

1

Retrieving Data Using the SQL `SELECT` Statement

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- List the capabilities of SQL `SELECT` statements
- Execute a basic `SELECT` statement

ORACLE

1 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

To extract data from the database, you need to use the SQL `SELECT` statement. However, you may need to restrict the columns that are displayed. This lesson describes all the SQL statements that are needed to perform these actions. Further, you may want to create `SELECT` statements that can be used more than once.

Lesson Agenda

- **Basic SELECT statement**
- Arithmetic expressions and NULL values in the SELECT statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

ORACLE

Capabilities of SQL SELECT Statements

Projection

Table 1

Selection

Table 1

Table 1

Join

Table 2

ORACLE

Capabilities of SQL SELECT Statements

A `SELECT` statement retrieves information from the database. With a `SELECT` statement, you can use the following capabilities:

- **Projection:** Select the columns in a table that are returned by a query. Select as few or as many of the columns as required.
- **Selection:** Select the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
- **Joining:** Bring together data that is stored in different tables by specifying the link between them. SQL joins are covered in more detail in the lesson titled “Displaying Data from Multiple Tables.”

Basic SELECT Statement

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

- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

ORACLE

1 - 5

Copyright © 2007, Oracle. All rights reserved.

Basic SELECT Statement

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

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

In the syntax:

SELECT	is a list of one or more columns
*	selects all columns
DISTINCT	suppresses duplicates
column expression	selects the named column or the expression
alias	gives the selected columns different headings
FROM table	specifies the table containing the columns

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

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

Selecting All Columns

```
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

ORACLE

1 - 6

Copyright © 2007, Oracle. All rights reserved.

Selecting All Columns

You can display all columns of data in a table by following the `SELECT` keyword with an asterisk (*). In the example in the slide, the department table contains four columns: `DEPARTMENT_ID`, `DEPARTMENT_NAME`, `MANAGER_ID`, and `LOCATION_ID`. The table contains eight rows, one for each department.

You can also display all columns in the table by listing all the columns after the `SELECT` keyword. For example, the following SQL statement (like the example in the slide) displays all columns and all rows of the `DEPARTMENTS` table:

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

Note: In SQL Developer, you can enter your SQL statement in a SQL Worksheet and click the “Execute Statement” icon or press [F9] to execute the statement. The output displayed in the Results tabbed page appears as shown in the slide.

Selecting Specific Columns

```
SELECT department_id, location_id
FROM 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

ORACLE

1 - 7

Copyright © 2007, Oracle. All rights reserved.

Selecting Specific Columns

You can use the `SELECT` statement to display specific columns of the table by specifying the column names, separated by commas. The example in the slide displays all the department numbers and location numbers from the `DEPARTMENTS` table.

In the `SELECT` clause, specify the columns that you want in the order in which you want them to appear in the output. For example, to display location before department number (from left to right), you use the following statement:

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

	LOCATION_ID	DEPARTMENT_ID
1	1700	10
2	1800	20
3	1500	50
4	1400	60

...

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can optionally be terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

ORACLE

1 - 8

Copyright © 2007, Oracle. All rights reserved.

Writing SQL Statements

By using the following simple rules and guidelines, you can construct valid statements that are both easy to read and edit:

- SQL statements are not case-sensitive (unless indicated).
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns names are entered in lowercase.

Executing SQL Statements

In SQL Developer, click the Run Script icon or press [F5] to run the command or commands in the SQL Worksheet. You can also click the Execute Statement icon or press [F9] to run a SQL statement in the SQL Worksheet. The Execute Statement icon executes the statement at the mouse pointer in the Enter SQL Statement box while the Run Script icon executes all the statements in the Enter SQL Statement box. The Execute Statement icon displays the output of the query on the Results tabbed page while the Run Script icon emulates the SQL*Plus display and shows the output on the Script Output tabbed page.

In SQL*Plus, terminate the SQL statement with a semicolon, and then press [Enter] to run the command.

Column Heading Defaults

- SQL Developer:
 - Default heading alignment: Left-aligned
 - Default heading display: Uppercase
- SQL*Plus:
 - Character and Date column headings are left-aligned.
 - Number column headings are right-aligned.
 - Default heading display: Uppercase

ORACLE

1 - 9

Copyright © 2007, Oracle. All rights reserved.

Column Heading Defaults

In SQL Developer, column headings are displayed in uppercase and are left-aligned.

```
SELECT last_name, hire_date, salary
FROM employees;
```

	LAST_NAME	HIRE_DATE	SALARY
1	King	17-JUN-87	24000
2	Kochhar	21-SEP-89	17000
3	De Haan	13-JAN-93	17000
4	Hunold	03-JAN-90	9000
5	Ernst	21-MAY-91	6000
6	Lorentz	07-FEB-99	4200
7	Mourgos	16-NOV-99	5800
8	Rajs	17-OCT-95	3500

...

You can override the column heading display with an alias. Column aliases are covered later in this lesson.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

ORACLE

1 - 11

Copyright © 2007, Oracle. All rights reserved.

Arithmetic Expressions

You may need to modify the way in which data is displayed, or you may want to perform calculations, or look at what-if scenarios. All these are possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

Arithmetic Operators

The slide lists the arithmetic operators that are available in SQL. You can use arithmetic operators in any clause of a SQL statement (except the FROM clause).

Note: With the DATE and TIMESTAMP data types, you can use the addition and subtraction operators only.

Using Arithmetic Operators

```
SELECT last_name, salary, salary + 300
FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

ORACLE

1 - 12

Copyright © 2007, Oracle. All rights reserved.

Using Arithmetic Operators

The example in the slide uses the addition operator to calculate a salary increase of \$300 for all employees. The slide also displays a `SALARY+300` column in the output.

Note that the resultant calculated column, `SALARY+300`, is not a new column in the `EMPLOYEES` table; it is for display only. By default, the name of a new column comes from the calculation that generated it—in this case, `salary+300`.

Note: The Oracle server ignores blank spaces before and after the arithmetic operator.

Operator Precedence

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators in an expression are of the same priority, then evaluation is done from left to right.

You can use parentheses to force the expression that is enclosed by the parentheses to be evaluated first.

Rules of Precedence:

- Multiplication and division occur before addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to override the default precedence or to clarify the statement.

Operator Precedence

```
SELECT last_name, salary, 12*salary+100  
FROM employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100

...

```
SELECT last_name, salary, 12*(salary+100)  
FROM employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200

...

ORACLE

1 - 13

Copyright © 2007, Oracle. All rights reserved.

Operator Precedence (continued)

The first example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation by multiplying the monthly salary with 12, plus a one-time bonus of \$100. Note that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression in the slide can be written as $(12*salary)+100$ with no change in the result.

Using Parentheses

You can override the rules of precedence by using parentheses to specify the desired order in which the operators are to be executed.

The second example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation as follows: adding a monthly bonus of \$100 to the monthly salary, and then multiplying that subtotal with 12. Because of the parentheses, addition takes priority over multiplication.

Defining a Null Value

- Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- Null is not the same as zero or a blank space.

```
SELECT last_name, job_id, salary, commission_pct
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
...				
19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

ORACLE

Defining a Null Value

If a row lacks a data value for a particular column, that value is said to be *null* or to contain a null.

Null is a value that is unavailable, unassigned, unknown, or inapplicable. Null is not the same as zero or a blank space. Zero is a number and blank space is a character.

Columns of any data type can contain nulls. However, some constraints (NOT NULL and PRIMARY KEY) prevent nulls from being used in the column.

In the `COMMISSION_PCT` column in the `EMPLOYEES` table, notice that only a sales manager or sales representative can earn a commission. Other employees are not entitled to earn commissions. A null represents that fact.

Note: By default, SQL Developer uses the literal, (null), to identify null values. However, you can set it to something more relevant to you. To do so, select Preferences from the Tools menu. In the Preferences dialog box, expand the Database node. Click Advanced Parameters and on the right pane, for the “Display Null value As,” enter the appropriate value.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name, 12*salary*commission_pct  
FROM employees;
```

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)
...		
12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640
...		
19	Higgins	(null)
20	Gietz	(null)

ORACLE

Null Values in Arithmetic Expressions

If any column value in an arithmetic expression is null, the result is null. For example, if you attempt to perform division by zero, you get an error. However, if you divide a number by null, the result is a null or unknown.

In the example in the slide, employee King does not get any commission. Because the `COMMISSION_PCT` column in the arithmetic expression is null, the result is null.

For more information, see the section on “Basic Elements of Oracle SQL” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- **Column aliases**
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Defining a Column Alias

A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name (There can also be the optional `AS` keyword between the column name and alias.)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

ORACLE

1 - 17

Copyright © 2007, Oracle. All rights reserved.

Defining a Column Alias

When displaying the result of a query, SQL Developer normally uses the name of the selected column as the column heading. This heading may not be descriptive and, therefore, may be difficult to understand. You can change a column heading by using a column alias.

Specify the alias after the column in the `SELECT` list using blank space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces or special characters (such as `#` or `$`), or if it is case-sensitive, enclose the alias in double quotation marks (“”).

Using Column Aliases

```
SELECT last_name AS name, commission_pct comm
FROM employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

```
SELECT last_name "Name", salary*12 "Annual Salary"
FROM employees;
```

	Name	Annual Salary
1	King	268000
2	Kochhar	204000
3	De Haan	204000

...

ORACLE

Using Column Aliases

The first example displays the names and the commission percentages of all the employees. Note that the optional `AS` keyword has been used before the column alias name. The result of the query is the same whether the `AS` keyword is used or not. Also, note that the SQL statement has the column aliases, `name` and `comm`, in lowercase, whereas the result of the query displays the column headings in uppercase. As mentioned in the previous slide, column headings appear in uppercase by default.

The second example displays the last names and annual salaries of all the employees. Because `Annual Salary` contains a space, it has been enclosed in double quotation marks. Note that the column heading in the output is exactly the same as the column alias.

Lesson Agenda

- Basic `SELECT` Statement
- Arithmetic Expressions and `NULL` values in `SELECT` statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Concatenation Operator

A concatenation operator:

- Links columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SELECT last_name||job_id AS "Employees"  
FROM employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP

...

ORACLE

1 - 20

Copyright © 2007, Oracle. All rights reserved.

Concatenation Operator

You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using the concatenation operator (||). Columns on either side of the operator are combined to make a single output column.

In the example, `LAST_NAME` and `JOB_ID` are concatenated, and given the alias `Employees`. Note that the last name of the employee and the job code are combined to make a single output column.

The `AS` keyword before the alias name makes the `SELECT` clause easier to read.

Null Values with the Concatenation Operator

If you concatenate a null value with a character string, the result is a character string. `LAST_NAME || NULL` results in `LAST_NAME`.

Note: You can also concatenate date expressions with other expressions or columns.

Literal Character Strings

- A literal is a character, a number, or a date that is included in the `SELECT` statement.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

ORACLE

1 - 21

Copyright © 2007, Oracle. All rights reserved.

Literal Character Strings

A literal is a character, a number, or a date that is included in the `SELECT` list. It is not a column name or a column alias. It is printed for each row returned. Literal strings of free-format text can be included in the query result and are treated the same as a column in the `SELECT` list.

Date and character literals *must* be enclosed within single quotation marks (' '); number literals need not be enclosed in a similar manner.

Using Literal Character Strings

```
SELECT last_name || ' is a ' || job_id  
       AS "Employee Details"  
FROM   employees;
```

	Employee Details
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
...	
18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

ORACLE

1 - 22

Copyright © 2007, Oracle. All rights reserved.

Using Literal Character Strings

The example in the slide displays the last names and job codes of all employees. The column has the heading Employee Details. Note the spaces between the single quotation marks in the SELECT statement. The spaces improve the readability of the output.

In the following example, the last name and salary for each employee are concatenated with a literal, to give the returned rows more meaning:

```
SELECT last_name || ': 1 Month salary = ' || salary Monthly  
FROM   employees;
```

	MONTHLY
1	King: 1 Month salary = 24000
2	Kochhar: 1 Month salary = 17000
3	De Haan: 1 Month salary = 17000
4	Hunold: 1 Month salary = 9000
5	Ernst: 1 Month salary = 6000
6	Lorentz: 1 Month salary = 4200
7	Mourgos: 1 Month salary = 5800
8	Rajs: 1 Month salary = 3500

...

Alternative Quote (q) Operator

- Specify your own quotation mark delimiter.
- Select any delimiter.
- Increase readability and usability.

```
SELECT department_name || ' Department' ||  
       q'['s Manager Id: ]'  
       || manager_id  
       AS "Department and Manager"  
FROM departments;
```

	Department and Manager
1	Administration Department's Manager Id:200
2	Marketing Department's Manager Id:201
3	Shipping Department's Manager Id:124
4	IT Department's Manager Id:103
5	Sales Department's Manager Id:149
6	Executive Department's Manager Id:100
7	Accounting Department's Manager Id:205
8	Contracting Department's Manager Id:

ORACLE

Alternative Quote (q) Operator

Many SQL statements use character literals in expressions or conditions. If the literal itself contains a single quotation mark, you can use the quote (q) operator and select your own quotation mark delimiter.

You can choose any convenient delimiter, single-byte or multibyte, or any of the following character pairs: [], { }, (), or <>.

In the example shown, the string contains a single quotation mark, which is normally interpreted as a delimiter of a character string. By using the q operator, however, brackets [] are used as the quotation mark delimiters. The string between the brackets delimiters is interpreted as a literal character string.

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

```
SELECT department_id
FROM employees;
```

1

	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60

...

```
SELECT DISTINCT department_id
FROM employees;
```

2

	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110

...

ORACLE

1 - 24

Copyright © 2007, Oracle. All rights reserved.

Duplicate Rows

Unless you indicate otherwise, SQL displays the results of a query without eliminating the duplicate rows. The first example in the slide displays all the department numbers from the EMPLOYEES table. Note that the department numbers are repeated.

To eliminate duplicate rows in the result, include the `DISTINCT` keyword in the `SELECT` clause immediately after the `SELECT` keyword. In the second example in the slide, the EMPLOYEES table actually contains 20 rows, but there are only seven unique department numbers in the table.

You can specify multiple columns after the `DISTINCT` qualifier. The `DISTINCT` qualifier affects all the selected columns, and the result is every distinct combination of the columns.

```
SELECT DISTINCT department_id, job_id
FROM employees;
```

	DEPARTMENT_ID	JOB_ID
1	110	AC_ACCOUNT
2	90	AD_VP
3	50	ST_CLERK
4	80	SA_REP
5	50	ST_MAN

...

Lesson Agenda

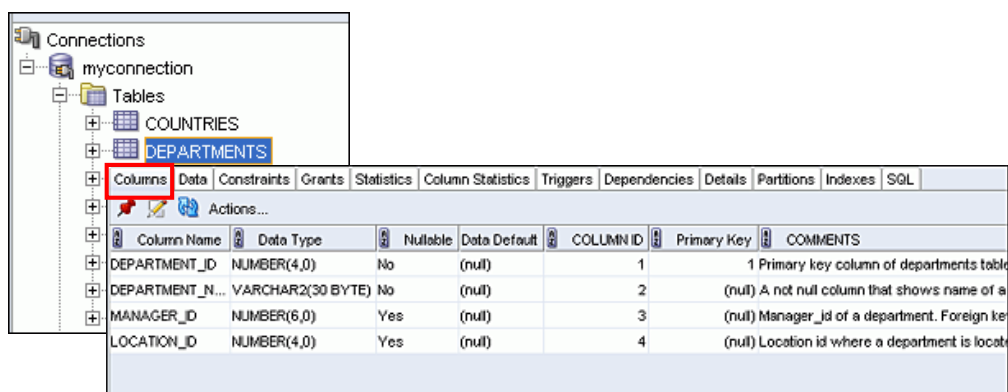
- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Displaying the Table Structure

- Use the `DESCRIBE` command to display the structure of a table.
- Or, select the table in the Connections tree and use the Columns tab to view the table structure.

```
DESC[RIBE] tablename
```



Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	1	Primary key column of departments table
DEPARTMENT_N...	VARCHAR2(30 BYTE)	No	(null)	2		(null) A not null column that shows name of a
MANAGER_ID	NUMBER(6,0)	Yes	(null)	3		(null) Manager_id of a department. Foreign ke
LOCATION_ID	NUMBER(4,0)	Yes	(null)	4		(null) Location id where a department is locot

ORACLE

1 - 26

Copyright © 2007, Oracle. All rights reserved.

Displaying the Table Structure

In SQL Developer, you can display the structure of a table by using the `DESCRIBE` command. The command displays the column names and the data types, and it shows you whether a column *must* contain data (that is, whether the column has a `NOT NULL` constraint).

In the syntax, *table name* is the name of any existing table, view, or synonym that is accessible to the user.

Using the SQL Developer GUI interface, you can select the table in the Connections tree and use the Columns tab to view the table structure.

Note: The `DESCRIBE` command is supported by both SQL*Plus and SQL Developer.

Using the DESCRIBE Command

```
DESCRIBE employees
```

```
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
```

ORACLE

1 - 27

Copyright © 2007, Oracle. All rights reserved.

Using the DESCRIBE Command

The example in the slide displays information about the structure of the EMPLOYEES table using the DESCRIBE command.

In the resulting display, *Null* indicates that the values for this column may be unknown. NOT NULL indicates that a column must contain data. *Type* displays the data type for a column.

The data types are described in the following table:

Data Type	Description
NUMBER (<i>p</i> , <i>s</i>)	Number value having a maximum number of digits <i>p</i> , with <i>s</i> digits to the right of the decimal point
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, A.D. 9999.
CHAR (<i>s</i>)	Fixed-length character value of size <i>s</i>

Summary

In this lesson, you should have learned how to:

- Write a `SELECT` statement that:
 - Returns all rows and columns from a table
 - Returns specified columns from a table
 - Uses column aliases to display more descriptive column headings

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

ORACLE

1 - 28

Copyright © 2007, Oracle. All rights reserved.

SELECT Statement

In this lesson, you should have learned how to retrieve data from a database table with the `SELECT` statement.

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

In the syntax:

<code>SELECT</code>	is a list of one or more columns
<code>*</code>	selects all columns
<code>DISTINCT</code>	suppresses duplicates
<code>column expression</code>	selects the named column or the expression
<code>alias</code>	gives the selected columns different headings
<code>FROM table</code>	specifies the table containing the columns

Practice 1: Overview

This practice covers the following topics:

- Selecting all data from different tables
- Describing the structure of tables
- Performing arithmetic calculations and specifying column names

ORACLE

1 - 29

Copyright © 2007, Oracle. All rights reserved.

Practice 1: Overview

In this practice, you write simple `SELECT` queries. The queries cover most of the `SELECT` clauses and operations that you learned in this lesson.

Practice 1

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 the following statement. Can you identify them?

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

Part 2

Note the following points before you begin with the practices:

- Save all your lab files at the following location: D:\labs\SQL1\labs
- Enter your SQL statements in a SQL Worksheet. To save a script in SQL Developer, make sure the required SQL worksheet is active and then from the File menu, select Save As or right-click in the SQL Worksheet and select Save file to save your SQL statement as a lab_<lessonno>_<stepno>.sql script. When you are modifying an existing script, make sure you use Save As to save it with a different filename.
- To run the query, click the Execute Statement icon in the SQL Worksheet. Alternatively, you can press [F9]. For DML and DDL statements, use the Run Script icon or press [F5].
- After you have executed the query, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

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.

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

```
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

Practice 1 (continued)

	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

5. You need to determine the structure of the 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
```

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_05.sql so that you can dispatch this file to the HR department.

Practice 1 (continued)

6. Test your query in the lab_01_05.sql file to ensure that it runs correctly.

Note: After you have executed the query, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

	EMPLOYEE_ID	LAST_NAME	JOB_ID	STARTDATE
1	100	King	AD_PRES	17-JUN-87
2	101	Kochhar	AD_VP	21-SEP-89
3	102	De Haan	AD_VP	13-JAN-93
4	103	Hunold	IT_PROG	03-JAN-90
5	104	Ernst	IT_PROG	21-MAY-91
6	107	Lorentz	IT_PROG	07-FEB-99
7	124	Mourgos	ST_MAN	16-NOV-99
8	141	Rajs	ST_CLERK	17-OCT-95
9	142	Davies	ST_CLERK	29-JAN-97
10	143	Matos	ST_CLERK	15-MAR-98

...

19	205	Higgins	AC_MGR	07-JUN-94
20	206	Gietz	AC_ACCOUNT	07-JUN-94

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

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	IT_PROG
7	MK_MAN
8	MK_REP
9	SA_MAN
10	SA_REP
11	ST_CLERK
12	ST_MAN

Practice 1 (continued)

Part 3

If you have time, complete the following exercises:

- The HR department wants more descriptive column headings for its report on employees. Copy the statement from `lab_01_05.sql` to a new SQL Worksheet. Name the column headings `Emp #`, `Employee`, `Job`, and `Hire Date`, respectively. Then run your query again.

	Emp #	Employee	Job	Hire Date
1	100	King	AD_PRES	17-JUN-87
2	101	Kochhar	AD_VP	21-SEP-89
3	102	De Haan	AD_VP	13-JAN-93
4	103	Hunold	IT_PROG	03-JAN-90
5	104	Ernst	IT_PROG	21-MAY-91
6	107	Lorentz	IT_PROG	07-FEB-99
7	124	Mourgos	ST_MAN	16-NOV-99
8	141	Rajs	ST_CLERK	17-OCT-95
9	142	Davies	ST_CLERK	29-JAN-97
10	143	Matos	ST_CLERK	15-MAR-98

...

19	205	Higgins	AC_MGR	07-JUN-94
20	206	Gietz	AC_ACCOUNT	07-JUN-94

- 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`.

	Employee and Title
1	Abel, SA_REP
2	Davies, ST_CLERK
3	De Haan, AD_VP
4	Ernst, IT_PROG
5	Fay, MK_REP
6	Gietz, AC_ACCOUNT
7	Grant, SA_REP
8	Hartstein, MK_MAN
9	Higgins, AC_MGR
10	Hunold, IT_PROG

...

19	Whalen, AD_ASST
20	Zlotkey, SA_MAN

Practice 1 (continued)

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

- To familiarize yourself with the data in the EMPLOYEES table, create a query to display all the data from that table. Separate each column output by a comma. Name the column title THE_OUTPUT.

	THE_OUTPUT
1	100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90
2	101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90
3	102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90
4	103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60
5	104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60
6	107,Diana,Lorentz,DLORENTZ,590.423.5567,IT_PROG,103,07-FEB-99,4200,,60
7	124,Kevin,Mourgos,KMOURGOS,650.123.5234,ST_MAN,100,16-NOV-99,5800,,50
8	141,Trenna,Rajs,TRAJS,650.121.8009,ST_CLERK,124,17-OCT-95,3500,,50
9	142,Curtis,Davies,CDAVIES,650.121.2994,ST_CLERK,124,29-JAN-97,3100,,50
10	143,Randall,Matos,RMATOS,650.121.2874,ST_CLERK,124,15-MAR-98,2600,,50

...

19	205,Shelley,Higgins,SHIGGINS,515.123.8080,AC_MGR,101,07-JUN-94,12000,,110
20	206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110

2

Restricting and Sorting Data

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution to restrict and sort output at run time

ORACLE

2 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

When retrieving data from the database, you may need to do the following:

- Restrict the rows of data that are displayed
- Specify the order in which the rows are displayed

This lesson explains the SQL statements that you use to perform the actions listed above.

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` conditions
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE


Limiting Rows Using a Selection

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

“retrieve all employees in department 90”



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Limiting Rows Using a Selection

In the example in the slide, assume that you want to display all the employees in department 90. The rows with a value of 90 in the `DEPARTMENT_ID` column are the only ones that are returned. This method of restriction is the basis of the `WHERE` clause in SQL.

Limiting the Rows that Are Selected

- Restrict the rows that are returned by using the `WHERE` clause:

```
SELECT *|{[DISTINCT] column|expression [alias],...}  
FROM table  
[WHERE condition(s)];
```

- The `WHERE` clause follows the `FROM` clause.

ORACLE

2 - 5

Copyright © 2007, Oracle. All rights reserved.

Limiting the Rows that Are Selected

You can restrict the rows that are returned from the query by using the `WHERE` clause. A `WHERE` clause contains a condition that must be met and it directly follows the `FROM` clause. If the condition is true, the row meeting the condition is returned.

In the syntax:

`WHERE` restricts the query to rows that meet a condition

`condition` is composed of column names, expressions, constants, and a comparison operator. A condition specifies a combination of one or more expressions and logical (Boolean) operators, and returns a value of `TRUE`, `FALSE`, or `UNKNOWN`.

The `WHERE` clause can compare values in columns, literal, arithmetic expressions, or functions. It consists of three elements:

- Column name
- Comparison condition
- Column name, constant, or list of values

Using the WHERE Clause

```
SELECT employee_id, last_name, job_id, department_id
FROM employees
WHERE department_id = 90 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

Using the WHERE Clause

In the example, the SELECT statement retrieves the employee ID, last name, job ID, and department number of all employees who are in department 90.

Note: You cannot use column alias in the WHERE clause.

Character Strings and Dates

- Character strings and date values are enclosed with single quotation marks.
- Character values are case-sensitive and date values are format-sensitive.
- The default date display format is DD-MON-RR.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-FEB-96' ;
```

ORACLE

2 - 7

Copyright © 2007, Oracle. All rights reserved.

Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed with single quotation marks (' '). Number constants, however, should not be enclosed with single quotation marks.

All character searches are case-sensitive. In the following example, no rows are returned because the EMPLOYEES table stores all the last names in mixed case:

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'WHALEN' ;
```

Oracle databases store dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds. The default date display is in the DD-MON-RR format.

Note: For details about the RR format and about changing the default date format, see the lesson titled “Using Single-Row Functions to Customize Output.” Also, you learn about the use of single-row functions such as UPPER and LOWER to override the case sensitivity in the same lesson.

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN (set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Comparison Operators

Comparison operators are used in conditions that compare one expression to another value or expression. They are used in the WHERE clause in the following format:

Syntax

```
... WHERE expr operator value
```

Example

```
... WHERE hire_date = '01-JAN-95'  
... WHERE salary >= 6000  
... WHERE last_name = 'Smith'
```

An alias cannot be used in the WHERE clause.

Note: The symbols != and ^= can also represent the *not equal to* condition.

Using Comparison Operators

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

ORACLE

2 - 9

Copyright © 2007, Oracle. All rights reserved.

Using Comparison Operators

In the example, the `SELECT` statement retrieves the last name and salary from the `EMPLOYEES` table for any employee whose salary is less than or equal to \$3,000. Note that there is an explicit value supplied to the `WHERE` clause. The explicit value of 3000 is compared to the salary value in the `SALARY` column of the `EMPLOYEES` table.

Range Conditions Using the BETWEEN Operator

Use the BETWEEN operator to display rows based on a range of values:

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500 ;
```

Lower limit

Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

ORACLE

2 - 10

Copyright © 2007, Oracle. All rights reserved.

Range Conditions Using the BETWEEN Operator

You can display rows based on a range of values using the BETWEEN operator. The range that you specify contains a lower limit and an upper limit.

The SELECT statement in the slide returns rows from the EMPLOYEES table for any employee whose salary is between \$2,500 and \$3,500.

Values that are specified with the BETWEEN operator are inclusive. However, you must specify the lower limit first.

You can also use the BETWEEN operator on character values:

```
SELECT last_name
FROM employees
WHERE last_name BETWEEN 'King' AND 'Smith';
```

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos
6	Rajs

Membership Condition Using the IN Operator

Use the IN operator to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	Kochhar	17000	100
2	De Haan	17000	100
3	Mourgos	5800	100
4	Zlotkey	10500	100
5	Hartstein	13000	100
6	Whalen	4400	101
7	Higgins	12000	101
8	Fay	6000	201

ORACLE

2 - 11

Copyright © 2007, Oracle. All rights reserved.

Membership Condition Using the IN Operator

To test for values in a specified set of values, use the IN operator. The condition defined using the IN operator is also known as the *membership condition*.

The slide example displays employee numbers, last names, salaries, and managers' employee numbers for all the employees whose manager's employee number is 100, 101, or 201.

The IN operator can be used with any data type. The following example returns a row from the EMPLOYEES table, for any employee whose last name is included in the list of names in the WHERE clause:

```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

If characters or dates are used in the list, they must be enclosed with single quotation marks ('').

Note: The IN operator is internally evaluated by the Oracle server as a set of OR conditions, such as a=value1 or a=value2 or a=value3. Therefore, using the IN operator has no performance benefits and is used only for logical simplicity.

Pattern Matching Using the LIKE Operator

- Use the LIKE operator to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
 - % denotes zero or many characters.
 - _ denotes one character.

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

ORACLE

2 - 12

Copyright © 2007, Oracle. All rights reserved.

Pattern Matching Using the LIKE Operator

You may not always know the exact value to search for. You can select rows that match a character pattern by using the LIKE operator. The character pattern–matching operation is referred to as a *wildcard* search. Two symbols can be used to construct the search string.

Symbol	Description
%	Represents any sequence of zero or more characters
_	Represents any single character

The SELECT statement in the slide returns the first name from the EMPLOYEES table for any employee whose first name begins with the letter “S.” Note the uppercase “S.” Consequently, names beginning with a lowercase “s” are not returned.

The LIKE operator can be used as a shortcut for some BETWEEN comparisons. The following example displays the last names and hire dates of all employees who joined between January, 1995 and December, 1995:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%95';
```

Combining Wildcard Characters

- You can combine the two wildcard characters (% , _) with literal characters for pattern matching:

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

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

- You can use the `ESCAPE` identifier to search for the actual % and _ symbols.

ORACLE

2 - 13

Copyright © 2007, Oracle. All rights reserved.

Combining Wildcard Characters

The % and _ symbols can be used in any combination with literal characters. The example in the slide displays the names of all employees whose last names have the letter “o” as the second character.

ESCAPE Identifier

When you need to have an exact match for the actual % and _ characters, use the `ESCAPE` identifier. This option specifies what the escape character is. If you want to search for strings that contain SA_, you can use the following SQL statement:

```
SELECT employee_id, last_name, job_id
FROM employees WHERE job_id LIKE '%SA\_%' ESCAPE '\\';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	149	Zlotkey	SA_MAN
2	174	Abel	SA_REP
3	176	Taylor	SA_REP
4	178	Grant	SA_REP

The `ESCAPE` identifier identifies the backslash (\) as the escape character. In the SQL statement, the escape character precedes the underscore (_). This causes the Oracle server to interpret the underscore literally.

Using the NULL Conditions

Test for nulls with the `IS NULL` operator.

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

	LAST_NAME	MANAGER_ID
1	King	(null)

ORACLE

2 - 14

Copyright © 2007, Oracle. All rights reserved.

Using the NULL Conditions

The NULL conditions include the `IS NULL` condition and the `IS NOT NULL` condition.

The `IS NULL` condition tests for nulls. A null value means that the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with `=`, because a null cannot be equal or unequal to any value. The slide example retrieves the last names and managers of all employees who do not have a manager.

Here is another example: To display the last name, job ID, and commission for all employees who are *not* entitled to receive a commission, use the following SQL statement:

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

	LAST_NAME	JOB_ID	COMMISSION_PCT
1	King	AD_PRES	(null)
2	Kochhar	AD_VP	(null)

...

15	Higgins	AC_MGR	(null)
16	Gietz	AC_ACCOUNT	(null)

Defining Conditions Using the Logical Operators

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are true
OR	Returns TRUE if <i>either</i> component condition is true
NOT	Returns TRUE if the condition is false

ORACLE

2 - 15

Copyright © 2007, Oracle. All rights reserved.

Defining Conditions Using the Logical Operators

A logical condition combines the result of two component conditions to produce a single result based on those conditions or it inverts the result of a single condition. A row is returned only if the overall result of the condition is true.

Three logical operators are available in SQL:

- AND
- OR
- NOT

All the examples so far have specified only one condition in the WHERE clause. You can use several conditions in a single WHERE clause using the AND and OR operators.

Using the AND Operator

AND requires both the component conditions to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
AND    job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149 Zlotkey	SA_MAN	10500
2	201 Hartstein	MK_MAN	13000

ORACLE

2 - 16

Copyright © 2007, Oracle. All rights reserved.

Using the AND Operator

In the example, both the component conditions must be true for any record to be selected. Therefore, only those employees who have a job title that contains the string 'MAN' and earn \$10,000 or more are selected.

All character searches are case-sensitive, that is no rows are returned if 'MAN' is not uppercase. Further, character strings must be enclosed with quotation marks.

AND Truth Table

The following table shows the results of combining two expressions with AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Using the OR Operator

OR requires either component condition to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12000

ORACLE

2 - 17

Copyright © 2007, Oracle. All rights reserved.

Using the OR Operator

In the example, either component condition can be true for any record to be selected. Therefore, any employee who has a job ID that contains the string 'MAN' *or* earns \$10,000 or more is selected.

OR Truth Table

The following table shows the results of combining two expressions with OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Using the NOT Operator

```
SELECT last_name, job_id
FROM employees
WHERE job_id
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

ORACLE

2 - 18

Copyright © 2007, Oracle. All rights reserved.

Using the NOT Operator

The slide example displays the last name and job ID of all employees whose job ID *is not* IT_PROG, ST_CLERK, or SA_REP.

NOT Truth Table

The following table shows the result of applying the NOT operator to a condition:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Note: The NOT operator can also be used with other SQL operators, such as BETWEEN, LIKE, and NULL.

```
... WHERE job_id NOT IN ('AC_ACCOUNT', 'AD_VP')
... WHERE salary NOT BETWEEN 10000 AND 15000
... WHERE last_name NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL
```

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Rules of Precedence

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

You can use parentheses to override rules of precedence.

ORACLE

2 - 20

Copyright © 2007, Oracle. All rights reserved.

Rules of Precedence

The rules of precedence determine the order in which expressions are evaluated and calculated. The table in the slide lists the default order of precedence. However, you can override the default order by using parentheses around the expressions that you want to calculate first.

Rules of Precedence

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

1

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

2

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

ORACLE

2 - 21

Copyright © 2007, Oracle. All rights reserved.

Rules of Precedence (continued)

1. Precedence of the AND Operator: Example

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *and* the salary is greater than \$15,000.
- The second condition is that the job ID is SA_REP.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee is a president *and* earns more than \$15,000, *or* if the employee is a sales representative.”

2. Using Parentheses: Example

In this example, there are two conditions:

- The first condition is that the job ID is AD_PRES *or* SA_REP.
- The second condition is that the salary is greater than \$15,000.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee is a president *or* a sales representative, *and* if the employee earns more than \$15,000.”

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- **Sorting rows using the `ORDER BY` clause**
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
 - ASC: Ascending order, default
 - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES	90	17-JUN-87
2	Whalen	AD_ASST	10	17-SEP-87
3	Kochhar	AD_VP	90	21-SEP-89
4	Hunold	IT_PROG	60	03-JAN-90
5	Ernst	IT_PROG	60	21-MAY-91
6	De Haan	AD_VP	90	13-JAN-93

...

ORACLE

2 - 23

Copyright © 2007, Oracle. All rights reserved.

Using the ORDER BY Clause

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. However, if you use the ORDER BY clause, it must be the last clause of the SQL statement. Further, you can specify an expression, an alias, or a column position as the sort condition.

Syntax

```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY      {column, expr, numeric_position} [ASC|DESC]];
```

In the syntax:

ORDER BY specifies the order in which the retrieved rows are displayed
ASC orders the rows in ascending order (this is the default order)
DESC orders the rows in descending order

If the ORDER BY clause is not used, the sort order is undefined, and the Oracle server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

Note: Use the keywords NULLS FIRST or NULLS LAST to specify whether returned rows containing null values should appear first or last in the ordering sequence.

Sorting

- Sorting in descending order:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC ;
```

1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM employees
ORDER BY annsal ;
```

2

ORACLE

2 - 24

Copyright © 2007, Oracle. All rights reserved.

Sorting

The default sort order is ascending:

- Numeric values are displayed with the lowest values first (for example, 1 to 999).
- Date values are displayed with the earliest value first (for example, 01-JAN-92 before 01-JAN-95).
- Character values are displayed in the alphabetical order (for example, “A” first and “Z” last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can also sort by a column that is not in the SELECT list.

Examples:

1. To reverse the order in which the rows are displayed, specify the DESC keyword after the column name in the ORDER BY clause. The slide example sorts the result by the most recently hired employee.
2. You can also use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.

Sorting

- Sorting by using the column's numeric position:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```

3

- Sorting by multiple columns:

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

4

ORACLE

Sorting (continued)

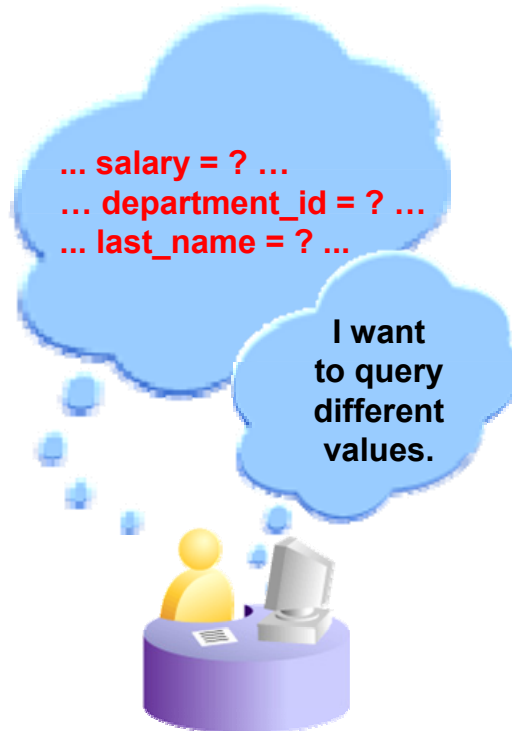
Examples:

3. You can sort query results by specifying the numeric position of the column in the `SELECT` clause. The slide example sorts the result by the `department_id` as this column is at the third position in the `SELECT` clause.
4. You can sort query results by more than one column. The sort limit is the number of columns in the given table. In the `ORDER BY` clause, specify the columns and separate the column names using commas. If you want to reverse the order of a column, specify `DESC` after its name.

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- **Substitution variables**
- `DEFINE` and `VERIFY` commands

Substitution Variables



ORACLE

2 - 27

Copyright © 2007, Oracle. All rights reserved.

Substitution Variables

So far, all the SQL statements were executed with predetermined columns, conditions and their values. Suppose that you want a query that lists the employees with various jobs and not just those whose `job_id` is `SA_REP`. You can edit the `WHERE` clause to provide a different value each time you run the command, but there is also an easier way.

By using a substitution variable in place of the exact values in the `WHERE` clause, you can run the same query for different values.

You can create reports that prompt users to supply their own values to restrict the range of data returned, by using substitution variables. You can embed *substitution variables* in a command file or in a single SQL statement. A variable can be thought of as a container in which values are temporarily stored. When the statement is run, the stored value is substituted.

Substitution Variables

- Use substitution variables to:
 - Temporarily store values with single-ampersand (&) and double-ampersand (&&) substitution
- Use substitution variables to supplement the following:
 - WHERE conditions
 - ORDER BY clauses
 - Column expressions
 - Table names
 - Entire SELECT statements

ORACLE

2 - 28

Copyright © 2007, Oracle. All rights reserved.

Substitution Variables (continued)

You can use single-ampersand (&) substitution variables to temporarily store values.

You can also predefine variables by using the DEFINE command. DEFINE creates and assigns a value to a variable.

Restricted Ranges of Data: Examples

- Reporting figures only for the current quarter or specified date range
- Reporting on data relevant only to the user requesting the report
- Displaying personnel only within a given department

Other Interactive Effects

Interactive effects are not restricted to direct user interaction with the WHERE clause. The same principles can also be used to achieve other goals, such as:

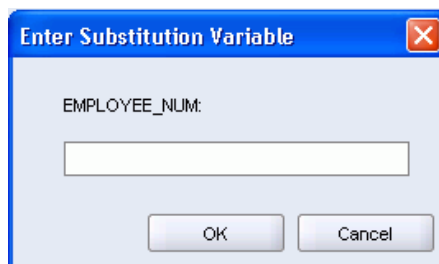
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

Note: Both SQL Developer and SQL* Plus support the substitution variables and the DEFINE/UNDEFINE commands. Though SQL Developer or SQL* Plus does not support validation checks (except for data type) on user input.

Using the Single-Ampersand Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value:

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ;
```



ORACLE

2 - 29

Copyright © 2007, Oracle. All rights reserved.

Using the Single-Ampersand Substitution Variable

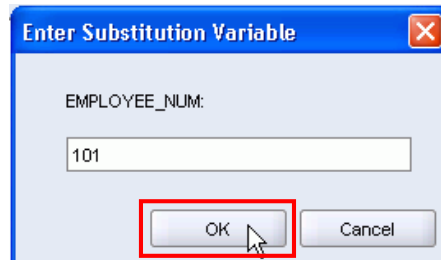
When running a report, users often want to restrict the data that is returned dynamically. SQL*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. However, you do not need to define the value of each variable.

Notation	Description
<i>&user_variable</i>	Indicates a variable in a SQL statement; if the variable does not exist, SQL*Plus or SQL Developer prompts the user for a value (the new variable is discarded after it is used.)

The example in the slide creates a SQL Developer substitution variable for an employee number. When the statement is executed, SQL Developer prompts the user for an employee number and then displays the employee number, last name, salary, and department number for that employee.

With the single ampersand, the user is prompted every time the command is executed if the variable does not exist.

Using the Single-Ampersand Substitution Variable



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE

2 - 30

Copyright © 2007, Oracle. All rights reserved.

Using the Single-Ampersand Substitution Variable (continued)

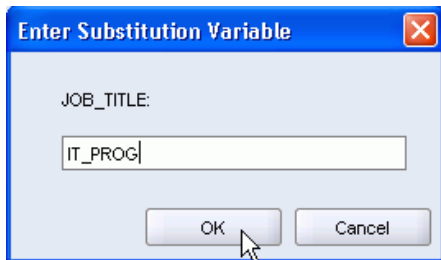
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

After you enter a value and click the OK button, the results are displayed in the Results tab of your SQL Developer session.

Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

```
SELECT last_name, department_id, salary*12
FROM employees
WHERE job_id = '&job_title' ;
```



	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Lorentz	60	50400

ORACLE

Character and Date Values with Substitution Variables

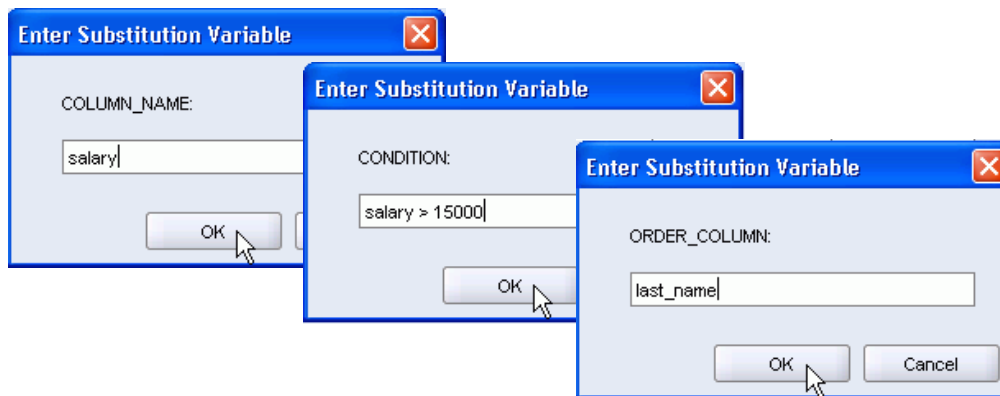
In a `WHERE` clause, date and character values must be enclosed with single quotation marks. The same rule applies to the substitution variables.

Enclose the variable with single quotation marks within the SQL statement itself.

The slide shows a query to retrieve the employee names, department numbers, and annual salaries of all employees based on the job title value of the SQL Developer substitution variable.

Specifying Column Names, Expressions, and Text

```
SELECT employee_id, last_name, job_id, &column_name  
FROM employees  
WHERE &condition  
ORDER BY &order_column ;
```



ORACLE

2 - 32

Copyright © 2007, Oracle. All rights reserved.

Specifying Column Names, Expressions, and Text

You can use the substitution variables not only in the `WHERE` clause of a SQL statement, but also as substitution for column names, expressions, or text.

Example:

The slide example displays the employee number, last name, job title, and any other column that is specified by the user at run time, from the `EMPLOYEES` table. For each substitution variable in the `SELECT` statement, you are prompted to enter a value, and then click `OK` to proceed.

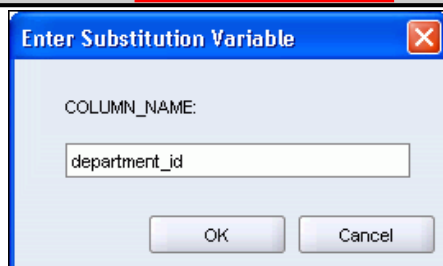
If you do not enter a value for the substitution variable, you get an error when you execute the preceding statement.

Note: A substitution variable can be used anywhere in the `SELECT` statement, except as the first word entered at the command prompt.

Using the Double-Ampersand Substitution Variable

Use double ampersand (&&) if you want to reuse the variable value without prompting the user each time:

```
SELECT employee_id, last_name, job_id, &&column_name
FROM employees
ORDER BY &column_name ;
```



A dialog box titled "Enter Substitution Variable" with a close button (X) in the top right corner. It contains a label "COLUMN_NAME:" followed by a text input field containing the value "department_id". Below the input field are two buttons: "OK" and "Cancel".

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200 Whalen	AD_ASST	10
2	201 Hartstein	MK_MAN	20
3	202 Fay	MK_REP	20

...

ORACLE

2 - 33

Copyright © 2007, Oracle. All rights reserved.

Using the Double-Ampersand Substitution Variable

You can use the double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time. The user sees the prompt for the value only once. In the example in the slide, the user is asked to give the value for the variable, `column_name`, only once. The value that is supplied by the user (`department_id`) is used for both display and ordering of data. If you run the query again, you will not be prompted for the value of the variable.

SQL Developer stores the value that is supplied by using the `DEFINE` command; it uses it again whenever you reference the variable name. After a user variable is in place, you need to use the `UNDEFINE` command to delete it:

```
UNDEFINE column_name
```

Lesson Agenda

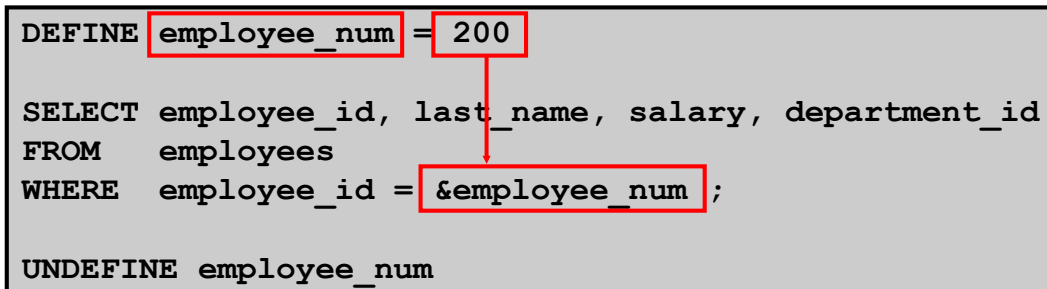
- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Using the DEFINE Command

- Use the DEFINE command to create and assign a value to a variable.
- Use the UNDEFINE command to remove a variable.

```
DEFINE employee_num = 200
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ;
UNDEFINE employee_num
```



Using the DEFINE Command

The example shown creates a substitution variable for an employee number by using the DEFINE command. At run time, this displays the employee number, name, salary, and department number for that employee.

Because the variable is created using the SQL Developer DEFINE command, the user is not prompted to enter a value for the employee number. Instead, the defined variable value is automatically substituted in the SELECT statement.

The EMPLOYEE_NUM substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

Using the VERIFY Command

Use the `VERIFY` command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values:

The screenshot illustrates the process of using the `VERIFY` command in SQL Developer. At the top, a code editor shows the following SQL statement with `SET VERIFY ON` highlighted in red:

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = &employee_num;
```

Below the code editor, an "Enter Substitution Variable" dialog box is open, showing the variable `EMPLOYEE_NUM` with the value `200` entered. The `OK` button is highlighted.

To the right, the "Script Output" tab is active, showing the SQL statement after substitution. The `WHERE` clause is highlighted in red, showing the variable replaced by its value:

```
SELECT employee_id, last_name, salary
FROM employees
WHERE employee id = 200
```

Below the SQL statement, the output is displayed as a table:

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400

The output also indicates "1 rows selected".

2 - 36

Copyright © 2007, Oracle. All rights reserved.

ORACLE

Using the VERIFY Command

To confirm the changes in the SQL statement, use the `VERIFY` command. Setting `SET VERIFY ON` forces SQL Developer to display the text of a command after it replaces substitution variables with values. To see the `VERIFY` output, you should use the Run Script (F5) icon in the SQL Worksheet. SQL Developer displays the text of a command after it replaces substitution variables with values, in the Script Output tab as shown in the slide.

The example in the slide displays the new value of the `EMPLOYEE_ID` column in the SQL statement followed by the output.

SQL*Plus System Variables

SQL*Plus uses various system variables that control the working environment. One of the variables is `VERIFY`. To obtain a complete list of all the system variables, you can issue the `SHOW ALL` command on the SQL*Plus command prompt.

Summary

In this lesson, you should have learned how to:

- Use the `WHERE` clause to restrict rows of output:
 - Use the comparison conditions
 - Use the `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Apply the logical `AND`, `OR`, and `NOT` operators
- Use the `ORDER BY` clause to sort rows of output:

```
SELECT *|{[DISTINCT] column|expression [alias],...}  
FROM table  
[WHERE condition(s)]  
[ORDER BY {column, expr, alias} [ASC|DESC]] ;
```

- Use ampersand substitution to restrict and sort output at run time

ORACLE

Summary

In this lesson, you should have learned about restricting and sorting rows that are returned by the `SELECT` statement. You should also have learned how to implement various operators and conditions.

By using the substitution variables, you can add flexibility to your SQL statements. This enables the queries to prompt for the filter condition for the rows during run time.

Practice 2: Overview

This practice covers the following topics:

- Selecting data and changing the order of the rows that are displayed
- Restricting rows by using the `WHERE` clause
- Sorting rows by using the `ORDER BY` clause
- Using substitution variables to add flexibility to your SQL `SELECT` statements

ORACLE

2 - 38

Copyright © 2007, Oracle. All rights reserved.

Practice 2: Overview

In this practice, you build more reports, including statements that use the `WHERE` clause and the `ORDER BY` clause. You make the SQL statements more reusable and generic by including the ampersand substitution.

Practice 2

The HR department needs your assistance in creating some queries.

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

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

2. Open a new SQL Worksheet. Create a report that displays the last name and department number for employee number 176. Run the query.

	LAST_NAME	DEPARTMENT_ID
1	Taylor	80

3. The HR department needs to find high-salary and low-salary employees. Modify `lab_02_01.sql` to display the last name and salary for any employee whose salary is not in the range of \$5,000 to \$12,000. Save your SQL statement as `lab_02_03.sql`.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Lorentz	4200
5	Rajs	3500
6	Davies	3100
7	Matos	2600
8	Vargas	2500
9	Whalen	4400
10	Hartstein	13000

Practice 2 (continued)

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

	LAST_NAME	JOB_ID	HIRE_DATE
1	Matos	ST_CLERK	15-MAR-98
2	Taylor	SA_REP	24-MAR-98

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


	LAST_NAME	DEPARTMENT_ID
1	Davies	50
2	Fay	20
3	Hartstein	20
4	Matos	50
5	Mourgos	50
6	Rajs	50
7	Vargas	50

6. Modify `lab_02_03.sql` to display 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`.



	Employee	Monthly Salary
1	Fay	6000
2	Mourgos	5800

Practice 2 (continued)




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

	 LAST_NAME	HIRE_DATE
1	Higgins	07-JUN-94
2	Gietz	07-JUN-94



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

	 LAST_NAME	 JOB_ID
1	King	AD_PRES

9. Create a report to display the last name, salary, and commission of all employees who earn commissions. Sort data in descending order of salary and commissions. Use the column's numeric position in the ORDER BY clause.

	 LAST_NAME	 SALARY	 COMMISSION_PCT
1	Abel	11000	0.3
2	Zlotkey	10500	0.2
3	Taylor	8600	0.2
4	Grant	7000	0.15

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. Save this query to a file named lab_02_10.sql. If you enter 12000 when prompted, the report displays the following results:

	 LAST_NAME	 SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

Practice 2 (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 last_name:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	104	Ernst	6000	60
2	107	Lorentz	4200	60

manager_id = 201, sorted by salary:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	202	Fay	6000	20

manager_id = 124, sorted by employee_id:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	141	Rajs	3500	50
2	142	Davies	3100	50
3	143	Matos	2600	50
4	144	Vargas	2500	50

Practice 2 (continued)

If you have time, complete the following exercises:

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

	LAST_NAME
1	Grant
2	Whalen

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

	LAST_NAME
1	Davies
2	De Haan
3	Hartstein
4	Whalen

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

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

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000
2	Taylor	SA_REP	8600
3	Davies	ST_CLERK	3100
4	Matos	ST_CLERK	2600

15. Modify lab_02_06.sql to display the last name, salary, and commission for all employees whose commission is 20%. Resave lab_02_06.sql as lab_02_15.sql. Rerun the statement in lab_02_15.sql.

	Employee	Monthly Salary	COMMISSION_PCT
1	Zlotkey	10500	0.2
2	Taylor	8600	0.2

3

Using Single-Row Functions to Customize Output

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Describe various types of functions available in SQL
- Use character, number, and date functions in `SELECT` statements

ORACLE

3 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

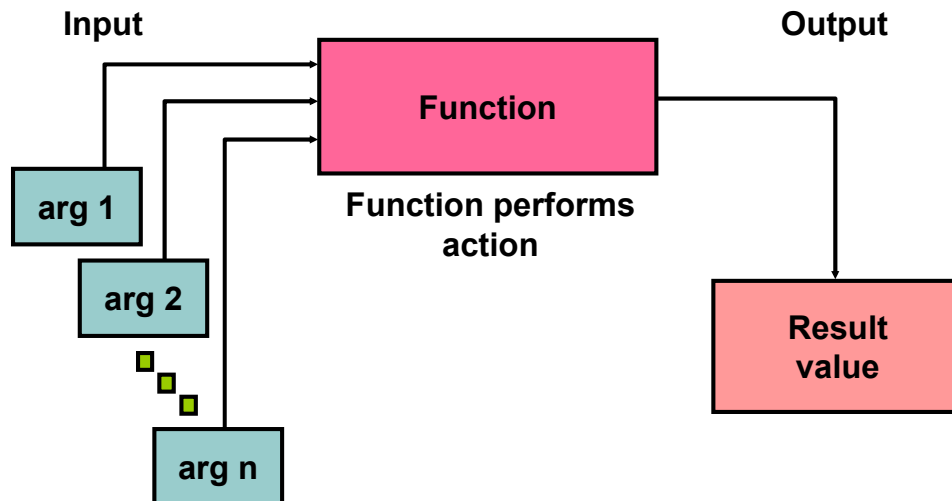
Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two lessons that explore functions. It focuses on single-row character, number, and date functions.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

ORACLE

SQL Functions



SQL Functions

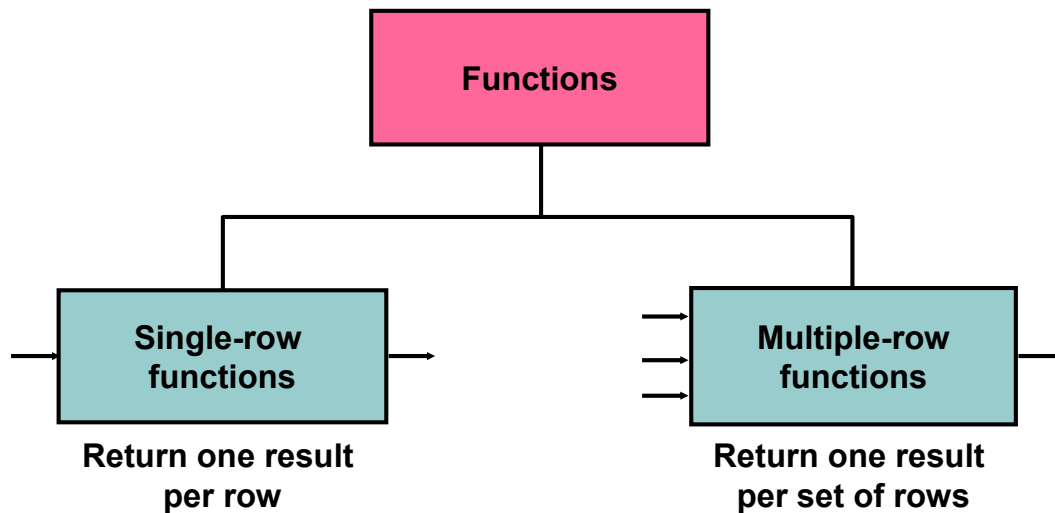
Functions are a very powerful feature of SQL. They can be used to do the following:

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

SQL functions sometimes take arguments and always return a value.

Note: If you want to know whether a function is a SQL:2003 compliant function, refer to the *Oracle Compliance To Core SQL:2003* section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Two Types of SQL Functions



ORACLE

3 - 5

Copyright © 2007, Oracle. All rights reserved.

Two Types of SQL Functions

There are two types of functions:

- Single-row functions
- Multiple-row functions

Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of single-row functions. This lesson covers the following ones:

- Character
- Number
- Date
- Conversion
- General

Multiple-Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are also known as *group functions* (covered in lesson 5 titled “Reporting Aggregated Data Using the Group Functions”).

Note: For more information and a complete list of available functions and their syntax, see the topic, *Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Single-Row Functions

Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2, ...)]
```

ORACLE

3 - 6

Copyright © 2007, Oracle. All rights reserved.

Single-Row Functions

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row that is returned by the query. An argument can be one of the following:

- User-supplied constant
- Variable value
- Column name
- Expression

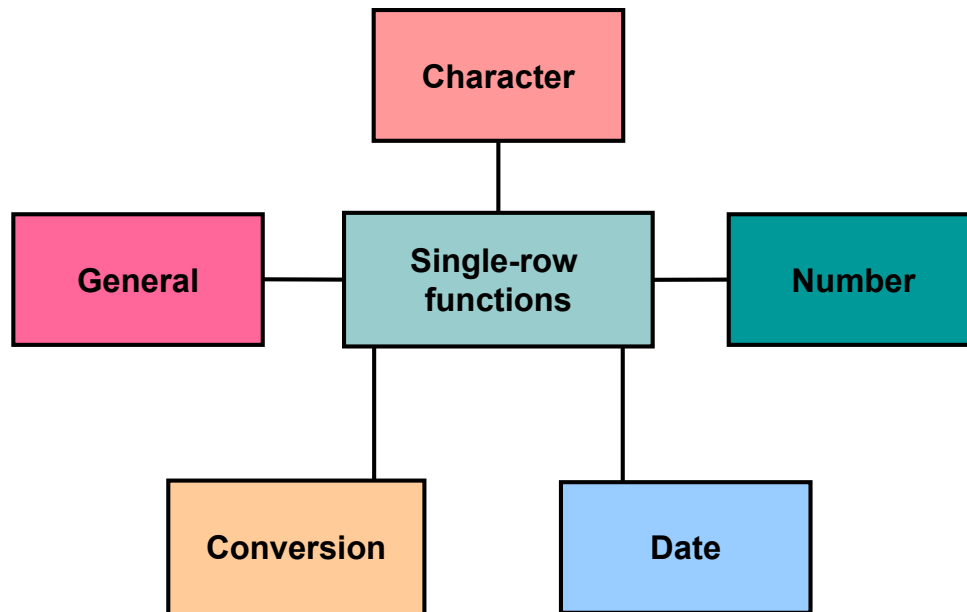
Features of single-row functions include:

- Acting on each row that is returned in the query
- Returning one result per row
- Possibly returning a data value of a different type than the one that is referenced
- Possibly expecting one or more arguments
- Can be used in SELECT, WHERE, and ORDER BY clauses; can be nested

In the syntax:

function_name is the name of the function
arg1, *arg2* is any argument to be used by the function. This can be represented by a column name or expression.

Single-Row Functions



ORACLE

3 - 7

Copyright © 2007, Oracle. All rights reserved.

Single-Row Functions (continued)

This lesson covers the following single-row functions:

- **Character functions:** Accept character input and can return both character and number values
- **Number functions:** Accept numeric input and return numeric values
- **Date functions:** Operate on values of the DATE data type (All date functions return a value of the DATE data type except the MONTHS_BETWEEN function, which returns a number.)

The following single-row functions are discussed in the next lesson titled “Using Conversion Functions and Conditional Expressions”:

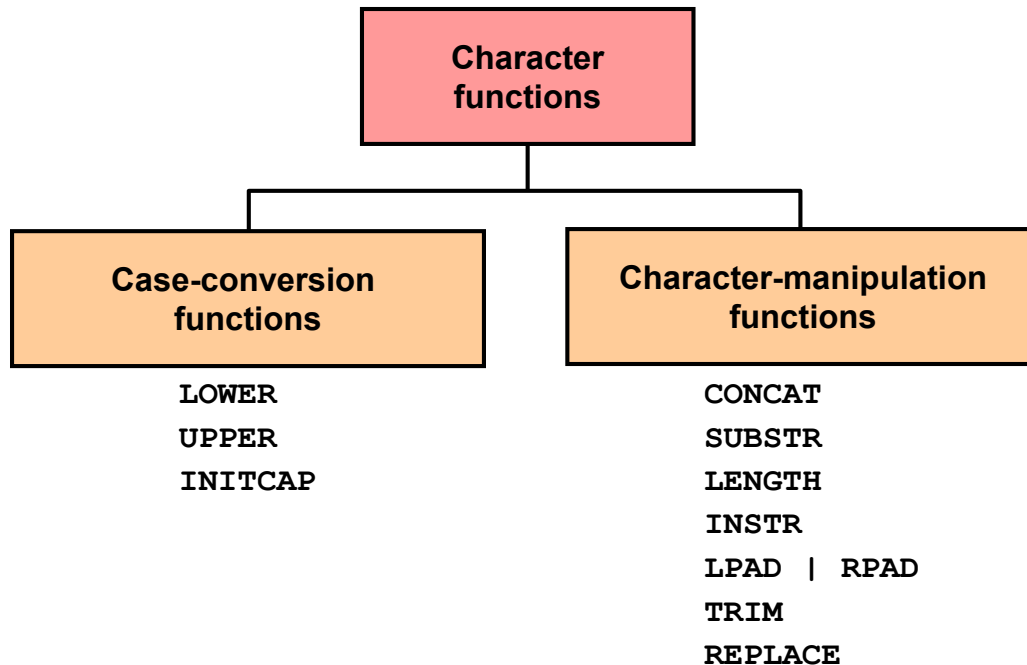
- **Conversion functions:** Convert a value from one data type to another
- **General functions:**
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
 - CASE
 - DECODE

Lesson Agenda

- Single-row SQL functions
- **Character functions**
- Number functions
- Working with dates
- Date functions

ORACLE

Character Functions



Character Functions

Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-conversion functions
- Character-manipulation functions

Function	Purpose
<code>LOWER (column expression)</code>	Converts alpha character values to lowercase
<code>UPPER (column expression)</code>	Converts alpha character values to uppercase
<code>INITCAP (column expression)</code>	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase
<code>CONCAT (column1 expression1, column2 expression2)</code>	Concatenates the first character value to the second character value; equivalent to concatenation operator ()
<code>SUBSTR (column expression, m[, n])</code>	Returns specified characters from character value starting at character position <i>m</i> , <i>n</i> characters long (If <i>m</i> is negative, the count starts from the end of the character value. If <i>n</i> is omitted, all characters to the end of the string are returned.)

Note: The functions discussed in this lesson are only some of the available functions.

Character Functions (continued)

Function	Purpose
LENGTH(<i>column expression</i>)	Returns the number of characters in the expression
INSTR(<i>column expression</i> , ' <i>string</i> ', [<i>m</i>], [<i>n</i>])	Returns the numeric position of a named string. Optionally, you can provide a position <i>m</i> to start searching, and the occurrence <i>n</i> of the string. <i>m</i> and <i>n</i> default to 1, meaning start the search at the beginning of the string and report the first occurrence.
LPAD(<i>column expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column expression</i> , <i>n</i> , ' <i>string</i> ')	Returns an expression left-padded to length of <i>n</i> characters with a character expression. Returns an expression right-padded to length of <i>n</i> characters with a character expression.
TRIM(<i>leading trailing both</i> , <i>trim_character</i> FROM <i>trim_source</i>)	Enables you to trim leading or trailing characters (or both) from a character string. If <i>trim_character</i> or <i>trim_source</i> is a character literal, you must enclose it in single quotation marks. This is a feature that is available in Oracle8i and later versions.
REPLACE(<i>text</i> , <i>search_string</i> , <i>replacement_string</i>)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Note: Some of the functions that are fully or partially SQL:2003 compliant are:

UPPER
LOWER
TRIM
LENGTH
SUBSTR
INSTR

Refer to the *Oracle Compliance To Core SQL:2003* section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)* for more information.

Case-Conversion Functions

These functions convert the case for character strings:

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

3 - 11

Copyright © 2007, Oracle. All rights reserved.

Case-Conversion Functions

LOWER, UPPER, and INITCAP are the three case-conversion functions.

- LOWER: Converts mixed-case or uppercase character strings to lowercase
- UPPER: Converts mixed-case or lowercase character strings to uppercase
- INITCAP: Converts the first letter of each word to uppercase and the remaining letters to lowercase

```
SELECT 'The job id for '||UPPER(last_name)||' is '  
||LOWER(job_id) AS "EMPLOYEE DETAILS"  
FROM employees;
```

	EMPLOYEE DETAILS
1	The job id for ABEL is sa_rep
2	The job id for DAVIES is st_clerk
3	The job id for DE HAAN is ad_vp
...	
19	The job id for WHALEN is ad_asst
20	The job id for ZLOTKEY is sa_man

Using Case-Conversion Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE last_name = 'higgins';
```

0 rows selected

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE LOWER(last_name) = 'higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205 Higgins	110

ORACLE

3 - 12

Copyright © 2007, Oracle. All rights reserved.

Using Case-Conversion Functions

The slide example displays the employee number, name, and department number of employee Higgins.

The WHERE clause of the first SQL statement specifies the employee name as `higgins`. Because all the data in the `EMPLOYEES` table is stored in proper case, the name `higgins` does not find a match in the table, and no rows are selected.

The WHERE clause of the second SQL statement specifies that the employee name in the `EMPLOYEES` table is compared to `higgins`, converting the `LAST_NAME` column to lowercase for comparison purposes. Because both names are now lowercase, a match is found and one row is selected. The WHERE clause can be rewritten in the following manner to produce the same result:

```
...WHERE last_name = 'Higgins'
```

The name in the output appears as it was stored in the database. To display the name in uppercase, use the `UPPER` function in the `SELECT` statement.

```
SELECT employee_id, UPPER(last_name), department_id
FROM employees
WHERE INITCAP(last_name) = 'Higgins';
```

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE', 'J', 'BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

ORACLE

3 - 13

Copyright © 2007, Oracle. All rights reserved.

Character-Manipulation Functions

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, and TRIM are the character-manipulation functions that are covered in this lesson.

- CONCAT: Joins values together (You are limited to using two parameters with CONCAT.)
- SUBSTR: Extracts a string of determined length
- LENGTH: Shows the length of a string as a numeric value
- INSTR: Finds the numeric position of a named character
- LPAD: Returns an expression left-padded to the length of *n* characters with a character expression
- RPAD: Returns an expression right-padded to the length of *n* characters with a character expression
- TRIM: Trims leading or trailing characters (or both) from a character string (If *trim_character* or *trim_source* is a character literal, you must enclose it within single quotation marks.)

Note: You can use functions such as UPPER and LOWER with ampersand substitution. For example, use UPPER('&job_title') so that the user does not have to enter the job title in a specific case.

Using the Character-Manipulation Functions

```

SELECT employee_id, CONCAT(first_name, last_name) NAME,
       job_id, LENGTH (last_name),
       INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
    
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
1	EllenAbel	SA_REP	4	0
2	JonathonTaylor	SA_REP	6	2
3	KimberelyGrant	SA_REP	5	3
4	PatFay	MK_REP	3	2

1

2

3

Using the Character-Manipulation Functions

The slide example displays employee first names and last names joined together, the length of the employee last name, and the numeric position of the letter “a” in the employee last name for all employees who have the string, REP, contained in the job ID starting at the fourth position of the job ID.

Example:

Modify the SQL statement in the slide to display the data for those employees whose last names end with the letter “n.”

```

SELECT employee_id, CONCAT(first_name, last_name) NAME,
       LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
    
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	LexDe Haan	7	5
2	JenniferWhalen	6	3
3	MichaelHartstein	9	2

Lesson Agenda

- Single-row SQL functions
- Character functions
- **Number functions**
- Working with dates
- Date Functions

ORACLE

Number Functions

- ROUND: Rounds value to a specified decimal
- TRUNC: Truncates value to a specified decimal
- MOD: Returns remainder of division

Function	Result
ROUND (45.926, 2)	45.93
TRUNC (45.926, 2)	45.92
MOD (1600, 300)	100

ORACLE

3 - 16

Copyright © 2007, Oracle. All rights reserved.

Number Functions

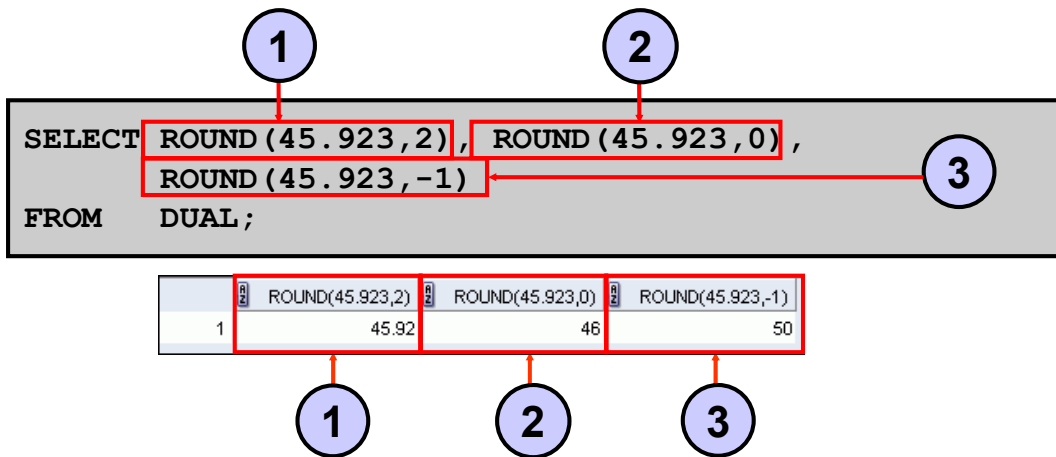
Number functions accept numeric input and return numeric values. This section describes some of the number functions.

Function	Purpose
ROUND (<i>column expression</i> , <i>n</i>)	Rounds the column, expression, or value to <i>n</i> decimal places or, if <i>n</i> is omitted, no decimal places (If <i>n</i> is negative, numbers to the left of decimal point are rounded.)
TRUNC (<i>column expression</i> , <i>n</i>)	Truncates the column, expression, or value to <i>n</i> decimal places or, if <i>n</i> is omitted, <i>n</i> defaults to zero
MOD (<i>m</i> , <i>n</i>)	Returns the remainder of <i>m</i> divided by <i>n</i>

Note: This list contains only some of the available number functions.

For more information, see the section on *Numeric Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Using the ROUND Function



DUAL is a dummy table that you can use to view results from functions and calculations.

ORACLE

Using the ROUND Function

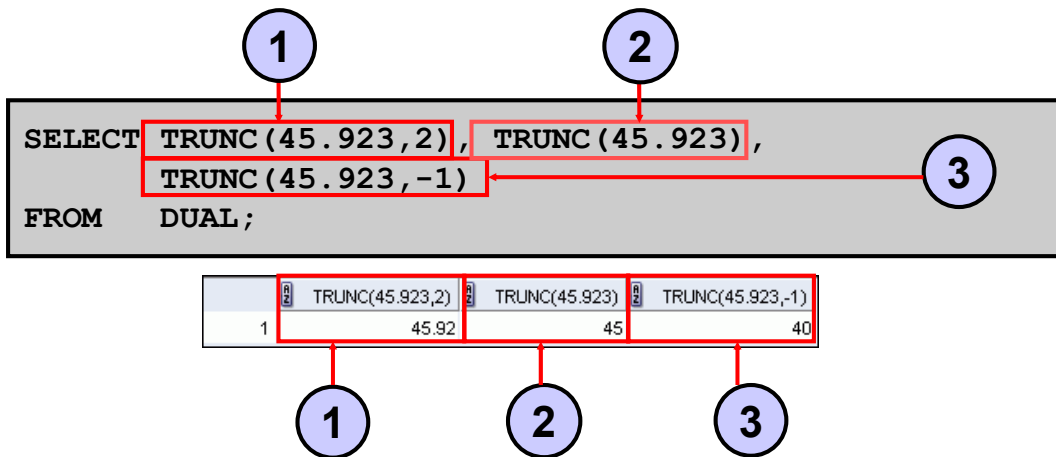
The ROUND function rounds the column, expression, or value to n decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left (rounded to the nearest unit of 100).

The ROUND function can also be used with date functions. You will see examples later in this lesson.

DUAL Table

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value only once (for example, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data). The DUAL table is generally used for completeness of the SELECT clause syntax, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from the actual tables.

Using the TRUNC Function



ORACLE

3 - 18

Copyright © 2007, Oracle. All rights reserved.

Using the TRUNC Function

The TRUNC function truncates the column, expression, or value to *n* decimal places.

The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places. Conversely, if the second argument is -2, the value is truncated to two decimal places to the left. If the second argument is -1, the value is truncated to one decimal place to the left.

Like the ROUND function, the TRUNC function can be used with date functions.

Using the MOD Function

For all employees with the job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM employees
WHERE job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

ORACLE

Using the MOD Function

The MOD function finds the remainder of the first argument divided by the second argument. The slide example calculates the remainder of the salary after dividing it by 5,000 for all employees whose job ID is SA_REP.

Note: The MOD function is often used to determine whether a value is odd or even.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- **Working with dates**
- Date functions

ORACLE

Working with Dates

- The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.
- The default date display format is DD-MON-RR.
 - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
 - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-87
2	Whalen	17-SEP-87

ORACLE

3 - 21

Copyright © 2007, Oracle. All rights reserved.

Working with Dates

The Oracle database stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The default display and input format for any date is DD-MON-RR. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9999 A.D.

In the example in the slide, the `HIRE_DATE` column output is displayed in the default format DD-MON-RR. However, dates are not stored in the database in this format. All the components of the date and time are stored. So, although a `HIRE_DATE` such as 17-JUN-87 is displayed as day, month, and year, there is also *time* and *century* information associated with the date. The complete data might be June 17, 1987, 5:10:43 PM.

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

ORACLE

3 - 22

Copyright © 2007, Oracle. All rights reserved.

RR Date Format

The RR date format is similar to the YY element, but you can use it to specify different centuries. Use the RR date format element instead of YY so that the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The table in the slide summarizes the behavior of the RR element.

Current Year	Given Date	Interpreted (RR)	Interpreted (YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

Oracle Date Format

This data is stored internally as follows:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	87	06	17	17	10	43

Centuries and the Year 2000

When a record with a date column is inserted into a table, the *century* information is picked up from the `SYSDATE` function. However, when the date column is displayed on the screen, the century component is not displayed (by default).

The `DATE` data type always stores year information as a four-digit number internally: two digits for the century and two digits for the year. For example, the Oracle database stores the year as 1987 or 2004, and not just as 87 or 04.

Using the SYSDATE Function

SYSDATE is a function that returns:

- Date
- Time

```
SELECT sysdate
FROM dual;
```

SYSDATE
1 31-MAY-07

ORACLE

Using the SYSDATE Function

SYSDATE is a date function that returns the current database server date and time. You can use SYSDATE just as you would use any other column name. For example, you can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

Note: SYSDATE returns the current date and time set for the operating system on which the database resides. Hence, if you are in a place in Australia and connected to a remote database in a location in the United States (US), sysdate function will return the US date and time. In that case, you can use the CURRENT_DATE function that returns the current date in the session time zone.

The CURRENT_DATE function and other related time zone functions are discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

ORACLE

3 - 25

Copyright © 2007, Oracle. All rights reserved.

Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

You can perform the following operations:

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Using Arithmetic Operators with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1041.168239087301587301587301587302
2	Kochhar	923.025381944444444444444444444444
3	De Haan	750.168239087301587301587301587302

ORACLE

3 - 26

Copyright © 2007, Oracle. All rights reserved.

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (`SYSDATE`) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

Note: `SYSDATE` is a SQL function that returns the current date and time. Your results may differ depending on the date and time set for the operating system of your local database when you run the SQL query.

If a more current date is subtracted from an older date, the difference is a negative number.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- **Date functions**

ORACLE

Date-Manipulation Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

ORACLE

3 - 28

Copyright © 2007, Oracle. All rights reserved.

Date-Manipulation Functions

Date functions operate on Oracle dates. All date functions return a value of the DATE data type except MONTHS_BETWEEN, which returns a numeric value.

- MONTHS_BETWEEN(*date1*, *date2*): Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result represents a portion of the month.
- ADD_MONTHS(*date*, *n*): Adds *n* number of calendar months to *date*. The value of *n* must be an integer and can be negative.
- NEXT_DAY(*date*, '*char*'): Finds the date of the next specified day of the week ('*char*') following *date*. The value of *char* may be a number representing a day or a character string.
- LAST_DAY(*date*): Finds the date of the last day of the month that contains *date*.

The above list is a subset of the available date functions. ROUND and TRUNC number functions can also be used to manipulate the date values as shown below:

- ROUND(*date* [, '*fmt*']): Returns *date* rounded to the unit that is specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is rounded to the nearest day.
- TRUNC(*date* [, '*fmt*']): Returns *date* with the time portion of the day truncated to the unit that is specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is truncated to the nearest day.

The format models are covered in detail in the next lesson titled “Using Conversion Functions and Conditional Expressions.”

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94')	19.6774194
ADD_MONTHS ('31-JAN-96', 1)	'29-FEB-96'
NEXT_DAY ('01-SEP-95', 'FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Using Date Functions

In the slide example, the ADD_MONTHS function adds one month to the supplied date value, “31-JAN-96” and returns “29-FEB-96.” The function recognizes the year 1996 as the leap year and hence returns the last day of the February month. If you change the input date value to “31-JAN-95,” the function returns “28-FEB-95.”

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and the last day of the hire month for all employees who have been employed for fewer than 100 months.

```
SELECT employee_id, hire_date,
       MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
       ADD_MONTHS (hire_date, 6) REVIEW,
       NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
FROM   employees
WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 100;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(HIRE_DATE,FRIDAY)	LAST_DAY(HIRE_DATE)
1	124	16-NOV-99	91.1099600...	16-MAY-00	19-NOV-99	30-NOV-99
2	149	29-JAN-00	88.6906052...	29-JUL-00	04-FEB-00	31-JAN-00
3	178	24-MAY-99	96.8518955...	24-NOV-99	28-MAY-99	31-MAY-99
4	99999	07-JUN-99	96.4002826...	07-DEC-99	11-JUN-99	30-JUN-99
5	113	11-JUN-07	0.25824335...	11-DEC-07	15-JUN-07	30-JUN-07

Using ROUND and TRUNC Functions with Dates

Assume SYSDATE = '25-JUL-03':

Function	Result
ROUND(SYSDATE, 'MONTH')	01-AUG-03
ROUND(SYSDATE, 'YEAR')	01-JAN-04
TRUNC(SYSDATE, 'MONTH')	01-JUL-03
TRUNC(SYSDATE, 'YEAR')	01-JAN-03

ORACLE

3 - 30

Copyright © 2007, Oracle. All rights reserved.

Using ROUND and TRUNC Functions with Dates

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month. If the format model is month, dates 1-15 result in the first day of the current month. Dates 16-31 result in the first day of the next month. If the format model is year, months 1-6 result in January 1 of the current year. Months 7-12 result in January 1 of the next year.

Example:

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and starting month using the ROUND and TRUNC functions.

```
SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM   employees
WHERE  hire_date LIKE '%97';
```

	EMPLOYEE_ID	HIRE_DATE	ROUND(HIRE_DATE,'MONTH')	TRUNC(HIRE_DATE,'MONTH')
1	142	29-JAN-97	01-FEB-97	01-JAN-97
2	202	17-AUG-97	01-SEP-97	01-AUG-97

Summary

In this lesson, you should have learned how to:

- Perform calculations on data using functions
- Modify individual data items using functions

ORACLE

3 - 31

Copyright © 2007, Oracle. All rights reserved.

Summary

Single-row functions can be nested to any level. Single-row functions can manipulate the following:

- Character data: LOWER, UPPER, INITCAP, CONCAT, SUBSTR, INSTR, LENGTH
- Number data: ROUND, TRUNC, MOD
- Date values: SYSDATE, MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY

Remember the following:

- Date values can also use arithmetic operators.
- ROUND and TRUNC functions can also be used with date values.

SYSDATE and DUAL

SYSDATE is a date function that returns the current date and time. It is customary to select SYSDATE from a dummy table called DUAL.

Practice 3: Overview

This practice covers the following topics:

- Writing a query that displays the current date
- Creating queries that require the use of numeric, character, and date functions
- Performing calculations of years and months of service for an employee

ORACLE

3 - 32

Copyright © 2007, Oracle. All rights reserved.

Practice 3: Overview

This practice provides a variety of exercises using different functions that are available for character, number, and date data types.

Practice 3

Part 1

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

Note: If your database is remotely located in a different time zone, the output will be the date for the operating system on which the database resides.

	Date
1	31-MAY-07

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. Save your SQL statement in a file named lab_03_02.sql.
3. Run your query in the lab_03_02.sql file.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
1	100	King	24000	27720
2	101	Kochhar	17000	19635
3	102	De Haan	17000	19635
4	103	Hunold	9000	10395
5	104	Ernst	6000	6930
6	107	Lorentz	4200	4851
7	124	Mourgos	5800	6699
8	141	Rajs	3500	4043
9	142	Davies	3100	3581
10	143	Matos	2600	3003

...

19	205	Higgins	12000	13860
20	206	Gietz	8300	9587

4. Modify your query lab_03_02.sql 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.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
1	100	King	24000	27720	3720
2	101	Kochhar	17000	19635	2635
3	102	De Haan	17000	19635	2635
4	103	Hunold	9000	10395	1395
5	104	Ernst	6000	6930	930

...

20	206	Gietz	8300	9587	1287
----	-----	-------	------	------	------

Practice 3 (continued)

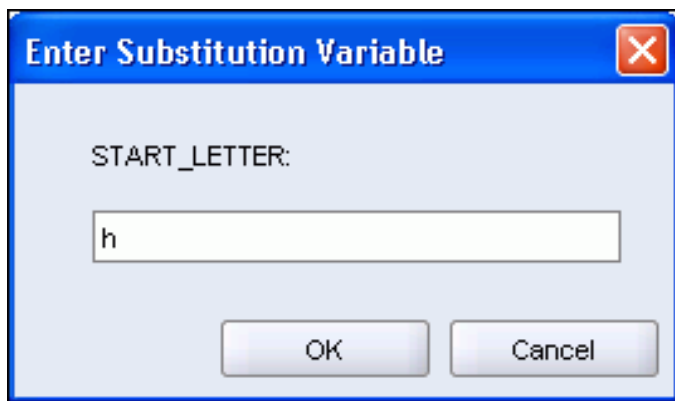
- Write a query that displays the last name (with the first letter in uppercase and all the other letters in lowercase) and the length of the last name for all employees whose name starts with the letters “J,” “A,” or “M.” Give each column an appropriate label. Sort the results by the employees’ last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

	A-Z	Name	A-Z	Length
1		Abel		4
2		Matos		5
3		Mourgos		7

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

	A-Z	Name	A-Z	Length
1		Hartstein		9
2		Higgins		7
3		Hunold		6

Modify the query such that the case of the entered letter does not affect the output. The entered letter must be capitalized before being processed by the `SELECT` query.



Enter Substitution Variable

START_LETTER:

h

OK Cancel

	A-Z	Name	A-Z	Length
1		Hartstein		9
2		Higgins		7
3		Hunold		6

Practice 3 (continued)

- 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 as MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note: Because this query depends on the date when it was executed, the values in the MONTHS_WORKED column will differ for you.

	LAST_NAME	MONTHS_WORKED
1	Zlotkey	88
2	Mourgos	90
3	Grant	96
4	Lorentz	100
5	Vargas	107
6	Taylor	110
7	Matos	111
8	Fay	117
9	Davies	124
10	Abel	133
11	Hartstein	135
12	Rajs	139
13	Higgins	156
14	Gietz	156
15	De Haan	173
16	Ernst	192
17	Hunold	209
18	Kochhar	212
19	Whalen	236
20	King	239

Practice 3 (continued)

If you have time, complete the following exercises:

7. 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 as SALARY.

	LAST_NAME	SALARY
1	King	\$\$\$\$\$\$\$\$\$24000
2	Kochhar	\$\$\$\$\$\$\$\$\$17000

...

20	Gietz	\$\$\$\$\$\$\$\$\$8300
----	-------	------------------------

8. 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 as EMPLOYEES_AND_THEIR_SALARIES.

	EMPLOYEES_AND_THEIR_SALARIES
1	King *****
2	Kochhar *****
3	De Haan *****
4	Hartstei *****
5	Higgins *****

...

19	Matos **
20	Vargas **

9. Create a query to display the last name and the number of weeks employed for all employees in department 90. Label the number of weeks column as TENURE. Truncate the number of weeks value to 0 decimal places. Show the records in descending order of the employee's tenure.
Note: The TENURE value will differ as it depends on the date on which you run the query.

	LAST_NAME	TENURE
1	King	1041
2	Kochhar	923
3	De Haan	750

4

Using Conversion Functions and Conditional Expressions

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Describe various types of conversion functions that are available in SQL
- Use the `TO_CHAR`, `TO_NUMBER`, and `TO_DATE` conversion functions
- Apply conditional expressions in a `SELECT` statement

ORACLE

4 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

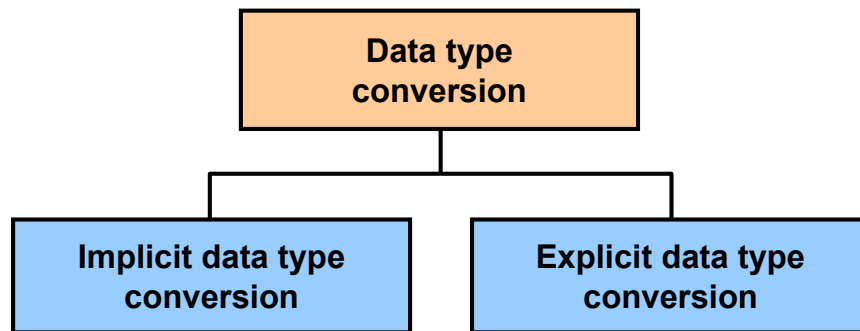
This lesson focuses on functions that convert data from one type to another (for example, conversion from character data to numeric data) and discusses the conditional expressions in SQL `SELECT` statements.

Lesson Agenda

- **Implicit and explicit data type conversion**
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
 - `NVL`
 - `NVL2`
 - `NULLIF`
 - `COALESCE`
- Conditional expressions:
 - `CASE`
 - `DECODE`

ORACLE

Conversion Functions



ORACLE

4 - 4

Copyright © 2007, Oracle. All rights reserved.

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle database can be defined by using the American National Standards Institute (ANSI), DB2, and SQL/DS data types. However, the Oracle server internally converts such data types to Oracle data types.

In some cases, the Oracle server receives data of one data type where it expects data of a different data type. When this happens, the Oracle server can automatically convert the data to the expected data type. This data type conversion can be done *implicitly* by the Oracle server or *explicitly* by the user.

Implicit data type conversions work according to the rules explained in the next two slides.

Explicit data type conversions are done by using the conversion functions. Conversion functions convert a value from one data type to another. Generally, the form of the function names follows the convention *data type* TO *data type*. The first data type is the input data type and the second data type is the output.

Note: Although implicit data type conversion is available, it is recommended that you do the explicit data type conversion to ensure the reliability of your SQL statements.

Implicit Data Type Conversion

In expressions, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

ORACLE

Implicit Data Type Conversion

Oracle server can automatically perform data type conversion in an expression. For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string '01-JAN-90' to a date. Therefore, a VARCHAR2 or CHAR value can be implicitly converted to a number or date data type in an expression.

Implicit Data Type Conversion

For expression evaluation, the Oracle server can automatically convert the following:

From	To
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR

ORACLE

4 - 6

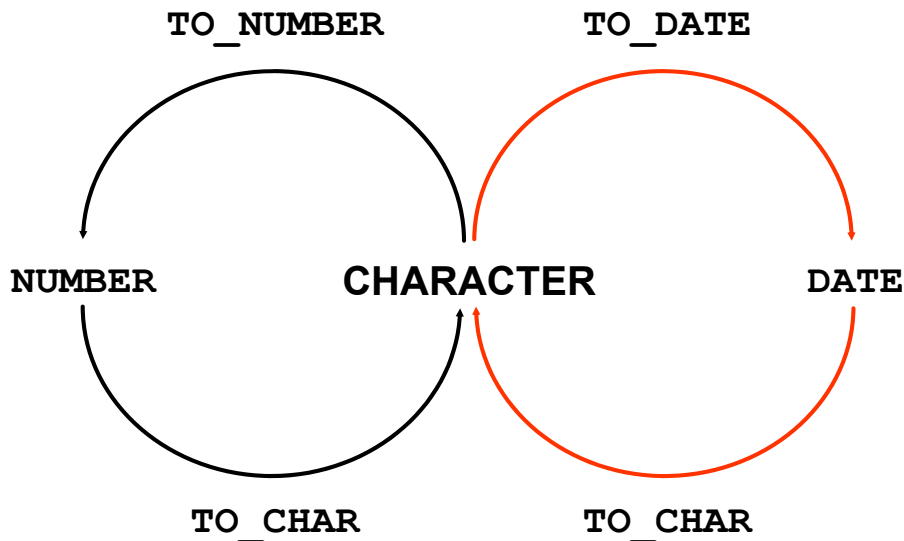
Copyright © 2007, Oracle. All rights reserved.

Implicit Data Type Conversion (continued)

In general, the Oracle server uses the rule for expressions when a data type conversion is needed. For example, the expression `grade = 2` results in the implicit conversion of the number 20000 to the string "2" because `grade` is a `CHAR(2)` column.

Note: CHAR to NUMBER conversions succeed only if the character string represents a valid number.

Explicit Data Type Conversion



ORACLE

4 - 7

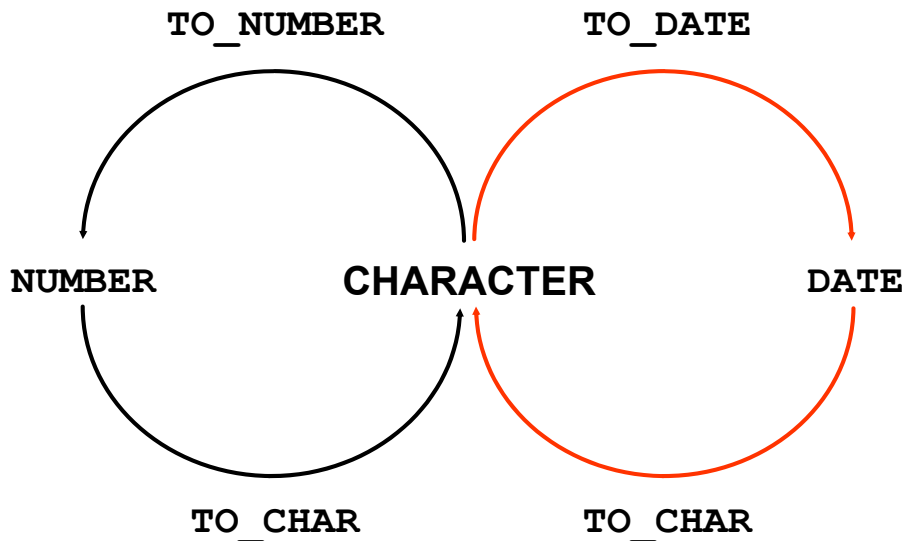
Copyright © 2007, Oracle. All rights reserved.

Explicit Data Type Conversion

SQL provides three functions to convert a value from one data type to another:

Function	Purpose
<code>TO_CHAR(number date, [fmt], [nlsparams])</code>	<p>Converts a number or date value to a VARCHAR2 character string with the format model <i>fmt</i></p> <p>Number conversion: The <i>nlsparams</i> parameter specifies the following characters, which are returned by number format elements:</p> <ul style="list-style-type: none"> • Decimal character • Group separator • Local currency symbol • International currency symbol <p>If <i>nlsparams</i> or any other parameter is omitted, this function uses the default parameter values for the session.</p>

Explicit Data Type Conversion



ORACLE

4 - 8

Copyright © 2007, Oracle. All rights reserved.

Explicit Data Type Conversion (continued)

Function	Purpose
<code>TO_CHAR(number date, [fmt], [nlsparams])</code>	Date conversion: The <code>nlsparams</code> parameter specifies the language in which the month and day names, and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session.
<code>TO_NUMBER(char, [fmt], [nlsparams])</code>	Converts a character string containing digits to a number in the format specified by the optional format model <code>fmt</code> . The <code>nlsparams</code> parameter has the same purpose in this function as in the <code>TO_CHAR</code> function for number conversion.
<code>TO_DATE(char, [fmt], [nlsparams])</code>	Converts a character string representing a date to a date value according to the <code>fmt</code> that is specified. If <code>fmt</code> is omitted, the format is DD-MON-YY. The <code>nlsparams</code> parameter has the same purpose in this function as in the <code>TO_CHAR</code> function for date conversion.

Explicit Data Type Conversion (continued)

Note: The list of functions mentioned in this lesson includes only some of the available conversion functions.

For more information, see the section on *Conversion Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- Nesting functions
- General functions:
 - `NVL`
 - `NVL2`
 - `NULLIF`
 - `COALESCE`
- Conditional expressions:
 - `CASE`
 - `DECODE`

ORACLE

Using the TO_CHAR Function with Dates

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed with single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an *fm* element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

ORACLE

4 - 11

Copyright © 2007, Oracle. All rights reserved.

Using the TO_CHAR Function with Dates

TO_CHAR converts a datetime data type to a value of VARCHAR2 data type in the format specified by the *format_model*. A format model is a character literal that describes the format of datetime stored in a character string. For example, the datetime format model for the string '11-NOV-1999' is 'DD-MON-YYYY'. You can use the TO_CHAR function to convert a date from its default format to the one that you specify.

Guidelines

- The format model must be enclosed with single quotation marks and is case-sensitive.
- The format model can include any valid date format element. But be sure to separate the date value from the format model with a comma.
- The names of days and months in the output are automatically padded with blanks.
- To remove padded blanks or to suppress leading zeros, use the fill mode *fm* element.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

	EMPLOYEE_ID	MONTH_HIRED
1	205	06/94

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

Sample Format Elements of Valid Date Formats

Element	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digit of the year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four-, three-, two-, or one-digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of the month padded with blanks to a length of nine characters
MON	Name of the month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of the year or month
DDD or DD or D	Day of the year, month, or week
DAY	Name of the day padded with blanks to a length of nine characters
DY	Name of the day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.
IW	Weeks in the year from ISO standard (1 to 53)

Elements of the Date Format Model

- Time elements format the time portion of the date:

HH24:MI:SS AM	15:45:32 PM
---------------	-------------

- Add character strings by enclosing them with double quotation marks:

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- Number suffixes spell out numbers:

ddspth	fourteenth
--------	------------

ORACLE

4 - 14

Copyright © 2007, Oracle. All rights reserved.

Elements of the Date Format Model

Use the formats that are listed in the following tables to display time information and literals, and to change numerals to spelled numbers.

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day, or hour (1–12), or hour (0–23)
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

Other Formats

Element	Description
/ . ,	Punctuation is reproduced in the result.
“of the”	Quoted string is reproduced in the result.

Specifying Suffixes to Influence Number Display

Element	Description
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, DDSP for FOUR)
SPTH or THSP	Spelled-out ordinal numbers (for example, DDSPTH for FOURTH)

Using the TO_CHAR Function with Dates

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY')  
       AS HIREDATE  
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 1987
2	Kochhar	21 September 1989
3	De Haan	13 January 1993
4	Hunold	3 January 1990
5	Ernst	21 May 1991
6	Lorentz	7 February 1999
7	Mourgos	16 November 1999
8	Rajs	17 October 1995
9	Davies	29 January 1997
10	Matos	15 March 1998
...		
19	Higgins	7 June 1994
20	Gietz	7 June 1994

ORACLE

4 - 16

Copyright © 2007, Oracle. All rights reserved.

Using the TO_CHAR Function with Dates

The SQL statement in the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 1987.

Example:

Modify the example in the slide to display the dates in a format that appears as “Seventeenth of June 1987 12:00:00 AM.”

```
SELECT last_name,  
       TO_CHAR(hire_date,  
               'fmDdspth "of" Month YYYY fmHH:MI:SS AM')  
       HIREDATE  
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	Seventeenth of June 1987 12:00:00 AM
2	Kochhar	Twenty-First of September 1989 12:00:00 AM

...

Notice that the month follows the format model specified; in other words, the first letter is capitalized and the rest are in lowercase.

Using the TO_CHAR Function with Numbers

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character:

Element	Result
9	Represents a number
0	Forces a zero to be displayed
\$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as a thousands indicator

ORACLE

Using the TO_CHAR Function with Numbers

When working with number values, such as character strings, you should convert those numbers to the character data type using the TO_CHAR function, which translates a value of NUMBER data type to VARCHAR2 data type. This technique is especially useful with concatenation.

Using the TO_CHAR Function with Numbers (continued)

Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns the decimal character in the specified position. The default is a period (.).	99D99	99.99
.	Decimal point in position specified	999999.99	1234.00
G	Returns the group separator in the specified position. You can specify multiple group separators in a number format model.	9,999	9G999
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the “Euro” (or other) dual currency	U9999	€1234
V	Multiply by 10 <i>n</i> times (<i>n</i> = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
B	Display zero values as blank, not 0	B9999.99	1234.00

Using the TO_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

	SALARY
1	\$6,000.00

ORACLE

Using the TO_CHAR Function with Numbers (continued)

- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places provided in the format model.

Using the TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the TO_NUMBER function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the TO_DATE function:

```
TO_DATE(char[, 'format_model'])
```

- These functions have an `fx` modifier. This modifier specifies the exact match for the character argument and date format model of a TO_DATE function.

ORACLE

4 - 20

Copyright © 2007, Oracle. All rights reserved.

Using the TO_NUMBER and TO_DATE Functions

You may want to convert a character string to either a number or a date. To accomplish this task, use the TO_NUMBER or TO_DATE functions. The format model that you select is based on the previously demonstrated format elements.

The `fx` modifier specifies the exact match for the character argument and date format model of a TO_DATE function:

- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- The character argument cannot have extra blanks. Without `fx`, the Oracle server ignores extra blanks.
- Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without `fx`, the numbers in the character argument can omit leading zeros.

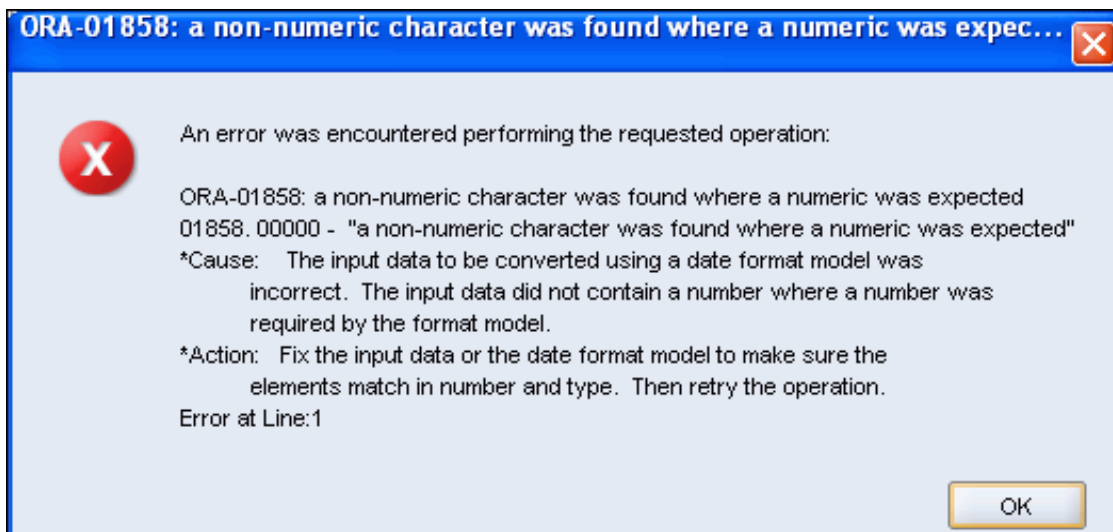
Using the TO_NUMBER and TO_DATE Functions (continued)

Example:

Display the name and hire date for all employees who started on May 24, 1999. There are two spaces after the month *May* and the number *24* in the following example. Because the `fx` modifier is used, an exact match is required and the spaces after the word *May* are not recognized:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May  24, 1999', 'fxMonth DD, YYYY');
```

The error:



Using the TO_CHAR and TO_DATE Function with RR Date Format

To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

	LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
1	King	17-Jun-1987
2	Kochhar	21-Sep-1989
3	Whalen	17-Sep-1987

Using the TO_CHAR and TO_DATE Function with RR Date Format

To find employees who were hired before 1990, the RR format can be used. Because the current year is greater than 1999, the RR format interprets the year portion of the date from 1950 to 1999.

The following command, on the other hand, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM employees
WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

```
0 rows selected
```

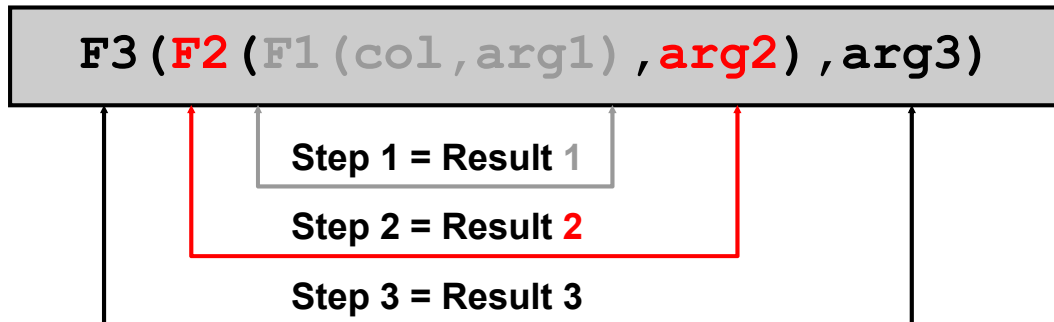

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- **Nesting functions**
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from the deepest level to the least deep level.



Nesting Functions

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

Nesting Functions

```
SELECT last name,  
       UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))  
FROM   employees  
WHERE  department_id = 60;
```

R	LAST_NAME	R	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold		HUNOLD_US
2	Ernst		ERNST_US
3	Lorentz		LORENTZ_US

Nesting Functions (continued)

The slide example displays the last names of employees in department 60. The evaluation of the SQL statement involves three steps:

1. The inner function retrieves the first eight characters of the last name.
Result1 = SUBSTR (LAST_NAME, 1, 8)
2. The outer function concatenates the result with _US.
Result2 = CONCAT (Result1, '_US')
3. The outermost function converts the results to uppercase.

The entire expression becomes the column heading because no column alias was given.

Example:

Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS  
                  (hire_date, 6), 'FRIDAY'),  
        'fmDay, Month ddth, YYYY')  
        "Next 6 Month Review"  
FROM   employees  
ORDER BY hire_date;
```

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- **General functions:**
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- **Conditional expressions:**
 - CASE
 - DECODE

ORACLE

General Functions

The following functions work with any data type and pertain to using nulls:

- NVL (*expr1*, *expr2*)
- NVL2 (*expr1*, *expr2*, *expr3*)
- NULLIF (*expr1*, *expr2*)
- COALESCE (*expr1*, *expr2*, ..., *exprn*)

ORACLE

4 - 27

Copyright © 2007, Oracle. All rights reserved.

General Functions

These functions work with any data type and pertain to the use of null values in the expression list.

Function	Description
NVL	Converts a null value to an actual value
NVL2	If <i>expr1</i> is not null, NVL2 returns <i>expr2</i> . If <i>expr1</i> is null, NVL2 returns <i>expr3</i> . The argument <i>expr1</i> can have any data type.
NULLIF	Compares two expressions and returns null if they are equal; returns the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Note: For more information about the hundreds of functions available, see the section on *Functions* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
 - `NVL(commission_pct, 0)`
 - `NVL(hire_date, '01-JAN-97')`
 - `NVL(job_id, 'No Job Yet')`

ORACLE

4 - 28

Copyright © 2007, Oracle. All rights reserved.

NVL Function

To convert a null value to an actual value, use the NVL function.

Syntax

```
NVL (expr1, expr2)
```

In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the target value for converting the null

You can use the NVL function to convert any data type, but the return value is always the same as the data type of *expr1*.

NVL Conversions for Various Data Types

Data Type	Conversion Example
NUMBER	<code>NVL(number_column, 9)</code>
DATE	<code>NVL(date_column, '01-JAN-95')</code>
CHAR or VARCHAR2	<code>NVL(character_column, 'Unavailable')</code>

Using the NVL Function

```
SELECT last name, salary, NVL(commission_pct, 0),
       (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200
11	Vargas	2500	0	30000
12	Zlotkey	10500	0.2	151200

...

1

2

ORACLE

4 - 29

Copyright © 2007, Oracle. All rights reserved.

Using the NVL Function

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to the result:

```
SELECT last_name, salary, commission_pct,
       (salary*12) + (salary*12*commission_pct) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
1	King	24000	(null)	(null)

...

11	Vargas	2500	(null)	(null)
12	Zlotkey	10500	0.2	151200
13	Abel	11000	0.3	171600

...

Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example in the slide, the NVL function is used to convert null values to zero.

Using the NVL2 Function

```
SELECT last name, salary, commission_pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

ORACLE

4 - 30

Copyright © 2007, Oracle. All rights reserved.

Using the NVL2 Function

The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned.

Syntax

```
NVL2(expr1, expr2, expr3)
```

In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the value that is returned if *expr1* is not null
- *expr3* is the value that is returned if *expr1* is null

In the example shown in the slide, the `COMMISSION_PCT` column is examined. If a value is detected, the second expression of `SAL+COMM` is returned. If the `COMMISSION_PCT` column holds a null value, the third expression of `SAL` is returned.

The argument *expr1* can have any data type. The arguments *expr2* and *expr3* can have any data types except `LONG`. If the data types of *expr2* and *expr3* are different, the Oracle server converts *expr3* to the data type of *expr2* before comparing them, unless *expr3* is a null constant. In the latter case, a data type conversion is not necessary. The data type of the return value is always the same as the data type of *expr2*, unless *expr2* is character data, in which case the return value's data type is `VARCHAR2`.

Using the NULLIF Function

```
SELECT first_name, LENGTH(first_name) "expr1",  
       last_name,  LENGTH(last_name)  "expr2",  
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result  
FROM employees;
```

	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis		Davies	6	(null)
3	Lex		De Haan	7	3
4	Bruce		Ernst	5	(null)
5	Pat		Fay	3	(null)
6	William		Gietz	5	7
7	Kimberely		Grant	5	9
...					
19	Jennifer		Whalen	6	8
20	Eleni		Zlotkey	7	5

Using the NULLIF Function

The NULLIF function compares two expressions. If they are equal, the function returns a null. If they are not equal, the function returns the first expression. However, you cannot specify the literal NULL for the first expression.

Syntax

```
NULLIF (expr1, expr2)
```

In the syntax:

- NULLIF compares *expr1* and *expr2*. If they are equal, then the function returns null. If they are not, then the function returns *expr1*. However, you cannot specify the literal NULL for *expr1*.

In the example shown in the slide, the length of the first name in the EMPLOYEES table is compared to the length of the last name in the EMPLOYEES table. When the lengths of the names are equal, a null value is displayed. When the lengths of the names are not equal, the length of the first name is displayed.

Note: The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed on a subsequent page:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.

ORACLE

4 - 32

Copyright © 2007, Oracle. All rights reserved.

Using the COALESCE Function

The COALESCE function returns the first non-null expression in the list.

Syntax

```
COALESCE (expr1, expr2, ... exprn)
```

In the syntax:

- *expr1* returns this expression if it is not null
- *expr2* returns this expression if the first expression is null and this expression is not null
- *exprn* returns this expression if the preceding expressions are null

Note that all expressions must be of the same data type.

Using the COALESCE Function

```
SELECT last name, employee id,  
COALESCE(TO_CHAR(commission_pct),TO_CHAR(manager_id),  
          'No commission and no manager')  
FROM employees;
```

	LAST_NAME	EMPLOYEE_ID	COALESCE(TO_CHAR(COMM
1	King	100	No commission and no manager
2	Kochhar	101	100
3	De Haan	102	100
4	Hunold	103	102
5	Ernst	104	103
6	Lorentz	107	103
7	Mourgos	124	100
8	Rajs	141	124

...

12	Zlotkey	149	.2
13	Abel	174	.3
14	Taylor	176	.2
15	Grant	178	.15
16	Whalen	200	101

...

ORACLE

Using the COALESCE Function (continued)

In the example shown in the slide, if the `manager_id` value is not null, it is displayed. If the `manager_id` value is null, then the `commission_pct` is displayed. If the `manager_id` and `commission_pct` values are null, then “No commission and no manager” is displayed. Note, `TO_CHAR` function is applied so that all expressions are of the same data type.

Using the COALESCE Function (continued)

Example:

For the employees who do not get any commission, your organization wants to give a salary increment of \$2,000 and for employees who get commission, the query should compute the new salary that is equal to the existing salary added to the commission amount.

```
SELECT last_name, salary, commission_pct,  
       COALESCE((salary+(commission_pct*salary)), salary+2000, salary) "New  
       Salary"  
FROM   employees;
```

Note: Examine the output. For employees who do not get any commission, the New Salary column shows the salary incremented by \$2,000 and for employees who get commission, the New Salary column shows the computed commission amount added to the salary.

	LAST_NAME	SALARY	COMMISSION_PCT	New Salary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000

...

9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12000	(null)	14000
20	Gietz	8300	(null)	10300

...

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Conditional Expressions

- Provide the use of the `IF-THEN-ELSE` logic within a SQL statement
- Use two methods:
 - `CASE` expression
 - `DECODE` function

ORACLE

4 - 36

Copyright © 2007, Oracle. All rights reserved.

Conditional Expressions

The two methods that are used to implement conditional processing (`IF-THEN-ELSE` logic) in a SQL statement are the `CASE` expression and the `DECODE` function.

Note: The `CASE` expression complies with the ANSI SQL. The `DECODE` function is specific to Oracle syntax.

CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
      [WHEN comparison_expr2 THEN return_expr2
      WHEN comparison_exprn THEN return_exprn
      ELSE else_expr]
END
```

ORACLE

4 - 37

Copyright © 2007, Oracle. All rights reserved.

CASE Expression

CASE expressions allow you to use the IF-THEN-ELSE logic in SQL statements without having to invoke procedures.

In a simple CASE expression, the Oracle server searches for the first WHEN . . . THEN pair for which *expr* is equal to *comparison_expr* and returns *return_expr*. If none of the WHEN . . . THEN pairs meet this condition, and if an ELSE clause exists, then the Oracle server returns *else_expr*. Otherwise, the Oracle server returns a null. You cannot specify the literal NULL for all the *return_exprs* and the *else_expr*.

All of the expressions (*expr*, *comparison_expr*, and *return_expr*) must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2.

Using the CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
SELECT last_name, job_id, salary,  
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary  
                 WHEN 'ST_CLERK' THEN 1.15*salary  
                 WHEN 'SA_REP' THEN 1.20*salary  
       ELSE salary END "REVISED_SALARY"  
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
...				

ORACLE

Using the CASE Expression

In the SQL statement in the slide, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be written with the DECODE function.

This is an example of a searched CASE expression. In a searched CASE expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an ELSE clause exists, the return expression in the ELSE clause is returned; otherwise, a NULL is returned.

```
SELECT last_name, salary,  
       (CASE WHEN salary<5000 THEN 'Low'  
            WHEN salary<10000 THEN 'Medium'  
            WHEN salary<20000 THEN 'Good'  
            ELSE 'Excellent'  
       END) qualified_salary  
FROM employees;
```


DECODE Function

Facilitates conditional inquiries by doing the work of a `CASE` expression or an `IF-THEN-ELSE` statement:

```
DECODE(col|expression, search1, result1  
      [, search2, result2, ..., ]  
      [, default])
```

ORACLE

DECODE Function

The `DECODE` function decodes an expression in a way similar to the `IF-THEN-ELSE` logic that is used in various languages. The `DECODE` function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

Using the DECODE Function

```
SELECT last name, job id, salary,  
       DECODE(job_id, 'IT_PROG', 1.10*salary,  
               'ST_CLERK', 1.15*salary,  
               'SA_REP', 1.20*salary,  
               salary)  
       REVISED_SALARY  
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
6 Lorentz	IT_PROG	4200	4620
7 Mourgos	ST_MAN	5800	5800
8 Rajs	ST_CLERK	3500	4025
...			
13 Abel	SA_REP	11000	13200
14 Taylor	SA_REP	8600	10320
...			

ORACLE

4 - 40

Copyright © 2007, Oracle. All rights reserved.

Using the DECODE Function

In the SQL statement in the slide, the value of `JOB_ID` is tested. If `JOB_ID` is `IT_PROG`, the salary increase is 10%; if `JOB_ID` is `ST_CLERK`, the salary increase is 15%; if `JOB_ID` is `SA_REP`, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be expressed in pseudocode as an IF-THEN-ELSE statement:

```
IF job_id = 'IT_PROG'      THEN salary = salary*1.10  
IF job_id = 'ST_CLERK'    THEN salary = salary*1.15  
IF job_id = 'SA_REP'      THEN salary = salary*1.20  
ELSE salary = salary
```

Using the DECODE Function

Display the applicable tax rate for each employee in department 80:

```
SELECT last name, salary,
       DECODE (TRUNC(salary/2000, 0),
              0, 0.00,
              1, 0.09,
              2, 0.20,
              3, 0.30,
              4, 0.40,
              5, 0.42,
              6, 0.44,
              0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

ORACLE

4 - 41

Copyright © 2007, Oracle. All rights reserved.

Using the DECODE Function (continued)

This slide shows another example using the DECODE function. In this example, you determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

<i>Monthly Salary Range</i>	<i>Tax Rate</i>
\$0.00–1,999.99	00%
\$2,000.00–3,999.99	09%
\$4,000.00–5,999.99	20%
\$6,000.00–7,999.99	30%
\$8,000.00–9,999.99	40%
\$10,000.00–11,999.99	42%
\$12,200.00–13,999.99	44%
\$14,000.00 or greater	45%

	LAST_NAME	SALARY	TAX_RATE
1	Zlotkey	10500	0.42
2	Abel	11000	0.42
3	Taylor	8600	0.4

Summary

In this lesson, you should have learned how to:

- Alter date formats for display using functions
- Convert column data types using functions
- Use `NVL` functions
- Use `IF-THEN-ELSE` logic and other conditional expressions in a `SELECT` statement

ORACLE

4 - 42

Copyright © 2007, Oracle. All rights reserved.

Summary

Remember the following:

- Conversion functions can convert character, date, and numeric values: `TO_CHAR`, `TO_DATE`, `TO_NUMBER`
- There are several functions that pertain to nulls, including `NVL`, `NVL2`, `NULLIF`, and `COALESCE`.
- `IF-THEN-ELSE` logic can be applied within a SQL statement by using the `CASE` expression or the `DECODE` function.

Practice 4: Overview

This practice covers the following topics:

- Creating queries that use `TO_CHAR`, `TO_DATE`, and other `DATE` functions
- Creating queries that use conditional expressions such as `DECODE` and `CASE`

ORACLE

4 - 43


Copyright © 2007, Oracle. All rights reserved.

Practice 4: Overview

This practice provides a variety of exercises using `TO_CHAR` and `TO_DATE` functions, and conditional expressions such as `DECODE` and `CASE`. Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.

Practice 4



1. 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.

	 Dream Salaries
1	King earns \$24,000.00 monthly but wants \$72,000.00.
2	Kochhar earns \$17,000.00 monthly but wants \$51,000.00.
3	De Haan earns \$17,000.00 monthly but wants \$51,000.00.
4	Hunold earns \$9,000.00 monthly but wants \$27,000.00.
5	Ernst earns \$6,000.00 monthly but wants \$18,000.00.

• • •

19	Higgins earns \$12,000.00 monthly but wants \$36,000.00.
20	Gietz earns \$8,300.00 monthly but wants \$24,900.00.

2. 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."

	 LAST_NAME	HIRE_DATE	 REVIEW
1	King	17-JUN-87	Monday, the Twenty-First of December, 1987
2	Kochhar	21-SEP-89	Monday, the Twenty-Sixth of March, 1990
3	De Haan	13-JAN-93	Monday, the Nineteenth of July, 1993
4	Hunold	03-JAN-90	Monday, the Ninth of July, 1990
5	Ernst	21-MAY-91	Monday, the Twenty-Fifth of November, 1991

• • •

19	Higgins	07-JUN-94	Monday, the Twelfth of December, 1994
20	Gietz	07-JUN-94	Monday, the Twelfth of December, 1994

Practice 4 (continued)

3. 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.

	LAST_NAME	HIRE_DATE	DAY
1	Grant	24-MAY-99	MONDAY
2	Gietz	07-JUN-94	TUESDAY
3	Taylor	24-MAR-98	TUESDAY
4	Higgins	07-JUN-94	TUESDAY
5	Rajs	17-OCT-95	TUESDAY

...

19	Lorentz	07-FEB-99	SUNDAY
20	Fay	17-AUG-97	SUNDAY

4. 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.

	LAST_NAME	COMM
1	King	No Commission
2	Kochhar	No Commission
3	De Haan	No Commission
4	Hunold	No Commission
5	Ernst	No Commission
6	Lorentz	No Commission

...

12	Zlotkey	.2
13	Abel	.3
14	Taylor	.2
15	Grant	.15
16	Whalen	No Commission
17	Hartstein	No Commission
18	Fay	No Commission
19	Higgins	No Commission
20	Gietz	No Commission

Practice 4 (continued)

If you have time, complete the following exercises:

- 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

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0

...

18	ST_CLERK	E
19	ST_CLERK	E
20	ST_MAN	B

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

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0

...

18	ST_CLERK	E
19	ST_CLERK	E
20	ST_MAN	B

5

Reporting Aggregated Data Using the Group Functions

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Identify the available group functions
- Describe the use of group functions
- Group data by using the `GROUP BY` clause
- Include or exclude grouped rows by using the `HAVING` clause

ORACLE

5 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

This lesson further addresses functions. It focuses on obtaining summary information (such as averages) for groups of rows. It discusses how to group rows in a table into smaller sets and how to specify search criteria for groups of rows.

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

What Are Group Functions?

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

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Maximum salary in
EMPLOYEES table

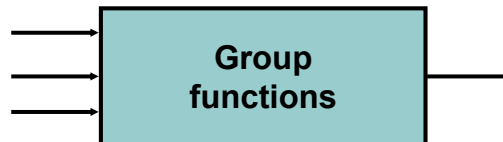
MAX(SALARY)
24000

What Are Group Functions?

Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may comprise the entire table or the table split into groups.

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



ORACLE

5 - 5

Copyright © 2007, Oracle. All rights reserved.

Types of Group Functions

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG ([DISTINCT <u>ALL</u>] <i>n</i>)	Average value of <i>n</i> , ignoring null values
COUNT ({ * [DISTINCT <u>ALL</u>] <i>expr</i> })	Number of rows, where <i>expr</i> evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX ([DISTINCT <u>ALL</u>] <i>expr</i>)	Maximum value of <i>expr</i> , ignoring null values
MIN ([DISTINCT <u>ALL</u>] <i>expr</i>)	Minimum value of <i>expr</i> , ignoring null values
STDDEV ([DISTINCT <u>ALL</u>] <i>x</i>)	Standard deviation of <i>n</i> , ignoring null values
SUM ([DISTINCT <u>ALL</u>] <i>n</i>)	Sum values of <i>n</i> , ignoring null values
VARIANCE ([DISTINCT <u>ALL</u>] <i>x</i>)	Variance of <i>n</i> , ignoring null values

Group Functions: Syntax

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

ORACLE

5 - 6

Copyright © 2007, Oracle. All rights reserved.

Group Functions: Syntax

The group function is placed after the `SELECT` keyword. You may have multiple group functions separated by commas.

Guidelines for using the group functions:

- `DISTINCT` makes the function consider only nonduplicate values; `ALL` makes it consider every value, including duplicates. The default is `ALL` and therefore does not need to be specified.
- The data types for the functions with an `expr` argument may be `CHAR`, `VARCHAR2`, `NUMBER`, or `DATE`.
- All group functions ignore null values. To substitute a value for null values, use the `NVL`, `NVL2`, or `COALESCE` functions.

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

	A	A	A	A
	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

ORACLE

Using the AVG and SUM Functions

You can use the AVG, SUM, MIN, and MAX functions against the columns that can store numeric data. The example in the slide displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.

Using the MIN and MAX Functions

You can use `MIN` and `MAX` for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	29-JAN-00

Using the MIN and MAX Functions

You can use the `MAX` and `MIN` functions for numeric, character, and date data types. The example in the slide displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetic list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

	MIN(LAST_NAME)	MAX(LAST_NAME)
1	Abel	Zlotkey

Note: The `AVG`, `SUM`, `VARIANCE`, and `STDDEV` functions can be used only with numeric data types. `MAX` and `MIN` cannot be used with `LOB` or `LONG` data types.

Using the COUNT Function

COUNT (*) returns the number of rows in a table:

1

```
SELECT COUNT (*)
FROM employees
WHERE department_id = 50;
```

	COUNT(*)
1	5

COUNT (expr) returns the number of rows with non-null values for expr:

2

```
SELECT COUNT (commission_pct)
FROM employees
WHERE department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

ORACLE

5 - 9

Copyright © 2007, Oracle. All rights reserved.

Using the COUNT Function

The COUNT function has three formats:

- COUNT (*)
- COUNT (expr)
- COUNT (DISTINCT expr)

COUNT (*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT (*) returns the number of rows that satisfy the condition in the WHERE clause.

In contrast, COUNT (expr) returns the number of non-null values that are in the column identified by expr.

COUNT (DISTINCT expr) returns the number of unique, non-null values that are in the column identified by expr.

Examples:

1. The example in the slide displays the number of employees in department 50.
2. The example in the slide displays the number of employees in department 80 who can earn a commission.

Using the DISTINCT Keyword

- `COUNT (DISTINCT expr)` returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the `EMPLOYEES` table:

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

ORACLE

Using the DISTINCT Keyword

Use the `DISTINCT` keyword to suppress the counting of any duplicate values in a column.

The example in the slide displays the number of distinct department values that are in the `EMPLOYEES` table.

Group Functions and Null Values

Group functions ignore null values in the column:

1 `SELECT AVG(commission_pct)
FROM employees;`

	AVG(COMMISSION_PCT)
1	0.2125

The NVL function forces group functions to include null values:

2 `SELECT AVG(NVL(commission_pct, 0))
FROM employees;`

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

ORACLE

Group Functions and Null Values

All group functions ignore null values in the column.

However, the NVL function forces group functions to include null values.

Examples:

1. The average is calculated based on *only* those rows in the table in which a valid value is stored in the COMMISSION_PCT column. The average is calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMMISSION_PCT column. The average is calculated as the total commission that is paid to all employees divided by the total number of employees in the company (20).

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	5800	3500
5	50	2500	
6	50	2600	
7	50	3100	
8	50	3500	6400
9	60	4200	
10	60	6000	
11	60	9000	10033
12	80	11000	
13	80	10500	
14	80	8600	
...			
19	110	12000	
20	(null)	7000	

**Average salary in
EMPLOYEES table for
each department**

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

ORACLE

Creating Groups of Data

Until this point in our discussion, all group functions have treated the table as one large group of information. At times, however, you need to divide the table of information into smaller groups. This can be done by using the `GROUP BY` clause.

Creating Groups of Data: GROUP BY Clause Syntax

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

You can divide rows in a table into smaller groups by using the GROUP BY clause.

ORACLE

5 - 14

Copyright © 2007, Oracle. All rights reserved.

Creating Groups of Data: GROUP BY Clause Syntax

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use the group functions to return summary information for each group.

In the syntax:

group_by_expression specifies columns whose values determine the basis for grouping rows

Guidelines

- If you include a group function in a SELECT clause, you cannot select individual results as well, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the *columns* in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.

Using the GROUP BY Clause

All columns in the `SELECT` list that are not in group functions must be in the `GROUP BY` clause.

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

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.3333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.3333333333...
7	60	6400
8	10	4400

ORACLE

5 - 15

Copyright © 2007, Oracle. All rights reserved.

Using the GROUP BY Clause

When using the `GROUP BY` clause, make sure that all columns in the `SELECT` list that are not group functions are included in the `GROUP BY` clause. The example in the slide displays the department number and the average salary for each department. Here is how this `SELECT` statement, containing a `GROUP BY` clause, is evaluated:

- The `SELECT` clause specifies the columns to be retrieved, as follows:
 - Department number column in the `EMPLOYEES` table
 - The average of all salaries in the group that you specified in the `GROUP BY` clause
- The `FROM` clause specifies the tables that the database must access: the `EMPLOYEES` table
- The `WHERE` clause specifies the rows to be retrieved. Because there is no `WHERE` clause, all rows are retrieved by default.
- The `GROUP BY` clause specifies how the rows should be grouped. The rows are grouped by department number, so the `AVG` function that is applied to the salary column calculates the average salary for each department.

Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

```
SELECT  AVG(salary)
FROM    employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	19333.333333333333333333333333...
3	9500
4	10150
5	3500
6	10033.333333333333333333333333...
7	6400
8	4400

ORACLE

5 - 16

Copyright © 2007, Oracle. All rights reserved.

Using the GROUP BY Clause (continued)

The GROUP BY column does not have to be in the SELECT clause. For example, the SELECT statement in the slide displays the average salaries for each department without displaying the respective department numbers. Without the department numbers, however, the results do not look meaningful.

You can also use the group function in the ORDER BY clause:

```
SELECT  department_id, AVG(salary)
FROM    employees
GROUP BY department_id
ORDER BY AVG(salary);
```

	DEPARTMENT_ID	AVG(SALARY)
1	50	3500
2	10	4400
3	60	6400
...		
7	110	10150
8	90	19333.333333333333333333333333...

Grouping by More than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_MAN	5800
5	50	ST_CLERK	2500
6	50	ST_CLERK	2600
7	50	ST_CLERK	3100
8	50	ST_CLERK	3500
9	60	IT_PROG	4200
10	60	IT_PROG	6000
11	60	IT_PROG	9000
12	80	SA_REP	11000
13	80	SA_MAN	10500
14	80	SA_REP	8600
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the **EMPLOYEES** table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

Grouping by More than One Column

Sometimes you need to see results for groups within groups. The slide shows a report that displays the total salary that is paid to each job title in each department.

The **EMPLOYEES** table is grouped first by the department number and then by the job title within that grouping. For example, the four stock clerks in department 50 are grouped together, and a single result (total salary) is produced for all stock clerks in the group.

Using the GROUP BY Clause on Multiple Columns

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

5 - 18

Copyright © 2007, Oracle. All rights reserved.

Using the Group By Clause on Multiple Columns

You can return summary results for groups and subgroups by listing more than one GROUP BY column. You can determine the default sort order of the results by the order of the columns in the GROUP BY clause. In the example in the slide, the SELECT statement containing a GROUP BY clause is evaluated as follows:

- The SELECT clause specifies the column to be retrieved:
 - Department number in the EMPLOYEES table
 - Job ID in the EMPLOYEES table
 - The sum of all salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMPLOYEES table
- The GROUP BY clause specifies how you must group the rows:
 - First, the rows are grouped by the department number.
 - Second, the rows are grouped by job ID in the department number groups.

So the SUM function is applied to the salary column for all job IDs in each department number group.

Illegal Queries Using Group Functions

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause:

```
SELECT department_id, COUNT(last_name)
FROM employees;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

A `GROUP BY` clause must be added to count the last names for each `department_id`.

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

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

Either add `job_id` in the `GROUP BY` or remove the `job_id` column from the `SELECT` list.

ORACLE

5 - 19

Copyright © 2007, Oracle. All rights reserved.

Illegal Queries Using Group Functions

Whenever you use a mixture of individual items (`DEPARTMENT_ID`) and group functions (`COUNT`) in the same `SELECT` statement, you must include a `GROUP BY` clause that specifies the individual items (in this case, `DEPARTMENT_ID`). If the `GROUP BY` clause is missing, then the error message “not a single-group group function” appears and an asterisk (*) points to the offending column. You can correct the error in the first example in the slide by adding the `GROUP BY` clause:

```
SELECT department_id, count(last_name)
FROM employees
GROUP BY department_id;
```

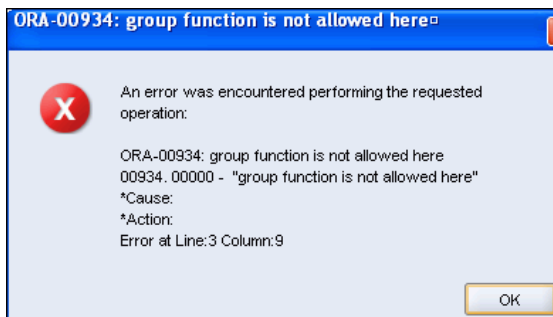
Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause. In the second example in the slide, `job_id` is neither in the `GROUP BY` clause nor is it being used by a group function, so there is a “not a `GROUP BY` expression” error. You can correct the error in the second slide example by adding `job_id` in the `GROUP BY` clause.

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

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;
```



**Cannot use the
`WHERE` clause to
restrict groups**

ORACLE

5 - 20

Copyright © 2007, Oracle. All rights reserved.

Illegal Queries Using Group Functions (continued)

The `WHERE` clause cannot be used to restrict groups. The `SELECT` statement in the example in the slide results in an error because it uses the `WHERE` clause to restrict the display of the average salaries of those departments that have an average salary greater than \$8,000.

However, you can correct the error in the example by using the `HAVING` clause to restrict groups:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 8000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.3333333333333333...
2	20	9500
3	110	10150
4	80	10033.333333333333333...

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than \$10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

ORACLE

Restricting Group Results

You use the `HAVING` clause to restrict groups in the same way that you use the `WHERE` clause to restrict the rows that you select. To find the maximum salary in each of the departments that have a maximum salary greater than \$10,000, you need to do the following:

1. Find the average salary for each department by grouping by department number.
2. Restrict the groups to those departments with a maximum salary greater than \$10,000.

Restricting Group Results with the HAVING Clause

When you use the `HAVING` clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the `HAVING` clause are displayed.

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING  group_condition]
[ORDER BY column];
```

ORACLE

5 - 22

Copyright © 2007, Oracle. All rights reserved.

Restricting Group Results with the HAVING Clause

You use the `HAVING` clause to specify the groups that are to be displayed, thus further restricting the groups on the basis of aggregate information.

In the syntax, *group_condition* restricts the groups of rows returned to those groups for which the specified condition is true.

The Oracle server performs the following steps when you use the `HAVING` clause:

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the `HAVING` clause are displayed.

The `HAVING` clause can precede the `GROUP BY` clause, but it is recommended that you place the `GROUP BY` clause first because it is more logical. Groups are formed and group functions are calculated before the `HAVING` clause is applied to the groups in the `SELECT` list.

Using the HAVING Clause

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary) > 10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

ORACLE

5 - 23

Copyright © 2007, Oracle. All rights reserved.

Using the HAVING Clause

The example in the slide displays the department numbers and maximum salaries for those departments with a maximum salary greater than \$10,000.

You can use the GROUP BY clause without using a group function in the SELECT list. If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

The following example displays the department numbers and average salaries for those departments with a maximum salary greater than \$10,000:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING max(salary) > 10000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333...
2	20	9500
3	110	10150
4	80	10033.333333333333...

Using the HAVING Clause

```
SELECT  job_id, SUM(salary) PAYROLL
FROM    employees
WHERE   job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING  SUM(salary) > 13000
ORDER BY SUM(salary);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

ORACLE

Using the HAVING Clause (continued)

The example in the slide displays the job ID and total monthly salary for each job that has a total payroll exceeding \$13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

Nesting Group Functions

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.333

ORACLE

Nesting Group Functions

Group functions can be nested to a depth of two functions. The example in the slide calculates the average salary for each `department_id` and then displays the maximum average salary.

Note that `GROUP BY` clause is mandatory when nesting group functions.

Summary

In this lesson, you should have learned how to:

- Use the group functions `COUNT`, `MAX`, `MIN`, `SUM`, and `AVG`
- Write queries that use the `GROUP BY` clause
- Write queries that use the `HAVING` clause

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING     group_condition]
[ORDER BY  column];
```

ORACLE

5 - 27

Copyright © 2007, Oracle. All rights reserved.

Summary

There are several group functions available in SQL, such as:

`AVG`, `COUNT`, `MAX`, `MIN`, `SUM`, `STDDEV`, and `VARIANCE`

You can create subgroups by using the `GROUP BY` clause. Further, groups can be restricted using the `HAVING` clause.

Place the `HAVING` and `GROUP BY` clauses after the `WHERE` clause in a statement. The order of the `GROUP BY` and `HAVING` clauses following the `WHERE` clause is not important. Place the `ORDER BY` clause at the end.

The Oracle server evaluates the clauses in the following order:

1. If the statement contains a `WHERE` clause, the server establishes the candidate rows.
2. The server identifies the groups that are specified in the `GROUP BY` clause.
3. The `HAVING` clause further restricts result groups that do not meet the group criteria in the `HAVING` clause.

Note: For a complete list of the group functions, see *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Practice 5: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve more than one result
- Restricting groups by using the `HAVING` clause

ORACLE

5 - 28

Copyright © 2007, Oracle. All rights reserved.

Practice 5: Overview

At the end of this practice, you should be familiar with using group functions and selecting groups of data.

Practice 5

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

- Group functions work across many rows to produce one result per group.
True/False
- Group functions include nulls in calculations.
True/False
- The WHERE clause restricts rows before inclusion in a group calculation.
True/False

The HR department needs the following reports:

- Find the highest, lowest, sum, and average salary of all employees. Label the columns as Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Save your SQL statement as lab_05_04.sql. Run the query.

	Maximum	Minimum	Sum	Average
1	24000	2500	175500	8775

- Modify the query in lab_05_04.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab_05_04.sql as lab_05_05.sql. Run the statement in lab_05_05.sql.

	JOB_ID	Maximum	Minimum	Sum	Average
1	IT_PROG	9000	4200	19200	6400
2	AC_MGR	12000	12000	12000	12000
3	AC_ACCOUNT	8300	8300	8300	8300
4	ST_MAN	5800	5800	5800	5800
5	AD_ASST	4400	4400	4400	4400
6	AD_VP	17000	17000	34000	17000
7	SA_MAN	10500	10500	10500	10500
8	MK_MAN	13000	13000	13000	13000
9	AD PRES	24000	24000	24000	24000
10	SA_REP	11000	7000	26600	8867
11	MK_REP	6000	6000	6000	6000
12	ST_CLERK	3500	2500	11700	2925

Practice 5 (continued)

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

	JOB_ID	COUNT(*)
1	AC_ACCOUNT	1
2	AC_MGR	1
3	AD_ASST	1
4	AD_PRES	1
5	AD_VP	2
6	IT_PROG	3
7	MK_MAN	1
8	MK_REP	1
9	SA_MAN	1
10	SA_REP	3
11	ST_CLERK	4
12	ST_MAN	1

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_05_06.sql. Run the query. Enter IT_PROG when prompted.

	JOB_ID	COUNT(*)
1	IT_PROG	3

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

	Number of Managers
1	8

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

	DIFFERENCE
1	21500

Practice 5 (continued)

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.

	MANAGER_ID	MIN(SALARY)
1	102	9000
2	205	8300
3	149	7000

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

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

	TOTAL	1995	1996	1997	1998
1	20	1	2	2	3

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

	Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
1	IT_PROG	(null)	(null)	(null)	(null)	19200
2	AC_MGR	(null)	(null)	(null)	(null)	12000
3	AC_ACCOUNT	(null)	(null)	(null)	(null)	8300
4	ST_MAN	(null)	5800	(null)	(null)	5800
5	AD_ASST	(null)	(null)	(null)	(null)	4400
6	AD_VP	(null)	(null)	(null)	34000	34000
7	SA_MAN	(null)	(null)	10500	(null)	10500
8	MK_MAN	13000	(null)	(null)	(null)	13000
9	AD PRES	(null)	(null)	(null)	24000	24000
10	SA_REP	(null)	(null)	19600	(null)	26600
11	MK_REP	6000	(null)	(null)	(null)	6000
12	ST_CLERK	(null)	11700	(null)	(null)	11700

6

Displaying Data from Multiple Tables

ORACLE

Copyright © 2007, 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
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using outer joins
- Generate a Cartesian product of all rows from two or more tables

ORACLE

6 - 2

Copyright © 2007, 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 the section on *SQL Queries and Subqueries: Joins* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- **Types of JOINS and its syntax**
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

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	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

6 - 4

Copyright © 2007, Oracle. All rights reserved.

Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the example in the slide, 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.

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural joins:
 - NATURAL JOIN clause
 - USING clause
 - ON clause
- Outer joins:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cross joins

ORACLE

6 - 5

Copyright © 2007, Oracle. All rights reserved.

Types of Joins

To join tables, you can use a join syntax that is compliant with the SQL:1999 standard.

Note: Before the Oracle9i release, the join syntax was different from the American National Standards Institute (ANSI) standards. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in the prior releases. For detailed information about the proprietary join syntax, see Appendix C: Oracle Join Syntax.

Note: The following slide discusses the SQL:1999 join syntax.

Joining Tables Using SQL:1999 Syntax

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

```
SELECT  table1.column, table2.column
FROM    table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

ORACLE

6 - 6

Copyright © 2007, Oracle. All rights reserved.

Joining Tables Using SQL:1999 Syntax

In the syntax:

table1.column denotes the table and the column from which data is retrieved

NATURAL JOIN joins two tables based on the same column name

JOIN *table2* USING *column_name* performs an equijoin based on the column name

JOIN *table2* ON *table1.column_name = table2.column_name* performs an equijoin based on the condition in the ON clause

LEFT/RIGHT/FULL OUTER is used to perform outer joins

CROSS JOIN returns a Cartesian product from the two tables

For more information, see the section titled *SELECT* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
 - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

ORACLE

6 - 7

Copyright © 2007, Oracle. All rights reserved.

Qualifying Ambiguous Column Names

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column in the `SELECT` list could be from either the `DEPARTMENTS` table or the `EMPLOYEES` table. 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.

However, qualifying column names with table names can be time consuming, particularly if the table names are lengthy. Instead, you can use *table aliases*. 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.

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

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the `FROM` clause, 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.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

ORACLE

6 - 9

Copyright © 2007, Oracle. All rights reserved.

Creating Natural Joins

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

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

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,  
       location_id, city  
FROM   departments  
NATURAL JOIN locations ;
```

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

ORACLE

6 - 10

Copyright © 2007, Oracle. All rights reserved.

Retrieving Records with Natural Joins

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

Natural Joins with a `WHERE` Clause

Additional restrictions on a natural join are implemented by using a `WHERE` clause. The following example limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT  department_id, department_name,  
        location_id, city  
FROM    departments  
NATURAL JOIN locations  
WHERE   department_id IN (20, 50);
```

Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, natural join can be applied using the `USING` clause to specify the columns that should be used for an equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

ORACLE

6 - 11

Copyright © 2007, Oracle. All rights reserved.

Creating Joins with the USING Clause

Natural joins use all columns with matching names and data types to join the tables. The `USING` clause can be used to specify only those columns that should be used for an equijoin.

Joining Column Names

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80
...	...

DEPARTMENTS

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

Primary key

Foreign key

Joining Column Names

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 the 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 the USING Clause

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

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

ORACLE

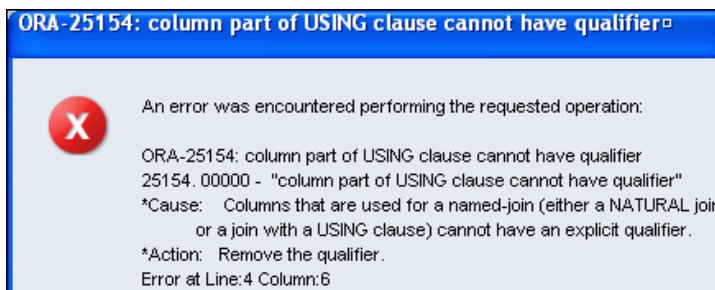
Retrieving Records with the USING Clause

In the example in the slide, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS tables are joined and thus the LOCATION_ID of the department where an employee works is shown.

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE  d.location_id = 1400;
```



ORACLE

6 - 14

Copyright © 2007, Oracle. All rights reserved.

Using Table Aliases with the USING clause

When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it. For example, in the query mentioned in the slide, you should not alias the `location_id` column in the WHERE clause because the column is used in the USING clause.

The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement. For example, the following statement is valid:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

Because, other columns that are common in both the tables, but not used in the USING clause, must be prefixed with a table alias otherwise you get the “column ambiguously defined” error.

In the following statement, `manager_id` is present in both the `employees` and `departments` table and if `manager_id` is not prefixed with a table alias, it gives a “column ambiguously defined” error.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

Creating Joins with the ON Clause

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

ORACLE

Creating Joins with the ON Clause

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

Retrieving Records with the ON Clause

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

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

...

ORACLE

Retrieving Records with the ON Clause

In this example, the `DEPARTMENT_ID` columns in the `EMPLOYEES` and `DEPARTMENTS` table are joined using the `ON` clause. Whenever a department ID in the `EMPLOYEES` table equals a department ID in the `DEPARTMENTS` table, the row is returned. The table alias is necessary to qualify the matching `column_names`.

You can also use the `ON` clause to join columns that have different names. The parenthesis around the joined columns as in the slide example, `(e.department_id = d.department_id)` is optional. So, even `ON e.department_id = d.department_id` will work.

Note: SQL Developer suffixes a `'_1'` to differentiate between the two `department_ids`.

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

ORACLE

6 - 17

Copyright © 2007, Oracle. All rights reserved.

Creating Three-Way Joins with the ON Clause

A three-way join is a join of three tables. In SQL:1999-compliant syntax, joins are performed from left to right. So, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

Note: The code example in the slide can also be accomplished with the USING clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id)
```

Applying Additional Conditions to a Join

Use the `AND` clause or the `WHERE` clause to apply additional conditions:

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149 ;
```

ORACLE

6 - 18

Copyright © 2007, Oracle. All rights reserved.

Applying Additional Conditions to a Join

You can apply additional conditions to the join.

The example shown performs a join on the `EMPLOYEES` and `DEPARTMENTS` tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the `ON` clause, you can add `AND` clauses. Alternatively, you can use a `WHERE` clause to apply additional conditions.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- **Self-join**
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Joining a Table to Itself

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	King	(null)
2	Kochhar	100
3	De Haan	100
4	Hunold	102
5	Ernst	103
6	Lorentz	103
7	Mourgos	100
8	Rajs	124
9	Davies	124
10	Matos	124

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

...

**MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.**

ORACLE

6 - 20

Copyright © 2007, 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, or perform a self-join. For example, to find the name of Lorentz's manager, you need to:

- 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 with 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.

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

ORACLE

6 - 21

Copyright © 2007, Oracle. All rights reserved.

Self-Joins Using the ON Clause

The ON clause can also be used to join columns that have different names, within the same table or in a different table.

The example shown is a self-join of the EMPLOYEES table, based on the EMPLOYEE_ID and MANAGER_ID columns.

Note: The parenthesis around the joined columns as in the slide example, (e.manager_id = m.employee_id) is **optional**. So, even ON e.manager_id = m.employee_id will work.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- **Nonequijoins**
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600
...		
19	Higgins	12000
20	Gietz	8300

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_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL. Hence, the GRADE_LEVEL column can be used to assign grades to each employee.

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. The SALARY column in the EMPLOYEES table ranges between the values in the LOWEST_SAL and HIGHEST_SAL columns of the JOB_GRADES table. Hence, each employee can be graded based on their salary. The relationship is obtained using an operator other than the equality (=) operator.

Retrieving Records with Nonequijoins

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

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

ORACLE

6 - 24

Copyright © 2007, Oracle. All rights reserved.

Retrieving Records with Nonequijoins

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_GRADES` 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 provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

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

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

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Returning Records with No Direct Match with Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos
...		
19	110	Higgins
20	110	Gietz

There are no employees in department 190.

ORACLE

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 EMPLOYEES and DEPARTMENTS tables, department ID 190 does not appear because there are no employees with that department ID recorded in the EMPLOYEES table. Therefore, instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) table is called a left (or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

ORACLE

6 - 27

Copyright © 2007, Oracle. All rights reserved.

INNER Versus OUTER Joins

Joining tables with the `NATURAL JOIN`, `USING`, or `ON` clauses results in an inner join. Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an outer join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of outer joins:

- `LEFT OUTER`
- `RIGHT OUTER`
- `FULL OUTER`

LEFT OUTER JOIN

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

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

LEFT OUTER JOIN

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

RIGHT OUTER JOIN

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

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
...			
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

ORACLE

RIGHT OUTER JOIN

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

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON     (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
...		
19 Taylor	80	Sales
20 Grant	(null)	(null)
21 (null)	190	Contracting

ORACLE

FULL OUTER JOIN

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

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoin
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

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.

ORACLE

6 - 32

Copyright © 2007, 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, therefore, always include a valid join condition unless you have a specific need to combine all rows from all tables.

However, 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	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60

...

19	Higgins	110
20	Gietz	110

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

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

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	90	1700
2	90	1700
3	90	1700
4	60	1700

...

159	110	1700
160	110	1700

ORACLE

Generating a Cartesian Product

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee last name and the department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition was 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.

Creating Cross Joins

- The `CROSS JOIN` clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
159	Whalen	Contracting
160	Zlotkey	Contracting

Creating Cross Joins

The example in the slide produces a Cartesian product of the `EMPLOYEES` and `DEPARTMENTS` tables.

Summary

In this lesson, you should have learned how to use joins to display data from multiple tables by using:

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

ORACLE

6 - 35

Copyright © 2007, Oracle. All rights reserved.

Summary

There are multiple ways to join tables.

Types of Joins

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

Cartesian Products

A Cartesian product results in the display of all combinations of rows. This is done by either omitting the `WHERE` clause or by specifying the `CROSS JOIN` clause.

Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.
- Table aliases are sometimes mandatory to avoid column ambiguity.

Practice 6: Overview

This practice covers the following topics:

- Joining tables using an equijoin
- Performing outer and self-joins
- Adding conditions

ORACLE

6 - 36

Copyright © 2007, Oracle. All rights reserved.

Practice 6: Overview

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999-compliant joins.

Practice 6

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.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	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	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 the employees.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping
9	Hunold	60	IT
10	Ernst	60	IT

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting

Practice 6 (continued)

- The HR department needs a report of employees in Toronto. Display the last name, job, department number, and the 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

- Create a report to display employees' last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Save your SQL statement as lab_06_04.sql. Run the query.

	Employee	EMP#	Manager	Mgr#
1	Kochhar	101	King	100
2	De Haan	102	King	100
3	Hunold	103	De Haan	102
4	Ernst	104	Hunold	103
5	Lorentz	107	Hunold	103
6	Mourgos	124	King	100
7	Rajs	141	Mourgos	124
8	Davies	142	Mourgos	124
9	Matos	143	Mourgos	124
10	Vargas	144	Mourgos	124

...

15	Whalen	200	Kochhar	101
16	Hartstein	201	King	100
17	Fay	202	Hartstein	201
18	Higgins	205	Kochhar	101
19	Gietz	206	Higgins	205

Practice 6 (continued)

- Modify `lab_06_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Save your SQL statement as `lab_06_05.sql`. Run the query in `lab_06_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	King	100	(null)	(null)
2	Kochhar	101	King	100
3	De Haan	102	King	100
4	Hunold	103	De Haan	102
5	Ernst	104	Hunold	103
6	Lorentz	107	Hunold	103
7	Mourgos	124	King	100
8	Rajs	141	Mourgos	124
9	Davies	142	Mourgos	124
10	Matos	143	Mourgos	124

...

18	Fay	202	Hartstein	201
19	Higgins	205	Kochhar	101
20	Gietz	206	Higgins	205

- 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_06_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
6	50	Davies	Vargas
7	50	Matos	Davies
8	50	Matos	Mourgos
9	50	Matos	Rajs
10	50	Matos	Vargas

...

42	110	Higgins	Gietz
----	-----	---------	-------

Practice 6 (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. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
Name                               Null    Type
-----
GRADE_LEVEL                        VARCHA2(3)
LOWEST_SAL                          NUMBER
HIGHEST_SAL                         NUMBER

3 rows selected
```

	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

...

18	De Haan	AD_VP	Executive	17000	E
19	King	AD PRES	Executive	24000	E

Practice 6 (continued)

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

8. The HR department wants to determine the names of all the employees who were 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	Lorentz	07-FEB-99
2	Mourgos	16-NOV-99
3	Matos	15-MAR-98
4	Vargas	09-JUL-98
5	Zlotkey	29-JAN-00
6	Taylor	24-MAR-98
7	Grant	24-MAY-99
8	Fay	17-AUG-97

9. The HR department needs to find the names and hire dates of all the employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_06_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



Using Subqueries to Solve Queries

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Define subqueries
- Describe the types of problems that the subqueries can solve
- List the types of subqueries
- Write single-row and multiple-row subqueries

ORACLE

7 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

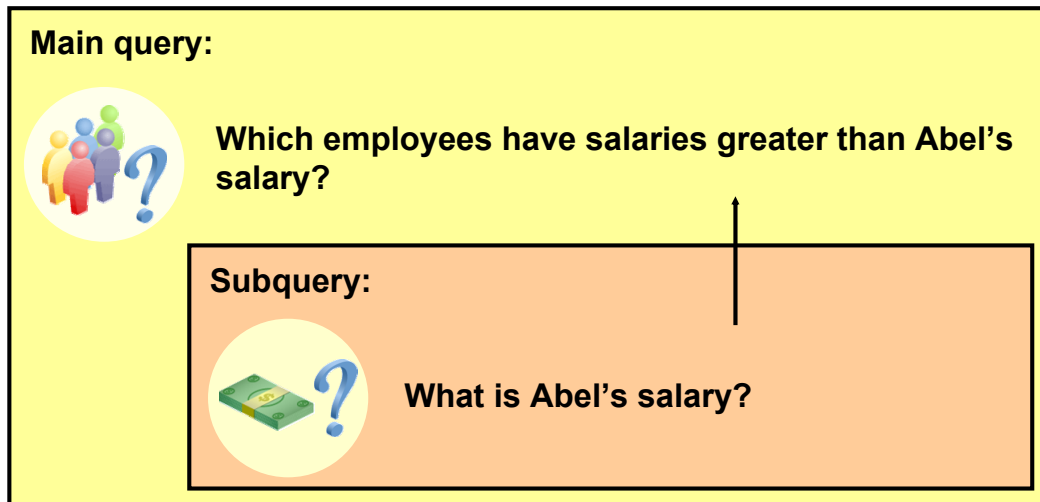
In this lesson, you learn about the more advanced features of the `SELECT` statement. You can write subqueries in the `WHERE` clause of another SQL statement to obtain values based on an unknown conditional value. This lesson also covers single-row subqueries and multiple-row subqueries.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - HAVING clause with subqueries
- Multiple-row subqueries
 - Use ALL or ANY operator
- Null values in a subquery

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?



ORACLE

7 - 4

Copyright © 2007, Oracle. All rights reserved.

Using a Subquery to Solve a Problem

Suppose you want to write a query to find out who earns a salary greater than Abel's salary.

To solve this problem, you need *two* queries: one to find how much Abel earns, and a second query to find who earns more than that amount.

You can solve this problem by combining the two queries, placing one query *inside* the other query.

The inner query (or *subquery*) returns a value that is used by the outer query (or *main query*). Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

Subquery Syntax

```
SELECT  select_list
FROM    table
WHERE   expr operator
       (SELECT    select_list
        FROM      table);
```

- The subquery (inner query) executes *before* the main query (outer query).
- The result of the subquery is used by the main query.

ORACLE

7 - 5

Copyright © 2007, Oracle. All rights reserved.

Subquery Syntax

A subquery is a `SELECT` statement that is embedded in the clause of another `SELECT` statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:

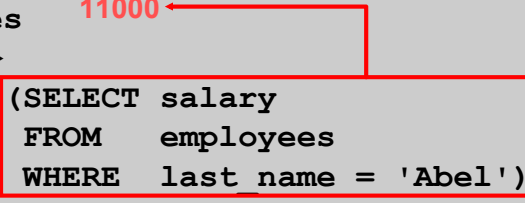
operator includes a comparison condition such as `>`, `=`, or `IN`

Note: Comparison conditions fall into two classes: single-row operators (`>`, `=`, `>=`, `<`, `<>`, `<=`) and multiple-row operators (`IN`, `ANY`, `ALL`).

The subquery is often referred to as a nested `SELECT`, sub-`SELECT`, or inner `SELECT` statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query.

Using a Subquery

```
SELECT last_name, salary
FROM employees
WHERE salary >
  (SELECT salary
   FROM employees
   WHERE last_name = 'Abel');
```



	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000
5	Higgins	12000

ORACLE

Using a Subquery

In the slide, the inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than employee Abel.

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability (However, the subquery can appear on either side of the comparison operator.).
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

ORACLE

7 - 7

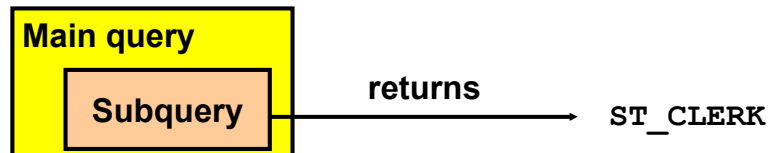
Copyright © 2007, Oracle. All rights reserved.

Guidelines for Using Subqueries

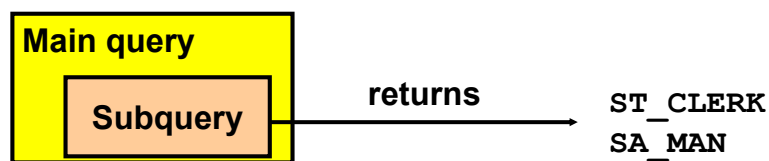
- A subquery must be enclosed in parentheses.
- Place the subquery on the right side of the comparison condition for readability. However, the subquery can appear on either side of the comparison operator.
- Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row operators.

Types of Subqueries

- Single-row subquery



- Multiple-row subquery



ORACLE

7 - 8

Copyright © 2007, Oracle. All rights reserved.

Types of Subqueries

- **Single-row subqueries:** Queries that return only one row from the inner SELECT statement
- **Multiple-row subqueries:** Queries that return more than one row from the inner SELECT statement

Note: There are also multiple-column subqueries, which are queries that return more than one column from the inner SELECT statement. These are covered in the *Oracle Database 11g: SQL Fundamentals II* course.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - HAVING clause with subqueries
- Multiple-row subqueries
 - Use ALL or ANY operator
- Null values in a subquery

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

ORACLE

7 - 10

Copyright © 2007, Oracle. All rights reserved.

Single-Row Subqueries

A single-row subquery is one that returns one row from the inner `SELECT` statement. This type of subquery uses a single-row operator. The slide gives a list of single-row operators.

Example:

Display the employees whose job ID is the same as that of employee 141:

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
      (SELECT job_id
       FROM employees
       WHERE employee_id = 141);
```

	LAST_NAME	JOB_ID
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = (SELECT job_id
                FROM employees
                WHERE last_name = 'Taylor')
AND salary > (SELECT salary
              FROM employees
              WHERE last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

ORACLE

7 - 11

Copyright © 2007, Oracle. All rights reserved.

Executing Single-Row Subqueries

A `SELECT` statement can be considered as a query block. The example in the slide displays employees who do the same job as “Taylor,” but earn more salary than him.

The example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results `SA_REP` and `8600`, respectively. The outer query block is then processed and uses the values that were returned by the inner queries to complete its search conditions.

Both inner queries return single values (`SA_REP` and `8600`, respectively), so this SQL statement is called a single-row subquery.

Note: The outer and inner queries can get data from different tables.

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM employees ← 2500
WHERE salary =
      (SELECT MIN(salary)
       FROM employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

ORACLE

7 - 12

Copyright © 2007, Oracle. All rights reserved.

Using Group Functions in a Subquery

You can display data from a main query by using a group function in a subquery to return a single row. The subquery is in parentheses and is placed after the comparison condition.

The example in the slide displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The MIN group function returns a single value (2500) to the outer query.

The HAVING Clause with Subqueries

- The Oracle server executes the subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > (SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
...		
7	10	4400

ORACLE

7 - 13

Copyright © 2007, Oracle. All rights reserved.

The HAVING Clause with Subqueries

You can use subqueries not only in the WHERE clause, but also in the HAVING clause. The Oracle server executes the subquery and the results are returned into the HAVING clause of the main query. The SQL statement in the slide displays all the departments that have a minimum salary greater than that of department 50.

Example:

Find the job with the lowest average salary.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
FROM employees
GROUP BY job_id);
```

	JOB_ID	AVG(SALARY)
1	ST_CLERK	2925

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
      (SELECT MIN(salary)
       FROM employees
       GROUP BY department_id);
```

ORA-01427: single-row subquery returns more than one ...



An error was encountered performing the requested operation:

ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"

*Cause:

*Action:

Error at Line:1

**Single-row operator
with multiple-row
subquery**

ORACLE

7 - 14

Copyright © 2007, Oracle. All rights reserved.

What Is Wrong with This Statement?

A common error with subqueries occurs when more than one row is returned for a single-row subquery.

In the SQL statement in the slide, the subquery contains a `GROUP BY` clause, which implies that the subquery will return multiple rows, one for each group that it finds. In this case, the results of the subquery are 4400, 6000, 2500, 4200, 7000, 17000, and 8300.

The outer query takes those results and uses them in its `WHERE` clause. The `WHERE` clause contains an equal (`=`) operator, a single-row comparison operator that expects only one value. The `=` operator cannot accept more than one value from the subquery and, therefore, generates the error.

To correct this error, change the `=` operator to `IN`.

No Rows Returned by the Inner Query

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
  (SELECT job_id
   FROM employees
   WHERE last_name = 'Haas');
```

0 rows selected

Subquery returns no rows because there is no employee named "Haas."

ORACLE

7 - 15

Copyright © 2007, Oracle. All rights reserved.

No Rows Returned by the Inner Query

A common problem with subqueries occurs when no rows are returned by the inner query.

In the SQL statement in the slide, the subquery contains a `WHERE` clause. Presumably, the intention is to find the employee whose name is Haas. The statement is correct, but selects no rows when executed.

Because, there is no employee named Haas. So the subquery returns no rows. The outer query takes the results of the subquery (null) and uses these results in its `WHERE` clause. The outer query finds no employee with a job ID equal to null, and so returns no rows. If a job existed with a value of null, the row is not returned because comparison of two null values yields a null; therefore, the `WHERE` condition is not true.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - `HAVING` clause with subqueries
- **Multiple-row subqueries**
 - Use `ALL` or `ANY` operator
- Null values in a subquery

ORACLE

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to <code>FALSE</code> if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to <code>TRUE</code> if the query returns no rows.

ORACLE

7 - 17

Copyright © 2007, Oracle. All rights reserved.

Multiple-Row Subqueries

Subqueries that return more than one row are called multiple-row subqueries. You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
                 FROM employees
                 GROUP BY department_id);
```

Example:

Find the employees who earn the same salary as the minimum salary for each department.

The inner query is executed first, producing a query result. The main query block is then processed and uses the values that were returned by the inner query to complete its search condition. In fact, the main query appears to the Oracle server as follows:

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300,
                8600, 17000);
```

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144 Vargas	ST_CLERK	2500
2	143 Matos	ST_CLERK	2600
3	142 Davies	ST_CLERK	3100
4	141 Raju	ST_CLERK	3500
5	200 Whalen	AD_ASST	4400

...

9	206 Gietz	AC_ACCOUNT	8300
10	176 Taylor	SA_REP	8600

ORACLE

7 - 18

Copyright © 2007, Oracle. All rights reserved.

Using the ANY Operator in Multiple-Row Subqueries

The ANY operator (and its synonym, the SOME operator) compares a value to *each* value returned by a subquery. The slide example displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is \$9,000.

<ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees          9000, 6000, 4200
WHERE  salary < ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141 Rajs	ST_CLERK	3500
2	142 Davies	ST_CLERK	3100
3	143 Matos	ST_CLERK	2600
4	144 Vargas	ST_CLERK	2500

ORACLE

7 - 19

Copyright © 2007, Oracle. All rights reserved.

Using the ALL Operator in Multiple-Row Subqueries

The ALL operator compares a value to *every* value returned by a subquery. The example in the slide displays employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG.

>ALL means more than the maximum and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - `HAVING` clause with subqueries
- Multiple-row subqueries
 - Use `ALL` or `ANY` operator
- Null values in a subquery

ORACLE

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                                (SELECT mgr.manager_id
                                FROM   employees mgr);
```

```
0 rows selected
```

ORACLE

7 - 21

Copyright © 2007, Oracle. All rights reserved.

Null Values in a Subquery

The SQL statement in the slide attempts to display all the employees who do not have any subordinates. Logically, this SQL statement should have returned 12 rows. However, the SQL statement does not return any rows. One of the values returned by the inner query is a null value, and, therefore, the entire query returns no rows.

The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL.

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id IN
                                (SELECT mgr.manager_id
                                FROM   employees mgr);
```

Null Values in a Subquery (continued)

Alternatively, a WHERE clause can be included in the subquery to display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN
      (SELECT manager_id
       FROM employees
       WHERE manager_id IS NOT NULL);
```


Summary

In this lesson, you should have learned how to:

- Identify when a subquery can help solve a problem
- Write subqueries when a query is based on unknown values

```
SELECT  select_list
FROM    table
WHERE   expr operator
        (SELECT select_list
         FROM   table);
```

ORACLE

Summary

In this lesson, you should have learned how to use subqueries. A subquery is a `SELECT` statement that is embedded in the clause of another SQL statement. Subqueries are useful when a query is based on a search criterion with unknown intermediate values.

Subqueries have the following characteristics:

- Can pass one row of data to a main statement that contains a single-row operator, such as `=`, `<>`, `>`, `>=`, `<`, or `<=`
- Can pass multiple rows of data to a main statement that contains a multiple-row operator, such as `IN`
- Are processed first by the Oracle server, after which the `WHERE` or `HAVING` clause uses the results
- Can contain group functions

Practice 7: Overview

This practice covers the following topics:

- Creating subqueries to query values based on unknown criteria
- Using subqueries to find out the values that exist in one set of data and not in another

ORACLE

7 - 24

Copyright © 2007, Oracle. All rights reserved.

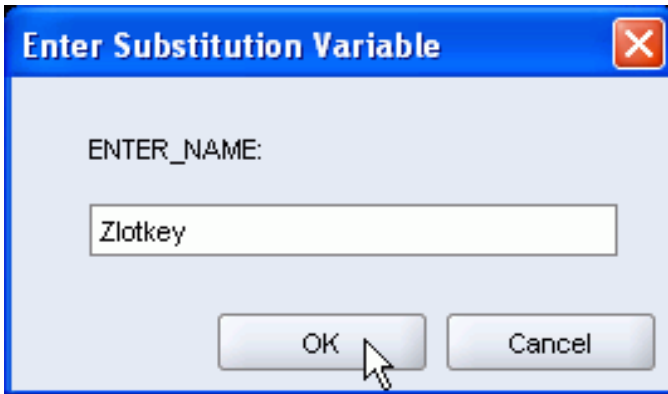
Practice 7: Overview

In this practice, you write complex queries using nested `SELECT` statements.

For practice questions, you may want to create the inner query first. Make sure that it runs and produces the data that you anticipate before you code the outer query.

Practice 7

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 they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).



	LAST_NAME	HIRE_DATE
1	Abel	11-MAY-96
2	Taylor	24-MAR-98

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 order of ascending salary.

	EMPLOYEE_ID	LAST_NAME	SALARY
1	103	Hunold	9000
2	149	Zlotkey	10500
3	174	Abel	11000
4	205	Higgins	12000
5	201	Hartstein	13000
6	101	Kochhar	17000
7	102	De Haan	17000
8	100	King	24000

Practice 7 (continued)

- 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 the letter “u.” Save your SQL statement as `lab_07_03.sql`. Run your query.

	EMPLOYEE_ID	LAST_NAME
1	124	Mourgos
2	141	Rajs
3	142	Davies
4	143	Matos
5	144	Vargas
6	103	Hunold
7	104	Ernst
8	107	Lorentz

- 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.

	LAST_NAME	DEPARTMENT_ID	JOB_ID
1	Whalen	10	AD_ASST
2	King	90	AD_PRES
3	Kochhar	90	AD_VP
4	De Haan	90	AD_VP
5	Higgins	110	AC_MGR
6	Gietz	110	AC_ACCOUNT

Modify the query so that the user is prompted for a location ID. Save this to a file named `lab_07_04.sql`.

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

	LAST_NAME	SALARY
1	Kochhar	17000
2	De Haan	17000
3	Mourgos	5800
4	Zlotkey	10500
5	Hartstein	13000

Practice 7 (continued)

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

	DEPARTMENT_ID	LAST_NAME	JOB_ID
1	90	King	AD_PRES
2	90	Kochhar	AD_VP
3	90	De Haan	AD_VP

If you have the time, complete the following exercise:

7. Modify the query in `lab_07_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_07_03.sql` as `lab_07_07.sql`. Run the statement in `lab_07_07.sql`.

	EMPLOYEE_ID	LAST_NAME	SALARY
1	103	Hunold	9000

8

Using the Set Operators

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

ORACLE

8 - 2

Copyright © 2007, Oracle. All rights reserved.

Objectives

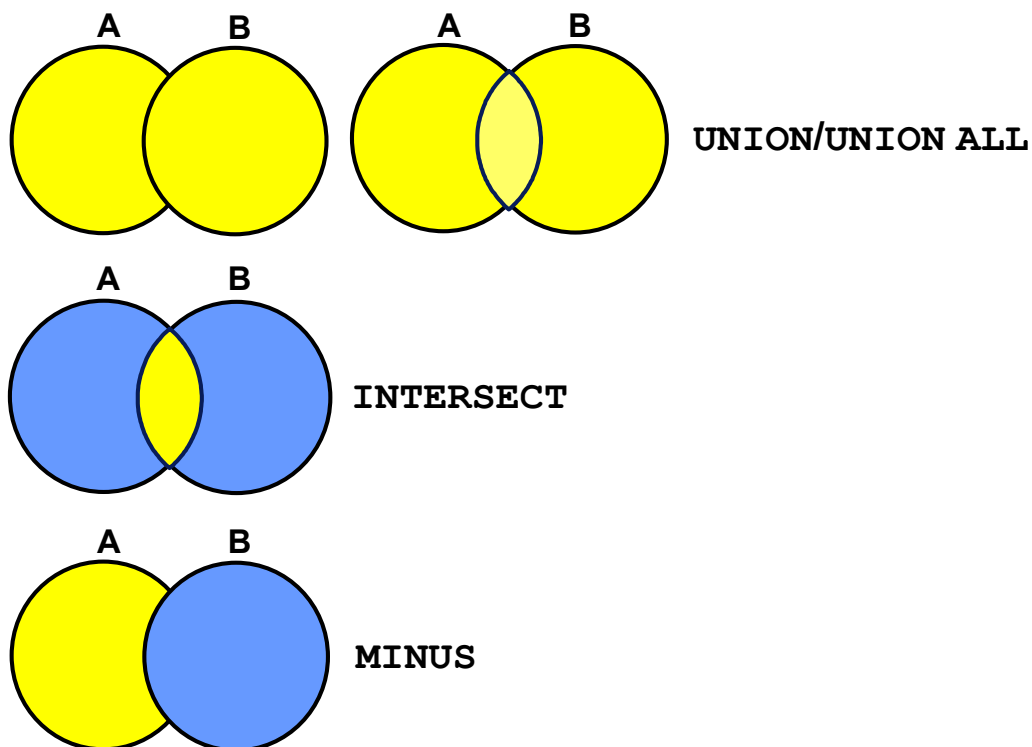
In this lesson, you learn how to write queries by using set operators.

Lesson Agenda

- **Set Operators: Types and guidelines**
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Set Operators



8 - 4

Copyright © 2007, Oracle. All rights reserved.

ORACLE

Set Operators

Set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound queries*.

Operator	Returns
UNION	Rows from both queries after eliminating duplications
UNION ALL	Rows from both queries, including all duplications
INTERSECT	Rows that are common to both queries
MINUS	Rows in the first query that are not present in the second query

All set operators have equal precedence. If a SQL statement contains multiple set operators, the Oracle server evaluates them from left (top) to right (bottom)—if no parentheses explicitly specify another order. You should use parentheses to specify the order of evaluation explicitly in queries that use the INTERSECT operator with other set operators.

Set Operator Guidelines

- The expressions in the `SELECT` lists must match in number.
- The data type of each column in the second query must match the data type of its corresponding column in the first query.
- Parentheses can be used to alter the sequence of execution.
- `ORDER BY` clause can appear only at the very end of the statement.

ORACLE

8 - 5

Copyright © 2007, Oracle. All rights reserved.

Set Operator Guidelines

- The expressions in the `SELECT` lists of the queries must match in number and data type. Queries that use `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS` operators in their `WHERE` clause must have the same number and data type of columns in their `SELECT` list. The data type of the columns in `SELECT` list of the queries in the compound query may not be exactly the same. The column in second query must be in the same data type group (such as numeric or character) as the corresponding column in the first query.
- Set operators can be used in subqueries.
- You should use parentheses to specify the order of evaluation in queries that use the `INTERSECT` operator with other set operators. This ensures compliance with emerging SQL standards that will give the `INTERSECT` operator greater precedence than the other set operators.

The Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in `UNION ALL`.
- Column names from the first query appear in the result.
- The output is sorted in ascending order by default except in `UNION ALL`.

ORACLE

8 - 6

Copyright © 2007, Oracle. All rights reserved.

The Oracle Server and Set Operators

When a query uses set operators, the Oracle server eliminates duplicate rows automatically except in the case of the `UNION ALL` operator. The column names in the output are decided by the column list in the first `SELECT` statement. By default, the output is sorted in ascending order of the first column of the `SELECT` clause.

The corresponding expressions in the `SELECT` lists of the component queries of a compound query must match in number and data type. If component queries select character data, the data type of the return values is determined as follows:

- If both queries select values of `CHAR` data type, of equal length, then the returned values have the `CHAR` data type of that length. If the queries select values of `CHAR` with different lengths, then the returned value is `VARCHAR2` with the length of the larger `CHAR` value.
- If either or both of the queries select values of `VARCHAR2` data type, then the returned values have the `VARCHAR2` data type.

If component queries select numeric data, then the data type of the return values is determined by numeric precedence. If all queries select values of the `NUMBER` type, then the returned values have the `NUMBER` data type. In queries using set operators, the Oracle server does not perform implicit conversion across data type groups. Therefore, if the corresponding expressions of component queries resolve to both character data and numeric data, the Oracle server returns an error.

Lesson Agenda

- Set Operators: Types and guidelines
- **Tables used in this lesson**
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Tables Used in This Lesson

The tables used in this lesson are:

- **EMPLOYEES**: Provides details regarding all current employees
- **JOB_HISTORY**: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

ORACLE

8 - 8

Copyright © 2007, Oracle. All rights reserved.

Tables Used in This Lesson

Two tables are used in this lesson. They are the **EMPLOYEES** table and the **JOB_HISTORY** table.

You are already familiar with the **EMPLOYEES** table that stores employee details such as a unique identification number, email address, job identification (such as **ST_CLERK**, **SA_REP**, and so on), salary, manager and so on.

Some of the employees have been with the company for a long time and have switched to different jobs. This is monitored using the **JOB_HISTORY** table. When an employee switches jobs, the details of the start date and end date of the former job, the `job_id` (such as **ST_CLERK**, **SA_REP**, and so on), and the department are recorded in the **JOB_HISTORY** table.

The structure and data from the **EMPLOYEES** and **JOB_HISTORY** tables are shown on the following pages.

Tables Used in This Lesson (continued)

There have been instances in the company, of people who have held the same position more than once during their tenure with the company. For example, consider the employee Taylor, who joined the company on 24-MAR-1998. Taylor held the job title SA_REP for the period 24-MAR-98 to 31-DEC-98 and the job title SA_MAN for the period 01-JAN-99 to 31-DEC-99. Taylor moved back into the job title of SA_REP, which is his current job title.

```
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)

Tables Used in This Lesson (continued)

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

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
1	100	King	AD_PRES	17-JUN-87	90
2	101	Kochhar	AD_VP	21-SEP-89	90
3	102	De Haan	AD_VP	13-JAN-93	90
4	103	Hunold	IT_PROG	03-JAN-90	60
5	104	Ernst	IT_PROG	21-MAY-91	60
6	107	Lorentz	IT_PROG	07-FEB-99	60
7	124	Mourgos	ST_MAN	16-NOV-99	50
8	141	Rajs	ST_CLERK	17-OCT-95	50
9	142	Davies	ST_CLERK	29-JAN-97	50
10	143	Matos	ST_CLERK	15-MAR-98	50
11	144	Vargas	ST_CLERK	09-JUL-98	50
12	149	Zlotkey	SA_MAN	29-JAN-00	80
13	174	Abel	SA_REP	11-MAY-96	80
14	176	Taylor	SA_REP	24-MAR-98	80
15	178	Grant	SA_REP	24-MAY-99	(null)
16	200	Whalen	AD_ASST	17-SEP-87	10
17	201	Hartstein	MK_MAN	17-FEB-96	20

...

```
DESCRIBE job_history
```

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)

Tables Used in This Lesson (continued)

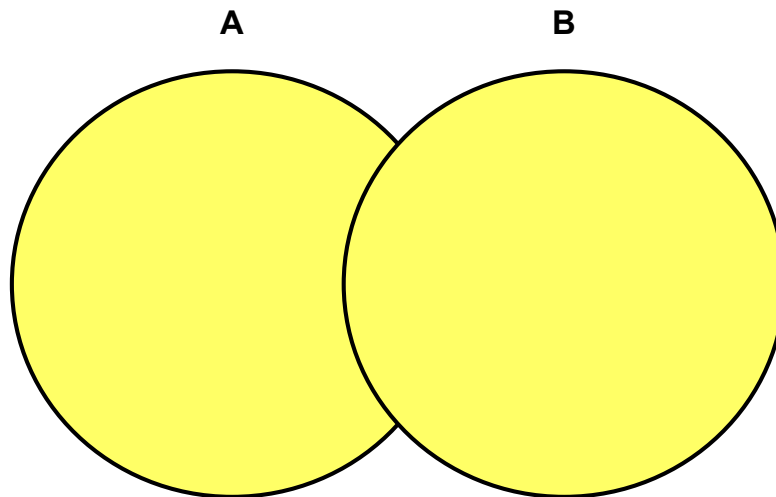
```
SELECT * FROM job_history;
```

	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

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- **UNION and UNION ALL operator**
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

UNION Operator



The **UNION** operator returns rows from both queries after eliminating duplications.

ORACLE

8 - 13

Copyright © 2007, Oracle. All rights reserved.

UNION Operator

The **UNION** operator returns all rows that are selected by either query. Use the **UNION** operator to return all rows from multiple tables and eliminate any duplicate rows.

Guidelines

- The number of columns being selected must be the same.
- The data types of the columns being selected must be in the same data type group (such as numeric or character).
- The names of the columns need not be identical.
- **UNION** operates over all of the columns being selected.
- **NULL** values are not ignored during duplicate checking.
- By default, the output is sorted in ascending order of the columns of the **SELECT** clause.

Using the UNION Operator

Display the current and previous job details of all employees.
Display each employee only once.

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

EMPLOYEE_ID	JOB_ID
1	100 AD_PRES
2	101 AC_ACCOUNT
...	...
22	200 AC_ACCOUNT
23	200 AD_ASST
24	201 MK_MAN
...	...

ORACLE

8 - 14

Copyright © 2007, Oracle. All rights reserved.

Using the UNION Operator

The UNION operator eliminates any duplicate records. If records that occur in both the EMPLOYEES and the JOB_HISTORY tables are identical, the records are displayed only once. Observe in the output shown in the slide that the record for the employee with the EMPLOYEE_ID 200 appears twice because the JOB_ID is different in each row.

Consider the following example:

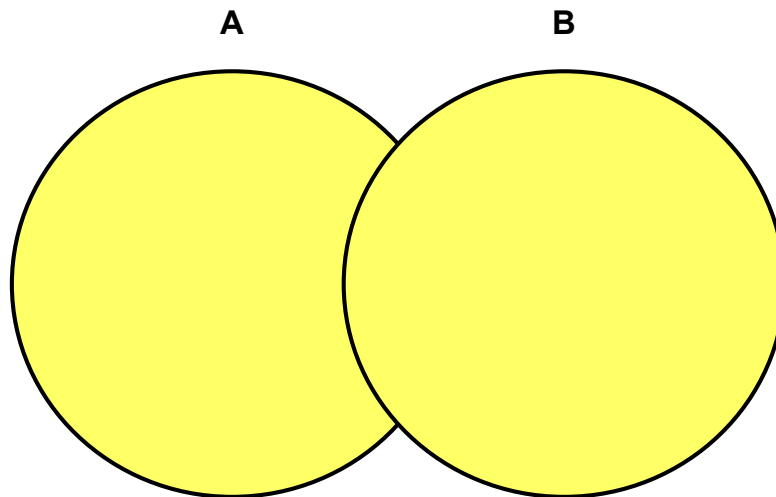
```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100 AD_PRES	90
2	101 AC_ACCOUNT	110
...
22	200 AC_ACCOUNT	90
23	200 AD_ASST	10
24	200 AD_ASST	90
...

Using the UNION Operator (continued)

In the preceding output, employee 200 appears three times. Why? Note the `DEPARTMENT_ID` values for employee 200. One row has a `DEPARTMENT_ID` of 90, another 10, and the third 90. Because of these unique combinations of job IDs and department IDs, each row for employee 200 is unique and therefore not considered to be a duplicate. Observe that the output is sorted in ascending order of the first column of the `SELECT` clause (in this case, `EMPLOYEE_ID`).

UNION ALL Operator



The **UNION ALL** operator returns rows from both queries, including all duplications.

ORACLE

8 - 16

Copyright © 2007, Oracle. All rights reserved.

UNION ALL Operator

Use the **UNION ALL** operator to return all rows from multiple queries.

Guidelines

The guidelines for **UNION** and **UNION ALL** are the same, with the following two exceptions that pertain to **UNION ALL**: Unlike **UNION**, duplicate rows are not eliminated and the output is not sorted by default.

Using the UNION ALL Operator

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100 AD_PRES	90
...		
16	144 ST_CLERK	50
17	149 SA_MAN	80
18	174 SA_REP	80
19	176 SA_REP	80
20	176 SA_MAN	80
21	176 SA_REP	80
22	178 SA_REP	(null)
...		
30	206 AC_ACCOUNT	110

ORACLE

8 - 17

Copyright © 2007, Oracle. All rights reserved.

Using the UNION ALL Operator

In the example, 30 rows are selected. The combination of the two tables totals to 30 rows. The UNION ALL operator does not eliminate duplicate rows. UNION returns all distinct rows selected by either query. UNION ALL returns all rows selected by either query, including all duplicates. Consider the query in the slide, now written with the UNION clause:

```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

The preceding query returns 29 rows. This is because it eliminates the following row (because it is a duplicate):

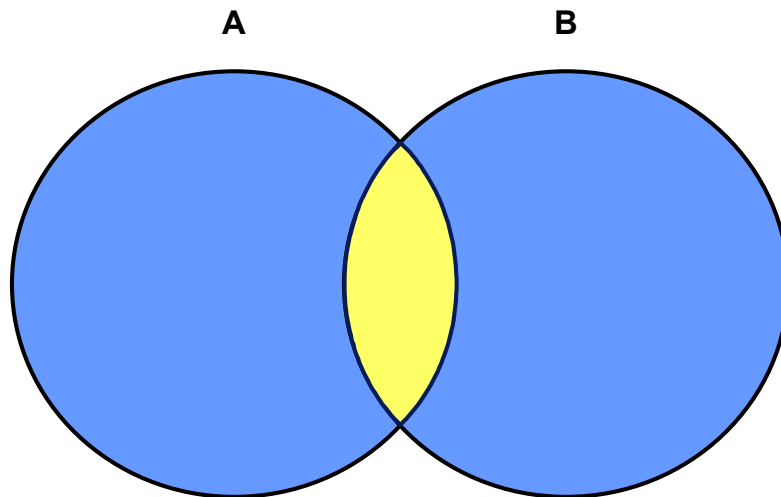
176 SA_REP	80
------------	----

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- **INTERSECT operator**
- MINUS operator
- Matching the SELECT statements
- Using ORDER BY clause in set operations

ORACLE

INTERSECT Operator



The `INTERSECT` operator returns rows that are common to both queries.

ORACLE

8 - 19

Copyright © 2007, Oracle. All rights reserved.

INTERSECT Operator

Use the `INTERSECT` operator to return all rows that are common to multiple queries.

Guidelines

- The number of columns and the data types of the columns being selected by the `SELECT` statements in the queries must be identical in all the `SELECT` statements used in the query. The names of the columns, however, need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- `INTERSECT` does not ignore `NULL` values.

Using the INTERSECT Operator

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their previous one (that is, they changed jobs but have now gone back to doing the same job they did previously).

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

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

ORACLE

8 - 20

Copyright © 2007, Oracle. All rights reserved.

Using the INTERSECT Operator

In the example in this slide, the query returns only those records that have the same values in the selected columns in both tables.

What will be the results if you add the DEPARTMENT_ID column to the SELECT statement from the EMPLOYEES table and add the DEPARTMENT_ID column to the SELECT statement from the JOB_HISTORY table, and run this query? The results may be different because of the introduction of another column whose values may or may not be duplicates.

Example:

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

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	176	SA_REP	80

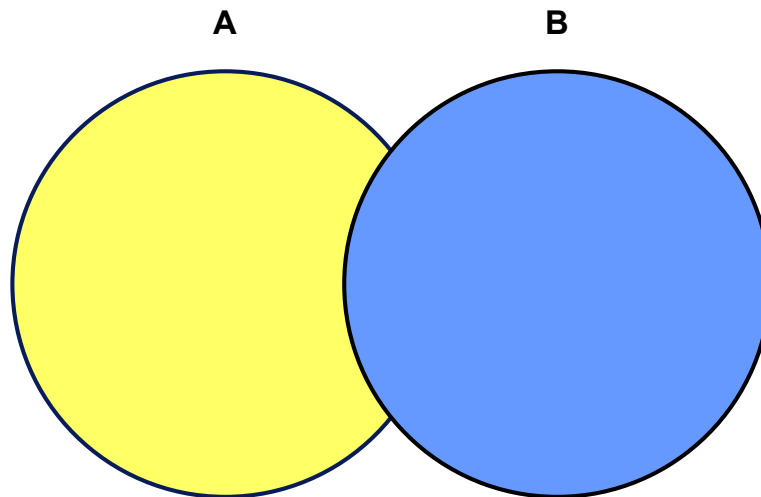
Employee 200 is no longer part of the results because the EMPLOYEES.DEPARTMENT_ID value is different from the JOB_HISTORY.DEPARTMENT_ID value.

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- **MINUS operator**
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

MINUS Operator



The **MINUS** operator returns all the distinct rows selected by the first query, but not present in the second query result set.

ORACLE

8 - 22

Copyright © 2007, Oracle. All rights reserved.

MINUS Operator

Use the **MINUS** operator to return all distinct rows selected by the first query, but not present in the second query result set (the first **SELECT** statement **MINUS** the second **SELECT** statement).

Note: The number of columns must be the same and the data types of the columns being selected by the **SELECT** statements in the queries must belong to the same data type group in all the **SELECT** statements used in the query. The names of the columns, however, need not be identical.

Using the MINUS Operator

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id
FROM employees
MINUS
SELECT employee_id
FROM job_history;
```

EMPLOYEE_ID
1
2
3
4
5

...

14
15

ORACLE

Using the MINUS Operator

In the example in the slide, the employee IDs in the `JOB_HISTORY` table are subtracted from those in the `EMPLOYEES` table. The results set displays the employees remaining after the subtraction; they are represented by rows that exist in the `EMPLOYEES` table, but do not exist in the `JOB_HISTORY` table. These are the records of the employees who have not changed their jobs even once.

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- **Matching the SELECT statements**
- Using ORDER BY clause in set operations

Matching the SELECT Statements

- Using the UNION operator, display the location ID, department name, and the state where it is located.
- You must match the data type (using the TO_CHAR function or any other conversion functions) when columns do not exist in one or the other table.

```
SELECT location_id, department_name "Department",
       TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
       state_province
FROM locations;
```

ORACLE

8 - 25

Copyright © 2007, Oracle. All rights reserved.

Matching the SELECT Statements

Because the expressions in the SELECT lists of the queries must match in number, you can use the dummy columns and the data type conversion functions to comply with this rule. In the slide, the name, Warehouse location, is given as the dummy column heading. The TO_CHAR function is used in the first query to match the VARCHAR2 data type of the state_province column that is retrieved by the second query. Similarly, the TO_CHAR function in the second query is used to match the VARCHAR2 data type of the department_name column that is retrieved by the first query.

The output of the query is shown:

	LOCATION_ID	Department	Warehouse location
1	1400	IT	(null)
2	1400	(null)	Texas
3	1500	Shipping	(null)
4	1500	(null)	California
5	1700	Accounting	(null)
6	1700	Administration	(null)
7	1700	Contracting	(null)
8	1700	Executive	(null)
9	1700	(null)	Washington

■ ■ ■

Matching the SELECT Statement: Example

Using the UNION operator, display the employee ID, job ID, and salary of all employees.

```
SELECT employee_id, job_id, salary
FROM employees
UNION
SELECT employee_id, job_id, 0
FROM job_history;
```

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000
...			
29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

ORACLE

Matching the SELECT Statement: Example

The EMPLOYEES and JOB_HISTORY tables have several columns in common (for example, EMPLOYEE_ID, JOB_ID, and DEPARTMENT_ID). But what if you want the query to display the employee ID, job ID, and salary using the UNION operator, knowing that the salary exists only in the EMPLOYEES table?

The code example in the slide matches the EMPLOYEE_ID and JOB_ID columns in the EMPLOYEES and JOB_HISTORY tables. A literal value of 0 is added to the JOB_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.

In the results shown in the slide, each row in the output that corresponds to a record from the JOB_HISTORY table contains a 0 in the SALARY column.

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Match the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Using the ORDER BY Clause in Set Operations

- The ORDER BY clause can appear only once at the end of the compound query.
- Component queries cannot have individual ORDER BY clauses.
- ORDER BY clause recognizes only the columns of the first SELECT query.
- By default, the first column of the first SELECT query is used to sort the output in an ascending order.

ORACLE

8 - 28

Copyright © 2007, Oracle. All rights reserved.

Using the ORDER BY Clause in Set Operations

The ORDER BY clause can be used only once in a compound query. If used, the ORDER BY clause must be placed at the end of the query. The ORDER BY clause accepts the column name or an alias. By default, the output is sorted in ascending order in the first column of the first SELECT query.

Note: The ORDER BY clause does not recognize the column names of the second SELECT query. To avoid confusion over column names, it is a common practice to ORDER BY column positions.

For example, in the following statement, the output will be shown in ascending order of the job_id.

```
SELECT employee_id, job_id, salary
FROM   employees
UNION
SELECT employee_id, job_id, 0
FROM   job_history
ORDER BY 2;
```

If you omit the ORDER BY, then by default the output will be sorted in the ascending order of employee_id. You cannot use the columns from the second query to sort the output.

Summary

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

- `UNION` to return all distinct rows
- `UNION ALL` to return all rows, including duplicates
- `INTERSECT` to return all rows that are shared by both queries
- `MINUS` to return all distinct rows that are selected by the first query, but not by the second
- `ORDER BY` only at the very end of the statement

ORACLE

8 - 29

Copyright © 2007, Oracle. All rights reserved.

Summary

- The `UNION` operator returns all the distinct rows selected by each query in the compound query. Use the `UNION` operator to return all rows from multiple tables and eliminate any duplicate rows.
- Use the `UNION ALL` operator to return all rows from multiple queries. Unlike the case with the `UNION` operator, duplicate rows are not eliminated and the output is not sorted by default.
- Use the `INTERSECT` operator to return all rows that are common to multiple queries.
- Use the `MINUS` operator to return rows returned by the first query that are not present in the second query.
- Remember to use the `ORDER BY` clause only at the very end of the compound statement.
- Make sure that the corresponding expressions in the `SELECT` lists match in number and data type.

Practice 8: Overview

In this practice, you create reports by using:

- The `UNION` operator
- The `INTERSECTION` operator
- The `MINUS` operator

ORACLE

8 - 30

Copyright © 2007, Oracle. All rights reserved.

Practice 8: Overview

In this practice, you write queries using the set operators.

Practice 8

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

	DEPARTMENT_ID
1	10
2	20
3	60
4	80
5	90
6	110
7	190

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 the set operators to create this report.

COUNTRY_ID	COUNTRY_NAME
1 DE	Germany

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

JOB_ID	DEPARTMENT_ID
1 AD_ASST	10
2 ST_MAN	50
3 ST_CLERK	50
4 MK_MAN	20
5 MK_REP	20

4. Create a report that lists the employee IDs and job IDs 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).

EMPLOYEE_ID	JOB_ID
1 176	SA_REP
2 200	AD_ASST

Practice 8 (continued)

5. The HR department needs a report with the following specifications:
- Last name and department ID of all employees from the EMPLOYEES table, regardless of whether or not they belong to a department
 - Department ID and department name of all departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

Write a compound query to accomplish this.

	LAST_NAME	DEPARTMENT_ID	TO_CHAR(NULL)
1	Abel	80	(null)
2	Davies	50	(null)
3	De Haan	90	(null)
4	Ernst	60	(null)
5	Fay	20	(null)
6	Gietz	110	(null)
7	Grant	(null)	(null)
8	Hartstein	20	(null)
9	Higgins	110	(null)
10	Hunold	60	(null)
11	King	90	(null)
12	Kochhar	90	(null)
13	Lorentz	60	(null)
14	Matos	50	(null)
15	Mourgos	50	(null)
16	Rajs	50	(null)
17	Taylor	80	(null)
18	Vargas	50	(null)
19	Whalen	10	(null)
20	Zlotkey	80	(null)
21	(null)	10	Administration
22	(null)	20	Marketing
23	(null)	50	Shipping
24	(null)	60	IT
25	(null)	80	Sales
26	(null)	90	Executive
27	(null)	110	Accounting
28	(null)	190	Contracting

9 Manipulating Data

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Objectives

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

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions

ORACLE

9 - 2

Copyright © 2007, Oracle. All rights reserved.

Objective

In this lesson, you learn how to use the data manipulation language (DML) statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Data Manipulation Language

- A DML statement is executed when you:
 - Add new rows to a table
 - Modify existing rows in a table
 - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

ORACLE

9 - 4

Copyright © 2007, Oracle. All rights reserved.

Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decreasing the savings account, increasing the checking account, and recording the transaction in the transaction journal. The Oracle server must guarantee that all the three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Note: Most of the DML statements in this lesson assume that no constraints on the table are violated. Constraints are discussed later in this course.

Note: In SQL Developer, click the Run Script icon or press [F5] to run the DML statements. The feedback messages will be shown on the Script Output tabbed page.

Adding a New Row to a Table

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

DEPARTMENTS

70 Public Relations	100	1700
---------------------	-----	------

New row

Insert new row into the DEPARTMENTS table.

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

9	70 Public Relations	100	1700
---	---------------------	-----	------

Adding a New Row to a Table

The graphic in the slide illustrates the addition of a new department to the DEPARTMENTS table.

INSERT Statement Syntax

- Add new rows to a table by using the `INSERT` statement:

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

- With this syntax, only one row is inserted at a time.

ORACLE

9 - 6

Copyright © 2007, Oracle. All rights reserved.

INSERT Statement Syntax

You can add new rows to a table by issuing the `INSERT` statement.

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	is the corresponding value for the column

Note: This statement with the `VALUES` clause adds only one row at a time to a table.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id,  
                        department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);
```

```
1 rows inserted
```

- Enclose character and date values within single quotation marks.

ORACLE

9 - 7

Copyright © 2007, Oracle. All rights reserved.

Inserting New Rows

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

```
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)

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values within single quotation marks; however, it is not recommended that you enclose numeric values within single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO departments (department_id,  
                          department_name)  
VALUES (30, 'Purchasing');
```

```
1 rows inserted
```

- Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO departments  
VALUES (100, 'Finance', NULL, NULL);
```

```
1 rows inserted
```

ORACLE

9 - 8

Copyright © 2007, Oracle. All rights reserved.

Inserting Rows with Null Values

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string (' ') in the VALUES list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the Null status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input are checked in the following order:

- Mandatory value missing for a NOT NULL column
- Duplicate value violating any unique or primary key constraint
- Any value violating a CHECK constraint
- Referential integrity maintained for foreign key constraint
- Data type mismatches or values too wide to fit in column

Note: Use of the column list is recommended as it makes the INSERT statement more readable and reliable, or less prone to mistakes.

Inserting Special Values

The SYSDATE function records the current date and time.

```
INSERT INTO employees (employee_id,
                       first_name, last_name,
                       email, phone_number,
                       hire_date, job_id, salary,
                       commission_pct, manager_id,
                       department_id)
VALUES (113,
       'Louis', 'Popp',
       'LPOPP', '515.124.4567',
       SYSDATE, 'AC_ACCOUNT', 6900,
       NULL, 205, 110);
```

1 rows inserted

ORACLE

9 - 9

Copyright © 2007, Oracle. All rights reserved.

Inserting Special Values

You can use functions to enter special values in your table.

The slide example records information for employee Popp in the EMPLOYEES table. It supplies the current date and time in the HIRE_DATE column. It uses the SYSDATE function that returns the current date and time of the database server. You may also use the CURRENT_DATE function to get the current date in the session time zone. You can also use the USER function when inserting rows in a table. The USER function records the current username.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	AC_ACCOUNT	11-JUN-07	(null)

Inserting Specific Date and Time Values

- Add a new employee.

```
INSERT INTO employees
VALUES      (114,
            'Den', 'Raphealy',
            'DRAPHEAL', '515.127.4561',
            TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
            'SA_REP', 11000, 0.2, 100, 60);
```

1 rows inserted

- Verify your addition.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	SA_REP	11000	0.2

ORACLE

9 - 10

Copyright © 2007, Oracle. All rights reserved.

Inserting Specific Date and Time Values

The DD-MON-RR format is generally used to insert a date value. With the RR format, the system provides the correct century automatically.

You may also supply the date value in the DD-MON-YYYY format. This is recommended because it clearly specifies the century and does not depend on the internal RR format logic of specifying the correct century.

If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the `TO_DATE` function.

The example in the slide records information for employee Raphealy in the `EMPLOYEES` table. It sets the `HIRE_DATE` column to be February 3, 1999.

Creating a Script

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
      (department_id, department_name, location_id)
VALUES  (&department_id, '&department_name', &location);
```

The image shows three overlapping dialog boxes titled "Enter Substitution Variable". The first dialog prompts for "DEPARTMENT_ID:" with the input field containing "40" and an "OK" button. The second dialog prompts for "DEPARTMENT_NAME:" with the input field containing "Human Resources" and an "OK" button. The third dialog prompts for "LOCATION:" with the input field containing "2500" and "OK" and "Cancel" buttons.

ORACLE

Creating a Script

You can save commands with substitution variables to a file and execute the commands in the file. The example in the slide records information for a department in the DEPARTMENTS table.

Run the script file and you are prompted for input for each of the ampersand (&) substitution variables. After entering a value for the substitution variable, click the OK button. The values that you input are then substituted into the statement. This enables you to run the same script file over and over, but supply a different set of values each time you run it.

Copying Rows from Another Table

- Write your `INSERT` statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows inserted

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.
- Inserts all the rows returned by the subquery in the table, `sales_reps`.

ORACLE

9 - 12

Copyright © 2007, Oracle. All rights reserved.

Copying Rows from Another Table

You can use the `INSERT` statement to add rows to a table where the values are derived from existing tables. In the slide example, for the `INSERT INTO` statement to work, you must have already created the `sales_reps` table using the `CREATE TABLE` statement. `CREATE TABLE` is discussed in the next lesson titled “Using DDL Statements to Create and Manage Tables.”

In place of the `VALUES` clause, you use a subquery.

Syntax

```
INSERT INTO table [ column (, column) ] subquery;
```

In the syntax:

table is the name of the table
column is the name of the column in the table to populate
subquery is the subquery that returns rows to the table

The number of columns and their data types in the column list of the `INSERT` clause must match the number of values and their data types in the subquery. Zero or more rows are added depending on the number of rows returned by the subquery. To create a copy of the rows of a table, use `SELECT *` in the subquery:

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the **EMPLOYEES** table: 

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

ORACLE

Changing Data in a Table

The slide illustrates changing the department number for employees in department 60 to department 80.

UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

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

- Update more than one row at a time (if required).

ORACLE

9 - 15

Copyright © 2007, Oracle. All rights reserved.

UPDATE Statement Syntax

You can modify the existing values in a table by using the UPDATE statement.

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	is the corresponding value or subquery for the column
<i>condition</i>	identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

For more information, see the section on “UPDATE” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Note: In general, use the primary key column in the WHERE clause to identify a single row for update. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the `WHERE` clause:

```
UPDATE employees
SET   department_id = 50
WHERE employee_id = 113;
```

1 rows updated

- Values for all the rows in the table are modified if you omit the `WHERE` clause:

```
UPDATE   copy_emp
SET      department_id = 110;
```

22 rows updated

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

ORACLE

9 - 16

Copyright © 2007, Oracle. All rights reserved.

Updating Rows in a Table

The `UPDATE` statement modifies the values of a specific row or rows if the `WHERE` clause is specified. The example in the slide shows the transfer of employee 113 (Popp) to department 50.

If you omit the `WHERE` clause, values for all the rows in the table are modified. Examine the updated rows in the `COPY_EMP` table.

```
SELECT last_name, department_id
FROM   copy_emp;
```

	LAST_NAME	DEPARTMENT_ID
1	King	110
2	Kochhar	110

...

For example, an employee who was a `SA_REP` has now changed his job to an `IT_PROG`. Therefore, his `JOB_ID` needs to be updated and the `commission` field needs to be set to `NULL`.

```
UPDATE employees
SET job_id = 'IT_PROG', commission_pct = NULL
WHERE employee_id = 114;
```

Note: The `COPY_EMP` table has the same data as the `EMPLOYEES` table.

Updating Two Columns with a Subquery

Update employee 113's job and salary to match those of employee 205.

```
UPDATE employees
SET job_id = (SELECT job_id
              FROM employees
              WHERE employee_id = 205),
    salary = (SELECT salary
              FROM employees
              WHERE employee_id = 205)
WHERE employee_id = 113;
```

1 rows updated

ORACLE

9 - 17

Copyright © 2007, Oracle. All rights reserved.

Updating Two Columns with a Subquery

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries. The syntax is as follows:

```
UPDATE table
SET column =
              (SELECT column
               FROM table
               WHERE condition)
[ ,
  column =
              (SELECT column
               FROM table
               WHERE condition) ]
[WHERE condition ] ;
```

The example in the slide can also be written as follows:

```
UPDATE employees
SET (job_id, salary) = (SELECT job_id, salary
                       FROM employees
                       WHERE employee_id = 205)
WHERE employee_id = 113;
```

Updating Rows Based on Another Table

Use the subqueries in the `UPDATE` statements to update row values in a table based on values from another table:

```
UPDATE copy_emp
SET    department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id        = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);

1 rows updated
```

ORACLE

Updating Rows Based on Another Table

You can use the subqueries in the `UPDATE` statements to update values in a table. The example in the slide updates the `COPY_EMP` table based on the values from the `EMPLOYEES` table. It changes the department number of all employees with employee 200's job ID to employee 100's current department number.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Removing a Row from a Table

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

Delete a row from the DEPARTMENTS table:

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

ORACLE

Removing a Row from a Table

The Contracting department has been removed from the DEPARTMENTS table (assuming no constraints on the DEPARTMENTS table are violated), as shown by the graphic in the slide.

DELETE Statement

You can remove existing rows from a table by using the DELETE statement:

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

ORACLE

9 - 21

Copyright © 2007, Oracle. All rights reserved.

DELETE Statement Syntax

You can remove existing rows from a table by using the DELETE statement.

In the syntax:

table is the name of the table
condition identifies the rows to be deleted, and is composed of column names, expressions, constants, subqueries, and comparison operators

Note: If no rows are deleted, the message “0 rows deleted” is returned (in the Script Output tab in SQL Developer)

For more information, see the section on “DELETE” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Deleting Rows from a Table

- Specific rows are deleted if you specify the `WHERE` clause:

```
DELETE FROM departments
WHERE department_name = 'Finance';
```

```
1 rows deleted
```

- All rows in the table are deleted if you omit the `WHERE` clause:

```
DELETE FROM copy_emp;
```

```
22 rows deleted
```

ORACLE

9 - 22

Copyright © 2007, Oracle. All rights reserved.

Deleting Rows from a Table

You can delete specific rows by specifying the `WHERE` clause in the `DELETE` statement. The first example in the slide deletes the Accounting department from the `DEPARTMENTS` table. You can confirm the delete operation by displaying the deleted rows using the `SELECT` statement.

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

```
0 rows selected
```

However, if you omit the `WHERE` clause, all rows in the table are deleted. The second example in the slide deletes all rows from the `COPY_EMP` table, because no `WHERE` clause was specified.

Example:

Remove rows identified in the `WHERE` clause.

```
DELETE FROM employees WHERE employee_id = 114;
```

```
1 rows deleted
```

```
DELETE FROM departments WHERE department_id IN (30, 40);
```

```
2 rows deleted
```

Deleting Rows Based on Another Table

Use the subqueries in the `DELETE` statements to remove rows from a table based on values from another table:

```
DELETE FROM employees
WHERE department_id =
  (SELECT department_id
   FROM departments
   WHERE department_name
         LIKE '%Public%');
```

1 rows deleted

ORACLE

9 - 23

Copyright © 2007, Oracle. All rights reserved.

Deleting Rows Based on Another Table

You can use the subqueries to delete rows from a table based on values from another table. The example in the slide deletes all employees in a department where the department name contains the string `Admin`. The subquery searches the `DEPARTMENTS` table to find the department number based on the department name containing the string `Public`. The subquery then feeds the department number to the main query, which deletes rows of data from the `EMPLOYEES` table based on this department number.

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

ORACLE

TRUNCATE Statement

A more efficient method of emptying a table is by using the TRUNCATE statement.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.

If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. Disabling constraints is covered in a subsequent lesson.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

ORACLE

9 - 26

Copyright © 2007, Oracle. All rights reserved.

Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that constitute one consistent change to the data. For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account of the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement

Database Transactions: Start and End

- Begin when the first DML SQL statement is executed.
- End with one of the following events:
 - A `COMMIT` or `ROLLBACK` statement is issued.
 - A DDL or DCL statement executes (automatic commit).
 - The user exits SQL Developer or SQL*Plus.
 - The system crashes.

ORACLE

9 - 27

Copyright © 2007, Oracle. All rights reserved.

Database Transaction: Start and End

When does a database transaction start and end?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A `COMMIT` or `ROLLBACK` statement is issued.
- A DDL statement, such as `CREATE`, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

Advantages of COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically-related operations

ORACLE

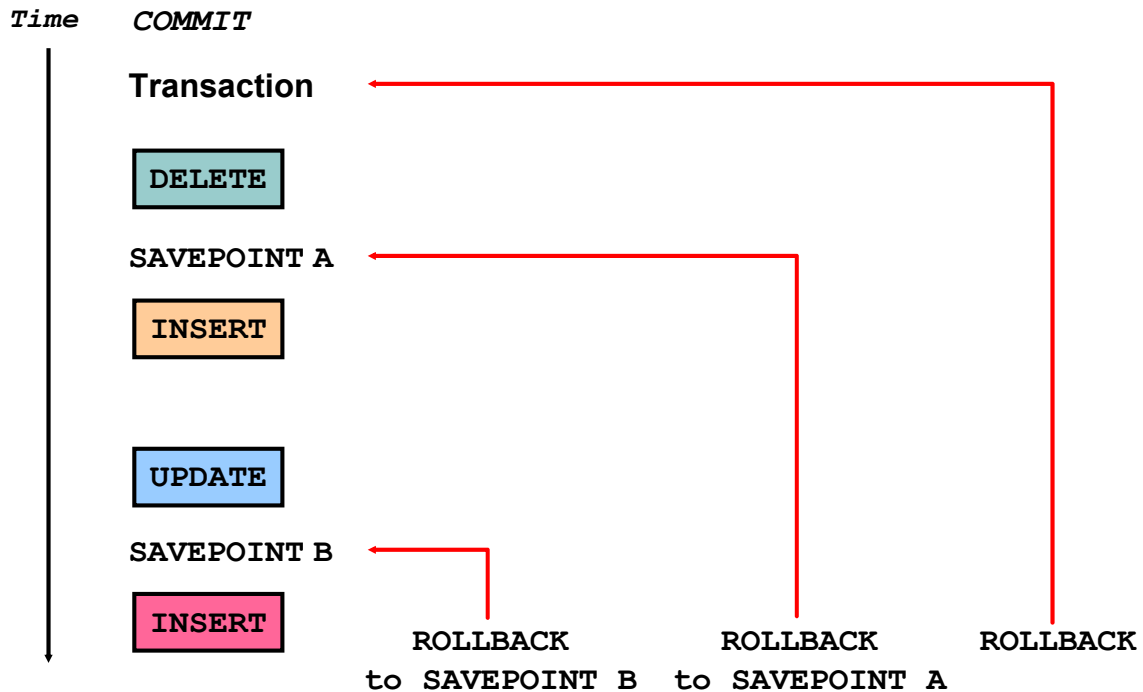
9 - 28

Copyright © 2007, Oracle. All rights reserved.

Advantages of COMMIT and ROLLBACK Statements

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

Explicit Transaction Control Statements



ORACLE

9 - 29

Copyright © 2007, Oracle. All rights reserved.

Explicit Transaction Control Statements

You can control the logic of transactions by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

Statement	Description
<code>COMMIT</code>	Ends the current transaction by making all pending data changes permanent
<code>SAVEPOINT name</code>	Marks a savepoint within the current transaction
<code>ROLLBACK</code>	<code>ROLLBACK</code> ends the current transaction by discarding all pending data changes.
<code>ROLLBACK TO SAVEPOINT name</code>	<code>ROLLBACK TO SAVEPOINT</code> rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the <code>TO SAVEPOINT</code> clause, the <code>ROLLBACK</code> statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

Note: You cannot `COMMIT` to a `SAVEPOINT`. `SAVEPOINT` is not ANSI-standard SQL.

Rolling Back Changes to a Marker

- Create a marker in the current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...  
SAVEPOINT update_done;  
SAVEPOINT update_done succeeded.  
INSERT...  
ROLLBACK TO update_done;  
ROLLBACK TO succeeded.
```

ORACLE

Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

Note, if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
 - A DDL statement is issued
 - A DCL statement is issued
 - Normal exit from SQL Developer or SQL*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure.

ORACLE

9 - 31

Copyright © 2007, Oracle. All rights reserved.

Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement issued SQL Developer or SQL*Plus exited normally, without explicitly issuing <code>COMMIT</code> or <code>ROLLBACK</code> commands
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus or system failure

Note: In SQL*Plus, the `AUTOCOMMIT` command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the `COMMIT` statement can still be issued explicitly. Also, the `COMMIT` statement is issued when a DDL statement is issued or when you exit SQL*Plus. The `SET AUTOCOMMIT ON/OFF` command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- On the right pane, check the Autocommit in SQL Worksheet option. Click OK.

Implicit Transaction Processing (continued)

System Failures

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL*Plus, a normal exit is accomplished by entering the `EXIT` command at the prompt. Closing the window is interpreted as an abnormal exit.

State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the `SELECT` statement.
- Other users *cannot* view the results of the DML statements issued by the current user.
- The affected rows are *locked*; other users cannot change the data in the affected rows.

ORACLE

9 - 33

Copyright © 2007, Oracle. All rights reserved.

State of the Data Before COMMIT or ROLLBACK

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before `COMMIT` or `ROLLBACK` statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in the affected rows.

State of the Data After COMMIT

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

ORACLE

9 - 34

Copyright © 2007, Oracle. All rights reserved.

State of the Data After COMMIT

Make all pending changes permanent by using the COMMIT statement. Here is what happens after a COMMIT statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.

Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- Commit the changes:

```
COMMIT;
COMMIT succeeded.
```

ORACLE

9 - 35

Copyright © 2007, Oracle. All rights reserved.

Committing Data

In the example in the slide, a row is deleted from the EMPLOYEES table and a new row is inserted into the DEPARTMENTS table. The changes are saved by issuing the COMMIT statement.

Example:

Remove departments 290 and 300 in the DEPARTMENTS table and update a row in the EMPLOYEES table. Save the data change.

```
DELETE FROM departments
WHERE department_id IN (290, 300);
```

```
UPDATE employees
SET department_id = 80
WHERE employee_id = 206;
```

```
COMMIT;
```

State of the Data After ROLLBACK

Discard all pending changes by using the `ROLLBACK` statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;  
ROLLBACK ;
```

ORACLE

State of the Data After ROLLBACK

Discard all pending changes by using the `ROLLBACK` statement, which results in the following:

- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.

State of the Data After ROLLBACK: Example

```
DELETE FROM test;
25,000 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test WHERE id = 100;
1 row deleted.

SELECT * FROM test WHERE id = 100;
No rows selected.

COMMIT;
Commit complete.
```

ORACLE

State of the Data After ROLLBACK: Example

While attempting to remove a record from the TEST table, you may accidentally empty the table. However, you can correct the mistake, reissue a proper statement, and make the data change permanent.

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

ORACLE

9 - 38

Copyright © 2007, Oracle. All rights reserved.

Statement-Level Rollback

A part of a transaction can be discarded through an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
 - Readers do not wait for writers
 - Writers do not wait for readers
 - Writers wait for writers

ORACLE

9 - 40

Copyright © 2007, Oracle. All rights reserved.

Read Consistency

Database users access the database in two ways:

- Read operations (`SELECT` statement)
- Write operations (`INSERT`, `UPDATE`, `DELETE` statements)

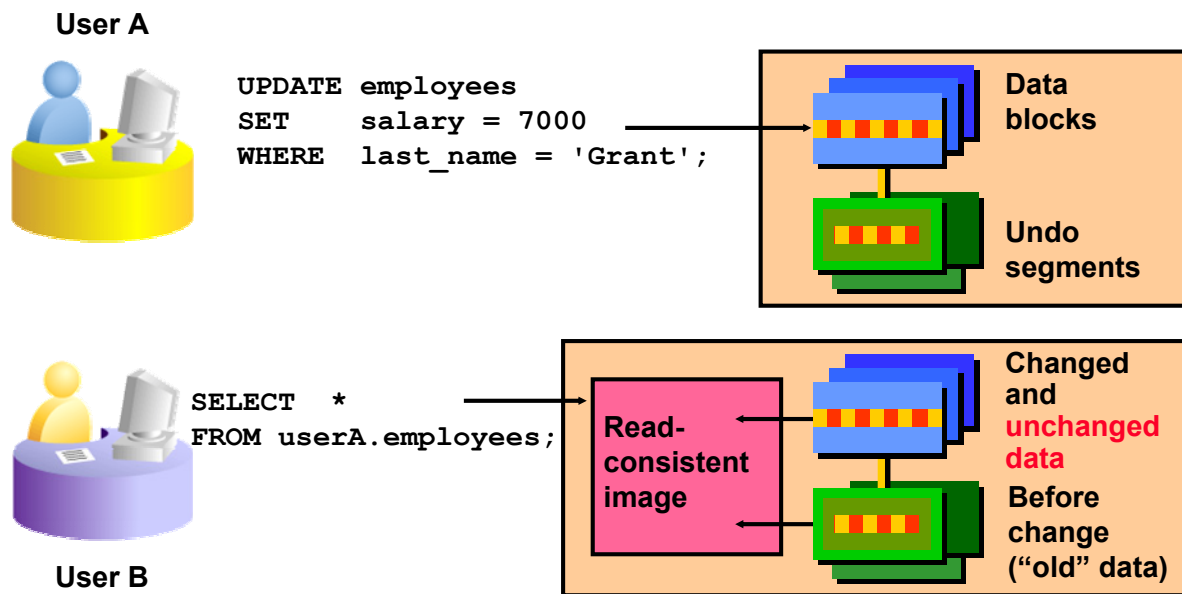
You need read consistency so that the following occur:

- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent manner.
- Changes made by one writer do not disrupt or conflict with the changes being made by another writer.

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Note: The same user can login as different sessions. Each session maintains read consistency in the manner described above, even if they are the same users.

Implementing Read Consistency



Implementing Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in the undo segments. The read-consistent image is constructed from the committed data in the table and the old data that is being changed and is not yet committed from the undo segment.

When an insert, update, or delete operation is made on the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before the changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a `SELECT` statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the `EMPLOYEES` table where `job_id` is `SA_REP`.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

- Lock is released only when you issue a `ROLLBACK` or a `COMMIT`.
- If the `SELECT` statement attempts to lock a row that is locked by another user, then the database waits until the row is available, and then returns the results of the `SELECT` statement.

ORACLE

FOR UPDATE Clause in a SELECT Statement

When you issue a `SELECT` statement against the database to query some records, no locks are placed on the selected rows. In general, this is required because the number of records locked at any given time is (by default) kept to the absolute minimum: only those records that have been changed but not yet committed are locked. Even then, others will be able to read those records as they appeared before the change (the “before image” of the data). There are times, however, when you may want to lock a set of records even before you change them in your program. Oracle offers the `FOR UPDATE` clause of the `SELECT` statement to perform this locking.

When you issue a `SELECT . . . FOR UPDATE` statement, the relational database management system (RDBMS) automatically obtains exclusive row-level locks on all the rows identified by the `SELECT` statement, thereby holding the records “for your changes only.” No one else will be able to change any of these records until you perform a `ROLLBACK` or a `COMMIT`.

You can append the optional keyword `NOWAIT` to the `FOR UPDATE` clause to tell the Oracle server not to wait if the table has been locked by another user. In this case, control will be returned immediately to your program or to your SQL Developer environment so that you can perform other work, or simply wait for a period of time before trying again. Without the `NOWAIT` clause, your process will block until the table is available, when the locks are released by the other user through the issue of a `COMMIT` or a `ROLLBACK` command.

FOR UPDATE Clause: Examples

- You can use the `FOR UPDATE` clause in a `SELECT` statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

- Rows from both the `EMPLOYEES` and `DEPARTMENTS` tables are locked.
- Use `FOR UPDATE OF column_name` to qualify the column you intend to change, then only the rows from that specific table are locked.

ORACLE

9 - 44

Copyright © 2007, Oracle. All rights reserved.

FOR UPDATE Clause: Examples

In the example in the slide, the statement locks rows in the `EMPLOYEES` table with `JOB_ID` set to `ST_CLERK` and `LOCATION_ID` set to 1500, and locks rows in the `DEPARTMENTS` table with departments in `LOCATION_ID` set as 1500.

You can use the `FOR UPDATE OF column_name` to qualify the column that you intend to change. The `OF` list of the `FOR UPDATE` clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state `FOR UPDATE` in the query and do not include one or more columns after the `OF` keyword, the database will lock all identified rows across all the tables listed in the `FROM` clause.

The following statement locks only those rows in the `EMPLOYEES` table with `ST_CLERK` located in `LOCATION_ID` 1500. No rows are locked in the `DEPARTMENTS` table:

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
ORDER BY e.employee_id;
```

FOR UPDATE Clause: Examples (continued)

In the following example, the database is instructed to wait for five seconds for the row to become available, and then return control to you.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE WAIT 5
ORDER BY employee_id;
```

Summary

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

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query

ORACLE

Summary

In this lesson, you should have learned how to manipulate data in the Oracle database by using the INSERT, UPDATE, DELETE, and TRUNCATE statements, as well as how to control data changes by using the COMMIT, SAVEPOINT, and ROLLBACK statements. You also learned how to use the FOR UPDATE clause of the SELECT statement to lock rows for your changes only.

Remember that the Oracle server guarantees a consistent view of data at all times.

Practice 9: Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions

ORACLE

9 - 47

Copyright © 2007, Oracle. All rights reserved.

Practice 9: Overview

In this practice, you add rows to the `MY_EMPLOYEE` table, update and delete data from the table, and control your transactions. You run a script to create the `MY_EMPLOYEE` table.

Practice 9

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.

Note: For all the DML statements, use the Run Script icon (or press [F5]) to execute the query. This way you get to see the feedback messages on the Script Output tab page. For SELECT queries, continue to use the Execute Statement icon or press [F9] to get the formatted output on the Results tab page.

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab_09_01.sql script to build the MY_EMPLOYEE table used in this practice.
2. Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE MY_EMPLOYEE
Name                               Null    Type
-----
ID                                 NOT NULL NUMBER(4)
LAST_NAME                          VARCHAR2(25)
FIRST_NAME                          VARCHAR2(25)
USERID                              VARCHAR2(8)
SALARY                              NUMBER(9,2)
```

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. *Do not enter all rows yet.*

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

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

Practice 9 (continued)

5. Confirm your addition to the table.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860

6. Write an INSERT statement in a dynamic reusable script file to load the remaining rows into the MY_EMPLOYEE table. The script should prompt for all the columns (ID, LAST_NAME, FIRST_NAME, USERID, and SALARY). Save this script to a lab_09_06.sql file.
7. Populate the table with the next two rows of the sample data listed in step 3 by running the INSERT statement in the script that you created.
8. Confirm your additions to the table.

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

9. Make the data additions permanent.

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.
11. Change the salary to \$1,000 for all employees who have a salary less than \$900.
12. Verify your changes to the table.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Dancs	Betty	bdancs	1000
3	Drexler	Ben	bbiri	1100
4	Newman	Chad	cnewman	1000

13. Delete Betty Dancs from the MY_EMPLOYEE table.
14. Confirm your changes to the table.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Drexler	Ben	bbiri	1100
3	Newman	Chad	cnewman	1000

Practice 9 (continued)

15. Commit all pending changes.

Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of the 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.

17. Confirm your addition to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

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

19. Delete all the rows from the MY_EMPLOYEE table.

20. Confirm that the table is empty.

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

22. Confirm that the new row is still intact.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

23. Make the data addition permanent.

If you have the time, complete the following exercise:

24. Modify the lab_09_06.sql script such that the USERID is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated USERID must be in lowercase. Hence, the script should not prompt for the USERID. Save this script to a file named lab_09_24.sql.

25. Run the script, lab_09_24.sql to insert the following record:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

26. Confirm that the new row was added with correct USERID.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

10

Using DDL Statements to Create and Manage Tables

ORACLE

Copyright © 2007, 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

10 - 2

Copyright © 2007, 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 discussed in this lesson. Exception messages that are generated from violating constraints during DML operations are shown and explained.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

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 name to an object

ORACLE

10 - 4

Copyright © 2007, Oracle. All rights reserved.

Database Objects

The 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 name to an object

Oracle Table Structures

- Tables can be created at any time, even when 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

10 - 5

Copyright © 2007, 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.
 - You may also use quoted identifiers to represent the name of an object. A quoted identifier begins and ends with double quotation marks (“”). If you name a schema object using a quoted identifier, then you must use the double quotation marks whenever you refer to that object. Quoted identifiers can be reserved words, although this is not recommended.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are not case-sensitive. For example, EMPLOYEES is treated to be the same name as eMPLOYees or eMpLOYEES. However, quoted identifiers are case-sensitive.

For more information, see the section on *Schema Object Names and Qualifiers* in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

CREATE TABLE Statement

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

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

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



ORACLE

10 - 7

Copyright © 2007, 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 that are a subset of the 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.

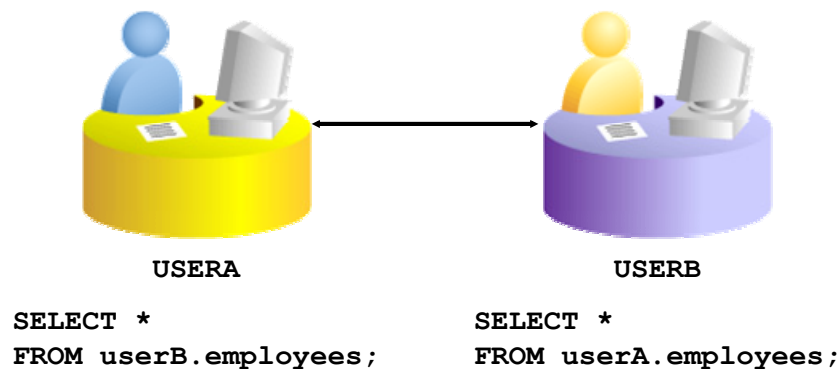
To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.

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

10 - 8

Copyright © 2007, Oracle. All rights reserved.

Referencing Another User's Tables

A schema is a collection of logical structures of data or *schema objects*. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.

Schema objects can be created and manipulated with SQL and 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`, `USERA` 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`, `USERB` 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

10 - 9

Copyright © 2007, Oracle. All rights reserved.

DEFAULT Option

When you define a table, you can specify that a column should be given a default value by using the `DEFAULT` option. This option prevents null values from entering the columns when 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.

Consider the following examples:

```
INSERT INTO hire_dates values(45, NULL);
```

The above statement will insert the null value rather than the default value.

```
INSERT INTO hire_dates(id) values(35);
```

The above statement will insert `SYSDATE` for the `HIRE_DATE` column.

Note: In SQL Developer, click the Run Script icon or press [F5] to run the DDL statements. The feedback messages will be shown on the Script Output tabbed page.

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
```

```
DESCRIBE dept
Name                               Null    Type
-----
DEPTNO                             NUMBER(2)
DNAME                               VARCHAR2(14)
LOC                                 VARCHAR2(13)
CREATE_DATE                         DATE
```

ORACLE

10 - 10

Copyright © 2007, 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.

To confirm that the table was created, run the DESCRIBE command.

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

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

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

10 - 12

Copyright © 2007, 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> (Precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and 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

10 - 14

Copyright © 2007, Oracle. All rights reserved.

Datetime Data Types

Data Type	Description
TIMESTAMP	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, and the second value of the DATE data type as well as the fractional seconds value There are several variations of this data type such as WITH TIMEZONE, WITH LOCALTIMEZONE.
INTERVAL YEAR TO MONTH	Enables storage of time 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 storage of time 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. The datetime data types are discussed in detail in the lesson titled “Managing Data in Different Time Zones” in the *Oracle Database 11g: SQL Fundamentals II* course.

Also, for more information about the datetime data types, see the topics *TIMESTAMP Datatype*, *INTERVAL YEAR TO MONTH Datatype*, and *INTERVAL DAY TO SECOND Datatype* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

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

10 - 16

Copyright © 2007, 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 referential integrity between the column and a column of the referenced table such that values in one table match values in another table.
CHECK	Specifies a condition that must be true

Constraint Guidelines

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

ORACLE

10 - 17

Copyright © 2007, 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, except that the name cannot be the same as another object owned by the same user. 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 creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

For more information, see the section on “Constraints” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Defining Constraints

- Syntax:

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

- Column-level constraint syntax:

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

- Table-level constraint syntax:

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

ORACLE

10 - 18

Copyright © 2007, Oracle. All rights reserved.

Defining Constraints

The slide gives the syntax for defining constraints when creating a table. You can create 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. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint.

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 be used 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

- Example of a column-level constraint:

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

1

- Example of a 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

10 - 19

Copyright © 2007, 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 be temporarily disabled.

Both examples in the slide 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	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	(null)
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	(null)
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	(null)
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	(null)
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	(null)
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	(null)
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	(null)
141	Trenna	Rajs	TRAJS	17-OCT-95	ST_CLERK	(null)
142	Curtis	Davies	CDAVIES	29-JAN-97	ST_CLERK	(null)
143	Randall	Matos	RMATOS	15-MAR-98	ST_CLERK	(null)
144	Peter	Vargas	PVARGAS	09-JUL-98	ST_CLERK	(null)
149	Eleni	Zlotkey	EZLOTKEY	29-JAN-00	SA_MAN	0.2
174	Ellen	Abel	EABEL	11-MAY-96	SA_REP	0.3

...

**NOT NULL constraint
(Primary Key enforces
NOT NULL constraint.)**

**NOT NULL
constraint**

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

ORACLE

10 - 20

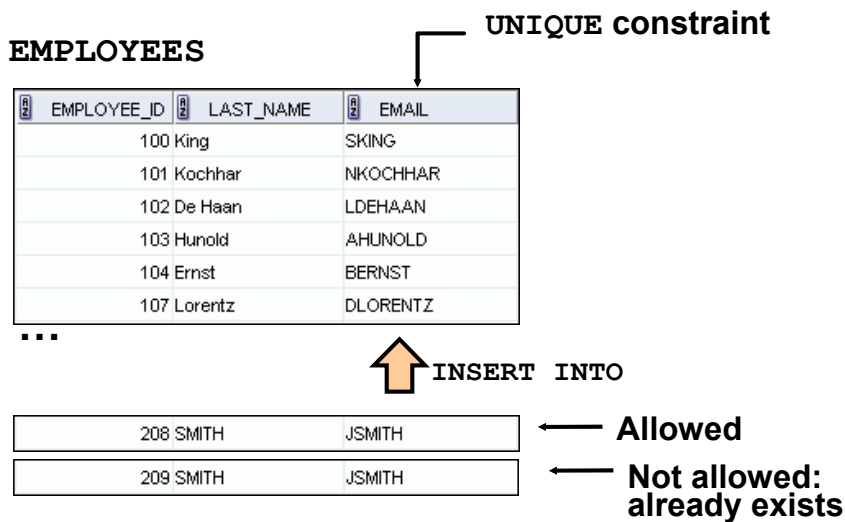
Copyright © 2007, Oracle. All rights reserved.

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. In the EMPLOYEES table, the EMPLOYEE_ID column inherits a NOT NULL constraint as it is defined as a primary key. Otherwise, the LAST_NAME, EMAIL, HIRE_DATE, and JOB_ID columns have the NOT NULL constraint enforced on them.

Note: Primary key constraint is discussed in detail later in this lesson.

UNIQUE Constraint



ORACLE

10 - 21

Copyright © 2007, Oracle. All rights reserved.

UNIQUE Constraint

A **UNIQUE** key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a 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 the **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 the **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

10 - 22

Copyright © 2007, Oracle. All rights reserved.

UNIQUE Constraint (continued)

UNIQUE constraints can be defined at the column level or table level. You define the constraint at the table level when you want to create a composite unique key. A composite key is defined when there is not a single attribute that can uniquely identify a row. In that case, you can have a unique key that is composed of two or more columns, the combined value of which is always unique and can identify rows.

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	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

Not allowed
(null value)

↑ INSERT INTO

Public Accounting	124	2500
50 Finance	124	1500

Not allowed
(50 already exists)

ORACLE

10 - 23

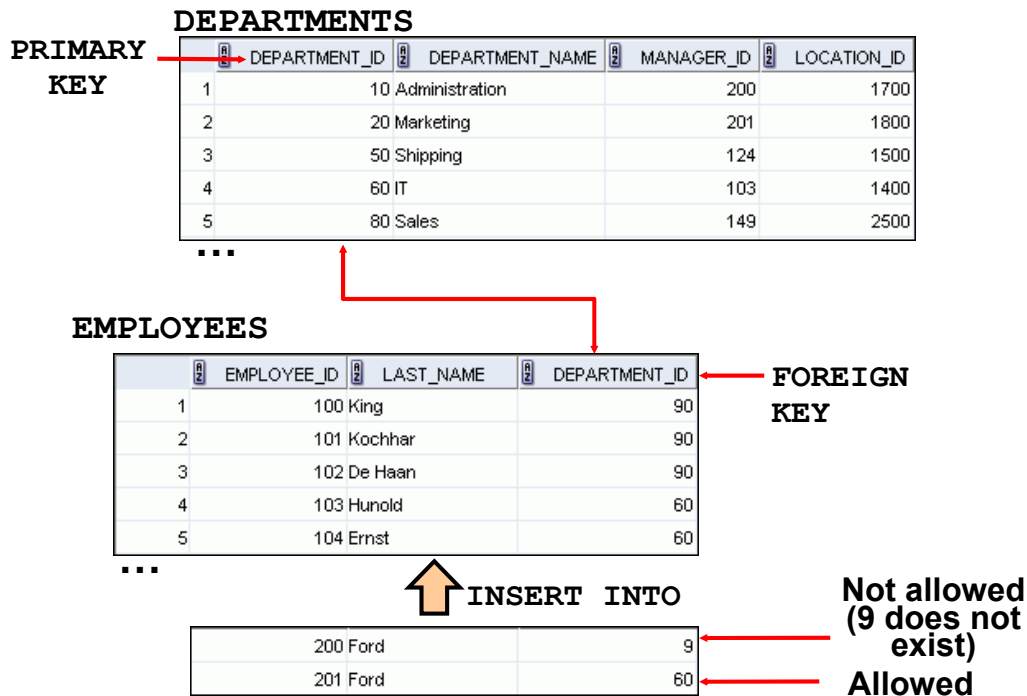
Copyright © 2007, Oracle. All rights reserved.

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 a set of columns that uniquely identifies each row in a table. This constraint enforces the 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

10 - 24

Copyright © 2007, Oracle. All rights reserved.

FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key and establishes a relationship with 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

10 - 25

Copyright © 2007, 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_DEPT_FK.

The foreign key can also be defined at the column level, provided that 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

10 - 26

Copyright © 2007, 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 the column in the parent table.
- **ON DELETE CASCADE** indicates that when a row in the parent table is deleted, the dependent rows in the child table are also deleted.
- **ON DELETE SET NULL** indicates that when a row in the parent table is deleted, the foreign key values are set to null.

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

10 - 27

Copyright © 2007, 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 the 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)
  CONSTRAINT emp_manager_fk  REFERENCES
    employees (employee_id)
, department_id   NUMBER(4)
  CONSTRAINT emp_dept_fk    REFERENCES
    departments (department_id));
```

ORACLE

10 - 28

Copyright © 2007, Oracle. All rights reserved.

CREATE TABLE: Example

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

Violating Constraints

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

```
Error starting at line 1 in command:
UPDATE employees
SET   department_id = 55
WHERE department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA16.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:      A foreign key value has no matching primary key value.
*Action:     Delete the foreign key or add a matching primary key.
```

Department 55 does not exist.

ORACLE

10 - 29

Copyright © 2007, Oracle. All rights reserved.

Violating Constraints

When you have constraints in place on columns, an error is returned if you try to violate the constraint rule. For example, if you try 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 not found” 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;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE      department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA16.EMP_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:      attempted to delete a parent key value that had a foreign
              dependency.
*Action:     delete dependencies first then parent or disable constraint.
```

ORACLE

10 - 30

Copyright © 2007, Oracle. All rights reserved.

Violating Constraints (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 rows deleted
```

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- **Creating a table using a subquery**
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

Creating a Table 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

10 - 32

Copyright © 2007, Oracle. All rights reserved.

Creating a Table Using a Subquery

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. Note that only the explicit `NOT NULL` constraint will be inherited. The `PRIMARY KEY` column will not pass the `NOT NULL` feature to the new column. Any other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

Creating a Table 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

ORACLE

10 - 33

Copyright © 2007, Oracle. All rights reserved.

Creating a Table Using a Subquery (continued)

The example in the slide 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 the column definitions by using the DESCRIBE command.

However, 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:

```
Error starting at line 1 in command:
CREATE TABLE dept80
AS SELECT employee_id, last_name,
salary*12,
hire_date FROM employees WHERE department_id = 80
Error at Command Line:3 Column:6
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 - "must name this expression with a column alias"
*Cause:
*Action:
```

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

ALTER TABLE Statement

Use the `ALTER TABLE` statement to:

- Add a new column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change table to read-only status

ORACLE

10 - 35

Copyright © 2007, 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 or its name needs to be changed.
- You need to remove columns.
- You want to put the table into the read-only mode

You can do this by using the `ALTER TABLE` statement.

Read-Only Tables

Use the `ALTER TABLE` syntax to put a table into the read-only mode:

- Prevents DDL or DML changes during table maintenance
- Change it back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

ORACLE

10 - 36

Copyright © 2007, Oracle. All rights reserved.

Read-Only Tables

With Oracle Database 11g, you can specify `READ ONLY` to place a table in the read-only mode. When the table is in the `READ-ONLY` mode, you cannot issue any DML statements that affect the table or any `SELECT ... FOR UPDATE` statements. You can issue DDL statements as long as they do not modify any data in the table. Operations on indexes associated with the table are allowed when the table is in the `READ ONLY` mode.

Specify `READ/WRITE` to return a read-only table to the read/write mode.

Note: You can drop a table that is in the `READ ONLY` mode. `DROP`

For information about the `ALTER TABLE` statement, see the course titled *Oracle Database 10g SQL Fundamentals II*.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement:
 - Access another user's tables
 - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
 - Read-only tables
- DROP TABLE statement

ORACLE

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the `PURGE` clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;
```

```
DROP TABLE dept80 succeeded.
```

ORACLE

10 - 38

Copyright © 2007, Oracle. All rights reserved.

Dropping a Table

The `DROP TABLE` statement moves a table to the recycle bin or removes the table and all its data from the database entirely. Unless you specify the `PURGE` clause, the `DROP TABLE` statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count towards the user's space quota. Dropping a table invalidates the dependent objects and removes object privileges on the 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 [PURGE]
```

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

Guidelines

- All the 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: Use the `FLASHBACK TABLE` statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

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

10 - 39

Copyright © 2007, 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.
- When executed, this statement cannot be rolled back.

Practice 10: 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
- Setting a table to read-only status
- Dropping tables

ORACLE

10 - 40

Copyright © 2007, Oracle. All rights reserved.

Practice 10: Overview

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table was added to the database. You also learn to set the status of a table as `READ ONLY` and then revert to `READ/WRITE`.

Note: For all the DDL and DML statements, click the Run Script icon (or press [F5]) to execute the query in SQL Developer. This way you get to see the feedback messages on the Script Output tab page. For `SELECT` queries, continue to click the Execute Statement icon or press [F9] to get the formatted output on the Results tab page.

Practice 10

1. Create the DEPT table based on the following table instance chart. Save the statement in a script called `lab_10_01.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

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

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

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. Save the statement in a script called `lab_10_03.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

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

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

Practice 10 (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. Alter the EMPLOYEES2 table status to read-only.
6. Try to insert the following row in the EMPLOYEES2 table:

ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_ID
34	Grant	Marcie	5678	10

You get the following error message:

```
Error starting at line 1 in command:
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
Error at Command Line:1 Column:12
Error report:
SQL Error: ORA-12081: update operation not allowed on table "ORA16"."EMPLOYEES2"
12081. 00000 - "update operation not allowed on table \"%s\".\"%s\""
*Cause:      An attempt was made to update a read-only materialized view.
*Action:     No action required. Only Oracle is allowed to update a
              read-only materialized view.
```

7. Revert the EMPLOYEES2 table to the read/write status. Now, try to insert the same row again. You should get the following messages:

```
ALTER TABLE employees2 succeeded.
1 rows inserted
```

8. Drop the EMPLOYEES2 table.

11

Creating Other Schema Objects

ORACLE

Copyright © 2007, 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

11 - 2

Copyright © 2007, 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.

Lesson Agenda

- Overview of views:
 - Creating, modifying, and retrieving data from a view
 - Data manipulation language (DML) operations on a view
 - Dropping a view
- Overview of sequences:
 - Creating, using, and modifying a sequence
 - Cache sequence values
 - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
 - Creating, dropping indexes
- Overview of synonyms
 - Creating, dropping synonyms

ORACLE

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 data retrieval queries
Synonym	Gives alternative names to objects

ORACLE

11 - 4

Copyright © 2007, Oracle. All rights reserved.

Database Objects

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

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

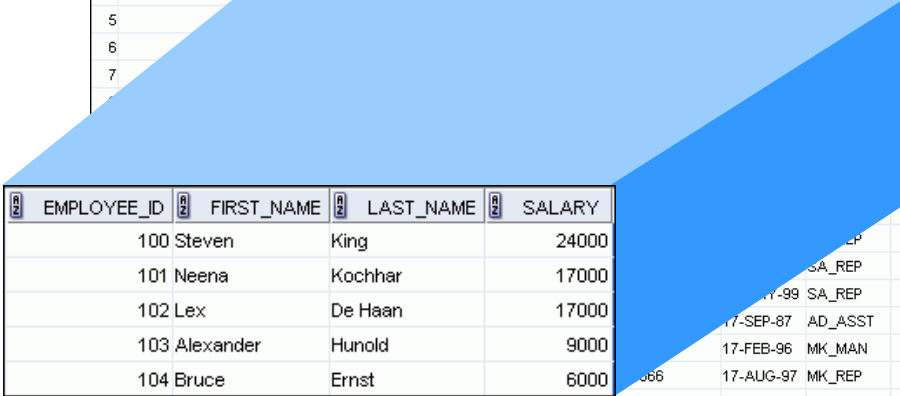
If you want to improve the performance of data retrieval queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

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	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
2	Neena	Kochhar	NKOCHH...	515.123.4568	21-SEP-89	AD_VP	17000
3	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
4	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
5							6000
6							4200
7							5800
							3500
							3100
							2600
							2500
							10500
							11000
							8600
							7000
							4400
							13000
							6000
19	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
20	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACC...	8300



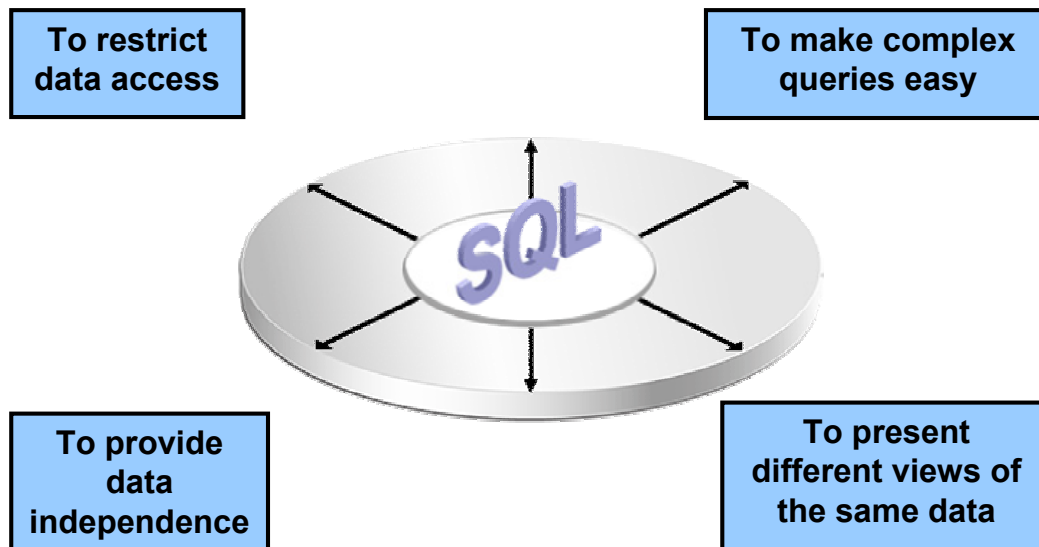
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	17000
102	Lex	De Haan	17000
103	Alexander	Hunold	9000
104	Bruce	Ernst	6000

ORACLE

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



Advantages of Views

- Views restrict access to the data because it displays 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 the section on “CREATE VIEW” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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

11 - 7

Copyright © 2007, 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

11 - 8

Copyright © 2007, Oracle. All rights reserved.

Creating a View

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

In the syntax:

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

Note: In SQL Developer, click the Run Script icon or press [F5] to run the data definition language (DDL) statements. The feedback messages will be shown on the Script Output tabbed page.

Creating a View

- Create the EMPVU80 view, which contains details of the 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 *iSQL*Plus* DESCRIBE command:

```
DESCRIBE empvu80
```

ORACLE

11 - 9

Copyright © 2007, 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)

Guidelines

- 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 the view created with the WITH CHECK OPTION, the system assigns a default name in the SYS_Cn format.
- You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it, or regrating the 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

11 - 10

Copyright © 2007, 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.

Alternatively, 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

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 OR REPLACE VIEW succeeded.
```

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

ORACLE

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 regranting 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.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE OR REPLACE VIEW dept_sum_vu
(name, minsal, maxsal, avgsal)
AS SELECT  d.department_name, MIN(e.salary),
          MAX(e.salary),AVG(e.salary)
FROM      employees e JOIN departments d
ON        (e.department_id = d.department_id)
GROUP BY d.department_name;
```

```
CREATE OR REPLACE VIEW succeeded.
```

ORACLE

11 - 13

Copyright © 2007, Oracle. All rights reserved.



Creating a Complex View

The example in the slide creates a complex view of department names, minimum salaries, maximum salaries, and the average salaries by department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression. You can view the structure of the view by using the DESCRIBE command. Display the contents of the view by issuing a SELECT statement.

```
SELECT *
FROM   dept_sum_vu;
```

	NAME	MINSAL	MAXSAL	AVGSAL
1	Administration	4400	4400	4400
2	Accounting	8300	12000	10150
3	IT	4200	9000	6400
4	Executive	17000	24000	19333.3333333333333333...
5	Shipping	2500	5800	3500
6	Sales	8600	11000	10033.3333333333333333...
7	Marketing	6000	13000	9500

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

Rules for 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

11 - 15

Copyright © 2007, Oracle. All rights reserved.

Rules for 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

11 - 16

Copyright © 2007, Oracle. All rights reserved.

Rules for 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 the 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 the section on “CREATE VIEW” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 OR REPLACE VIEW succeeded.

- Any attempt to INSERT a row with a department_id other than 20, or to UPDATE the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

ORACLE

11 - 17

Copyright © 2007, 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. Therefore it enables integrity constraints and data validation checks to be enforced on data being inserted or updated. 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;
```

causes:

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"
```

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 the 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

11 - 18

Copyright © 2007, 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 OR REPLACE VIEW succeeded.

ORACLE

11 - 19

Copyright © 2007, 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;
```

Similarly, any attempt to insert a row or modify a row using the view with a read-only constraint results in the same error.

```
Error report:
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

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

11 - 20

Copyright © 2007, 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. However, dropping views has no effect on the tables on which the view was based. On the other hand, views or other applications based on the 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 11: 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

ORACLE

11 - 21

Copyright © 2007, Oracle. All rights reserved.

Practice 11: 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.

Lesson Agenda

- Overview of views:
 - Creating, modifying, and retrieving data from a view
 - DML operations on a view
 - Dropping a view
- Overview of sequences:
 - Creating, using, and modifying a sequence
 - Cache sequence values
 - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
 - Creating, dropping indexes
- Overview of synonyms
 - Creating, dropping synonyms

ORACLE

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

11 - 23

Copyright © 2007, Oracle. All rights reserved.

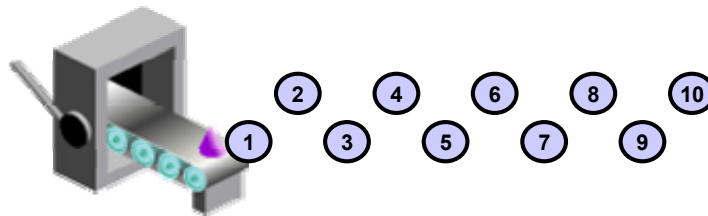
Sequences

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 shareable 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

11 - 24

Copyright © 2007, 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. A 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 independent 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

11 - 25

Copyright © 2007, Oracle. All rights reserved.

CREATE SEQUENCE Statement: Syntax

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

11 - 26

Copyright © 2007, Oracle. All rights reserved.

Creating a Sequence (continued)

<code>CYCLE NOCYCLE</code>	Specifies whether the sequence continues to generate values after reaching its maximum or minimum value (<code>NOCYCLE</code> is the default option.)
<code>CACHE n NOCACHE</code>	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 the section on “CREATE SEQUENCE” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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

11 - 27

Copyright © 2007, 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. However, 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 the sections on “Pseudocolumns” and “CREATE SEQUENCE” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 rows inserted

- View the current value for the DEPT_DEPTID_SEQ sequence:

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

ORACLE

11 - 29

Copyright © 2007, 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 using the *sequence_name*.CURRVAL, as shown in the second slide example. The output of the query is shown below:

	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

11 - 30

Copyright © 2007, 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, then 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. However, 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

11 - 31

Copyright © 2007, 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 the section on “ALTER SEQUENCE” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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

11 - 32

Copyright © 2007, 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;
```

- The error:

```
Error report:  
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value  
04009. 00000 - "MAXVALUE cannot be made to be less than the current value"  
*Cause:      the current value exceeds the given MAXVALUE  
*Action:     make sure that the new MAXVALUE is larger than the current value
```

Lesson Agenda

- Overview of views:
 - Creating, modifying, and retrieving data from a view
 - DML operations on a view
 - Dropping a view
- Overview of sequences:
 - Creating, using, and modifying a sequence
 - Cache sequence values
 - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
 - Creating, dropping indexes
- Overview of synonyms
 - Creating, dropping synonyms

ORACLE

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

11 - 34

Copyright © 2007, Oracle. All rights reserved.

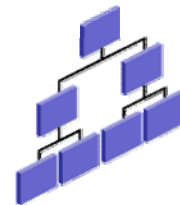
Indexes

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 a 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 input/output (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

11 - 35

Copyright © 2007, 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 disk I/O by using an indexed path to locate data quickly. An 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, the corresponding indexes are also dropped.

For more information, see the section on “Schema Objects: Indexes” in *Oracle Database Concepts 11g, Release 1 (11.1)*.

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

11 - 36

Copyright © 2007, Oracle. All rights reserved.

How Are Indexes Created?

You can create two types of indexes.

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` 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 the `FOREIGN KEY` column index for a join in a query to improve the speed of retrieval.

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 [UNIQUE] [BITMAP] 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

11 - 37

Copyright © 2007, Oracle. All rights reserved.

Creating an Index

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

In the syntax:

- `index` Is the name of the index
- `table` Is the name of the table
- `column` Is the name of the column in the table to be indexed

Specify `UNIQUE` to indicate that the value of the column (or columns) upon which the index is based must be unique. Specify `BITMAP` to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Bitmap indexes store the `rowids` associated with a key value as a bitmap.

For more information, see the section on “`CREATE INDEX`” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 <code>WHERE</code> 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

11 - 38

Copyright © 2007, Oracle. All rights reserved.

Index Creation Guidelines

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 `emp_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

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.

Lesson Agenda

- Overview of views:
 - Creating, modifying, and retrieving data from a view
 - DML operations on a view
 - Dropping a view
- Overview of sequences:
 - Creating, using, and modifying a sequence
 - Cache sequence values
 - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
 - Creating, dropping indexes
- Overview of synonyms
 - Creating, dropping synonyms

ORACLE

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

11 - 41

Copyright © 2007, 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.

Creating a Synonym for an Object

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

11 - 42

Copyright © 2007, 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 the section on “CREATE SYNONYM” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 d_sum succeeded.
```

ORACLE

11 - 43

Copyright © 2007, Oracle. All rights reserved.

Creating and Removing Synonyms

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
```

```
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;
```

For more information, see the section on "DROP SYNONYM" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 speed of query retrieval
- Use synonyms to provide alternative names for objects

ORACLE

11 - 44

Copyright © 2007, Oracle. All rights reserved.

Summary

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

Practice 11: Overview of Part 2

This practice covers the following topics:

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

ORACLE

11 - 45

Copyright © 2007, Oracle. All rights reserved.

Practice 11: 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 11

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 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	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
5	104	Ernst	60

...

19	205	Higgins	110
20	206	Gietz	110

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	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60

...

19	Higgins	110
20	Gietz	110

Practice 11 (continued)

- 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 have been 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.
- 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

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

Practice 11 (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_11_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`.