Using the ORDER BY Clause

Sort the 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  order_id, order_date, order_status
FROM  orders
ORDER BY  order_date ;
```

	ORDER_ID	ORDER_DATE	ORDER_STATUS
1	2442	27-JUL-90 11.52.59.062632000 PM	9
2	2445	28-JUL-90 03.04.30.362632000 AM	8
3	2418	21-MAR-96 05.48.21.062632000 AM	4
4	2357	09-JAN-98 09.49.44.123456000 AM	1

...

2-1

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  order_id, round(order_date), order_status
FROM  orders
ORDER BY  order_date  desc;
```

1

Sorting by column alias:

```
SELECT  order_id, round(order_date), order_status "Order Status"
FROM  orders
ORDER BY  order_date desc;
```

2

2-2

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 example in the slide sorts the result by the most recently acquired order
2. You can also use a column alias in the `ORDER BY` clause.

Note: The `DESC` keyword used here for sorting in descending order should not be confused with the `DESC` keyword used to describe table structures.

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;
```

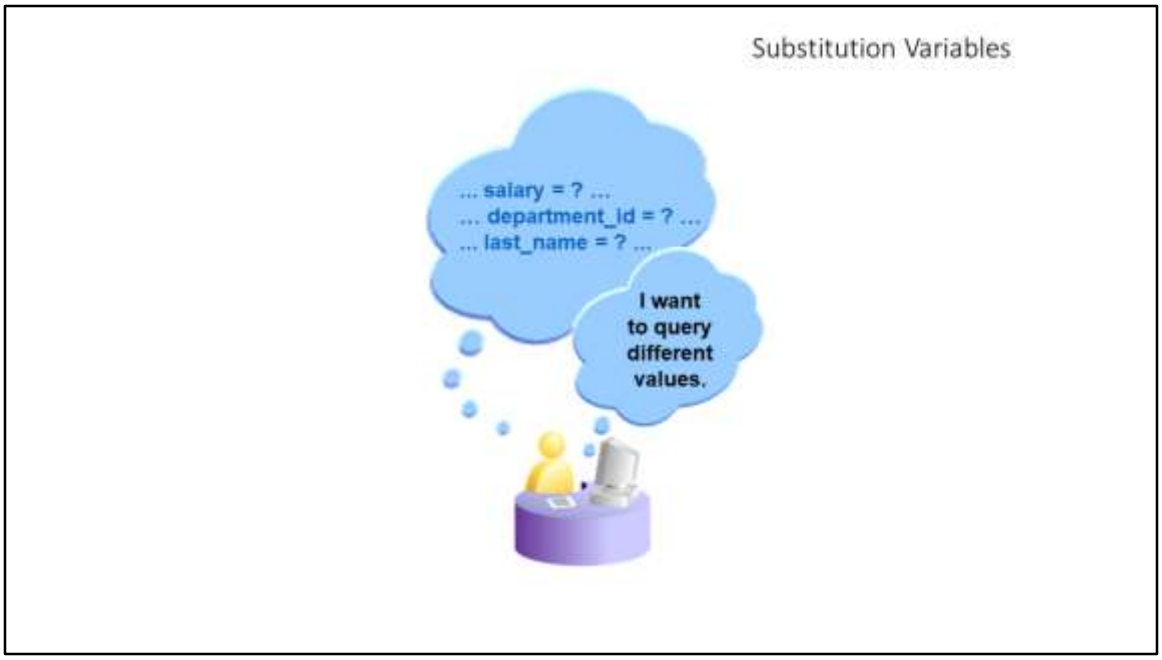
4

2-3

Sorting (continued)

Examples

3. You can sort query results by specifying the numeric position of the column in the `SELECT` clause. The example in the slide 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. The result of the query example shown in the slide is sorted by department_id in ascending order and also by salary in descending order.

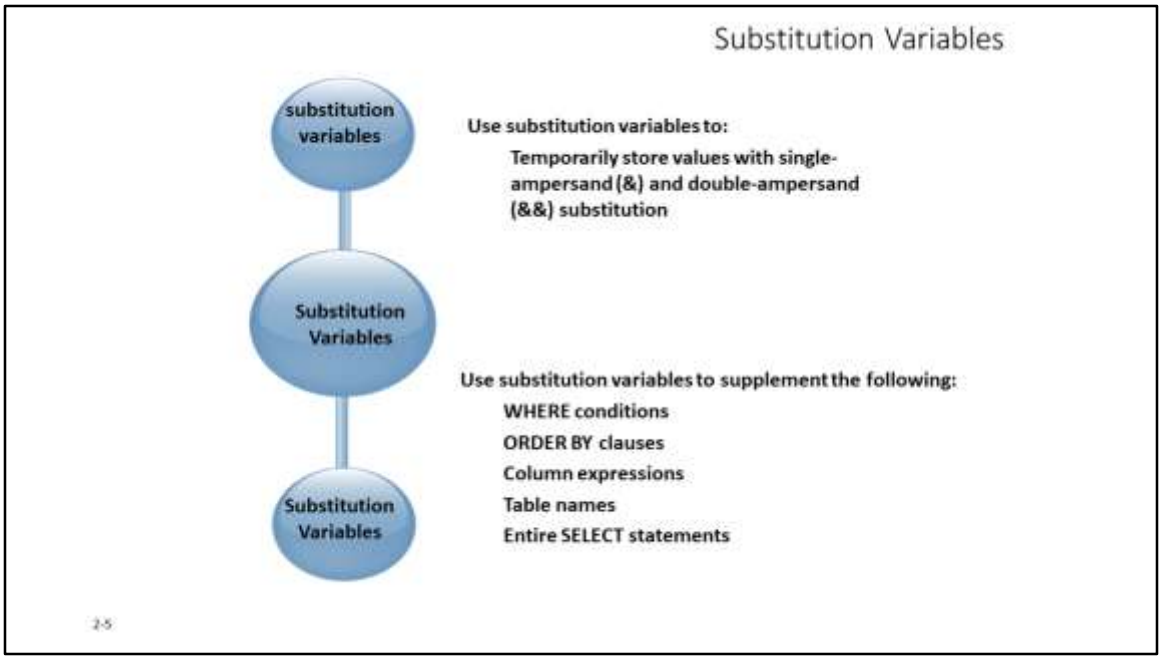


Substitution Variables

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 (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

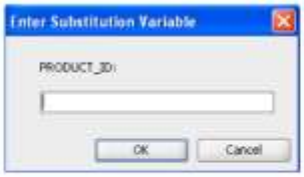
Passing values from one SQL statement to another

Note: Both SQL Developer and SQL* Plus support substitution variables and the `DEFINE/UNDEFINE` commands. Neither SQL Developer nor SQL* Plus support validation checks (except for data type) on user input. If used in scripts that are deployed to users, substitution variables can be subverted for SQL injection attacks.

Using the Single-Ampersand Substitution Variable

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

```
SELECT product_id, warehouse_id, quantity_on_hand
FROM   inventories
WHERE  product_id = &product_id;
```



2-6

Using the Single-Ampersand Substitution Variable

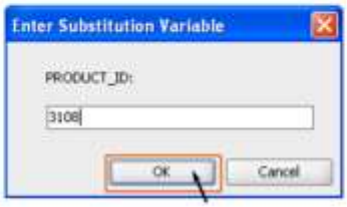
When running a report, users often want to restrict the data that is returned dynamically. SQL*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. However, you do not need to define the value of each variable.

Notation	Description
&user_variable	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 product ID. When the statement is executed, SQL Developer prompts the user for a product ID and then displays the product_ID, warehouse_ID and quantity_on_hand for that product ID.

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



	PRODUCT_ID	WAREHOUSE_ID	QUANTITY_ON_HAND
1	3108	8	122
2	3108	9	110
3	3108	2	194
4	3108	4	170
5	3108	6	146

2-7

Using the Single-Ampersand Substitution Variable (continued)

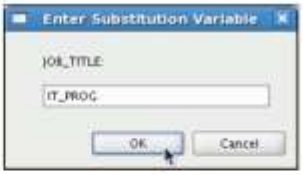
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

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

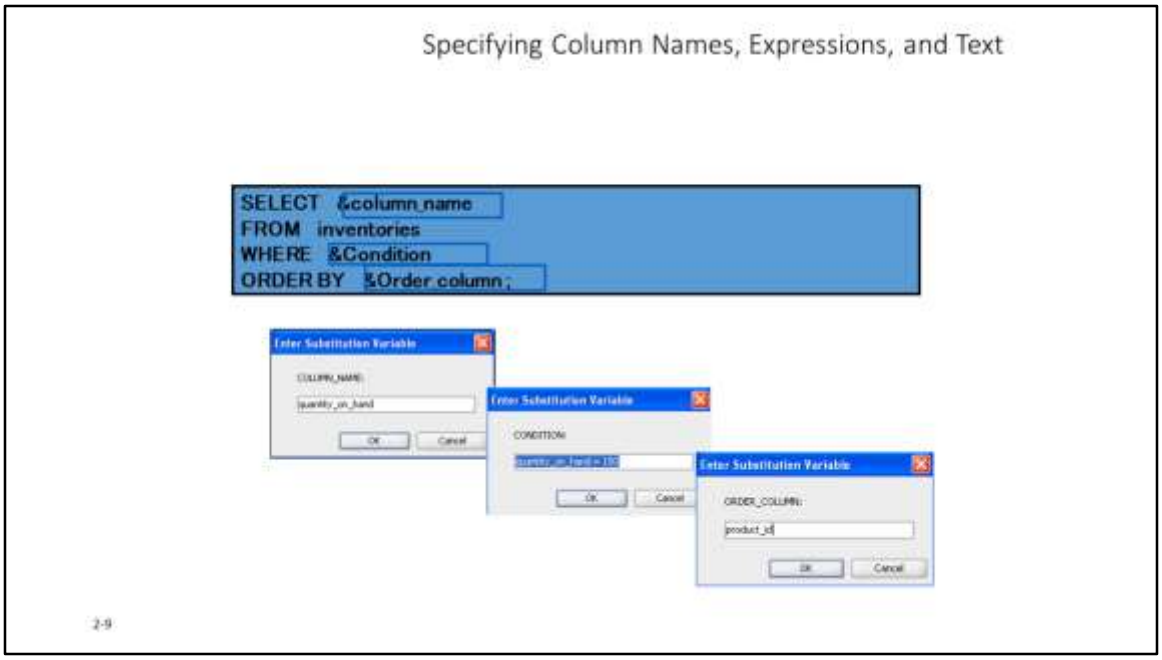
2-8

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

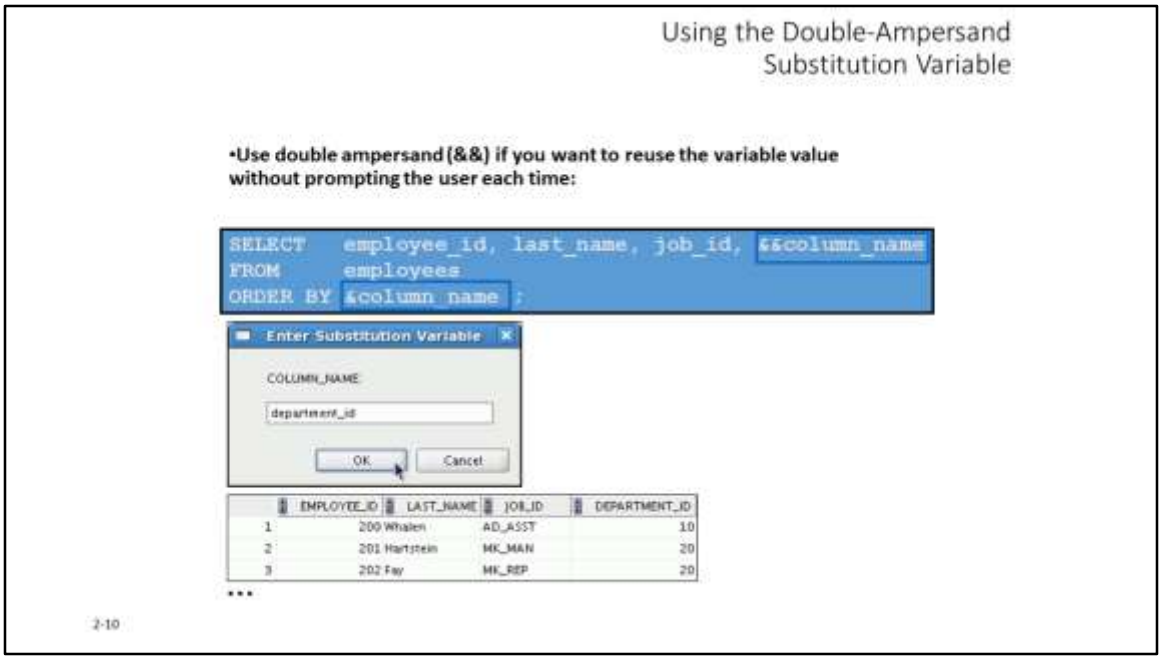
You can use the substitution variables not only in the `WHERE` clause of a SQL statement, but also as substitution for column names, expressions, or text.

Example

The example in the slide displays any column that is specified by the user at run time, from the `INVENTORIES` 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

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
```

Using the DEFINE Command

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

```
DEFINE order_num = 2458

SELECT order_id, order_date, order_mode, order_total
FROM orders
WHERE order_id = &order_num;

UNDEFINE order_num
```

2-11

Using the DEFINE Command

The example shown creates a substitution variable for an order number by using the DEFINE command. At run time, this displays the order_id, order_date, order_mode and order_total for that order.

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

The ORDER_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;
```

2-12

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.

Quiz

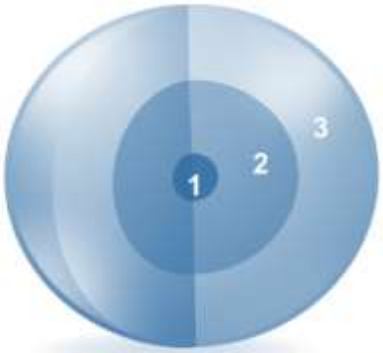
- Which of the following are valid operators for the `WHERE` clause?

1. `>=`
2. `IS NULL`
3. `!=`
4. `IS LIKE`
5. `IN BETWEEN`
6. `<>`

2-13

Answer: 1, 2, 3, 6

Session Summary



1. 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
2. Use ampersand substitution to restrict and sort output at run time
3. Use the ORDER BY clause to sort rows of output:

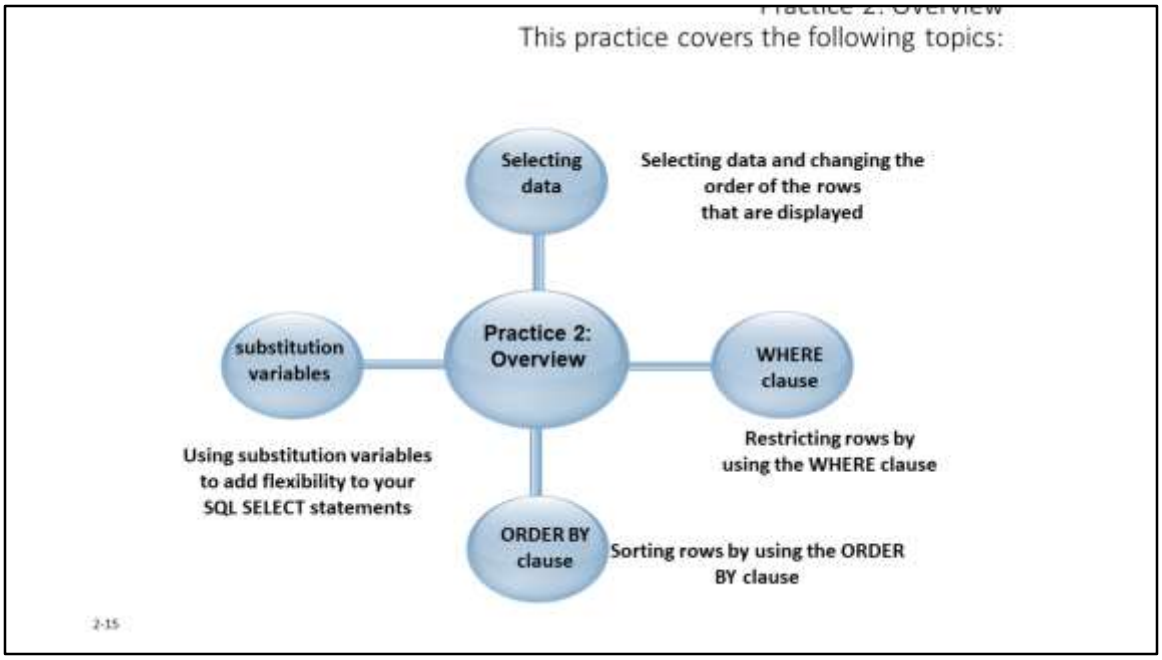
```
SELECT *|({DISTINCT} column|expression [alias],...}
FROM   table
[WHERE  condition(s)]
[ORDER BY {column, expr, alias} [ASC|DESC]];
```

2-14

Session 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

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.