

## 2 Restricting and Sorting Data

ORACLE

Copyright © 2009, 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

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

Copyright © 2009, Oracle. All rights reserved.

## 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 these actions.

## Limiting Rows Using a Selection

### EMPLOYEES

	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
4	205	Higgins	AC_MGR	110
5	206	Gietz	AC_ACCOUNT	110
6	100	King	AD_PRES	90
7	101	Kochhar	AD_VP	90

...

20	178	Grant	SA_REP	(null)
----	-----	-------	--------	--------

**“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

Copyright © 2009, Oracle. All rights reserved.

### 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

Copyright © 2009, 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:

<code>WHERE</code>	restricts the query to rows that meet a condition
<code>condition</code>	is composed of column names, expressions, constants, and a comparison operator

The `WHERE` clause can compare values in columns, literal values, 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the WHERE Clause

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

## Character Strings and Dates

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

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Character Strings and Dates

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

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

**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.”

## Comparison Conditions

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Comparison Conditions

Comparison conditions 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 Conditions

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Using Comparison Conditions

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.



## Using the BETWEEN Condition

Use the BETWEEN condition 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

Copyright © 2009, Oracle. All rights reserved.

### Using the BETWEEN Condition

You can display rows based on a range of values using the BETWEEN range condition. 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 condition are inclusive. You must specify the lower limit first.

You can also use the BETWEEN condition 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

## Using the IN Condition

Use the IN membership condition 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	201	Hartstein	13000	100
2	101	Kochhar	17000	100
3	102	De Haan	17000	100
4	124	Mourgos	5800	100
5	149	Zlotkey	10500	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the IN Condition

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

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

The IN condition 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 in single quotation marks (' ').

## Using the LIKE Condition

- Use the LIKE condition 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%';
```

	FIRST_NAME
1	Shelley
2	Steven

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the LIKE Condition

You may not always know the exact value to search for. You can select rows that match a character pattern by using the LIKE condition. 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 employee first name from the EMPLOYEES table for any employee whose first name begins with the letter S. Note the uppercase S. Names beginning with an s are not returned.

The LIKE condition 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';
```

## Using the LIKE Condition

- You can combine pattern-matching characters:

```
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

Copyright © 2009, 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 Option

When you need to have an exact match for the actual `%` and `_` characters, use the `ESCAPE` option. 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` option identifies the backslash (`\`) as the escape character. In the pattern, 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

Copyright © 2009, 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 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 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	Whalen	AD_ASST	(null)
2	Hartstein	MK_MAN	(null)
3	Fay	MK_REP	(null)
4	Higgins	AC_MGR	(null)

...

16	Vargas	ST_CLERK	(null)
----	--------	----------	--------

## Logical Conditions

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 following condition is false

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Logical Conditions

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 one WHERE clause using the AND and OR operators.

## Using the AND Operator

AND requires both 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	201	Hartstein	MK_MAN	13000
2	149	Zlotkey	SA_MAN	10500

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the AND Operator

In the example, both conditions must be true for any record to be selected. Therefore, only 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. No rows are returned if 'MAN' is not uppercase. Character strings must be enclosed in 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 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	201	Hartstein	MK_MAN	13000
2	205	Higgins	AC_MGR	12000
3	100	King	AD_PRES	24000
4	101	Kochhar	AD_VP	17000
5	102	De Haan	AD_VP	17000
6	124	Mourgos	ST_MAN	5800
7	149	Zlotkey	SA_MAN	10500
8	174	Abel	SA_REP	11000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the OR Operator

In the example, either 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

Copyright © 2009, 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
```

## 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

Copyright © 2009, Oracle. All rights reserved.

### Rules of Precedence

The rules of precedence determine the order in which expressions are evaluated and calculated. The table lists the default order of precedence. 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

R	LAST_NAME	R	JOB_ID	R	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

R	LAST_NAME	R	JOB_ID	R	SALARY
1	King		AD_PRES		24000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rules of Precedence (continued)

#### 1. Example of the Precedence of the AND Operator

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. Example of Using Parentheses

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 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.”

## 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
...				
	Zlotkey	SA_MAN		80 29-JAN-00

ORACLE

Copyright © 2009, 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. If you use the ORDER BY clause, it must be the last clause of the SQL statement. 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.

## 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

- Sorting by multiple columns:

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

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Default Ordering of Data

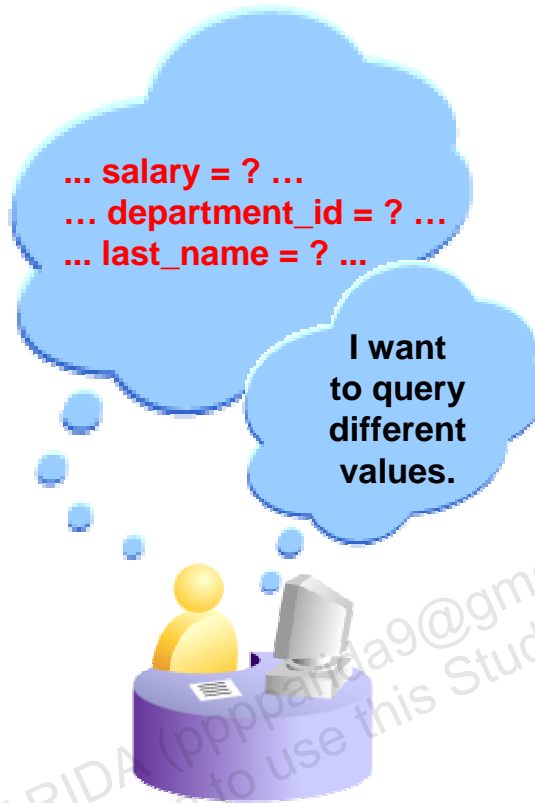
The default sort order is ascending:

- 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 alphabetical order (for example, A first and Z last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can sort by a column that is not in the SELECT list.

### Examples

1. To reverse the order in which 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 use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.
3. 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.

## Substitution Variables



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Substitution Variables

The examples so far have been hard-coded. In a finished application, the user would trigger the report, and the report would run without further prompting. The range of data would be predetermined by the fixed `WHERE` clause in the script file.

Using SQL Developer, 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 the values are temporarily stored. When the statement is run, the 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

Copyright © 2009, Oracle. All rights reserved.

### Substitution Variables (continued)

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

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

#### Examples of Restricted Ranges of Data

- 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 be used to achieve other goals, such as:

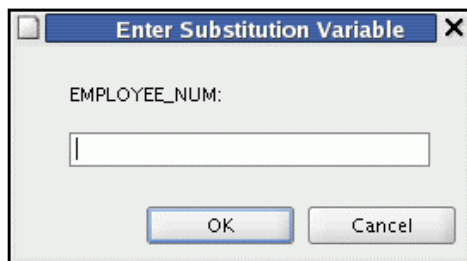
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

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

## Using the & 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

Copyright © 2009, Oracle. All rights reserved.

### Single-Ampersand Substitution Variable

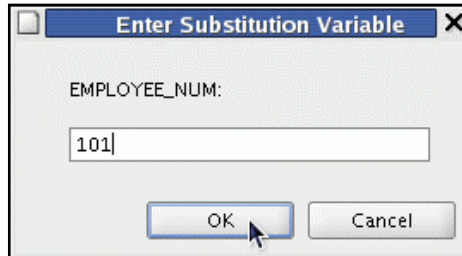
When running a report, users often want to restrict the data that is returned dynamically. SQL\*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. You do not need to define the value of each variable.

Notation	Description
<i>&amp;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 & Substitution Variable



A dialog box titled "Enter Substitution Variable" with a close button (X) in the top right corner. Inside the dialog, the text "EMPLOYEE\_NUM:" is followed by a text input field containing the value "101". At the bottom of the dialog are two buttons: "OK" and "Cancel". A mouse cursor is pointing at the "OK" button.

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 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 the 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

Copyright © 2009, Oracle. All rights reserved.

### Specifying Character and Date Values with Substitution Variables

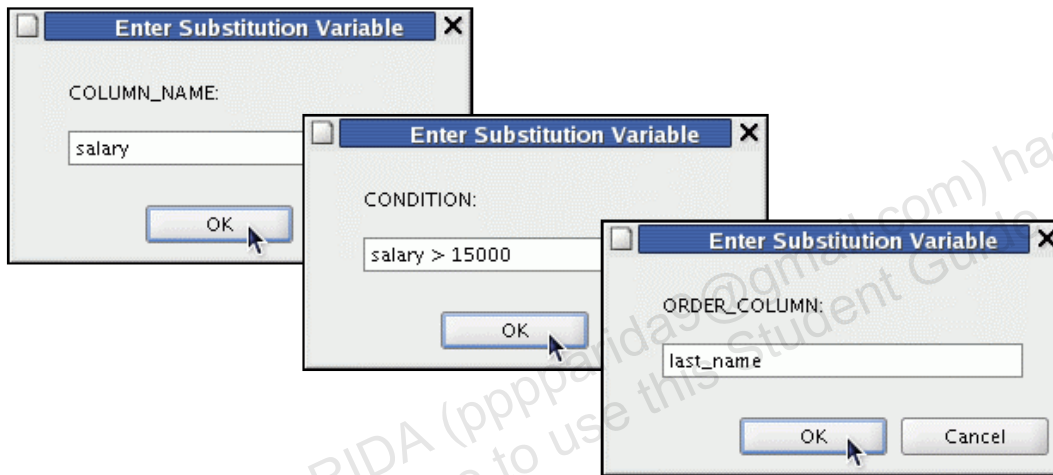
In a WHERE clause, date and character values must be enclosed in single quotation marks. The same rule applies to the substitution variables.

Enclose the variable in 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

Copyright © 2009, Oracle. All rights reserved.

### Specifying Column Names, Expressions, and Text

Not only can you use the substitution variables in the WHERE clause of a SQL statement, but these variables can also be used to substitute for column names, expressions, or text.

#### Example

The slide example displays the employee number, 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 you then click the OK button 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 && Substitution Variable

Use the 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 ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
...				
20	178	Grant	SA_REP	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Double-Ampersand Substitution Variable

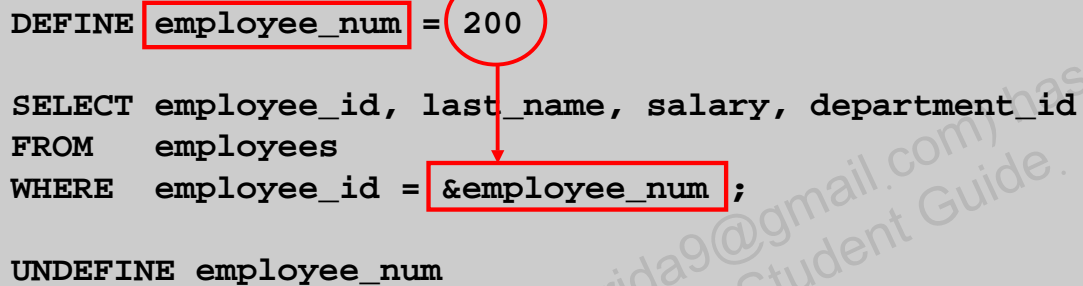
You can use the double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time. The user sees the prompt for the value only once. In the example in the slide, the user is asked to give the value for variable `column_name` only once. The value that is supplied by the user (`department_id`) is used for both display and ordering of data.

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 as follows:

```
UNDEFINE column_name
```

## 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.



The diagram illustrates the use of the `DEFINE` command. It shows a SQL script where a variable `employee_num` is defined with the value 200. This variable is then used in a `SELECT` statement's `WHERE` clause. A red box highlights `employee_num` in the `DEFINE` statement, and another red box highlights `&employee_num` in the `WHERE` clause. A red circle around the value 200 has an arrow pointing down to the `&employee_num` variable, indicating that the value 200 is substituted for the variable.

```
DEFINE employee_num = 200

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

UNDEFINE employee_num
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 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.

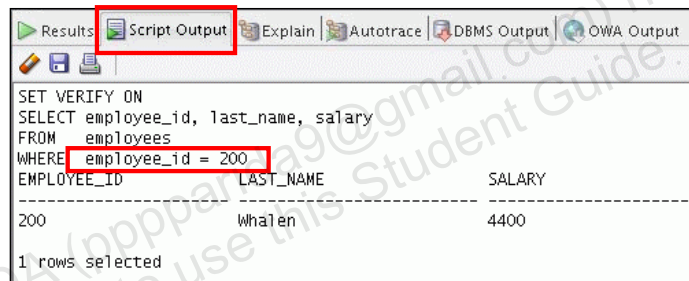
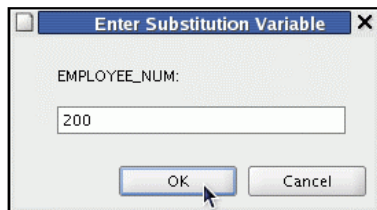
The `EMPLOYEE_NUM` substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

## 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:

```
SET VERIFY ON
```

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 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 conditions
  - Apply the logical AND, OR, and NOT operators
- Use the ORDER BY clause to sort rows of output:

```
SELECT  *|{[DISTINCT] column/expr [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

Copyright © 2009, Oracle. All rights reserved.

## 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. You can query users at run time and enable them to specify criteria.

## 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

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

Copyright © 2009, Oracle. All rights reserved.

### Practice 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 ampersand substitution.



## Practice 2

The HR department needs your assistance with creating some queries.

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

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

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

	A Z	LAST_NAME	A Z	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 \$5,000–\$12,000 range. Place your SQL statement in a text file named `lab_02_03.sql`.

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

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

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

**Practice 2 (continued)**

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

	A Z LAST_NAME	A Z 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.

	A Z Employee	A Z Monthly Salary
1	Fay	6000
2	Mourgos	5800

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

	A Z LAST_NAME	A Z 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.

	A Z LAST_NAME	A Z 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 the data in descending order of salary and commissions.

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

**Practice 2 (continued)**

10. Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. (You can use the query that you created in practice exercise 1 and modify it.) 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	Hartstein	13000
2	King	24000
3	Kochhar	17000
4	De Haan	17000

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

manager ID = 103, sorted by employee last name:

	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.”

	A-Z	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.

	A-Z	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 that of a sales representative or a stock clerk, and whose salaries are not equal to \$2,500, \$3,500, or \$7,000.

	A-Z	LAST_NAME	A-Z	JOB_ID	A-Z	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 amount is 20%. Resave lab\_02\_06.sql as lab\_02\_15.sql. Rerun the statement in lab\_02\_15.sql.

	A-Z	Employee	A-Z	Monthly Salary	A-Z	COMMISSION_PCT
1		Zlotkey		10500		0.2
2		Taylor		8600		0.2