file named lab\_06\_04.sql.

```

SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM   departments
                        WHERE  location_id = &Enter_location);

```

**Practice 6: Solutions (continued)**

5. Create a report for the HR department that displays the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM   employees
WHERE  manager_id = (SELECT employee_id
                     FROM   employees
                     WHERE  last_name = 'King');
```

6. Create a report for the HR department that displays the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM   departments
                        WHERE  department_name = 'Executive');
```

If you have time, complete the following exercise:

7. Modify the query in lab\_06\_03.sql to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*. Resave lab\_06\_03.sql to lab\_06\_07.sql. Run the statement in lab\_06\_07.sql.

```
SELECT employee_id, last_name, salary
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM   employees
                        WHERE  last_name like '%u%')
AND    salary > (SELECT AVG(salary)
                 FROM   employees);
```

**Practice 7: Solutions**

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use set operators to create this report.

```
SELECT department_id
FROM departments
MINUS
SELECT department_id
FROM employees
WHERE job_id = 'ST_CLERK';
```

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.

```
SELECT country_id, country_name
FROM countries
MINUS
SELECT country_id, country_name
FROM countries
NATURAL JOIN locations
NATURAL JOIN departments;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display the job ID and department ID using set operators.

```
COLUMN dummy NOPRINT
SELECT job_id, department_id, 'x' dummy
FROM employees
WHERE department_id = 10
UNION
SELECT job_id, department_id, 'y' dummy
FROM employees
WHERE department_id = 50
UNION
SELECT job_id, department_id, 'z' dummy
FROM employees
WHERE department_id = 20
ORDER BY dummy;
COLUMN dummy PRINT
```

4. Create a report that lists the employee ID and job ID of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

**Practice 7: Solutions (continued)**

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department
- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

Write a compound query to accomplish this.

```
SELECT last_name,department_id,TO_CHAR(null)
FROM   employees
UNION
SELECT TO_CHAR(null),department_id,department_name
FROM   departments;
```

## Practice 8: Solutions

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the MY\_EMPLOYEE table, before giving the statements to the HR department.

Insert data into the MY\_EMPLOYEE table.

1. Run the statement in the lab\_08\_01.sql script to build the MY\_EMPLOYEE table to be used for the lab.

```
CREATE TABLE my_employee
(id NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
last_name VARCHAR2(25),
first_name VARCHAR2(25),
userid VARCHAR2(8),
salary NUMBER(9,2));
```

2. Describe the structure of the MY\_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
```

3. Create an INSERT statement to add the first row of data to the MY\_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```



**Practice 8: Solutions (continued)**

4. Populate the MY\_EMPLOYEE table with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
INSERT INTO my_employee (id, last_name, first_name,
                        userid, salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. Confirm your addition to the table.

```
SELECT *
FROM my_employee;
```

6. Write an INSERT statement in a dynamic reusable script file named loademp.sql to load rows into the MY\_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID. Save this script to a file named lab\_08\_06.sql.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
      lower(substr('&p_first_name', 1, 1) ||
      substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

7. Populate the table with the next two rows of sample data listed in step 3 by running the INSERT statement in the script that you created.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
      lower(substr('&p_first_name', 1, 1) ||
      substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

8. Confirm your additions to the table.

```
SELECT *
FROM my_employee;
```

9. Make the data additions permanent.

```
COMMIT;
```

**Practice 8: Solutions (continued)**

Update and delete data in the MY\_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.

```
UPDATE my_employee
SET    last_name = 'Drexler'
WHERE  id = 3;
```

11. Change the salary to \$1,000 for all employees with a salary less than \$900.

```
UPDATE my_employee
SET    salary = 1000
WHERE  salary < 900;
```

12. Verify your changes to the table.

```
SELECT last_name, salary
FROM   my_employee;
```

13. Delete Betty Dancs from the MY\_EMPLOYEE table.

```
DELETE
FROM my_employee
WHERE last_name = 'Dancs';
```

14. Confirm your changes to the table.

```
SELECT *
FROM   my_employee;
```

15. Commit all pending changes.

```
COMMIT;
```

Control data transaction to the MY\_EMPLOYEE table.

16. Populate the table with the last row of sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
       lower(substr('&p_first_name', 1, 1) ||
       substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

**Practice 8: Solutions (continued)**

17. Confirm your addition to the table.

```
SELECT  *  
FROM    my_employee;
```

18. Mark an intermediate point in the processing of the transaction.

```
SAVEPOINT step_18;
```

19. Empty the entire table.

```
DELETE  
FROM    my_employee;
```

20. Confirm that the table is empty.

```
SELECT  *  
FROM    my_employee;
```

21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.

```
ROLLBACK TO step_18;
```

22. Confirm that the new row is still intact.

```
SELECT  *  
FROM    my_employee;
```

23. Make the data addition permanent.

```
COMMIT;
```

**Practice 9: Solutions**

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab\_09\_01.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

```
CREATE TABLE dept
(id    NUMBER(7) CONSTRAINT department_id_pk PRIMARY KEY,
name  VARCHAR2(25));

DESCRIBE dept
```

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

```
INSERT INTO dept
SELECT department_id, department_name
FROM departments;
```

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab\_09\_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

```
CREATE TABLE emp
(id          NUMBER(7),
last_name    VARCHAR2(25),
first_name   VARCHAR2(25),
dept_id      NUMBER(7)
CONSTRAINT emp_dept_id_FK REFERENCES dept (id)
);

DESCRIBE emp
```

4. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPARTMENT\_ID columns. Name the columns in your new table ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPT\_ID, respectively.

```
CREATE TABLE employees2 AS
SELECT employee_id id, first_name, last_name, salary,
       department_id dept_id
FROM employees;
```

5. Drop the EMP table.

```
DROP TABLE emp;
```

## Practice 10: Solutions

### Part 1

1. The staff in the HR department wants to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES\_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to be EMPLOYEE.

```
CREATE OR REPLACE VIEW employees_vu AS
  SELECT employee_id, last_name employee, department_id
  FROM employees;
```

2. Confirm that the view works. Display the contents of the EMPLOYEES\_VU view.

```
SELECT    *
FROM      employees_vu;
```

3. Using your EMPLOYEES\_VU view, write a query for the HR department to display all employee names and department numbers.

```
SELECT    employee, department_id
FROM      employees_vu;
```

4. Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. They have requested that you label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW dept50 AS
  SELECT    employee_id empno, last_name employee,
            department_id deptno
  FROM      employees
  WHERE     department_id = 50
  WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

5. Display the structure and contents of the DEPT50 view.

```
DESCRIBE dept50

SELECT    *
FROM      dept50;
```

6. Test your view. Attempt to reassign Matos to department 80.

```
UPDATE    dept50
SET        deptno = 80
WHERE     employee = 'Matos';
```

The error is due to the fact that the view “DEPT50” is created with CHECK OPTION CONSTRAINT. This ensures that the deptno column in the view is protected from being changed.

## Practice 10: Solutions (continued)

You cannot make modifications to the `deptno` column that will result in the row being removed from the view.

### Part 2

7. You need a sequence that can be used with the primary key column of the `DEPT` table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by 10. Name the sequence `DEPT_ID_SEQ`.

```
CREATE SEQUENCE dept_id_seq
  START WITH 200
  INCREMENT BY 10
  MAXVALUE 1000;
```

8. To test your sequence, write a script to insert two rows in the `DEPT` table. Name your script `lab_10_08.sql`. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

9. Create a nonunique index on the `NAME` column in the `DEPT` table.

```
CREATE INDEX dept_name_idx ON dept (name);
```

10. Create a synonym for your `EMPLOYEES` table. Call it `EMP`.

```
CREATE SYNONYM emp FOR EMPLOYEES;
```

**Practice 11: Solutions**

1. For a specified table, create a script that reports the column names, data types, and lengths of data types, as well as whether nulls are allowed. Prompt the user to enter the table name. Give appropriate aliases to the columns DATA\_PRECISION and DATA\_SCALE. Save this script in a file named lab\_11\_01.sql.

```
SELECT column_name, data_type, data_length,
       data_precision PRECISION, data_scale SCALE, nullable
FROM   user_tab_columns
WHERE  table_name = UPPER('&tab_name');
```

2. Create a script that reports the column name, constraint name, constraint type, search condition, and status for a specified table. You must join the USER\_CONSTRAINTS and USER\_CONS\_COLUMNS tables to obtain all of this information. Prompt the user to enter the table name. Save the script in a file named lab\_11\_02.sql.

```
SELECT ucc.column_name, uc.constraint_name, uc.constraint_type,
       uc.search_condition, uc.status
FROM   user_constraints uc JOIN user_cons_columns ucc
ON     uc.table_name = ucc.table_name
AND    uc.constraint_name = ucc.constraint_name
AND    uc.table_name = UPPER('&tab_name');
```

3. Add a comment to the DEPARTMENTS table. Then query the USER\_TAB\_COMMENTS view to verify that the comment is present.

```
COMMENT ON TABLE departments IS
  'Company department information including name, code, and location.';

SELECT COMMENTS
FROM   user_tab_comments
WHERE  table_name = 'DEPARTMENTS';
```

4. Find the names of all synonyms that are in your schema.

```
SELECT *
FROM   user_synonyms;
```

**Practice 11: Solutions (continued)**

5. You need to determine the names and definitions of all the views in your schema. Create a report that retrieves view information (the view name and text) from the USER\_VIEWS data dictionary view.

**Note:** Another view already exists. The EMP\_DETAILS\_VIEW was created as part of your schema. Also, if you completed practice 10, you see the DEPT50 view.

**Note:** To see more contents of a LONG column, use the command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

```
SET LONG 600
```

```
SELECT    view_name, text
FROM      user_views;
```

6. Find the names of your sequences. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab\_11\_06.sql. Run the statement in your script.

```
SELECT    sequence_name, max_value, increment_by, last_number
FROM      user_sequences;
```



**Practice C: Solutions**

1. Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output.

```
SELECT location_id, street_address, city, state_province, country_name
FROM   locations, countries
WHERE  locations.country_id = countries.country_id;
```

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id, d.department_name
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id
AND    LOWER(l.city) = 'toronto';
```

4. Create a report to display the employee last name and employee number along with the last name of the employee's manager and the manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab\_c\_04.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w, employees m
WHERE  w.manager_id = m.employee_id;
```

5. Modify lab\_c\_04.sql to display all employees, including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab\_c\_05.sql. Run the query in lab\_c\_05.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w, employees m
WHERE  w.manager_id = m.employee_id (+);
```

**Practice C: Solutions (continued)**

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named lab\_c\_06.sql.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e, employees c
WHERE  e.department_id = c.department_id
AND    e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB\_GRADES table, first show the structure of the JOB\_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e, departments d, job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all employees hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e , employees davies
WHERE  davies.last_name = 'Davies'
AND    davies.hire_date < e.hire_date;
```

9. The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively. Save the script to a file named lab\_c\_09.sql.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w , employees m
WHERE  w.manager_id = m.employee_id
AND    w.hire_date < m.hire_date;
```

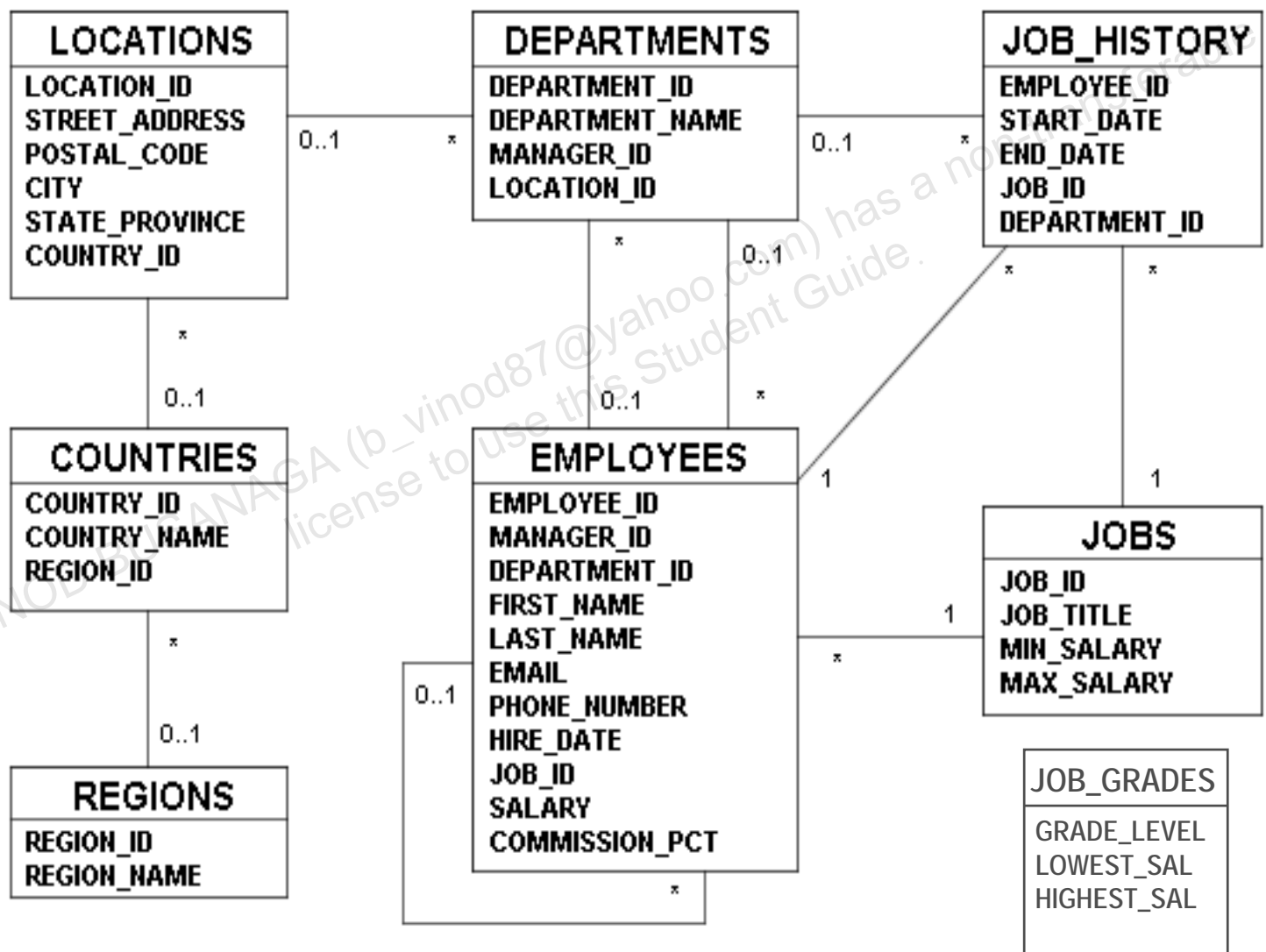
---

# B

---

## Table Descriptions and Data

# Human Resources (HR) Data Set



## Human Resources (HR) Data Set

The Human Resources (HR) schema is part of the Oracle Common Schema that can be installed in an Oracle Database. The practices in this course use the data from the HR schema.

### Table Descriptions

REGIONS contains rows representing a region (such as Americas, Asia, and so on).

COUNTRIES contains rows for countries, each of which are associated with a region.

LOCATIONS contains the addresses of specific offices, warehouses, and/or production sites of a company in a particular country.

DEPARTMENTS shows details of the departments in which employees work. Each department can have a relationship representing the department manager in the EMPLOYEES table.

EMPLOYEES contains details about each employee who works for a department. Some employees may not be assigned to any department.

JOBS contains the job types that can be held by each employee.

JOB\_HISTORY contains the job history of the employees. If an employee changes departments within the job or changes jobs within the department, a new row is inserted in this table with the old job information of the employee.

JOB\_GRADES identifies a salary range per job grade. The salary ranges do not overlap.

**COUNTRIES Table**

```
DESCRIBE countries
```

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

```
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

**DEPARTMENTS Table**

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)
4 rows selected		

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

**EMPLOYEES Table**

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)
11 rows selected		

SELECT \* FROM employees;

R	EMP...	R	FIRST...	R	LAST...	R	EMAIL	R	PHONE_NUMBER	R	HIRE_D...	R	JOB_ID	R	SAL...	R	COM...	R	MAN...	R	DEP...
1	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10										
2	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20										
3	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000	(null)	201	20										
4	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110										
5	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110										
6	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90										
7	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	(null)	100	90										
8	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	(null)	100	90										
9	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	(null)	102	60										
10	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	(null)	103	60										
11	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	(null)	103	60										
12	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800	(null)	100	50										
13	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500	(null)	124	50										
14	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100	(null)	124	50										
15	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600	(null)	124	50										
16	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500	(null)	124	50										
17	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	0.2	100	80										
18	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	0.3	149	80										
19	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	0.2	149	80										
20	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	0.15	149	(null)										



**JOBS Table**

```
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)
4 rows selected		

```
SELECT * FROM jobs;
```

	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20000	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	20000
7	SA_REP	Sales Representative	6000	12000
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2000	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

**JOB\_GRADES Table**

```
DESCRIBE job_grades
```

Name	Null	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER
3 rows selected		

```
SELECT * FROM job_grades;
```

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1 A		1000	2999
2 B		3000	5999
3 C		6000	9999
4 D		10000	14999
5 E		15000	24999
6 F		25000	40000

**JOB\_HISTORY Table**

```
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)
5 rows selected		

```
SELECT * FROM job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-93	24-JUL-98	IT_PROG	60
2	101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
3	101	28-OCT-93	15-MAR-97	AC_MGR	110
4	201	17-FEB-96	19-DEC-99	MK_REP	20
5	114	24-MAR-98	31-DEC-99	ST_CLERK	50
6	122	01-JAN-99	31-DEC-99	ST_CLERK	50
7	200	17-SEP-87	17-JUN-93	AD_ASST	90
8	176	24-MAR-98	31-DEC-98	SA_REP	80
9	176	01-JAN-99	31-DEC-99	SA_MAN	80
10	200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

**LOCATIONS Table**

```
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)
6 rows selected		

```
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

**REGIONS Table**

```
DESCRIBE regions
```

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

```
SELECT * FROM regions;
```

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

VINOD BUSANAGA (b\_vinod87@yahoo.com) has a non-transferable  
license to use this Student Guide.

# Oracle Join Syntax

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- Write **SELECT** statements to access data from more than one table using equijoins and nonequijoins
- Use outer joins to view data that generally does not meet a join condition
- Join a table to itself by using a self-join

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

**Note:** Information on joins is found in “SQL Queries and Subqueries: Joins” in *Oracle SQL Reference*.



## Obtaining Data from Multiple Tables

**EMPLOYEES**

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

**DEPARTMENTS**

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700



	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	100	90	Executive
2	101	90	Executive
...			
17	202	20	Marketing
18	205	110	Accounting
19	206	110	Accounting

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Data from Multiple Tables

Sometimes you need to use data from more than one table. In the slide example, the report displays data from two separate tables:

- Employee IDs exist in the EMPLOYEES table.
- Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
- Department names exist in the DEPARTMENTS table.

To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data from both of them.

## Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a `WHERE` clause.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product*, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition unless you have a specific need to combine all rows from all tables.

Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

## Generating a Cartesian Product

**EMPLOYEES (20 rows)**

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200 Whalen	10
2	201 Hartstein	20
...		
19	176 Taylor	80
20	178 Grant	(null)

**DEPARTMENTS (8 rows)**

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10 Administration	1700
2	20 Marketing	1800
3	50 Shipping	1500
4	60 IT	1400
5	80 Sales	2500
6	90 Executive	1700
7	110 Accounting	1700
8	190 Contracting	1700



**Cartesian product:**  
**20 x 8 = 160 rows**

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	10
2	101	10
...		
156	200	190
157	201	190
158	202	190
159	205	190
160	206	190

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Cartesian Products (continued)

A Cartesian product is generated if a join condition is omitted. The example in the slide displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition has been specified, all rows (20 rows) from the EMPLOYEES table are joined with all rows (8 rows) in the DEPARTMENTS table, thereby generating 160 rows in the output.

```
SELECT last_name, department_name dept_name
FROM employees, departments;
```

LAST_NAME	DEPT_NAME
1 Abel	Administration
2 Davies	Administration
3 De Haan	Administration
4 Ernst	Administration

...

160 Zlotkey	Contracting
-------------	-------------

## Types of Joins

### Oracle-proprietary joins (8i and earlier releases)

- Equijoin
- Nonequijoin
- Outer join
- Self-join

### SQL:1999-compliant joins

- Cross join
- Natural join
- Using clause
- Full (or two-sided) outer join
- Arbitrary join condition for outer join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Types of Joins

Before the release of Oracle9i Database, the join syntax was proprietary.

**Note:** The SQL:1999-compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in prior releases. For detailed information about the SQL:1999-compliant join syntax, see Lesson 5.

## Joining Tables Using Oracle Syntax

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 © 2009, Oracle. All rights reserved.

### Defining Joins

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 corresponding columns (that is, 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*                denotes the table and column from which data is retrieved  
*table1.column1* =            is the condition that joins (or relates) the tables together  
*table2.column2*

### 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.

## Equijoins

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60
11	107	60
12	124	50
13	141	50
14	142	50
15	143	50
16	144	50
17	149	80
18	174	80
19	176	80
20	178	(null)

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	20	Marketing
4	50	Shipping
5	50	Shipping
6	50	Shipping
7	50	Shipping
8	50	Shipping
9	60	T
10	60	T
11	60	T
12	80	Sales
13	80	Sales
14	80	Sales
15	90	Executive
16	90	Executive
17	90	Executive
18	110	Accounting
19	110	Accounting

Primary key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Equijoins

To determine an employee's department name, you compare the value 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 tables must be equal. Frequently, this type of join involves primary and foreign key complements.

**Note:** Equijoins are also called *simple joins* or *inner joins*.

## Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.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	205	Higgins	110	110	1700
...					
18	174	Abel	80	80	2500
19	176	Taylor	80	80	2500

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Retrieving Records with Equijoins

In the slide example:

- The **SELECT** clause specifies the column names to retrieve:
  - Employee last name, employee number, and department number, which are columns in the EMPLOYEES table
  - Department number, department name, 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:
 

```
EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
```

Because the DEPARTMENT\_ID column is common to both tables, it must be prefixed by the table name to avoid ambiguity.

## Additional Search Conditions Using the AND Operator

**EMPLOYEES**

	LAST_NAME	DEPARTMENT_ID
1	Whalen	10
2	Hartstein	20
3	Fay	20
4	Vargas	50
5	Matos	50
6	Davies	50
7	Rajs	50
8	Mourgos	50
9	Hunold	60
10	Ernst	60

...

**DEPARTMENTS**

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	20	Marketing
4	50	Shipping
5	50	Shipping
6	50	Shipping
7	50	Shipping
8	50	Shipping
9	60	IT
10	60	IT

...

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Additional Search Conditions

In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. For example, to display employee Matos's department number and department name, you need an additional condition in the WHERE clause.

```

SELECT last_name, employees.department_id,
       department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id
AND    last_name = 'Matos';

```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping



## Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Qualifying Ambiguous Column Names

You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT\_ID column could be from either the DEPARTMENTS table or the EMPLOYEES table. 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 the table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

The requirement to qualify ambiguous column names is also applicable to columns that may be ambiguous in other clauses, such as the SELECT clause or the ORDER BY clause.

## Using Table Aliases

- Use table aliases to simplify queries.
- Use table prefixes to improve performance.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. 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, therefore using less memory.

Notice how table aliases are identified in the FROM clause in the example. The table name is specified in full, followed by a space and then the table alias. The EMPLOYEES table has been given an alias of e, and the DEPARTMENTS table has 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, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid for only the current SELECT statement.

## Joining More Than Two Tables

EMPLOYEES

	LAST_NAME	DEPARTMENT_ID
1	Whalen	10
2	Hartstein	20
3	Fay	20
4	Higgins	110
5	Gietz	110
6	King	90
7	Kochhar	90
8	De Haan	90
9	Hunold	60
10	Ernst	60

DEPARTMENTS

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

LOCATIONS

	LOCATION_ID	CITY
1	1400	Southlake
2	1500	South San Francisco
3	1700	Seattle
4	1800	Toronto
5	2500	Oxford

...

To join  $n$  tables together, you need a minimum of  $n-1$  join conditions. For example, to join three tables, a minimum of two joins is required.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Additional Search Conditions

Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

	LAST_NAME	DEPARTMENT_NAME	CITY
1	Whalen	Administration	Seattle
2	Hartstein	Marketing	Toronto
3	Fay	Marketing	Toronto
4	Higgins	Accounting	Seattle
5	Gietz	Accounting	Seattle
6	King	Executive	Seattle
7	Kochhar	Executive	Seattle

...

## Nonequijoins

**EMPLOYEES**

R	LAST_NAME	R	SALARY
1	Whalen		4400
2	Hartstein		13000
3	Fay		6000
4	Higgins		12000
5	Gietz		8300
6	King		24000
7	Kochhar		17000
8	De Haan		17000
9	Hunold		9000
10	Ernst		6000

...

**JOB\_GRADES**

R	GRADE_LEVEL	R	LOWEST_SAL	R	HIGHEST_SAL
1	A		1000		2999
2	B		3000		5999
3	C		6000		9999
4	D		10000		14999
5	E		15000		24999
6	F		25000		40000

Salary in the **EMPLOYEES** table must be between lowest salary and highest salary in the **JOB\_GRADES** table.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Nonequijoins

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the **EMPLOYEES** table and the **JOB\_GRADES** table is an example of a nonequijoin. A relationship between the two tables is that the **SALARY** column in the **EMPLOYEES** table must be between the values in the **LOWEST\_SALARY** and **HIGHEST\_SALARY** columns of the **JOB\_GRADES** table. The relationship is obtained using an operator other than equality (=).

## Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  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	Mourgos	5800	B
8	Ernst	6000	C

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Nonequijoins (continued)

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

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. 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 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  $\leq$  and  $\geq$ ) can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN.

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

## Outer Joins

DEPARTMENTS

R	DEPARTMENT_NAME	R	DEPARTMENT_ID
1	Administration		10
2	Marketing		20
3	Shipping		50
4	IT		60
5	Sales		80
6	Executive		90
7	Accounting		110
8	Contracting		190

EMPLOYEES

R	DEPARTMENT_ID	R	LAST_NAME
1	10		Whalen
2	20		Hartstein
3	20		Fay
4	110		Higgins
5	110		Gietz
6	90		King
7	90		Kochhar
8	90		De Haan
9	60		Hunold
10	60		Ernst

...

There are no employees in department 190.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of the EMPLOYEES and DEPARTMENTS tables, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

R	LAST_NAME	R	DEPARTMENT_ID	R	DEPARTMENT_NAME
1	Whalen		10		Administration
2	Hartstein		20		Marketing
3	Fay		20		Marketing
4	Higgins		110		Accounting

...

19	Taylor		80		Sales
----	--------	--	----	--	-------

## Outer Joins Syntax

- You use an outer join to see rows that do not meet the join condition.
- The outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column (+) = table2.column;
```

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column (+);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using Outer Joins to Return Records with No Direct Match

The missing rows can be returned if an *outer join* operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the “side” of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined.

In the syntax:

`table1.column =` is the condition that joins (or relates) the tables together

`table2.column (+)` is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides. (Place the outer join symbol following the name of the column in the table without the matching rows.)

## Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
...			
18	Abel	80	Sales
19	Taylor	80	Sales
20	(null)	(null)	Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using Outer Joins to Return Records with No Direct Match (continued)

The slide example displays employee last names, department IDs, and department names. The Contracting department does not have any employees. The empty value is shown in the output.

#### Outer Join Restrictions

- The outer join operator can appear on only *one* side of the expression - the side that has information missing. It returns those rows from one table that have no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.



## Self-Joins

**EMPLOYEES (WORKER)**

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

**EMPLOYEES (MANAGER)**

	EMPLOYEE_ID	LAST_NAME
1	100	King
2	101	Kochhar
3	102	De Haan
4	103	Hunold
5	104	Ernst
6	107	Lorentz
7	124	Mourgos
8	141	Rajs
9	142	Davies
10	143	Matos

...

**MANAGER\_ID in the WORKER table is equal to  
EMPLOYEE\_ID in the MANAGER table.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself; this type of join is called a *self-join*.

For example, to find the name of Lorentz's manager, you need to do the following:

- Find Lorentz in the EMPLOYEES table by looking at the LAST\_NAME column.
- Find the manager number for Lorentz by looking at the MANAGER\_ID column.  
Lorentz's manager number is 103.
- Find the name of the manager who has EMPLOYEE\_ID 103 by looking at the LAST\_NAME column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the LAST\_NAME column and the MANAGER\_ID value of 103. The second time you look in the EMPLOYEE\_ID column to find 103 and the LAST\_NAME column to find Hunold.

## Joining a Table to Itself

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

	WORKER.LAST_NAME  'WORKSFOR'  MANAGER.LAST_NAME
1	Fay works for Hartstein
2	Gietz works for Higgins
3	Zlotkey works for King
4	Mourgos works for King
5	De Haan works for King
6	Kochhar works for King

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Joining a Table to Itself (continued)

The slide example joins the EMPLOYEES table to itself. To simulate two tables in the FROM clause, there are two aliases, namely w and m, for the same table, EMPLOYEES.

In this example, the WHERE clause contains the join that means “where a worker’s manager number matches the employee number for the manager.”

## Summary

In this appendix, you should have learned how to use joins to display data from multiple tables by using Oracle-proprietary syntax for versions 8i and earlier.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

There are multiple ways to join tables.

#### Types of Joins

- Cartesian products
- Equijoins
- Nonequijoins
- Outer joins
- Self-joins

#### Cartesian Products

A Cartesian product results in a display of all combinations of rows. This is done by omitting the WHERE clause.

#### Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.

## Practice C: Overview

This practice covers writing queries to join tables using Oracle syntax.

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Practice C: Overview

This practice is designed to give you a variety of exercises that join tables using the Oracle syntax that was covered in this appendix.

**Practice C**

1. Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output.

R2	LOCATION_ID	R2	STREET_ADDRESS	R2	CITY	R2	STATE_PROVINCE	R2	COUNTRY_NAME
1	1400		2014 Jabberwocky Rd		Southlake		Texas		United States of America
2	1500		2011 Interiors Blvd		South San Francisco		California		United States of America
3	1700		2004 Charade Rd		Seattle		Washington		United States of America
4	1800		460 Bloor St. W.		Toronto		Ontario		Canada
5	2500		Magdalen Centre, The Oxford Science Park		Oxford		Oxford		United Kingdom

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

R2	LAST_NAME	R2	DEPARTMENT_ID	R2	DEPARTMENT_NAME
1	Whalen		10		Administration
2	Hartstein		20		Marketing
3	Fay		20		Marketing
4	Higgins		110		Accounting
5	Gietz		110		Accounting
6	King		90		Executive
7	Kochhar		90		Executive
8	De Haan		90		Executive
9	Hunold		60		IT
10	Ernst		60		IT
11	Lorentz		60		IT
12	Mourgos		50		Shipping
13	Rajs		50		Shipping
14	Davies		50		Shipping
15	Matos		50		Shipping
16	Vargas		50		Shipping
17	Zlotkey		80		Sales
18	Abel		80		Sales
19	Taylor		80		Sales

**Practice C (continued)**

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

4. Create a report to display the employee last name and employee number along with the employee's manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab\_c\_04.sql.

	Employee	EMP#	Manager	Mgr#
1	Fay	202	Hartstein	201
2	Gietz	206	Higgins	205
3	Zlotkey	149	King	100
4	Mourgos	124	King	100
5	De Haan	102	King	100
6	Kochhar	101	King	100
7	Hartstein	201	King	100
8	Higgins	205	Kochhar	101
9	Whalen	200	Kochhar	101
10	Hunold	103	De Haan	102
11	Lorentz	107	Hunold	103
12	Ernst	104	Hunold	103
13	Vargas	144	Mourgos	124
14	Matos	143	Mourgos	124
15	Davies	142	Mourgos	124
16	Rajs	141	Mourgos	124
17	Grant	178	Zlotkey	149
18	Taylor	176	Zlotkey	149
19	Abel	174	Zlotkey	149

**Practice C (continued)**

5. Modify `lab_c_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named `lab_c_05.sql`. Run the query in `lab_c_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	Fay	202	Hartstein	201
2	Gietz	206	Higgins	205
3	Zlotkey	149	King	100
4	Mourgos	124	King	100
5	De Haan	102	King	100
6	Kochhar	101	King	100

...

19	Abel	174	Zlotkey	149
20	King	100	(null)	(null)

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_c_06.sql`.

	DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs

...

41	110	Gietz	Higgins
42	110	Higgins	Gietz

**Practice C (continued)**

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB\_GRADES table, first show the structure of the JOB\_GRADES table. Second, create a query that displays the last name, job, department name, salary, and grade for all employees.

Name	Null	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1	Vargas	ST_CLERK	Shipping	2500	A
2	Matos	ST_CLERK	Shipping	2600	A
3	Davies	ST_CLERK	Shipping	3100	B
4	Rajs	ST_CLERK	Shipping	3500	B
5	Lorentz	IT_PROG	IT	4200	B
6	Whalen	AD_ASST	Administration	4400	B
7	Mourgos	ST_MAN	Shipping	5800	B
8	Ernst	IT_PROG	IT	6000	C
9	Fay	MK_REP	Marketing	6000	C
10	Gietz	AC_ACCOUNT	Accounting	8300	C
11	Taylor	SA_REP	Sales	8600	C
12	Hunold	IT_PROG	IT	9000	C
13	Zlotkey	SA_MAN	Sales	10500	D
14	Abel	SA_REP	Sales	11000	D
15	Higgins	AC_MGR	Accounting	12000	D
16	Hartstein	MK_MAN	Marketing	13000	D
17	De Haan	AD_VP	Executive	17000	E
18	Kochhar	AD_VP	Executive	17000	E
19	King	AD_PRES	Executive	24000	E



**Practice C (continued)**

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all employees hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

	 LAST_NAME	 HIRE_DATE
1	Fay	17-AUG-97
2	Lorentz	07-FEB-99
3	Mourgos	16-NOV-99
4	Matos	15-MAR-98
5	Vargas	09-JUL-98
6	Zlotkey	29-JAN-00
7	Taylor	24-MAR-98
8	Grant	24-MAY-99

9. The HR department needs to find the name and hire date for all employees who were hired before their managers, along with their manager's name and hire date. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively. Save the script to a file named lab\_c\_09.sql.

	LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar	21-SEP-89
2	Hunold	03-JAN-90	De Haan	13-JAN-93
3	Vargas	09-JUL-98	Mourgos	16-NOV-99
4	Matos	15-MAR-98	Mourgos	16-NOV-99
5	Davies	29-JAN-97	Mourgos	16-NOV-99
6	Rajs	17-OCT-95	Mourgos	16-NOV-99
7	Grant	24-MAY-99	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Abel	11-MAY-96	Zlotkey	29-JAN-00

VINOD BUSANAGA (b\_vinod87@yahoo.com) has a non-transferable  
license to use this Student Guide.

# D

## Using SQL\*Plus

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

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

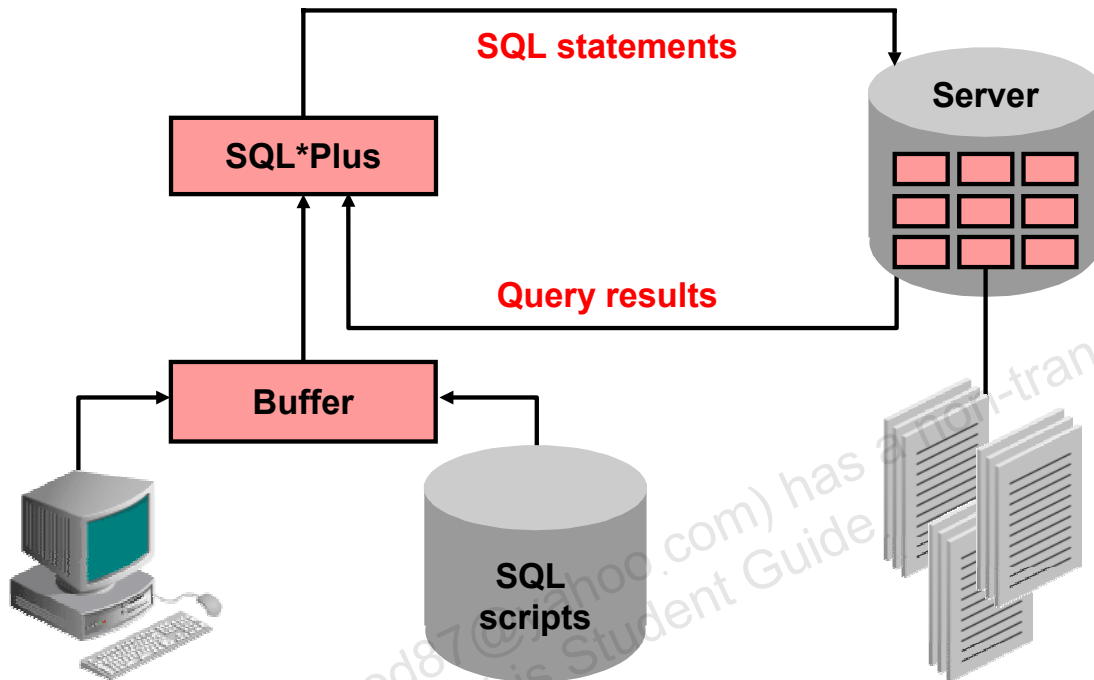
**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

You might want to create `SELECT` statements that can be used again and again. This appendix also covers the use of SQL\*Plus commands to execute SQL statements. You learn how to format output using SQL\*Plus commands, edit SQL commands, and save scripts in SQL\*Plus.

## SQL and SQL\*Plus Interaction



Copyright © 2009, Oracle. All rights reserved.

### SQL and SQL\*Plus

SQL is a command language for communication with the Oracle9i Server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new SQL statement. SQL\*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle9i Server for execution. It contains its own command language.

#### Features of SQL

- Can be used by a range of users, including those with little or no programming experience
- Is a nonprocedural language
- Reduces the amount of time required for creating and maintaining systems
- Is an English-like language

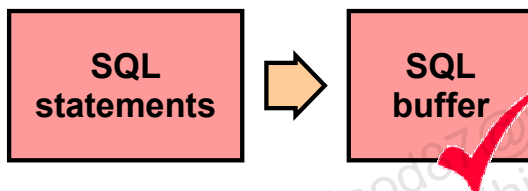
#### Features of SQL\*Plus

- 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

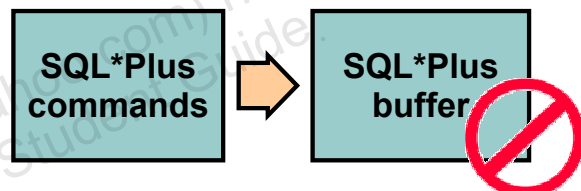
### SQL

- 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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL and SQL\*Plus (continued)

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

## Overview of SQL\*Plus

- 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 file to buffer to edit

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL\*Plus

SQL\*Plus is an environment in which you can do the following:

- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repeated use in the future

SQL\*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session
Format	Format query results
File manipulation	Save, load, and run script files
Execution	Send SQL statements from the SQL buffer to the Oracle server
Edit	Modify SQL statements in the buffer
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions

## Logging In to SQL\*Plus

- From a Linux desktop icon
- From a Linux terminal

```
sqlplus [username[/password
        [@database]]]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Logging In to SQL\*Plus

How you invoke SQL\*Plus depends on which type of operating system or environment you are running.

To log in using a desktop icon:

1. Double-click the sqlplus desktop icon.
2. Enter the username, password, and database name.

To log in from a Linux terminal:

1. Log on to your machine and open a terminal.
2. Enter the SQL\*Plus command shown in the slide.

In the syntax:

*username* Your database username  
*password* Your database password (Your password is visible if you enter it here.)  
*@database* 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 Table Structure

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

```
DESC[RIBE] tablename
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Displaying Table Structure

In SQL\*Plus, you can display the structure of a table 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 JOB\_GRADES table, use this command:

```
SQL> DESCRIBE job_grades
```

Name	Null?	Type
-----	-----	-----
GRADE_LEVEL		VARCHAR2 (3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

## Displaying Table Structure

```
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)

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Displaying Table Structure (continued)

The example in the slide displays the information about the structure of the DEPARTMENTS table.

In the result:

**Null?** Specifies whether a column must contain data (NOT NULL indicates that a column must contain data.)

**Type** Displays the data type for a column

The following table describes the data types:

Data Type	Description
NUMBER ( <i>p</i> , <i>s</i> )	Number value that has a maximum number of digits <i>p</i> , which is the number of digits to the right of the decimal point <i>s</i>
VARCHAR2 ( <i>s</i> )	Variable-length character value of maximum size <i>s</i>
DATE	Date and time value between January 1, 4712 B.C., and December 31, 9999 A.D.
CHAR ( <i>s</i> )	Fixed-length character value of size <i>s</i>

## 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 © 2009, Oracle. All rights reserved.

### SQL\*Plus Editing Commands

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

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

#### 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 then appears.

## SQL\*Plus Editing Commands

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL\*Plus Editing Commands (continued)

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>
0 <i>text</i>	Inserts a line before line 1

**Note:** 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 (-).

## Using LIST, n, and APPEND

```
SQL> LIST
```

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

```
SQL> 1
```

```
1* SELECT last_name
```

```
SQL> A , job_id
```

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

```
SQL> L
```

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



Copyright © 2009, Oracle. All rights reserved.

### Using LIST, n, and APPEND

- 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; APPEND can be abbreviated to A.

## Using the CHANGE Command

```
SQL> L
```

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

```
SQL> c/employees/departments
```

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

```
SQL> L
```

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

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Using the CHANGE Command

- 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 © 2009, Oracle. All rights reserved.

### SQL\*Plus File Commands

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

## Using the SAVE and START Commands

```
SQL> L
1  SELECT last_name, manager_id, department_id
2* FROM    employees
SQL> SAVE my_query
```

```
Created file my_query
```

```
SQL> START my_query
```

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
20 rows selected.		

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### SAVE

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

### START

Use the START command to run a script in SQL\*Plus.

### EDIT

Use the EDIT command to edit an existing script. This opens an editor with the script file in it. When you have made the changes, quit the editor to return to the SQL\*Plus command line.



## Summary

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

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

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

SQL\*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

VINOD BUSANAGA (b\_vinod87@yahoo.com) has a non-transferable  
license to use this Student Guide.

# Using SQL Developer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- List the key features of Oracle SQL Developer
- Install Oracle SQL Developer
- Identify menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Execute SQL statements and SQL scripts
- Create and save reports

**ORACLE**

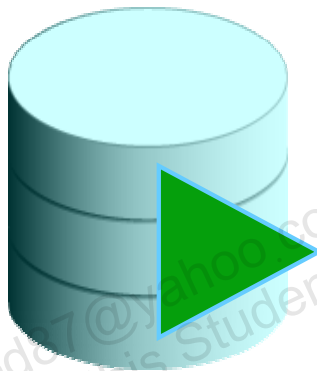
Copyright © 2009, Oracle. All rights reserved.

### Objectives

In this appendix, you are introduced to the graphical tool SQL Developer. You learn how to use SQL Developer for your database development tasks. You learn how to use SQL Worksheet to execute SQL statements and SQL scripts.

## What Is Oracle SQL Developer?

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

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## What Is Oracle SQL Developer?

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, 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 using standard Oracle Database authentication. When connected, you can perform operations on objects in the database.

## Key Features

- Developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Default connectivity by using the JDBC Thin driver
- Does not require an installer
- Connects to any Oracle Database version 9.2.0.1 and later
- Bundled with JRE 1.5

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Key Features of SQL Developer

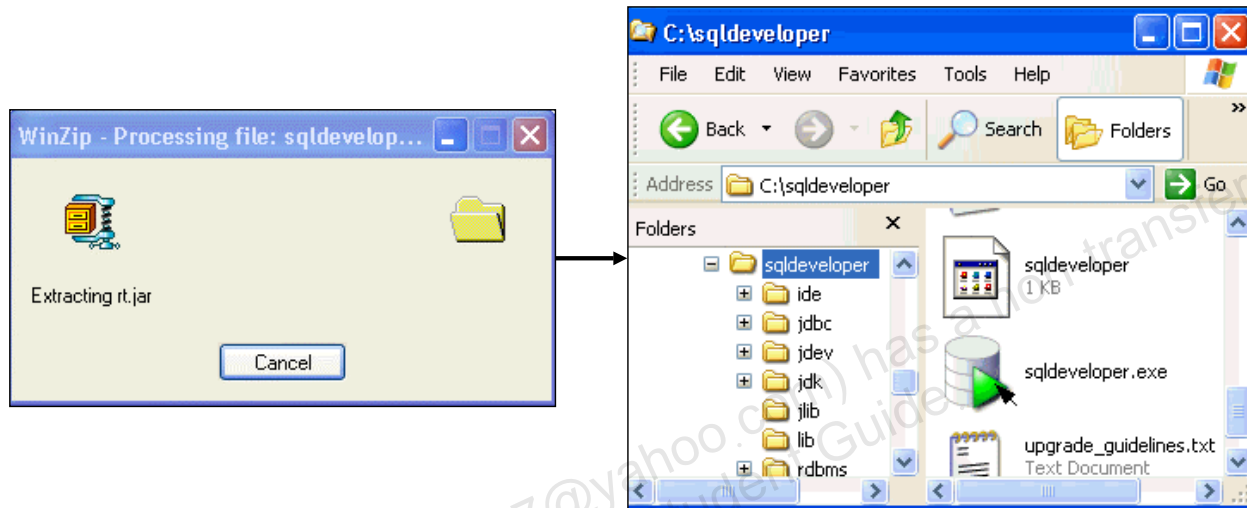
Oracle SQL Developer is developed in Java, leveraging the Oracle JDeveloper IDE. The tool runs on the Windows, Linux, and Mac OS X platforms. You can install SQL Developer on the Database Server and connect remotely from your desktop, thus avoiding client server network traffic.

Default connectivity to the database is through the JDBC Thin driver; so, no Oracle Home is required. SQL Developer does not require an installer and you need to just unzip the downloaded file.

With SQL Developer, users can connect to Oracle Database 9.2.0.1 and later versions, and all Oracle Database editions including Express Edition. SQL Developer is bundled with JRE 1.5, with an additional `tools.jar` to support Windows clients. Non-Windows clients need only JDK 1.5.

## Installing SQL Developer

Download the Oracle SQL Developer kit and unzip it into any directory on your machine.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Installing SQL Developer

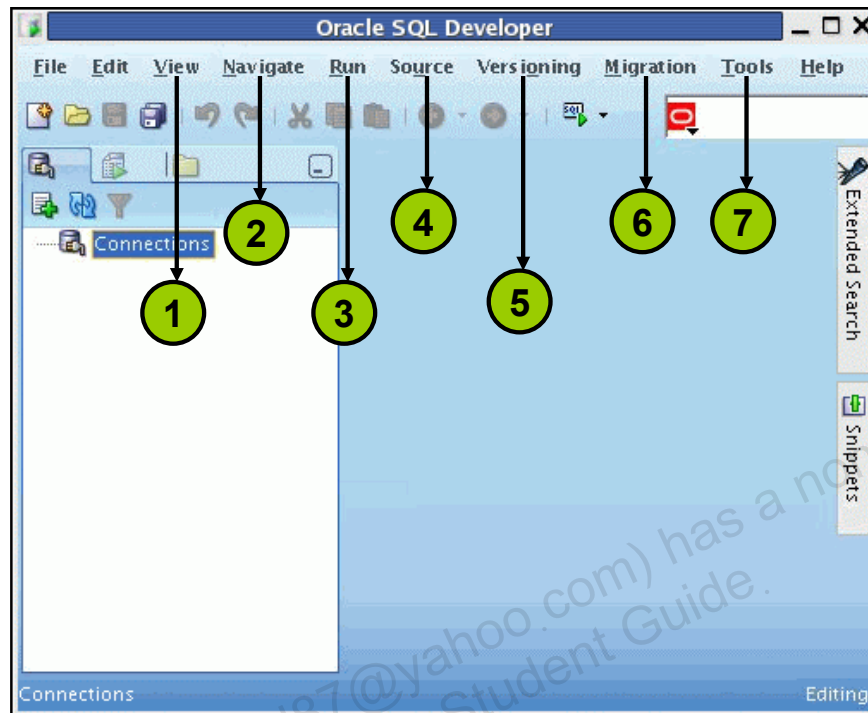
Oracle SQL Developer does not require an installer. To install SQL Developer, you need an unzip tool.

To install SQL Developer, perform the following steps:

1. Create a folder as <local drive>:\SQL Developer.
2. Download the SQL Developer kit from <http://www.oracle.com/technology/software/products/sql/index.html>.
3. Unzip the downloaded SQL Developer kit into the folder created in step 1.

To start SQL Developer, go to <local drive>:\SQL Developer, and double-click `sqldeveloper.exe`.

## Menus for SQL Developer



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Menus for SQL Developer

SQL Developer has two main navigation tabs.

- **Connections Navigator:** By using this tab, you can browse database objects and users to which you have access.
- **Reporting Tab:** By using this tab, you can run predefined reports or create and add your own reports.

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. The menus at the top contain standard entries, plus entries for features specific to SQL Developer.

1. **View:** Contains options that affect what is displayed in the SQL Developer interface
2. **Navigate:** Contains options for navigating to panes and for the execution of subprograms
3. **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected
4. **Source:** Contains options for use when editing functions and procedures
5. **Versioning:** Enables you to work with the files placed under source code control
6. **Migration:** Enables you to migrate from another database, such as Microsoft SQL Server and Microsoft Access, to an Oracle Database
7. **Tools:** Invokes SQL Worksheet, Preferences, and any added External Tools



## 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
  - For multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an XML file.
- Each additional database connection created is listed in the connections navigator hierarchy.



Copyright © 2009, Oracle. All rights reserved.

### Creating a Database Connection

A connection is a SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use SQL Developer, you must have at least one database connection which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

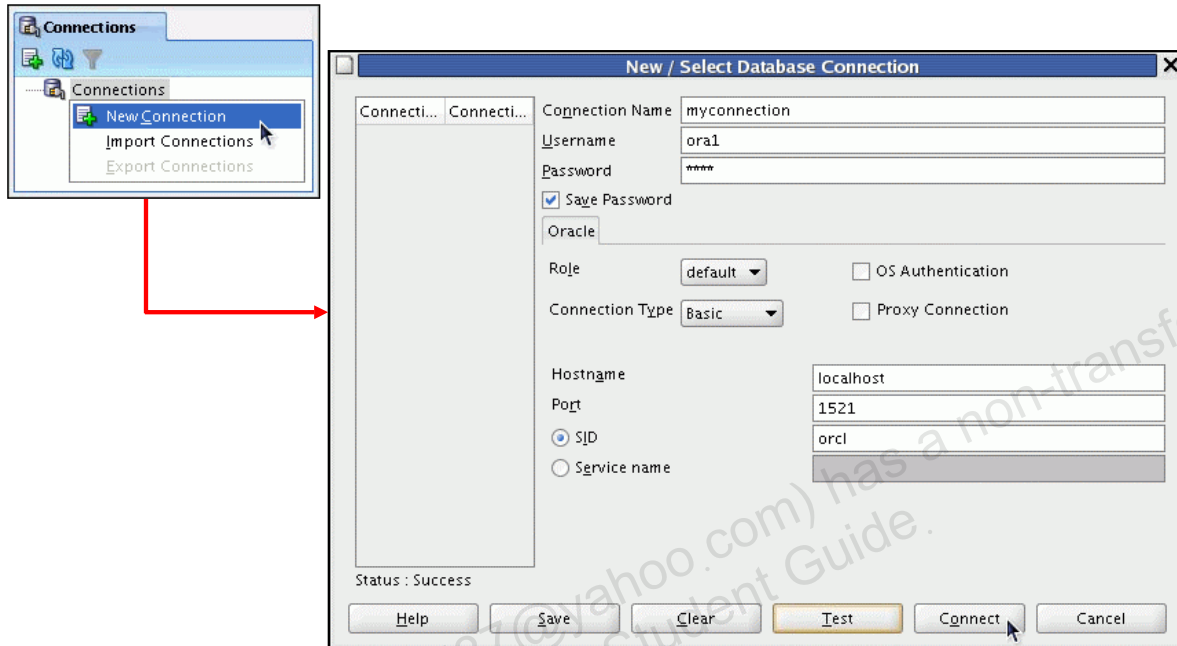
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory. But, it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and display the database connections dialog box, SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

**Note:** On Windows systems, if the `tnsnames.ora` file exists but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it later.

You can create additional connections to connect to the same database but as different users, or to connect to different databases. Each database connection is listed in the Connections navigator hierarchy.

## Creating a Database Connection



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Database Connection (continued)

To create a database connection, perform the following steps:

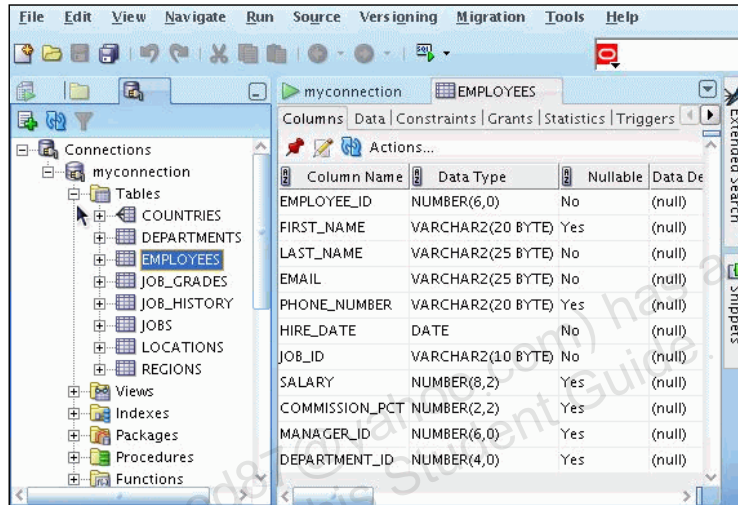
1. Double-click `<your_path>\sqldeveloper\sqldeveloper.exe`.
2. On the **Connections** tabbed page, right-click **Connections** and select **New Connection**.
3. Enter the connection name, username, password, host name, port number, and SID for the database you want to connect.
4. Click **Test** to make sure that the connection has been set correctly.
5. Click **Connect**.

**Note:** 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 will not be prompted for the password.

## Browsing Database Objects

Use the Database Navigator to:

- Browse through many objects in a database schema
- Do a quick review of the definitions of objects



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Browsing Database Objects

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

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, the details about columns, constraints, grants, statistics, triggers and more are all displayed in an easy-to-read tabbed page.

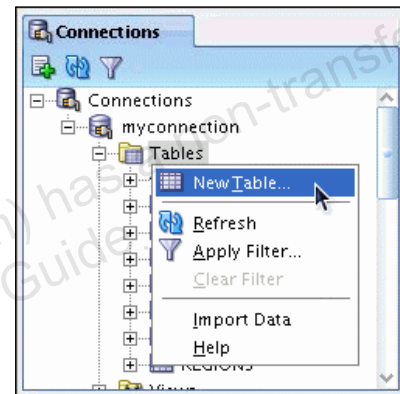
If you want to see the definition of the EMPLOYEES table as shown on the slide, perform the following steps:

1. Expand the connection node in the Connections Navigator.
2. Expand **Tables**.
3. Double-click **EMPLOYEES**.

Using the Data tab, you can enter new rows, update data, and commit these changes to the database.

## 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 many context-sensitive menus.
- View the DDL for adjustments, such as creating a new object or editing an existing schema object.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

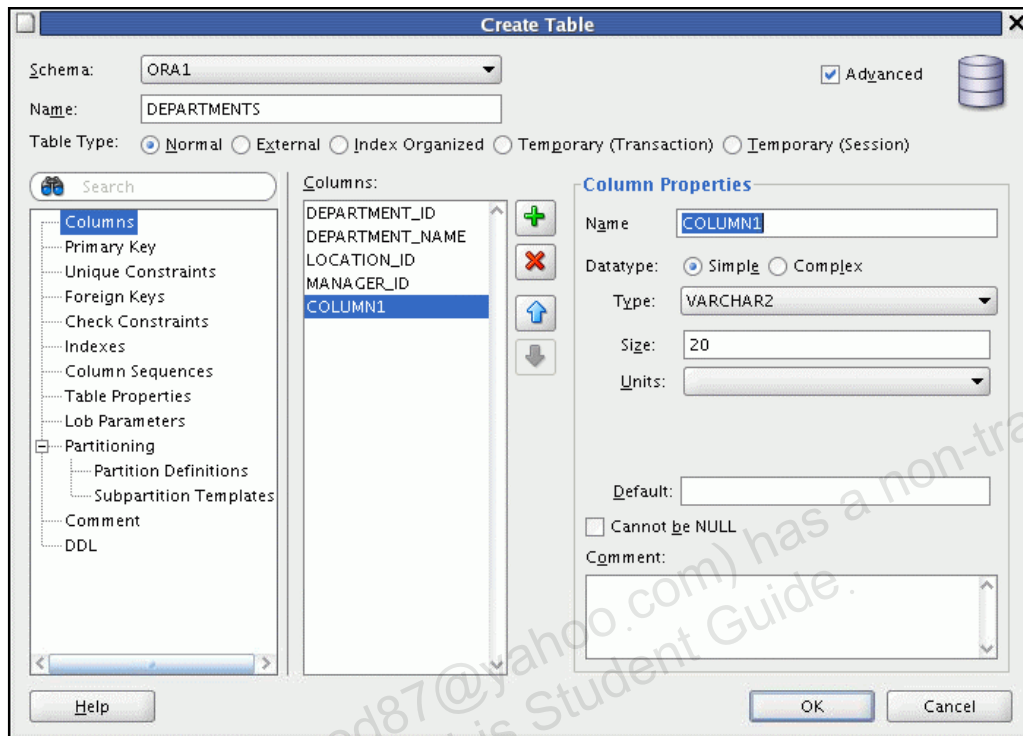
### Creating a Schema Object

SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects using the context menus. Once created, you can edit the objects using an edit dialog or one of 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 creating 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 for creating and editing database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

## Creating a New Table: Example



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 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 tabs, in which you can specify an extended set of features while creating the table.

The example in the slide shows creating the DEPENDENT table by selecting the **Advanced** check box.

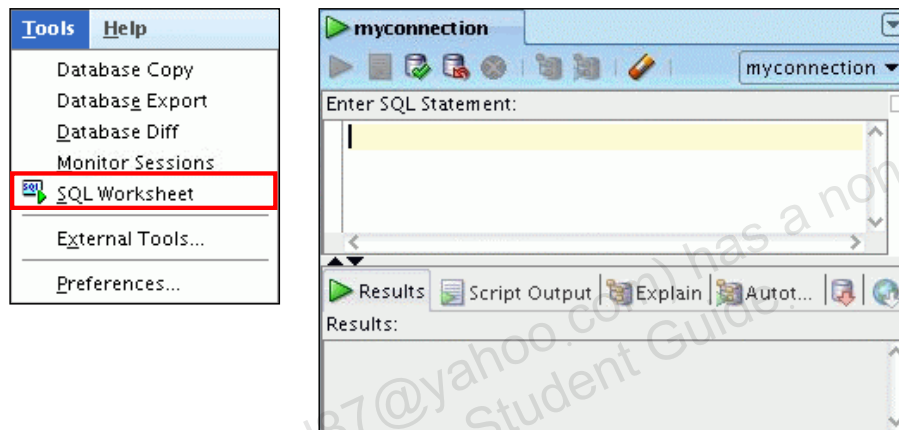
To create a new table, perform the following steps:

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

Although it is not required, you should also specify a primary key by selecting Primary Key in the dialog box. Sometimes, you may want to edit the table that you have created. To edit a table, 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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using SQL Worksheet

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

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

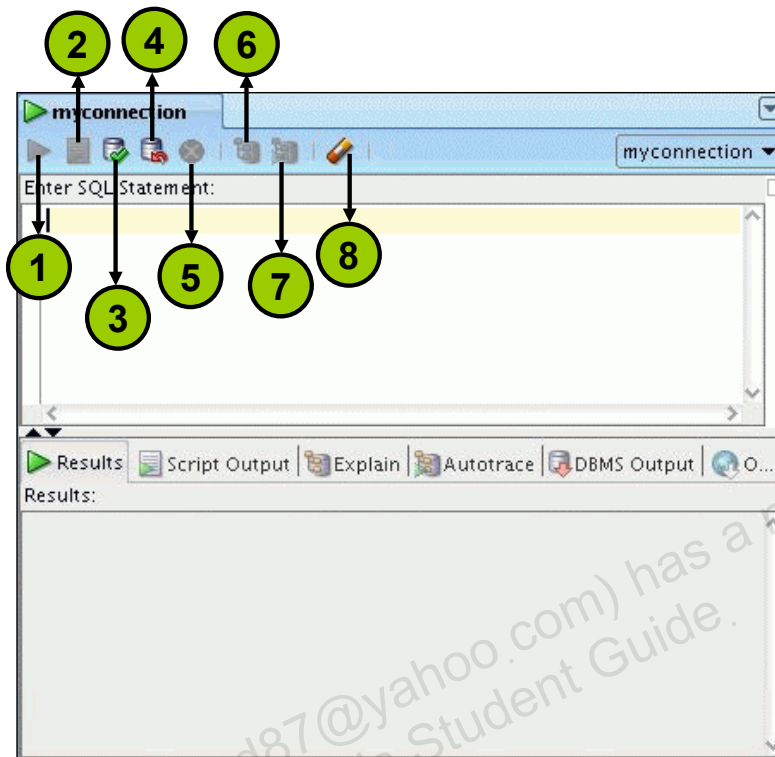
- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL worksheet by using any of the following two options:

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



## Using SQL Worksheet



ORACLE

Copyright © 2009, Oracle. All rights reserved.

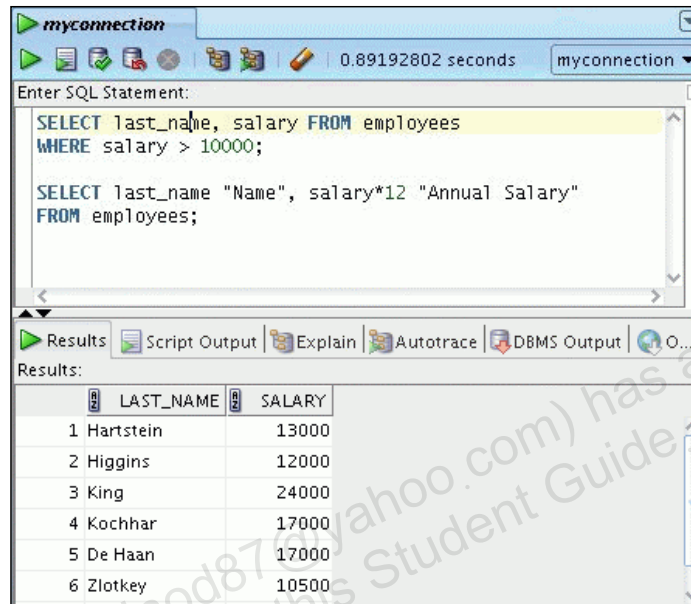
### Using SQL Worksheet (continued)

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

1. **Execute Statement:** Executes the statement at the cursor in the Enter SQL Statement box. You can use bind variables in the SQL statements but not substitution variables.
2. **Run Script:** Executes all 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. **Commit:** Writes any changes to the database, and ends the transaction
4. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction
5. **Cancel:** Stops the execution of any statements currently being executed
6. **Execute Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
7. **Autotrace:** Generates trace information for the statement, which you can see by clicking the Autotrace tab
8. **Clear:** Erases the statement or statements in the Enter SQL Statement box

## Executing SQL Statements

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Executing SQL Statements

In SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

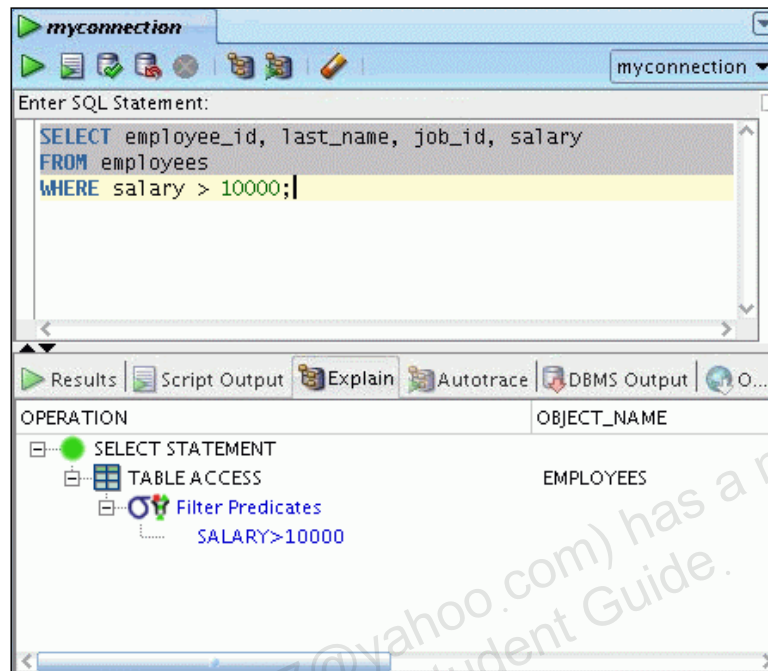
When you type in the statement, the SQL keywords are automatically highlighted. To execute a SQL statement, ensure that your cursor is within the statement and click the **Execute Statement** icon. Alternatively, you can press the **F9** key.

To execute multiple SQL statements and see the results, click the **Run Script** icon. Alternatively, you can press the **F5** key.

In the example in the slide, as there are multiple SQL statements, the first statement is terminated with a semicolon. The cursor is in the first statement and, therefore, when the statement is executed, results corresponding to the first statement are displayed in the Results box.



## Viewing the Execution Plan



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Viewing the Execution Plan

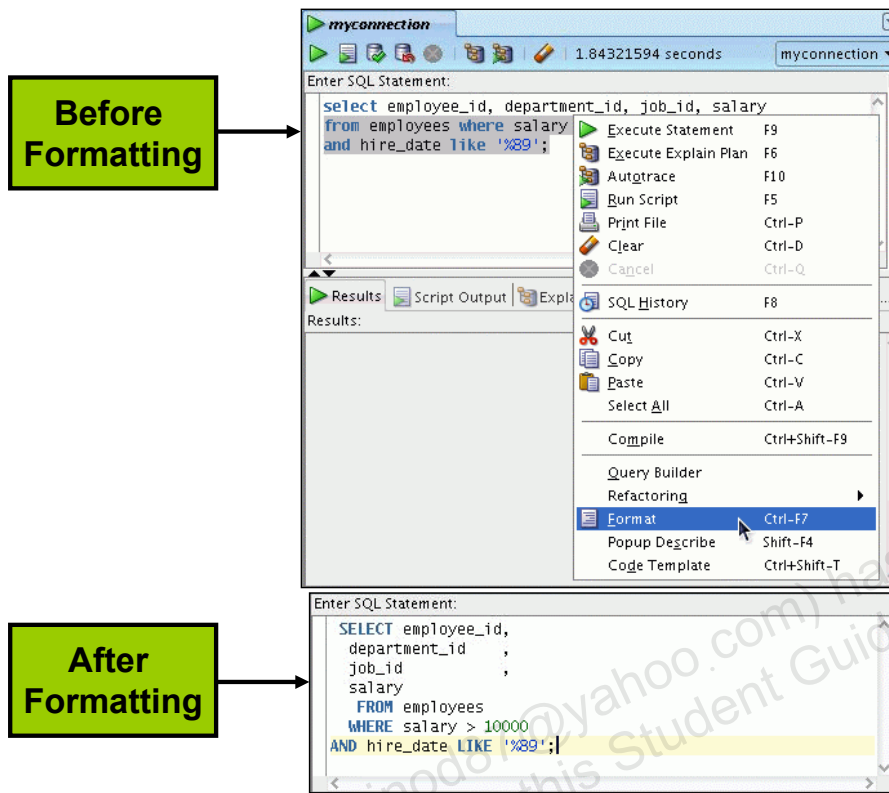
You can execute a SQL script and view the execution plan. To execute a SQL script file, perform the following steps:

1. From the **File** menu, Select **Open**.
2. In the Open dialog box, double-click the **.sql** file.
3. Click the **Run Script** icon.

When you double-click the **.sql** file, the SQL statements are loaded into the Enter SQL Statement box. You can execute the script or each line individually. The results are displayed in the Script Output area.

The example in the slide shows the execution plan. The Execute Explain Plan icon generates the execution plan. An execution plan is the sequence of operations that are performed to execute the statement. You can see the execution plan by clicking the **Explain** tab.

## Formatting the SQL Code



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Formatting the SQL Code

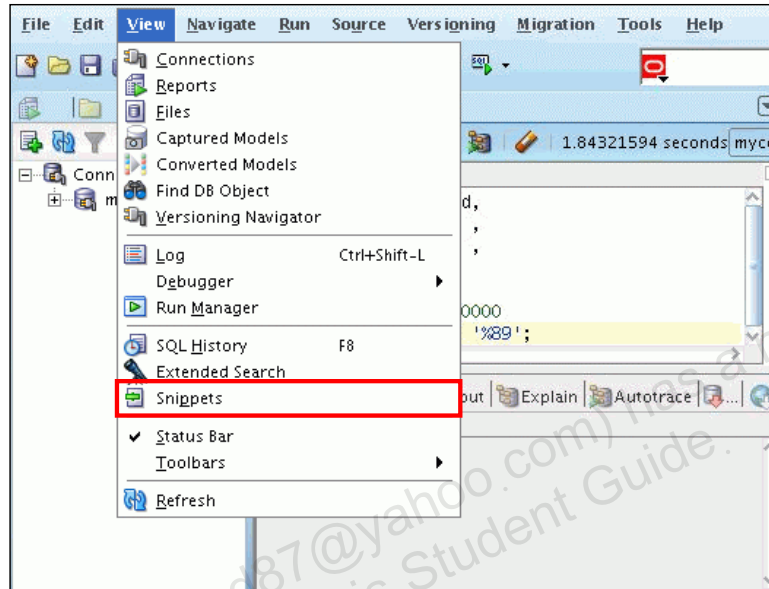
You may want to beautify the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has the feature of formatting the SQL code.

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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

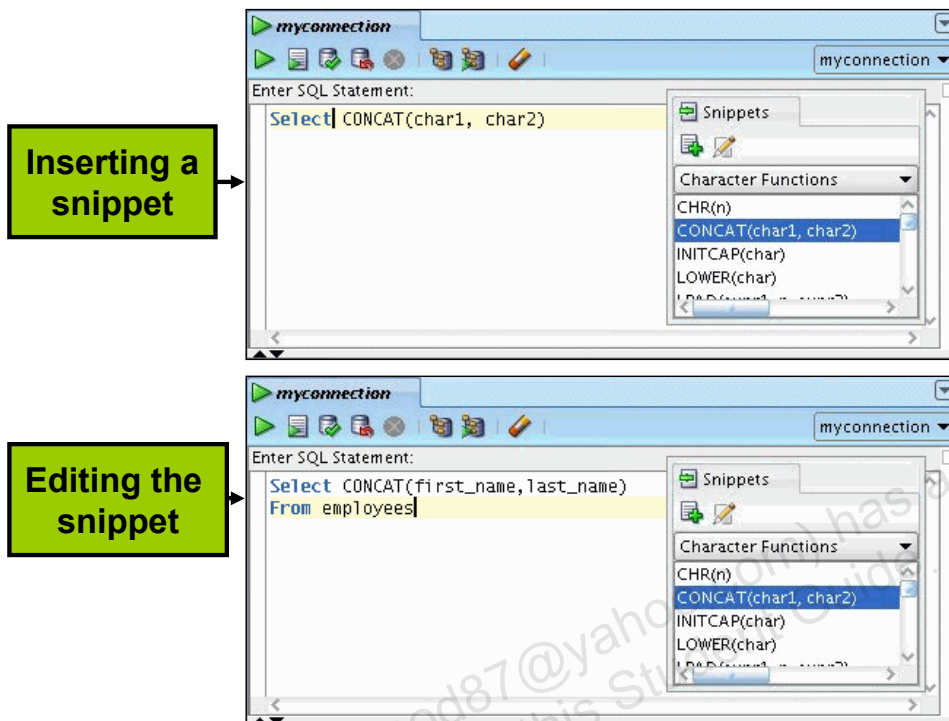
### Using Snippets

You may want to use certain code fragments when you are using the SQL Worksheet or creating or editing a PL/SQL function or procedure. SQL Developer has the feature called Snippets. Snippets are code fragments, such as SQL functions, Optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag and drop snippets into 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 © 2009, Oracle. All rights reserved.

### Using Snippets: Example

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 such as in the following:

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

## Using SQL\*Plus

- SQL Worksheet does not support all SQL\*Plus statements.
- SQL\*Plus statements that are not supported by SQL Worksheet are:
  - append
  - archive
  - attribute
  - break
  - change
  - clear

**ORACLE**

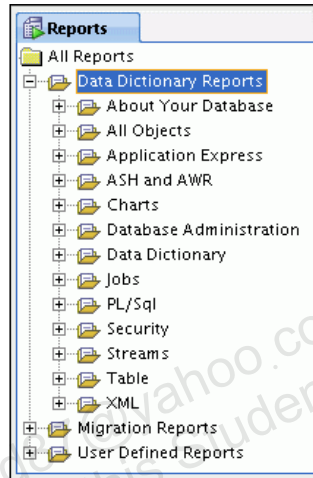
Copyright © 2009, Oracle. All rights reserved.

### Using SQL\*Plus

SQL Worksheet supports some SQL\*Plus statements. SQL\*Plus statements must be interpreted by the SQL Worksheet before being passed to the database; any SQL\*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database. For example, some of the SQL\*Plus statements that are not supported by SQL Worksheet are listed in the slide. For the complete list of SQL\*Plus statements that are supported and not supported by SQL Worksheet, refer to SQL Developer online Help.

## Database Reporting

- SQL Developer provides you with a number of predefined reports about your database and objects.
- The Reports are organized into categories.
- You can create your own customized reports too.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Database Reporting

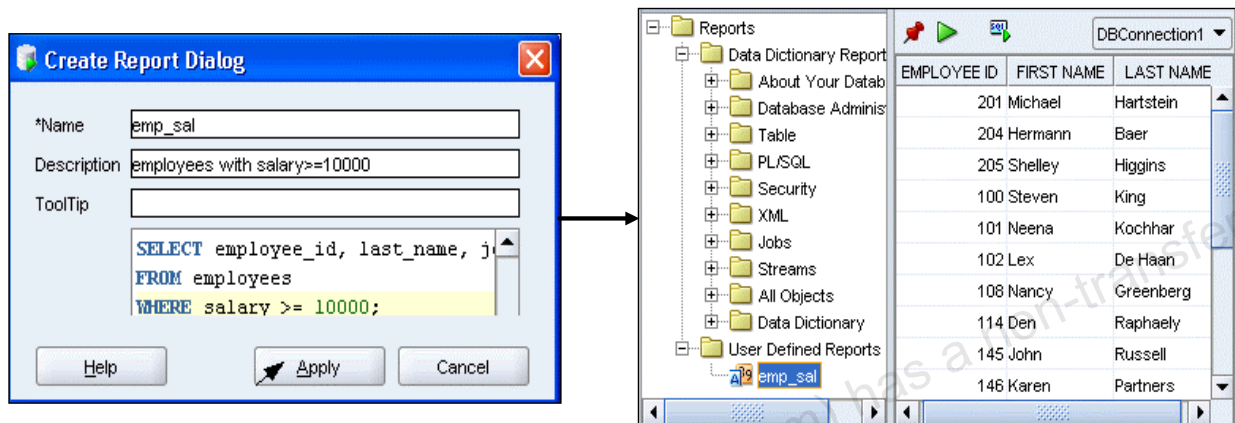
SQL Developer provides many reports about the database and its objects. These reports are grouped into the following categories:

- About Your Database reports
- Object reports
- Application Express reports
- Charts
- Database Administration reports
- Data Dictionary reports
- Jobs reports
- PL/SQL reports
- Security reports
- Streams reports
- Table reports
- XML reports

To display a report, click the Reports tabbed page and then select the report type. 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 © 2009, Oracle. All rights reserved.

### Creating a User-Defined Report

User-defined reports are any reports that are 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` under the directory for user-specific information.



## Summary

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

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.



---

# Index

---

**Note:** A bolded number or letter refers to an entire lesson or appendix.

## A

Adding Data Through a View 10-15

ADD\_MONTHS Function 3-22, 3-23, 3-46, 3-60

Advanced Features of the SELECT Statement 6-2

Alias 1-4, 1-8, 1-14, 1-15, 1-16, 1-17, 1-25, 1-37, 1-40, 2-7,  
2-20, 2-21, 2-31, 3-46, 4-12, 5-8, 5-11, 5-12, 5-20, 5-29, 7-17,  
7-21, 9-33, 10-7, 10-9, 10-11, 11-22, C-11, C-12, C-15, C-20, C-21

ALL Operator 6-16, 7-11, 7-12, 7-18, 7-22

ALL\_COL\_COMMENTS Dictionary View 11-19

ALL\_OBJECTS Dictionary View 11-5, 11-7

ALL\_TAB\_COMMENTS Dictionary View 11-19

ALTER SEQUENCE Statement 10-30, 10-31

ALTER TABLE Statement 9-34, 10-27

American National Standards Institute (ANSI) i-12, i-25, i-28, i-30, 3-26, 3-54,  
5-4, 8-27, D-4

Ampersand Substitution 2-2, 2-24, 2-25, 2-28, 2-31, 2-32, 3-11  
Double-ampersand (&&) 2-23, 2-28

ANY Operator 6-15

Arguments Function 3-3, 3-5, 3-15, 3-50

Arithmetic Expressions 1-9, 1-13, 1-16, 2-4

Arithmetic Operators 1-9, 1-10, 3-20, 3-21, 3-60

AS Keyword 1-14, 1-15, 1-16

AS Subquery 9-32, 10-7

Attributes i-18, i-19, i-20

AUTOCOMMIT Command 8-29

AVG Function 4-13

## B

BETWEEN Range Condition 2-9

**C**

Calculations 1-9, 1-14, 1-38, 3-3, 3-14, 3-20, 3-25, 3-60, 3-61, 4-26, D-5  
 Cardinality i-20  
 Cartesian Product 5-2, 5-5, 5-26, 5-27, 5-28, 5-29, C-4, C-5, C-21  
 CASE Expression 3-51, 3-54, 3-55, 3-56, 3-57, 3-60  
 Case-manipulation Functions 3-7, 3-9, 3-10  
 CAT View 11-8  
 Character strings 1-16, 1-17, 1-18, 2-6, 2-15, 3-9, 3-11, 3-35, 3-38  
 Character-manipulation Functions 3-7, 3-11, 3-12  
 CHECK Constraint 8-7, 9-28, 10-21, 11-12, 11-13  
 COALESCE Function 3-52, 3-53, 4-5  
 Column Alias 1-8, 1-14, 1-15, 1-17, 1-37, 2-21, 3-46, 4-12, 5-11,  
 5-12, 9-33, 10-9, 10-11, C-11, C-12  
 COMMENT Statement 11-19  
 COMMENTS Column 11-6, 11-19  
 COMMIT Statement 8-29, 8-32, 8-33  
 Composite Unique Key 9-22, 9-23  
 CONCAT Function 1-16, 1-18, 1-42, 3-11-12, 3-38, 3-46, 3-60-61, 8-41, C-7, E-18  
 Concatenation Operator 1-16  
 Conditional Processing 3-54  
 CONSTRAINT\_TYPE 9-19, 11-13  
 CONSTRAINTS i-9, i-30, 1-12, 1-27, 8-7, 8-17, 8-21, 9-2, 9-10,  
 9-17-23, 9-26, 9-28, 9-30-32, 9-35, 9-36, 10-16, 10-38, 11-11-14,  
 11-20, 11-22, E-9  
 Conversion Functions 3-2, 3-6, 3-9, 3-26, 3-31, 3-60, 7-19  
 COUNT Function 4-8  
 CREATE INDEX Statement 10-36  
 CREATE PUBLIC SYNONYM Statement 10-41  
 CREATE SEQUENCE Statement 10-24  
 CREATE SYNONYM Statement 10-39, 10-40, 10-41  
 CREATE TABLE Statement 8-11, 9-5, 9-32, 9-36, 9-37  
 Creating a Database Connection 1-24, 1-25, 1-26, E-7, E-8  
 Creating a User-Defined Report E-21  
 Cross Joins 5-4, 5-28, 5-29  
 CURRVAL 9-7, 9-28, 10-26, 10-27, 10-28  
 CYCLE Option (with Sequences) 10-25, 10-30

**D**

Data from More Than One Table 5-2, 5-3, 5-5, C-2, C-3, C-7

Data Structures i-9, i-14, i-17, 9-3

Data Types i-6, i-9, 1-9, 1-35, 1-36, 3-3, 3-25, 3-26, 3-48,  
3-50, 3-60, 3-61, 4-5, 4-7, 5-6, 5-8, 7-8, 7-13, 7-15, 8-7,  
8-11, 9-2, 9-9-12, 9-14, 9-36, 11-10-12, D-7, D-8

Data Warehouse Applications i-9

Database I, 1-2, 1-3, 1-22-29, 1-34, 1-37, 1-39, 2-2, 2-6, 3-10, 3-17-20, 3-26,  
4-13, 4-16, 5-29, 6-7, 8-3, 8-13, 8-18, 8-24, 8-25, 8-29, 8-31, 8-32,  
8-37, 8-38, 8-39, 9-2-6, 9-13, 9-18, 9-33-37, 10-3, 10-16, 10-20, 10-22,  
10-23, 10-33, 10-34, 10-39, 10-41, 10-42, 11-3-12, 11-17-19, B-3,  
C-6, C-7, C-9, C-21, D-3-6, D-15, E

Database Structures 9-5

Date i-3, i-9, i-13, i-14, i-18, i-28, 1-8, 1-9, 1-17, 1-27, 1-40, 1-42, 2-6, 2-7, 2-10,  
2-11, 2-20, 2-21, 2-23, 2-26, 2-33, 2-34, 3-2-6, 3-14-27, 3-32-37, 3-41-48,  
3-60-65, 4-5, 4-7, 4-24, 5-35, 6-21, 7-4, 7-6, 7-19, 8-2, 8-3, 8-6, 8-8, 8-9,  
8-12-16, 8-22, 8-23, 8-27, 8-28, 8-33, 8-37-42, 9-7-17, 9-23, 9-26-30, 9-33,  
10-6, 10-7, 10-16, 10-17, 10-27, 10-37, 11-4, 11-7, 11-8, 11-15, C-27, E-9

DATE Data Type 3-6, 3-18, 3-22, 3-25, 3-61, 4-7, 7-19, 9-12

Datetime Data Type 9-11, 9-12, 9-14

DBA\_OBJECTS 11-5

DBMS i-2, i-13, i-14, i-21, i-23, i-26, i-27, i-30

DCL Statement 8-25, 8-29, 9-5

DDL Statement 8-11, 8-24, 8-25, 8-29, 8-36, 9-1, 9-5, 9-8, 9-35

DECODE Function 3-54, 3-56, 3-57, 3-58, 3-59, 3-60, 3-67

Default Date Display 2-6, 3-17

DEFAULT Option 9-7, 10-24, 10-25

Default Sort Order 2-21, 4-16

DELETE Statement 8-18, 8-19, 8-20, 8-21, 8-37, 8-39

DESC Keyword 2-21

DESCRIBE Command 1-35, 1-36, 8-7, 9-8, 9-33, 10-8, 10-12, 11-11, D-7

Dictionary Views i-3, 11-1, 11-2, 11-3, 11-6, 11-19, 11-20, 11-21

**D**

DISTINCT Keyword 1-20, 4-9, 7-11, 10-13, 10-14, 10-15, 10-27

DML Operations on Data Through a View 10-13

Double-ampersand 2-23, 2-28

DROP ANY INDEX Privilege 10-38

DROP ANY VIEW Privilege 10-20

DROP INDEX Statement 10-38

DROP SYNONYM Statement 10-41

DROP TABLE Statement 9-35

DROP VIEW Statement 10-20

DUAL Table 3-14, 3-19

Duplicate Rows i-23, 1-20, 4-8, 7-8, 7-11, 7-12, 7-18, 7-22

**E**

Entity Relationship i-17, i-19, i-20

Equijoins 5-2, 5-9, 5-19, 5-20, 5-29, C-2, C-8, C-9, C-14, C-15, C-21

ESCAPE Option 2-12

Execute SQL 1-2, 1-28, 1-37, D-2, D-5, D-15, E-2, E-12, E-22

Execution Plan 1-29, E-13, E-15

Explicit Data Type Conversion 3-26, 3-29, 3-30, 3-31

**F**

Foreign Key i-21, i-22, i-24, 5-9, 8-7, 9-17, 9-25, 9-26, 9-27,  
9-31, 10-35, 11-12, 11-13, C-7, C-8

Format Model 3-22, 3-24, 3-32, 3-33, 3-35, 3-37, 3-40, 3-41

FROM Clause 1-4, 1-9, 2-4, 3-14, 4-13, 4-16, 5-12, 6-4, C-9, C-12, C-20

FULL OUTER Join 5-5, 5-22, 5-25

Functions i-3, i-7, i-10, 1-23, 2-4, 2-6, 3-1-15, 3-22-26, 3-29, 3-31, 3-41, 3-42,  
3-45, 3-46, 3-47, 3-60, 3-61, 4-1-7, 4-10-13, 4-17, 4-18, 4-20, 4-23-26,  
6-10, 6-19, 7-19, 8-8, 9-7, 9-28, 10-6, 10-12-15, 11-3, 11-8, E-6, E-17, E-18

fx Modifier 3-41, 3-42

**G**

Generate Unique Numbers 10-3, 10-23

GROUP BY Clause 4-2, 4-11, 4-12, 4-13, 4-14, 4-16, 4-17, 4-20,  
4-21, 4-24, 6-12, 10-13, 10-14, 10-15

GROUP BY Column 4-14, 4-16

Group Functions 3-4, 4-1-13, 4-17, 4-18, 4-20, 4-23-26, 6-10, 6-19, 10-12-15

Group Functions in a Subquery 6-10

**H**

HAVING Clause 4-2, 4-18-25, 6-4, 6-11, 6-19

**I**

IF-THEN-ELSE Logic 3-54, 3-55, 3-57, 3-60

Implicit Data Type Conversion 3-26, 3-27, 3-28

IN Condition 2-7, 2-10, 5-2, 5-4, 5-13, 5-18, 5-19, 5-21, 5-22,  
5-26, 5-27, 10-37, C-2, C-4-7, C-13, C-14, C-16, C-17

Index 1-27, 9-3, 9-6, 9-23, 9-24, 9-35, 10-2, 10-3, 10-33, 10-34-38,  
10-42, 10-43, 10-46, 11-3, 11-7, 11-8, E-5, E-9

INSERT Statement 8-5, 8-11, 8-22, 8-23, 8-41, 9-8, 9-19, 10-27, 10-28

Integrity Constraint i-30, 8-7, 8-21, 9-17, 9-19, 9-22, 9-30,  
9-31, 9-32, 10-16, 11-3

International Standards Organization (ISO) i-28

INTERSECT Operator 7-3, 7-13, 7-14, 7-22

INTERVAL YEAR TO MONTH 9-11, 9-14, 9-15

IS NOT NULL Condition 2-13

IS NULL Condition 2-13

**J**

Java i-4, i-7, i-10, i-27, E-4

Joining Tables 5-5, 5-22, 5-30, C-7

**K**

Keywords 1-4, 1-7, 1-31, 1-32, 5-6, 9-26, 9-27, D-4, E-14, E-16

**L**

LEFT OUTER Join 5-23

LIKE Condition 2-11, 2-12

Literal 1-17, 1-18, 1-19, 2-4, 2-11, 2-12, 3-11, 3-35, 3-51,  
3-55, 7-20, 9-7

Logical Condition 2-14

Logical Subsets 10-4

**M**

MAX Function 4-6, 4-7

MIN Function 4-7

MINUS Operator 7-15, 7-16, 7-17, 7-22, 7-23

MOD Function 3-16

Modifier 3-41, 3-42

MONTHS\_BETWEEN Function 3-6

Multiple-column Subqueries 6-7

Multiple-row Functions 3-4

Multiple-row Subqueries 6-2, 6-6, 6-7, 6-14, 6-15, 6-16

**N**

Naming 1-33, 9-4, 9-18, 11-5

Nested Functions 3-45, 3-61

Nested SELECT 6-4, 6-20

NEXTVAL 9-7, 9-28, 10-26, 10-27, 10-28

NEXTVAL and CURRVAL Pseudocolumns 10-26, 10-27

Nonunique Index 10-35, 10-43, 10-46

NOT NULL Constraint 1-35, 9-19, 9-21, 9-22, 9-32, 11-10

NOT Operator 2-17, 2-31, 6-16

NULL Conditions 2-13, 2-31

Null Value 1-24, 1-12, 1-13, 1-16, 2-13, 2-21, 3-47-51, 3-57, 4-5, 4-8-10,  
6-13, 6-17, 7-8, 7-13, 8-7, 9-7, 9-21, 9-24, 10-37

NULLIF Function 3-51

NUMBER Data Type 3-38, 7-19, 8-6

Number Functions 3-6, 3-13

NVL Function 3-48, 3-49, 3-52, 3-60, 4-10

NVL2 Function 3-50

**O**

Object Relational i-2, i-9, i-13, i-30  
 Object Relational Database Management System i-2, i-30  
 Object-oriented Programming i-9  
 OLTP i-9  
 ON clause 5-5, 5-13-18, 5-22, 7-12, 10-16, 10-17  
 ON DELETE CASCADE 9-27  
 ON DELETE SET NULL 9-27  
 Online Transaction Processing i-9  
 OR REPLACE Option 10-8, 10-11  
 Oracle Application Server 10g i-5, i-7, i-30  
 Oracle Database 10g i-3, i-5, i-6, i-27, i-30, 6-7, 9-34  
 Oracle Enterprise Manager 10g Grid Control i-5, i-8, i-30  
 Oracle Instance i-27  
 ORDBMS i-2  
 Order i-18, i-23, 1-6, 1-11, 2-2, 2-18, 2-20, 2-21, 2-23, 2-27,  
     2-28, 2-31-34, 3-5, 3-46, 3-53, 3-63, 3-65, 3-66,  
     4-5, 4-12, 4-14, 4-16, 4-20, 4-22, 4-24, 4-27, 5-33, 6-6, 6-21,  
     7-2, 7-3, 7-8, 7-10, 7-12, 7-13, 7-17, 7-18, 7-21, 7-22, 7-24,  
     8-6, 9-10, 9-13, 10-11, 10-27, 11-8, 11-16, C-11, C-25  
 ORDER BY Clause 2-20, 2-21, 2-23, 2-31, 2-32, 3-5, 4-14, 4-24,  
     6-6, 7-17, 7-21, 7-22, 9-10, 10-27, C-11  
 Order of Precedence 1-11, 2-18  
 Order of Rows i-23, 2-20, 7-2, 7-21  
 Outer Query 6-3, 6-4, 6-5, 6-9, 6-10, 6-12, 6-13, 6-20

**P**

PRIMARY KEY Constraint 9-20, 9-24  
 Projection 1-3



**R**

RDBMS i-2, i-14, i-21, i-23, i-26, i-27, i-30  
 Read Consistency 8-31, 8-37, 8-38  
 READ ONLY Option 10-18  
 Read-only Constraint 10-19  
 REFERENCES 1-23, 1-27, 1-34, 8-29, 9-25-29, 11-13, E-6, E-9  
 Referential Integrity Constraint 8-21  
 Relational Database i  
 Relational Database Management System i-2, i-14, i-27, i-30  
 Restrict the Rows 1-3, 2-2, 2-4, 4-19, C-10  
 Retrieve Data from a View 10-10  
 Return a Value 3-3, 3-6, 3-14, 3-22  
 RIGHT OUTER Join 5-24  
 ROLLBACK Statement 8-2, 8-25-27, 8-29, 8-31, 8-34, 8-36, 8-39  
 ROUND and TRUNC Functions 3-24  
 ROUND Function 3-14, 3-15  
 RR Date Format 3-43, 3-44  
 Rules of Precedence 1-10, 1-11, 2-18, 2-19

**S**

SAVEPOINT Statement 8-28  
 Schema 1-22-27, 9-2, 9-5, 9-6, 9-19, 9-29, 9-36, 10-1, 10-34, 10-40,  
 11-2-4, 11-7, 11-21-23, B-3, E-3, E-7, E-9, E-10  
 SELECT Statement i-3, 1-1-6, 1-17, 1-18, 1-37, 1-39, 2-5, 2-8, 2-9, 2-11,  
 2-19, 2-20, 2-23, 2-27, 2-29, 2-31, 2-32, 3-2, 3-10, 4-8, 4-13-18,  
 5-2, 5-12, 6-2-9, 6-19, 6-20, 7-8, 7-13-20, 8-19, 8-31, 8-37, 8-38, 9-32,  
 10-4, 10-7, 10-12, 10-27, 11-6, 11-15, C-2, C-7, C-12, D-2  
 Selection 1-3, 2-3  
 Sequences 2-21, 9-6, 10-2, 10-3, 10-22, 10-23, 10-29, 10-38, 10-42,  
 10-43, 11-3, 11-16, 11-17, 11-20, 11-23  
 Set operators 7-1, 7-2, 7-3, 7-17, 7-18, 7-23, 7-24  
 SET VERIFY ON 2-30  
 Sets of Rows 4-3  
 SGA i-27

**S**

Single-row Functions 2-6, 3-1, 3-4, 3-5, 3-6, 3-45, 3-60, 4-3  
 Single-row Operator 6-4, 6-6, 6-8, 6-12, 6-14, 6-19  
 Single-row Subqueries 6-2, 6-6, 6-7, 6-8, 6-9  
 SOME Operator 6-15  
 Sorted i-3, 2-35, 7-8, 7-10, 7-11, 7-18, 7-21, 7-22  
 Sorting 2-1, 2-21, 2-31, 2-32  
 SQL Developer i-2, 1-2, 1-7, 1-8, 1-14, 1-20, 1-21, 1-22,  
 1-23, 1-24, 1-25, 1-27, 1-28, 1-32, 1-34, 1-35, 1-37, 1-38, 1-39,  
 1-42, 2-22-30, 8-25, 8-29, 8-30, E  
 Statement-level Rollback 8-36  
 Structured Query Language i-25, i-26, 1-2  
 Sub-SELECT 6-4  
 Subqueries in UPDATE statements 8-16  
 Subqueries to Delete Rows 8-20  
 Subquery 6-3-19, 8-11, 8-13, 8-15, 8-20, 8-22, 8-23, 9-10, 9-32, 9-33, 9-36,  
 10-7, 10-8, 10-9, 10-11, 10-27  
 Substitution Variables 2-22, 2-23, 2-26, 2-27, 2-30-32, 8-10, E-13  
 SUM Function 4-6, 4-16  
 Summary Results for Groups 4-16  
 Synonym i-19, i-20, 1-35, 6-15, 9-3, 9-6, 9-35, 10-2, 10-3, 10-39, 10-40-43,  
 10-46, 11-3, 11-8, 11-9, 11-18, 11-20, 11-21, 11-22, D-7  
 SYSDATE Function 3-18, 3-19, 8-8  
 System Development Life Cycle i-11, i-12, i-17  
 System Failure 8-24, 8-29, 8-30  
 System Global Area i-27

**T**

Table Alias 5-12, 5-20, 5-29, C-12, C-15, C-21  
 Table Prefixes 5-11, C-11, C-12  
 Three-way Join 5-18  
 TO\_CHAR Function 3-32, 3-37, 3-38, 3-39, 3-40  
 TO\_NUMBER or TO\_DATE Functions 3-41  
 Transactions 8-2, 8-24, 8-25, 8-27, 8-36, 8-40, 9-35  
 TRUNC Function 3-15, 3-24  
 Tuple i-23  
 Types of Indexes 10-35

**U**

UNION ALL Operator 7-11, 7-12, 7-18, 7-22  
 UNION Clause 7-12  
 UNION Operator 7-8, 7-9, 7-10, 7-19, 7-20, 7-21, 7-22, 7-23  
 UNIQUE Constraint 9-22, 9-23, 10-33, 10-35, 10-37, 11-13  
 Unique Identifier i-19, i-20  
 Unique Index 9-23, 9-24, 10-35, 10-37, 10-43, 10-46  
 UNIQUE Key Integrity Constraint 9-22  
 UPDATE Statement 8-13, 8-14, 8-15, 8-16, 10-27  
 UPPER Function 3-10  
 USER\_CONS\_COLUMNS Dictionary View 11-14  
 USER\_SYNONYMS Dictionary View 11-18  
 USING Clause 5-4, 5-8, 5-9, 5-10, 5-11, C-6  
 Using Snippets E-17, E-18  
 Using SQL Worksheet E-12, E-13

**V**

VALUES Clause 8-5, 8-7, 8-11, 8-22, 10-27

VARIANCE 4-4, 4-7, 4-24

VERIFY Command 2-30

Views i-3, i-8, 1-27, 9-6, 9-35, 10-2, 10-3, 10-4, 10-5, 10-6,  
10-13, 10-16, 10-20, 10-21, 10-38, 10-40, 10-42, 11-1, 11-2, 11-3, 11-4,  
11-5, 11-6, 11-7, 11-8, 11-15, 11-19, 11-20, 11-21, 11-23, E-9

**W**

When to Create an Index 10-37

WHERE Clause 2-3, 2-4, 2-5, 2-6, 2-7, 2-8, 2-10, 2-14, 2-22,  
2-23, 2-26, 2-27, 2-31, 2-32, 3-10, 4-8, 4-12, 4-13, 4-18, 4-19,  
4-24, 4-26, 5-7, 5-8, 5-13, 5-17, 5-29, 6-2, 6-4, 6-11, 6-12,  
6-13, 6-18, 7-17, 8-14, 8-19, 10-37, 11-4, C-4, C-7, C-9, C-10,  
C-11, C-17, C-20, C-21

Wildcard Search 2-11

WITH CHECK OPTION 10-7, 10-8, 10-16, 10-17, 11-13

**X**

XML i-4, i-6, i-27, 1-24, 1-25, E-7, E-8, E-20, E-21