3

Restricting and Sorting Data

ORACLE

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

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

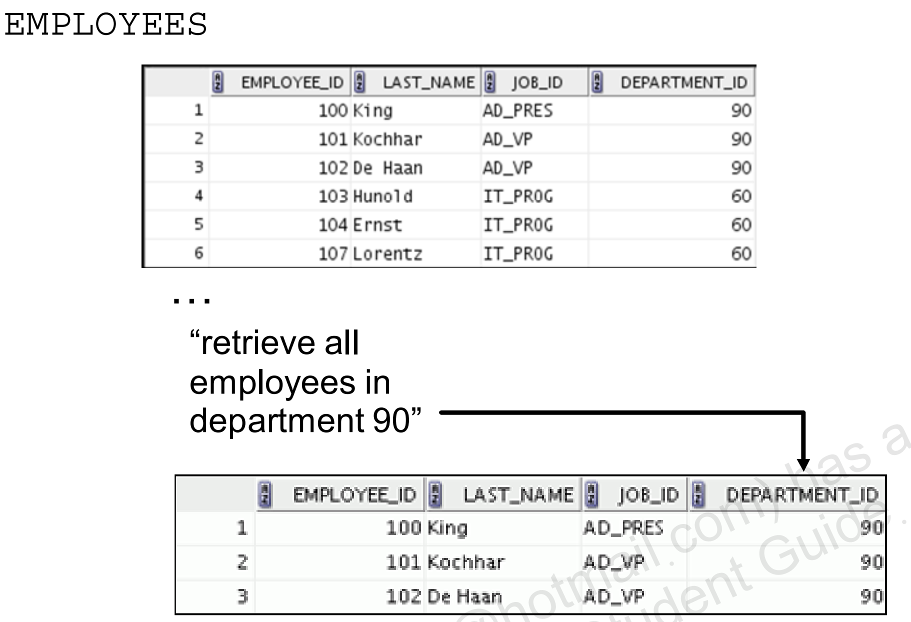
This lesson explains the SQL statements that you use to perform the actions listed in the slide.

Lesson Agenda

- Limiting rows with:
  - The WHERE clause
  - The comparison operators 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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands

ORACLE

Limiting Rows by 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

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 Rows That Are Selected

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

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

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

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:

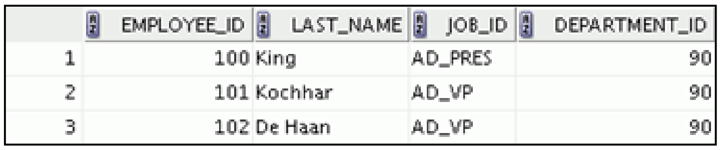
`WHERE`	Restricts the query to rows that meet a condition
`logical expression`	Is composed of column names, constants, and a comparison operator. It specifies a combination of one or more expressions and Boolean operators, and returns a value of `TRUE`, `FALSE`, or `KNOWN`.

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

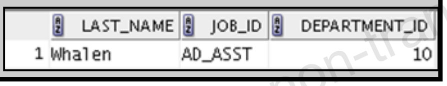
In the example, the `SELECT` statement retrieves the employee `ID`, last name, job `ID`, and department number of all employees who are in department 90.

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

Character Strings and Dates

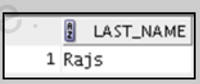
- Character strings and date values are enclosed within single quotation marks.
- Character values are case-sensitive and date values are format-sensitive.
- The default date display format is `DD-MON-RR`.

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



	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-OCT-03' ;
```



	LAST_NAME
1	Rajs

Character strings and dates in the `WHERE` clause must be enclosed within single quotation marks (`''`). Number constants, however, need not be enclosed within 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 are used in conditions that compare one expression with another value or expression. They are used in the `WHERE` clause in the following format:

Syntax

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

Example

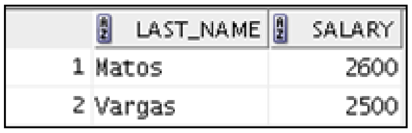
```
... WHERE hire_date = '01-JAN-05'
... WHERE salary >= 6000
... WHERE last_name = 'Smith'
```

Remember, 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
```

In the first example in the slide, 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.

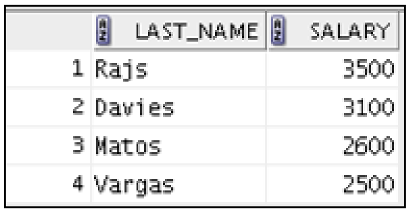
In the second code example, the `SELECT` statement retrieves all rows where the last name is either Ernst or Abel. Because * is used in the `SELECT` statement, all fields from the employees table would appear in the result set.

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

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.

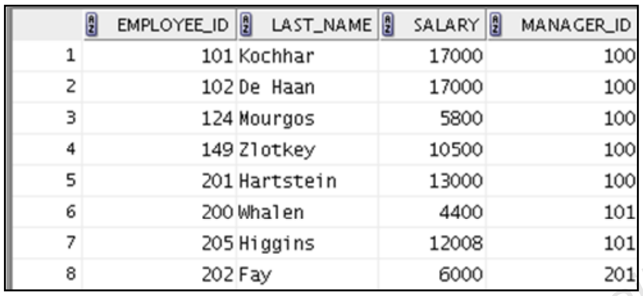
You can also use the `BETWEEN` operator on character values:

```
SELECT last_name FROM   employees
WHERE  last_name BETWEEN 'King' AND 'Whalen 10
```

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	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

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 example in the slide 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.

Note: The set of values can be specified in any random order—for example, (201,100,101).

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:

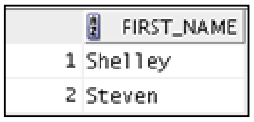
```
SELECT employee_id, manager_id, department_id
FROM   employees
WHERE  last_name IN ('Hartstein', 'Vargas');
```

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

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 more characters.
  – _ denotes one character.

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



```
   FIRST_NAME
 1 Shelley
 2 Steven
```

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.

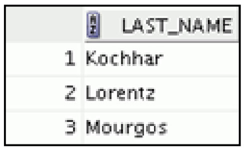
The LIKE operator can be used as a shortcut for some BETWEEN comparisons. The following example displays the last names and hire dates of all employees who joined between January, 2005 and December, 2005:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date LIKE '%05';
```

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.

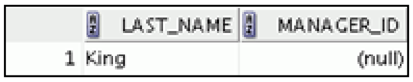
The % and _ symbols can be used in any combination with literal characters. The example in the slide displays the names of all employees whose last names have the letter "o" as the second character.

When you need to have an exact match for the actual % and _ characters, use the ESCAPE identifier.

Using 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)

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 example in the slide retrieves the `last_name` and `manager_id` 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;
```

. . .

Defining Conditions Using Logical Operators

Operator	Meaning
`AND`	Returns `TRUE` if *both* component conditions are true
`OR`	Returns `TRUE` if *either* component condition is true
`NOT`	Returns `TRUE` if the condition is false

A logical condition combines the results of two or more 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:

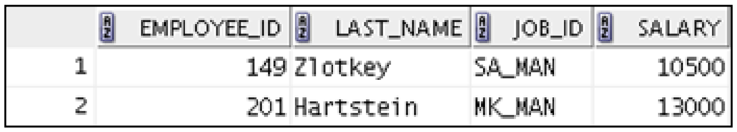
- `AND`
- `OR`
- `NOT`

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
```

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 within quotation marks.

AND Truth Table

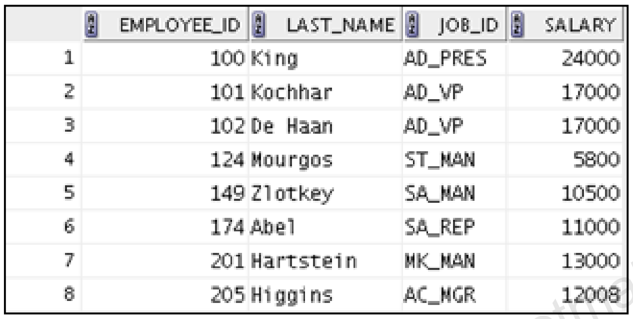
The following table shows the results of combining two expressions with AND:

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	12008

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 both is selected.

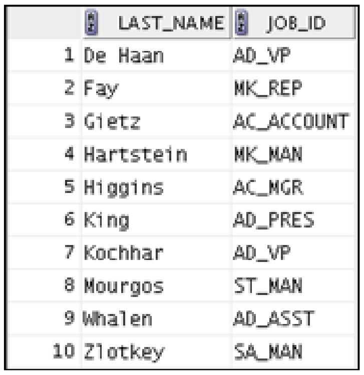
`OR` Truth Table

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

The example in the slide 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

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
- SQL row limiting clause in a query
- 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 operator
8	`AND` logical operator
9	`OR` logical operator

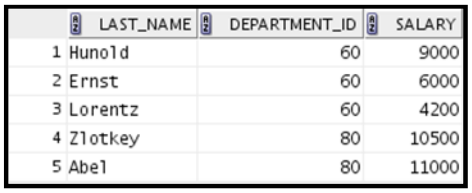
You can use parentheses to override rules of precedence.

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, department_id, salary
FROM   employees
WHERE  department_id = 60
OR     department_id = 80
AND    salary > 10000;
```

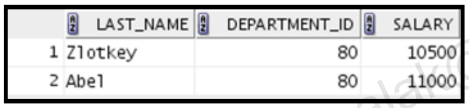
1



	LAST_NAME	DEPARTMENT_ID	SALARY
1	Hunold	60	9000
2	Ernst	60	6000
3	Lorentz	60	4200
4	Zlotkey	80	10500
5	Abel	80	11000

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (department_id = 60
OR     department_id = 80)
AND    salary > 10000;
```

2



	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

1. **Precedence of the `AND` Operator: Example**

   In this example, there are two conditions:
   - The first condition is that the department `ID` is `80` *and* the salary is greater than $10,000.
   - The second condition is that the department `ID` is `60`.

   Therefore, the `SELECT` statement reads as follows:

   "Select the row if an employee's department ID is 80 *and* earns more than $10,000, *or* if the employee's department ID is 60."
2. **Using Parentheses: Example**

   In this example, there are two conditions:
   - The first condition is that the department `ID` is 80 *or* 60.
   - The second condition is that the salary is greater than $10,000.

   Therefore, the `SELECT` statement reads as follows:

   "Select the row if an employee's department ID is 80 *or* 60, *and* if the employee earns more than $10,000."

Lesson Agenda

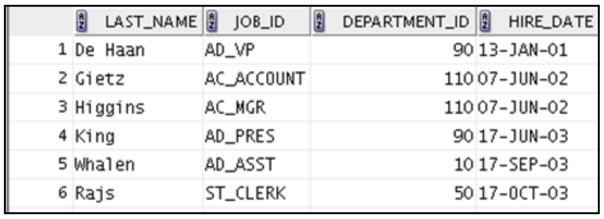
- Limiting rows with:
  - The `WHERE` clause
  - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- SQL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

Using the ORDER BY Clause

Sort the retrieved rows with the ORDER BY clause:

- ASC: Ascending order, default
- DESC: Descending order

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



	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP	90	13-JAN-01
2	Gietz	AC_ACCOUNT	110	07-JUN-02
3	Higgins	AC_MGR	110	07-JUN-02
4	King	AD_PRES	90	17-JUN-03
5	Whalen	AD_ASST	10	17-SEP-03
6	Rajs	ST_CLERK	50	17-OCT-03

…

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. You can specify an expression, an alias, or a column position as the sort condition. You can specify multiple expressions in the *order_by_clause*. Oracle Database first sorts rows based on their values for the first expression. Rows with the same value for the first expression are then sorted based on their values for the second expression, and so on.

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 department_id DESC ;
```

1

- Sorting by column alias:

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

2

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 department_id.
2. You can also use a column alias in the `ORDER BY` clause. The slide example sorts the data by annual salary.

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 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

Examples

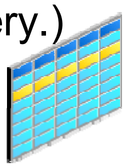
3. You can sort query results by specifying the numeric position of the column in the `SELECT` clause. The example in the slide sorts the result by the `department_id` as this column is at the third position in the `SELECT` clause.
4. You can sort query results by more than one column. You list the columns (or `SELECT` list column sequence numbers) in the `ORDER BY` clause, delimited by commas. The results are ordered by the first column, then the second, and so on for as many columns as the `ORDER BY` clause includes. If you want any results sorted in descending order, your `ORDER BY` clause must use the `DESC` keyword directly after the name or the number of the relevant column. 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.

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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands

SQL Row Limiting Clause

- You can use the `row_limiting_clause` to limit the rows that are returned by a query.
- You can use this clause to implement Top-N reporting.
- You can specify the number of rows or percentage of rows to return with the `FETCH FIRST` keyword.
- You can use the `OFFSET` keyword to specify that the returned rows begin with a row after the first row of the full result set.
- The `WITH TIES` keyword includes additional rows with the same ordering keys as the last row of the row-limited result set. (You must specify `ORDER BY` in the query.)



In Oracle Database 12*c* Release 1, SQL `SELECT` syntax has been enhanced to allow a `row_limiting_clause`, which limits the number of rows that are returned in the result set. The `row_limiting_clause` provides both easy-to-understand syntax and expressive power. Limiting the number or rows returned can be valuable for reporting, analysis, data browsing, and other tasks. Queries that order data and then limit row output are widely used and are often referred to as Top-N queries. Top-N queries sort their result set and then return only the first n rows.

You can specify the number of rows or percentage of rows to return with the `FETCH FIRST` keywords. You can use the `OFFSET` keyword to specify that the returned rows begin with a row after the first row of the full result set. The `WITH TIES` keywords includes rows with the same ordering keys as the last row of the row-limited result set (you must specify the `ORDER BY` clause with the `WITH TIES` clause in the query). `FETCH FIRST` and `OFFSET` keywords simplify syntax and comply with the ANSI SQL standard.

There are certain limitations of the SQL row limiting clause:

- You cannot specify this clause with the `for_update_clause`.
- You cannot specify this clause in the subquery of a `DELETE` or `UPDATE` statement.
- If you specify this clause, then the select list cannot contain the sequence pseudocolumns `CURRVAL` or `NEXTVAL`.

Using SQL Row Limiting Clause in a Query

You specify the `row_limiting_clause` in the SQL `SELECT` statement by placing it after the `ORDER BY` clause.

Syntax:

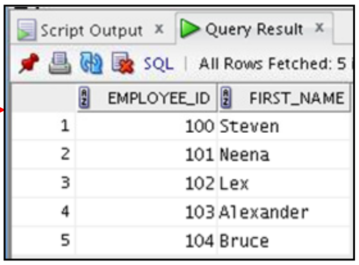
```
SELECT …
   FROM …
 [ WHERE …  ]
 [ ORDER BY …  ]
 [OFFSET offset { ROW | ROWS }]
[FETCH { FIRST | NEXT } [{ row_count | percent PERCENT
}] { ROW | ROWS }
  { ONLY | WITH TIES }]
```

You specify the `row_limiting_clause` in the SQL `SELECT` statement by placing it after the `ORDER BY` clause. Note that an `ORDER BY` clause is required if you want to sort the rows for consistency.

- `OFFSET`: Use this clause to specify the number of rows to skip before row limiting begins. The value for offset must be a number. If you specify a negative number, offset is treated as 0. If you specify `NULL` or a number greater than or equal to the number of rows that are returned by the query, 0 rows are returned.
- `ROW` | `ROWS`: Use these keywords interchangeably. They are provided for semantic clarity.
- `FETCH`: Use this clause to specify the number of rows or percentage of rows to return.
- `FIRST` | `NEXT`: Use these keywords interchangeably. They are provided for semantic clarity.
- `row_count` | *percent* `PERCENT`: Use `row_count` to specify the number of rows to return. Use *percent* `PERCENT` to specify the percentage of the total number of selected rows to return. The value for percent must be a number.
- `ONLY` | `WITH TIES`: Specify `ONLY` to return exactly the specified number of rows or percentage of rows. Specify `WITH TIES` to return additional rows with the same sort key as the last row fetched. If you specify `WITH TIES`, then you must specify the `order_by_clause`. If you do not specify the `order_by_clause`, then no additional rows will be returned.

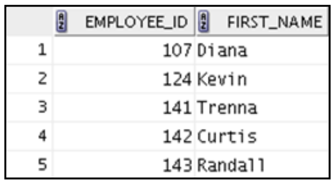
SQL Row Limiting Clause: Example

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;
```



```
   EMPLOYEE_ID FIRST_NAME
1          100 Steven
2          101 Neena
3          102 Lex
4          103 Alexander
5          104 Bruce
```

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```



```
   EMPLOYEE_ID FIRST_NAME
1          107 Diana
2          124 Kevin
3          141 Trenna
4          142 Curtis
5          143 Randall
```

The first code example returns the five employees with the lowest `employee_id`.

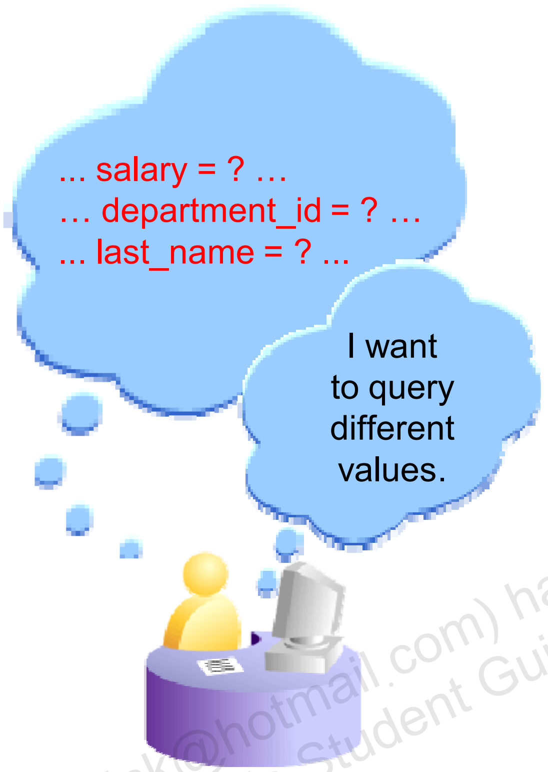
The second code example returns the five employees with the next set of lowest `employee_id`.

Note: If `employee_id` is assigned sequentially by the date when the employee joined the organization, these examples give us the top 5 employees and then employees 6-10, all in terms of seniority.

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
- SQL row limiting clause in a query
- Substitution variables
- `DEFINE` and `VERIFY` commands

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

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:

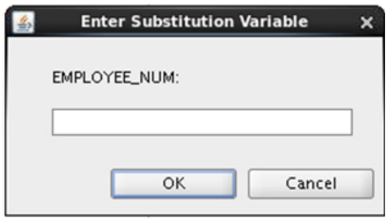
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

Note: Both SQL Developer and SQL* Plus support substitution variables and the `DEFINE/UNDEFINE` commands.

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 ;
```



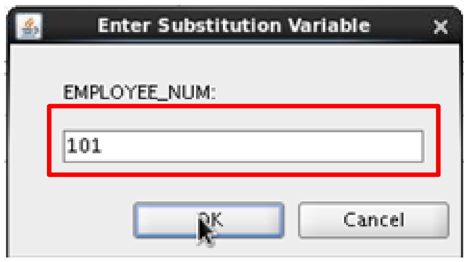
When running a report, users often want to restrict the data that is returned dynamically. SQL*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. However, you do not need to define the value of each variable.

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.)

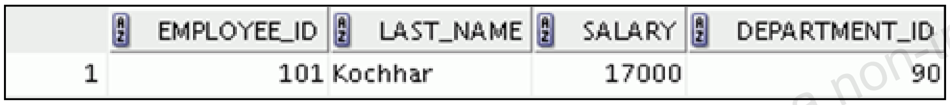
The example in the slide creates a SQL Developer substitution variable for an employee number. When the statement is executed, SQL Developer prompts the user for an employee number and then displays the employee number, last name, salary, and department number for that employee.

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_NUM: 101



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

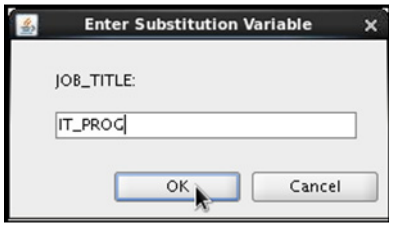
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

After you enter a value and click the OK button, the results are displayed.

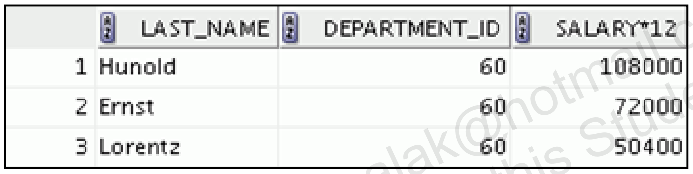
Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

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



JOB_TITLE: IT_PROG



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

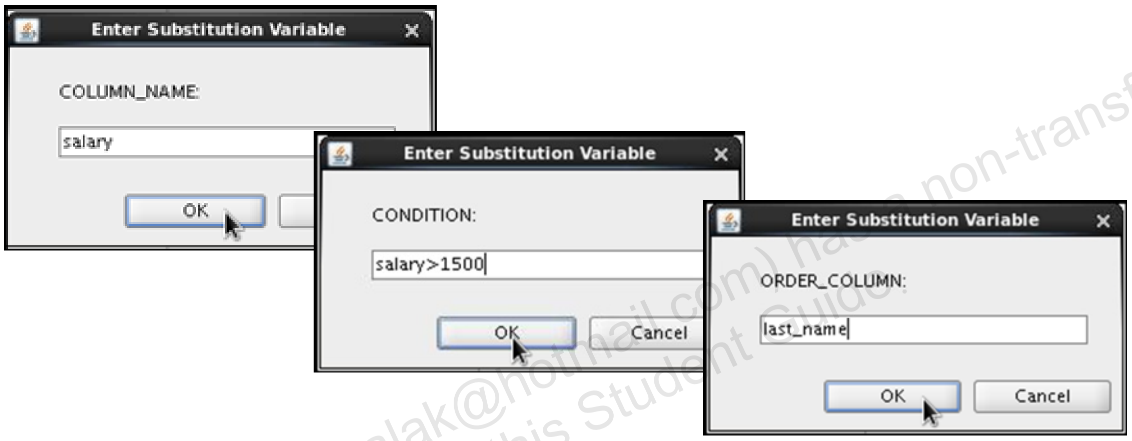
In a `WHERE` clause, date and character values must be enclosed within single quotation marks. The same rule applies to the substitution variables.

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 ;
```


Enter Substitution Variable
COLUMN_NAME:
salary
OK
Enter Substitution Variable
CONDITION:
salary>1500
OK
Cancel
Enter Substitution Variable
ORDER_COLUMN:
last_name
OK
Cancel

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 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.

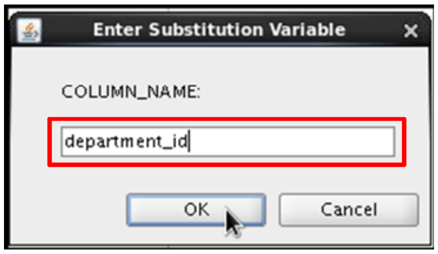
If you do not enter a value for the substitution variable, you get an error when you execute the preceding statement.

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

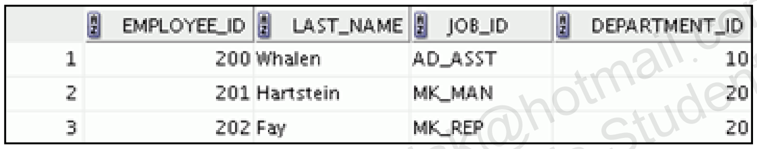
Using the Double-Ampersand Substitution Variable

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 ;
```



department_id



	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

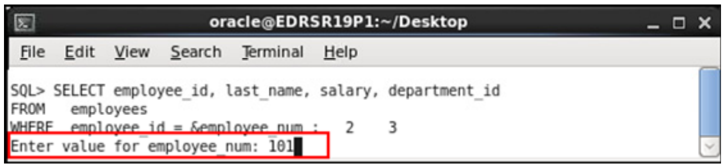
. . .

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.

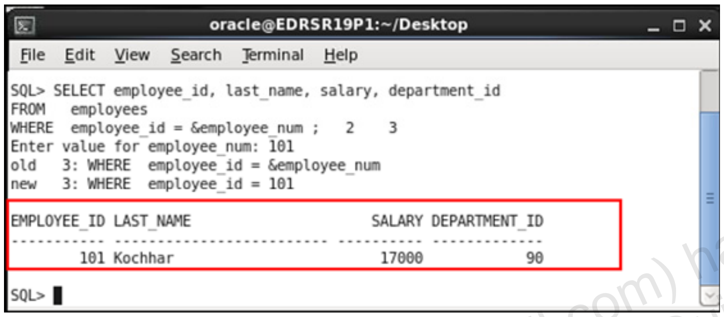
SQL Developer stores the value that is supplied by using the `DEFINE` command; it uses it again whenever you reference the variable name. After a user variable is in place, you need to use the `UNDEFINE` command to delete it:

```
UNDEFINE column_name;
```

Using the Ampersand Substitution Variable in SQL*Plus



```
SQL> SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;   2    3
Enter value for employee_num: 101
```



```
SQL> SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;   2    3
Enter value for employee_num: 101
old   3: WHERE  employee_id = &employee_num
new   3: WHERE  employee_id = 101

EMPLOYEE_ID LAST_NAME                     SALARY DEPARTMENT_ID
----------- ------------------------- ---------- -------------
        101 Kochhar                        17000            90

SQL>
```

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

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
- SQL row limiting clause in a query
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

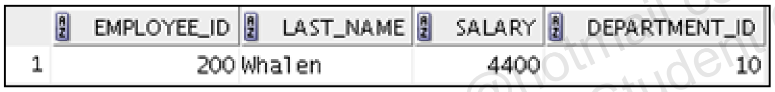
Using the `DEFINE` Command

- 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
```



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

ORACLE

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.

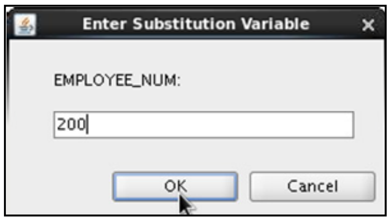
Because the variable is created using the SQL Developer `DEFINE` command, the user is not prompted to enter a value for the employee number. Instead, the defined variable value is automatically substituted in the `SELECT` statement.

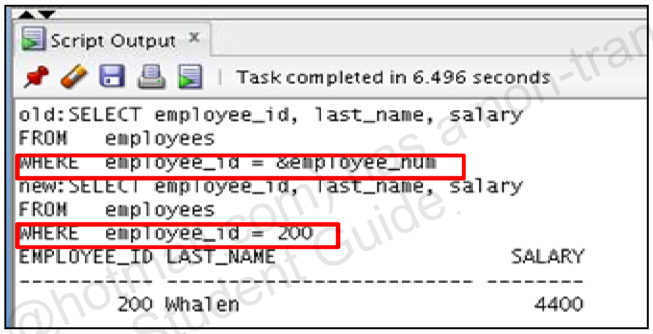
The `EMPLOYEE_NUM` substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

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;
```





```
old:SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = &employee_num
new:SELECT employee_id, last_name, salary
FROM   employees
WHERE  employee_id = 200
EMPLOYEE_ID LAST_NAME                    SALARY
----------- ------------------------- ---------
        200 Whalen                         4400
```

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 four of the following are valid operators for the `WHERE` clause?

a. `>=`

b. `IS NULL`

c. `!=`

d. `IS LIKE`

e. `IN BETWEEN`

f. `<>`

Answer: a, b, c, f

Summary

In this lesson, you should have learned how to:

- 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

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 3: 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

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.