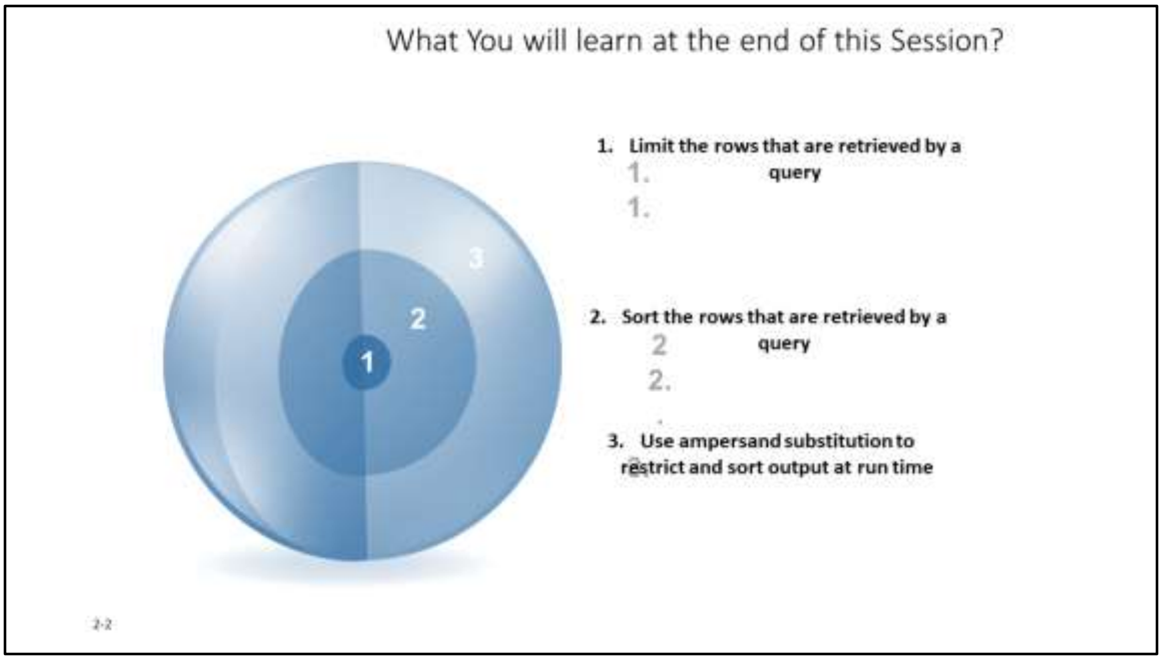
Lesson 1

Restricting and Sorting Data



What You will learn at the end of this Session?

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

2-2

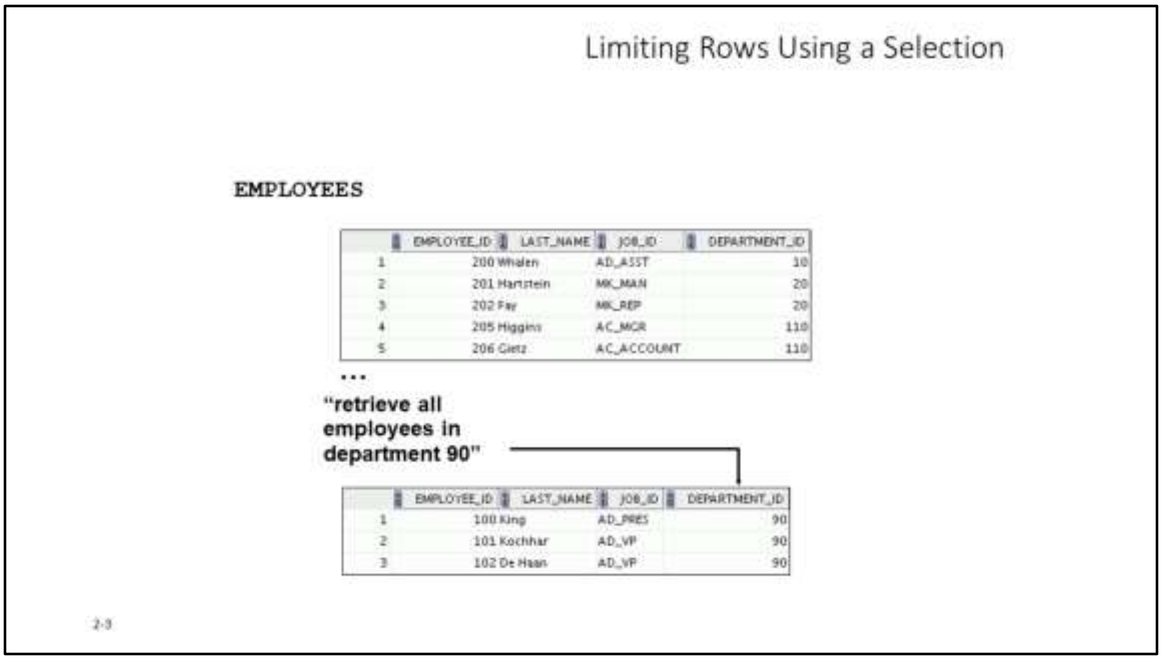
What You will learn at the end of this session?

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

Restrict the rows of data that are displayed

Specify the order in which the rows are displayed

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



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

...

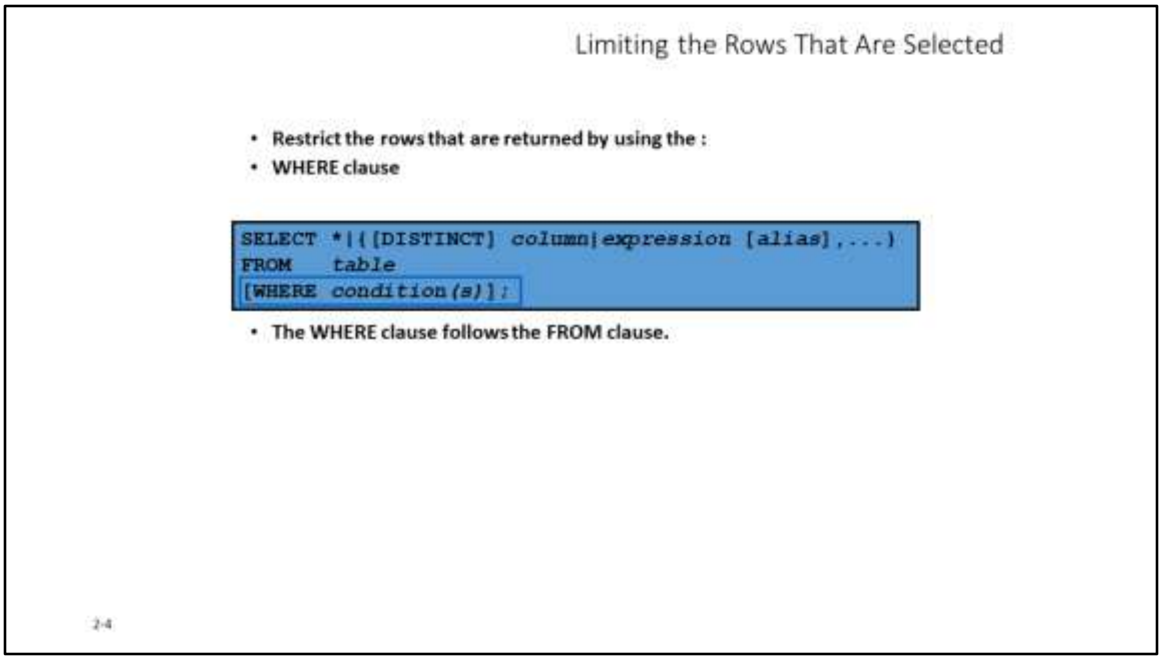
"retrieve all employees in department 90"

	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

2-3

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.

2-4

Limiting the Rows That Are Selected

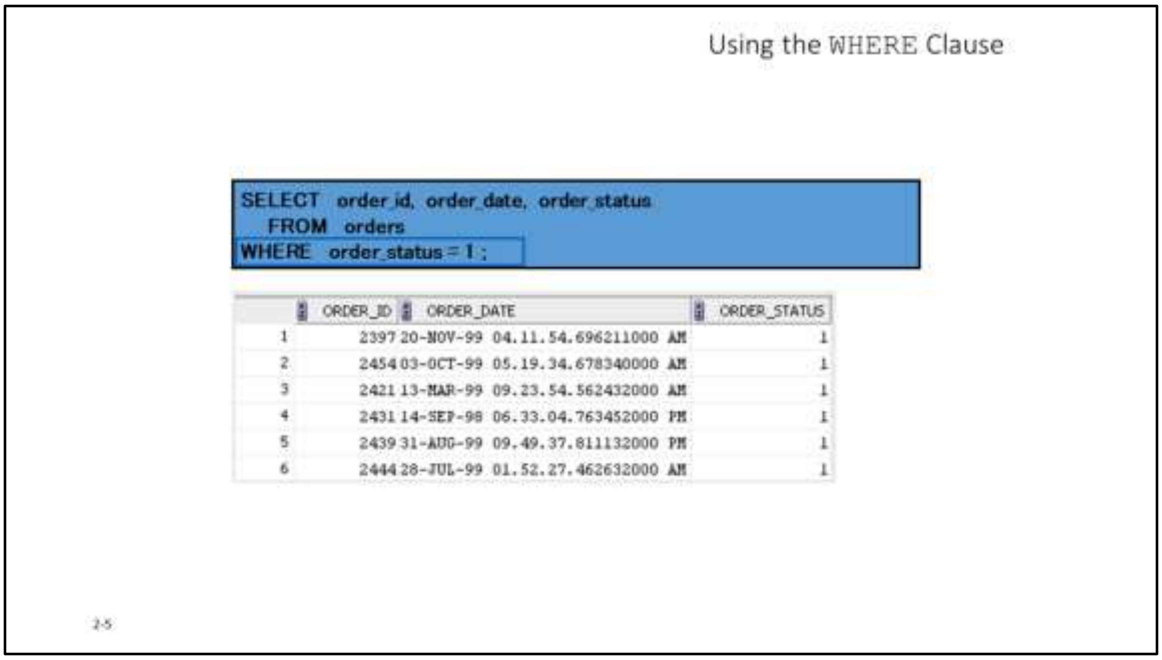
You can restrict the rows that are returned from the query by using the `WHERE` clause. A `WHERE` clause contains a condition that must be met and it directly follows the `FROM` clause. If the condition is true, the row meeting the condition is returned.

In the syntax:

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

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

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



Using the WHERE Clause

```
SELECT   order_id,  order_date,  order_status
   FROM   orders
WHERE   order_status = 1 ;
```

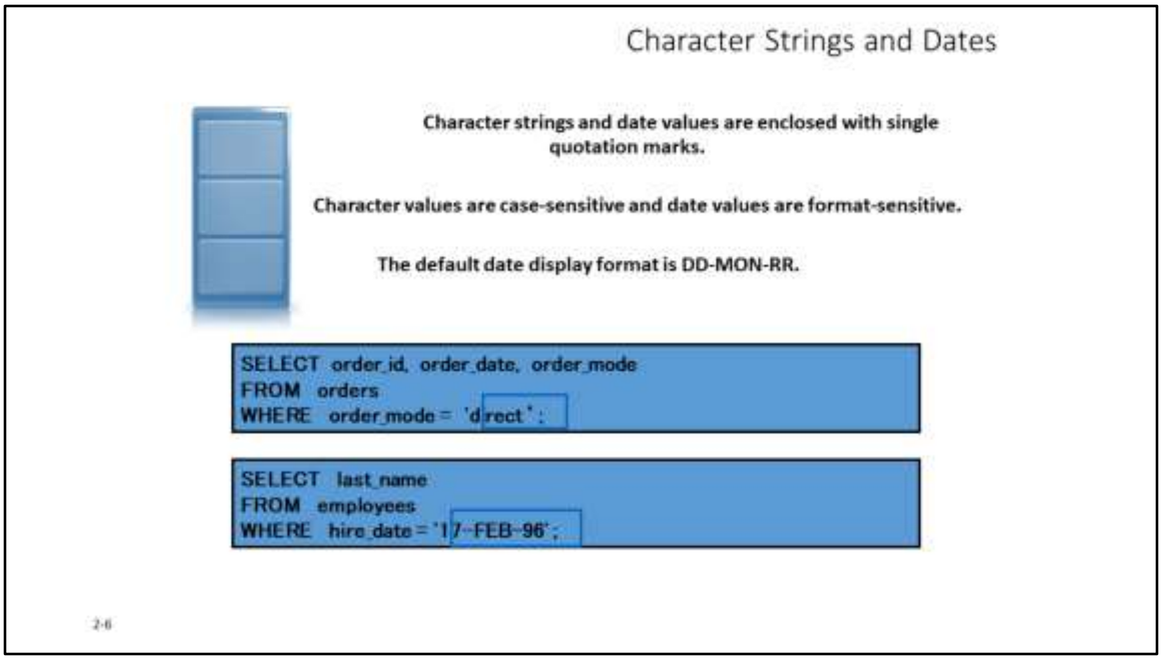
	ORDER_ID	ORDER_DATE	ORDER_STATUS
1	2397	20-NOV-99 04.11.54.696211000 AM	1
2	2454	03-OCT-99 05.19.34.678340000 AM	1
3	2421	13-MAR-99 09.23.54.562432000 AM	1
4	2431	14-SEP-98 06.33.04.763452000 PM	1
5	2439	31-AUG-99 09.49.37.811132000 PM	1
6	2444	28-JUL-99 01.52.27.462632000 AM	1

2-5

Using the WHERE Clause

In the example, the SELECT statement retrieves the order ID, order date and order status of all orders whose status is 1.

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



Character Strings and Dates

Character strings and date values are enclosed with single quotation marks.

Character values are case-sensitive and date values are format-sensitive.

The default date display format is DD-MON-RR.

```
SELECT  order_id,  order_date,  order_mode
FROM   orders
WHERE   order_mode =  'direct' ;
```

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

2-6

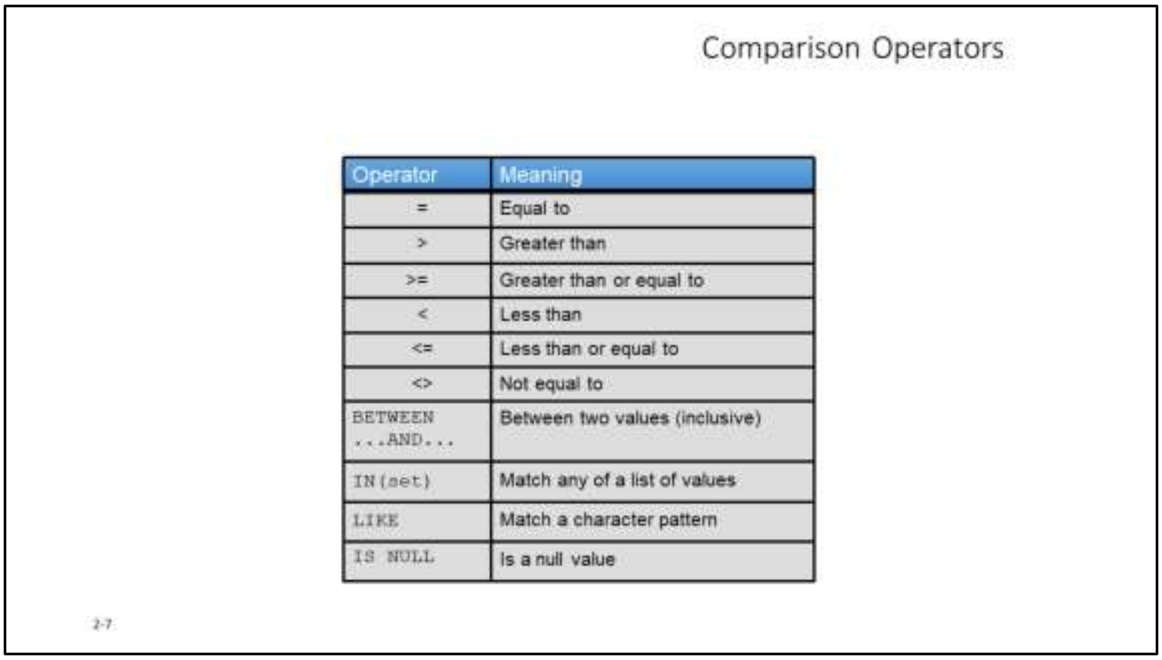
Character Strings and Dates

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

All character searches are case-sensitive.

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

2-7

Comparison Operators

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

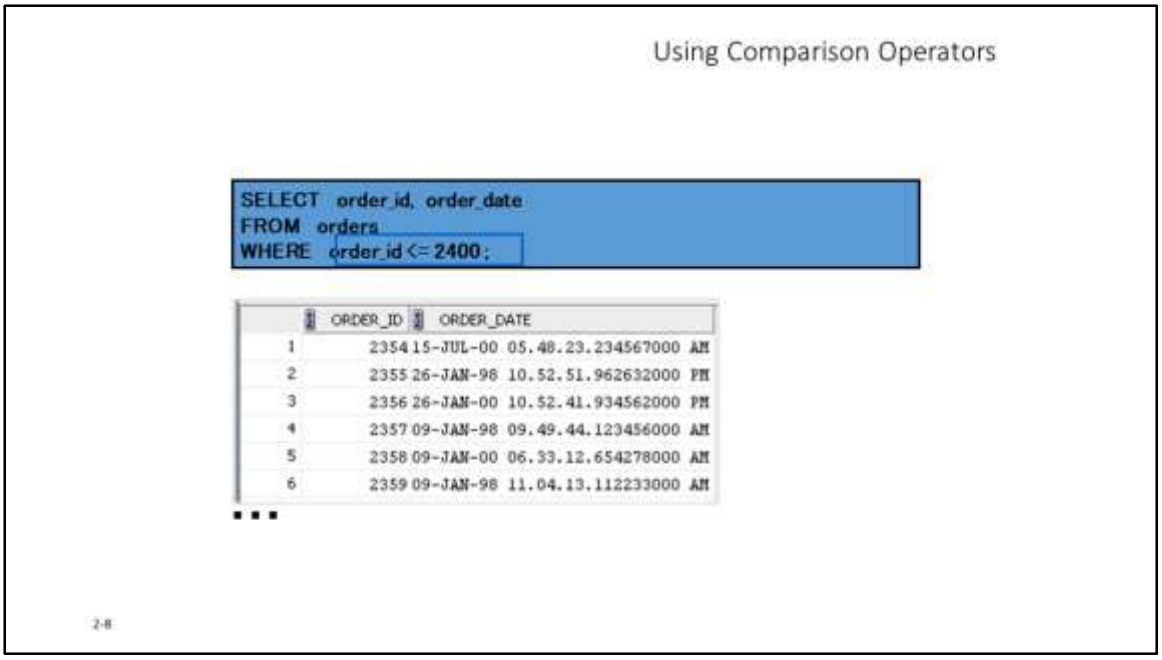
```
... WHERE expr operator value
```

Example

```
... WHERE hire_date = '01-JAN-95'
... 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

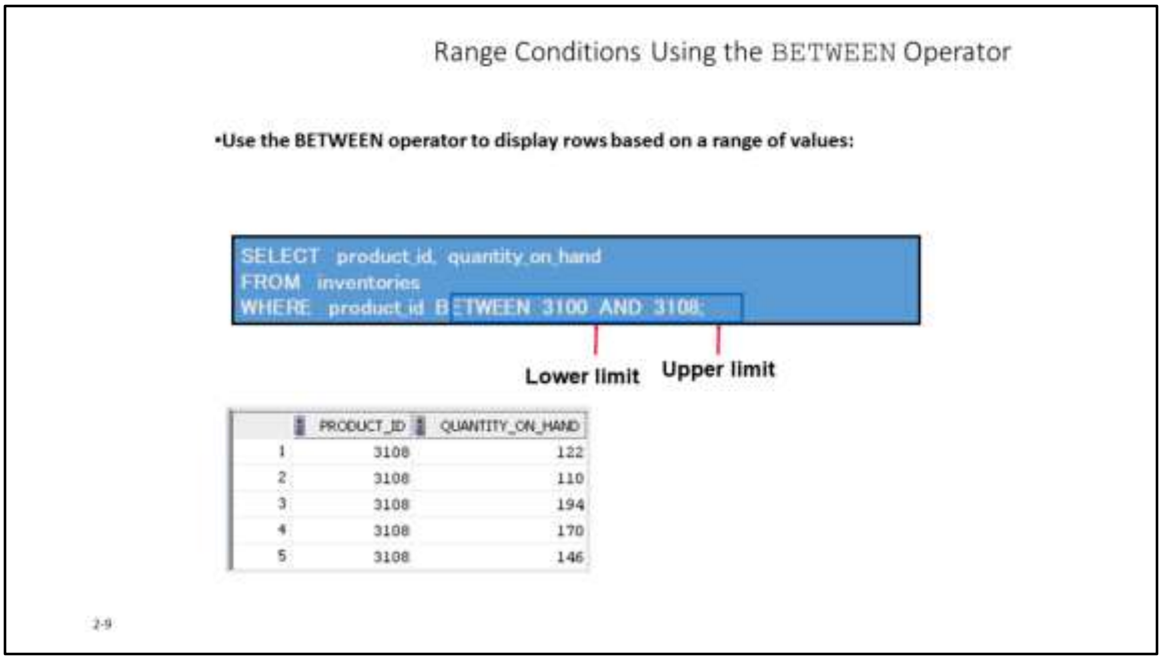
```
SELECT  order_id, order_date
FROM  orders
WHERE  order_id <= 2400 ;
```

```
      ORDER_ID  ORDER_DATE
1         2354  15-JUL-00 05.48.23.234567000 AM
2         2355  26-JAN-98 10.52.51.962632000 PM
3         2356  26-JAN-00 10.52.41.934562000 PM
4         2357  09-JAN-98 09.49.44.123456000 AM
5         2358  09-JAN-00 06.33.12.654278000 AM
6         2359  09-JAN-98 11.04.13.112233000 AM
...
```

2-8

Using Comparison Operators

In the example, the `SELECT` statement retrieves the order ID and order date from the `orders` table for any order whose order ID is less than or equal to 2400. Note that there is an explicit value supplied to the `WHERE` clause. The explicit value of `2400` is compared to the order ID value in the `order_id` column of the `orders` table.



Range Conditions Using the BETWEEN Operator

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

```
SELECT   product_id, quantity_on_hand
FROM   inventories
WHERE   product_id BETWEEN 3100 AND 3108;
```

Lower limit
Upper limit

	PRODUCT_ID	QUANTITY_ON_HAND
1	3108	122
2	3108	110
3	3108	194
4	3108	170
5	3108	146

2-9

Range Conditions Using the BETWEEN Operator

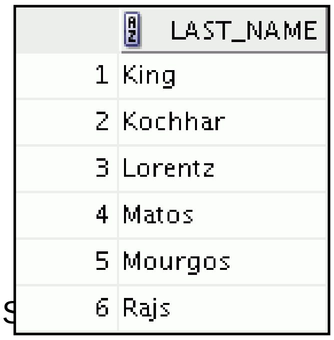
You can display rows based on a range of values using the BETWEEN operator. The range that you specify contains a lower limit and an upper limit.

The SELECT statement in the slide returns rows from the INVENTORIES table for any product whose product ID is between 3100 and 3108.

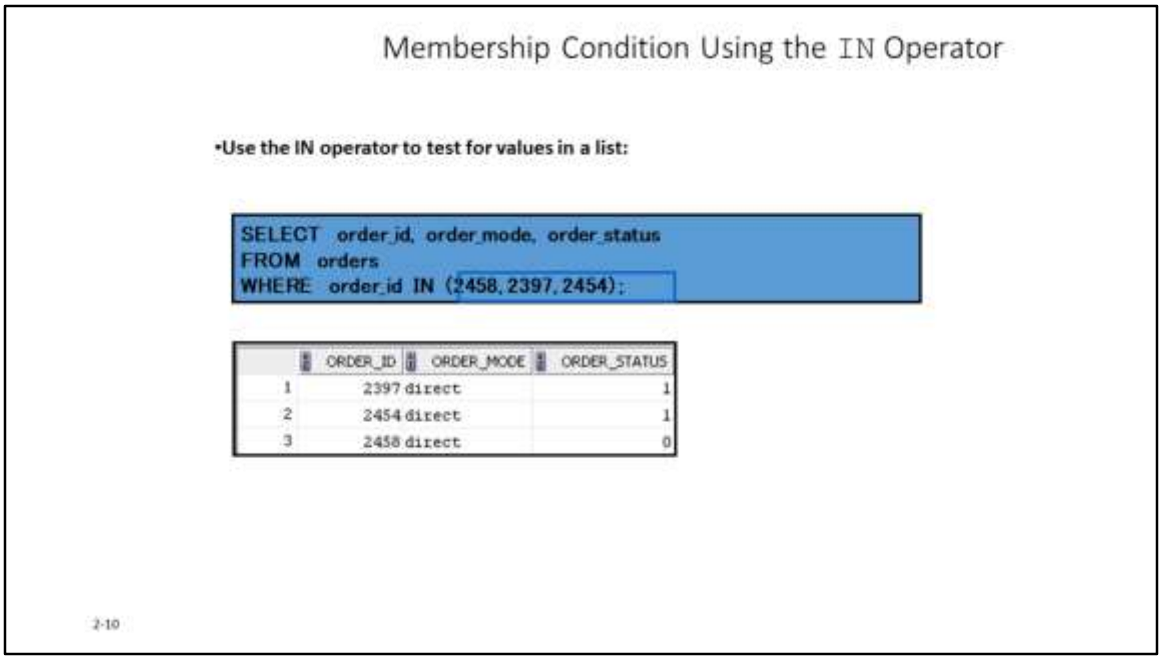
Values that are specified with the BETWEEN operator are inclusive. However, you must specify the lower limit first.

You can also use the BETWEEN operator on character values:

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



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



Membership Condition Using the IN Operator

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

```
SELECT  order_id, order_mode, order_status
FROM  orders
WHERE  order_id IN (2458, 2397, 2454);
```

	ORDER_ID	ORDER_MODE	ORDER_STATUS
1	2397	direct	1
2	2454	direct	1
3	2458	direct	0

2-10

Membership Condition Using the `IN` Operator

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

The slide example displays order ID, order mode and order status for all the orders whose order_ID is 2458, 2397 or 2454.

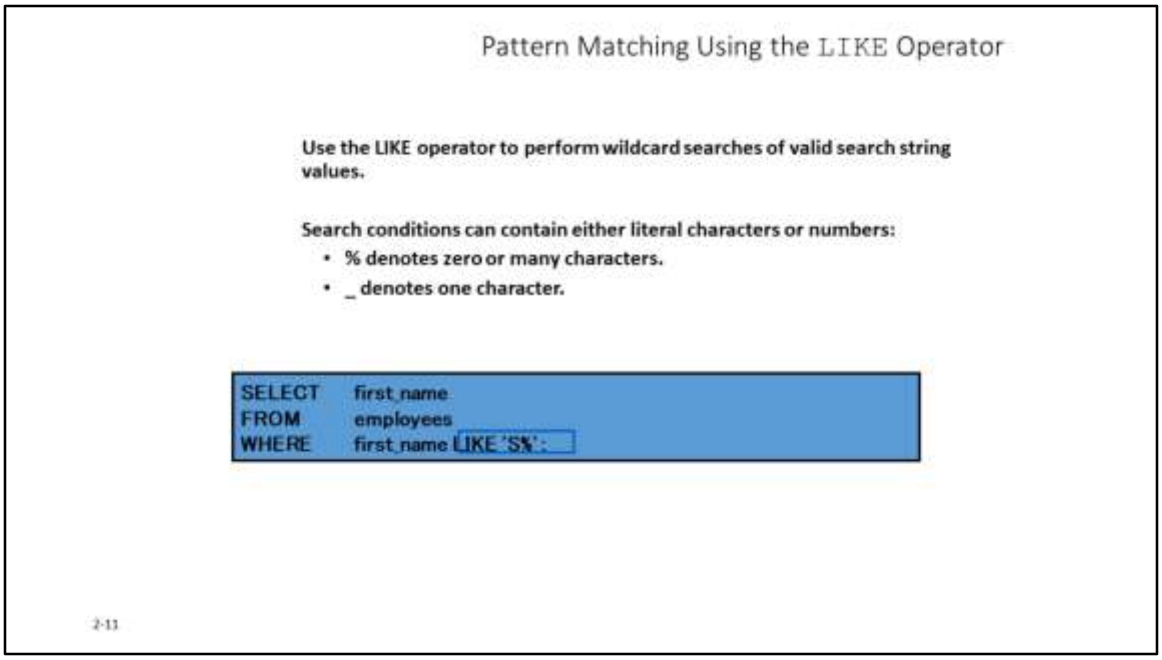
Note: The set of values can be specified in any random order—for example, (201,100,101).

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 the list, they must be enclosed with single quotation marks (`''`).

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



Pattern Matching Using the LIKE Operator

Use the LIKE operator to perform wildcard searches of valid search string values.

Search conditions can contain either literal characters or numbers:

- % denotes zero or many characters.
- _ denotes one character.

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

2-11

Pattern Matching Using the `LIKE` Operator

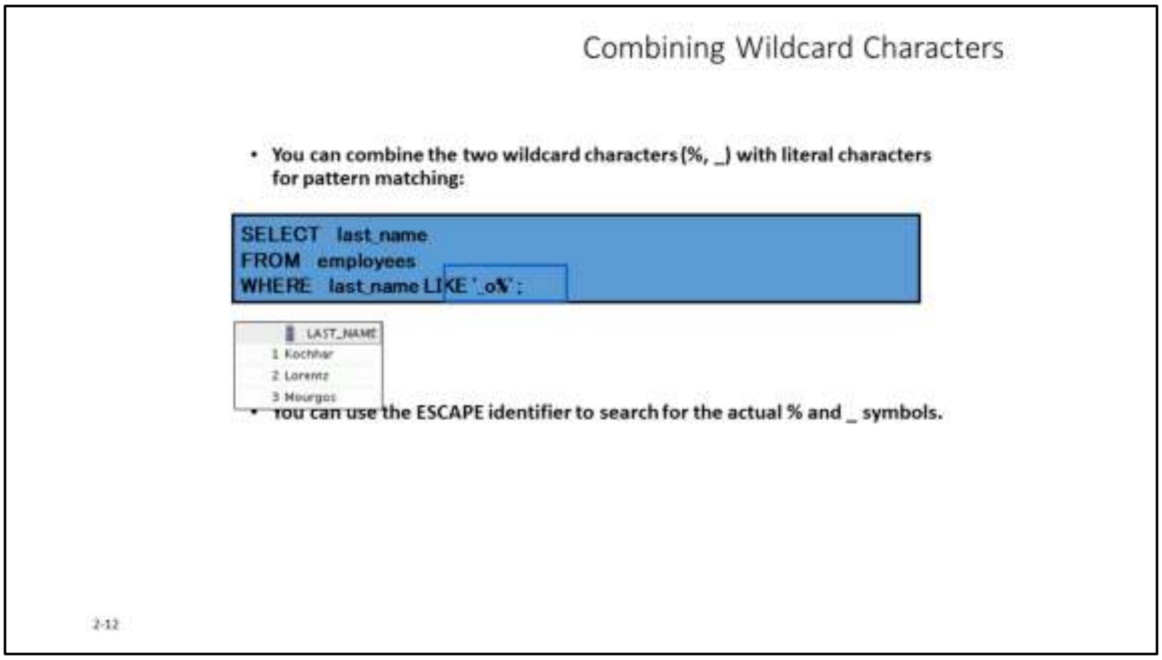
You may not always know the exact value to search for. You can select rows that match a character pattern by using the `LIKE` operator. The character pattern–matching operation is referred to as a *wildcard* search. Two symbols can be used to construct the search string.

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

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

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

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



Combining Wildcard Characters

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

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

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

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

2-12

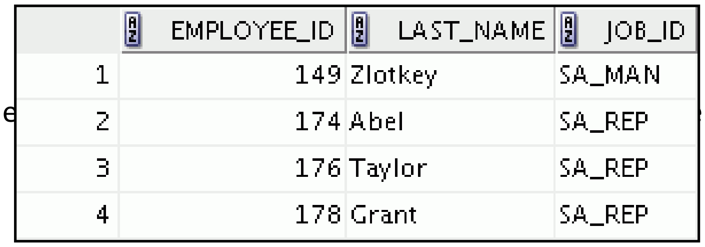
Combining Wildcard Characters

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

ESCAPE Identifier

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

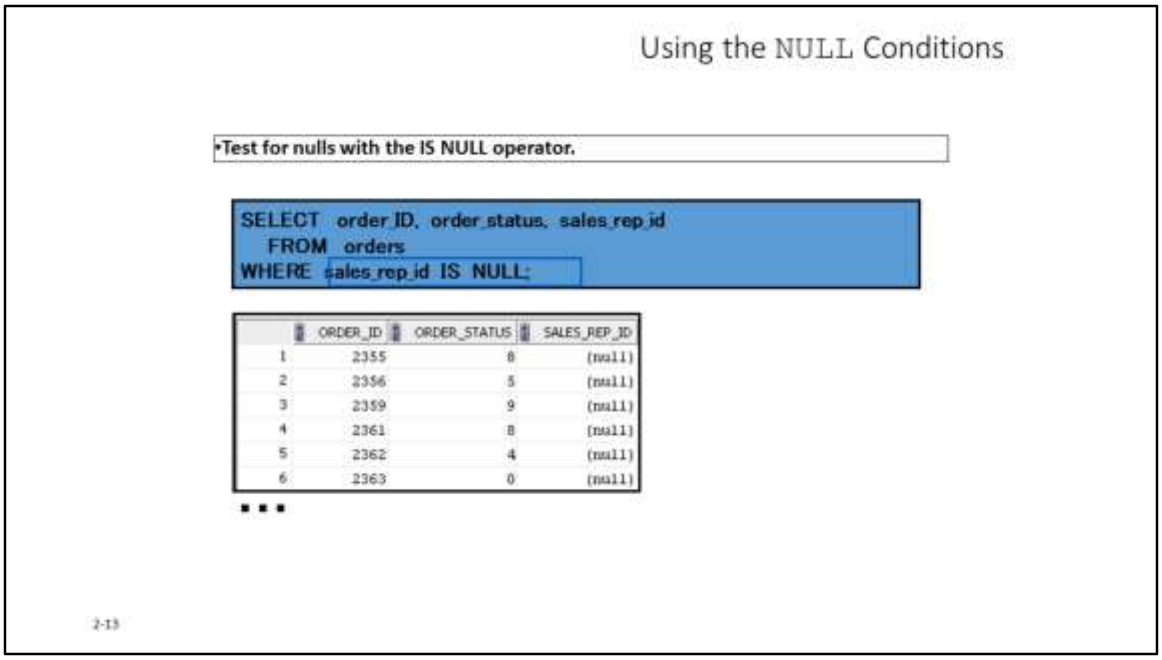
```
SELECT employee_id, last_name, job_id
FROM   employees WHERE  job_id LIKE '%SA\_%' ESCAPE '\';
```



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

The escape character. In the

SQL statement, the escape character precedes the underscore (_). This causes the Oracle server to interpret the underscore literally.



Using the NULL Conditions

•Test for nulls with the IS NULL operator.

```
SELECT  order_ID, order_status, sales_rep_id
  FROM  orders
WHERE  sales_rep_id  IS  NULL;
```

	ORDER_ID	ORDER_STATUS	SALES_REP_ID
1	2355	8	(null)
2	2356	5	(null)
3	2359	9	(null)
4	2361	8	(null)
5	2362	4	(null)
6	2363	0	(null)

▪ ▪ ▪

2-13

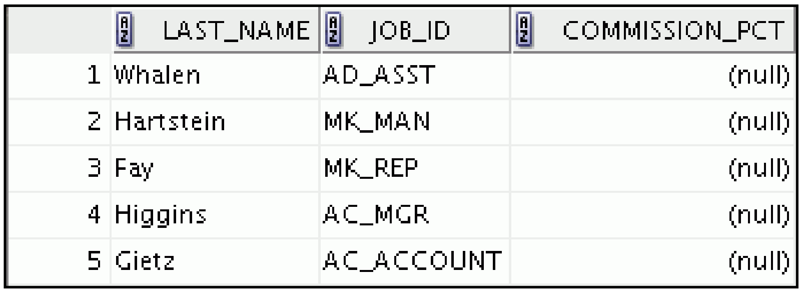
Using the NULL Conditions

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

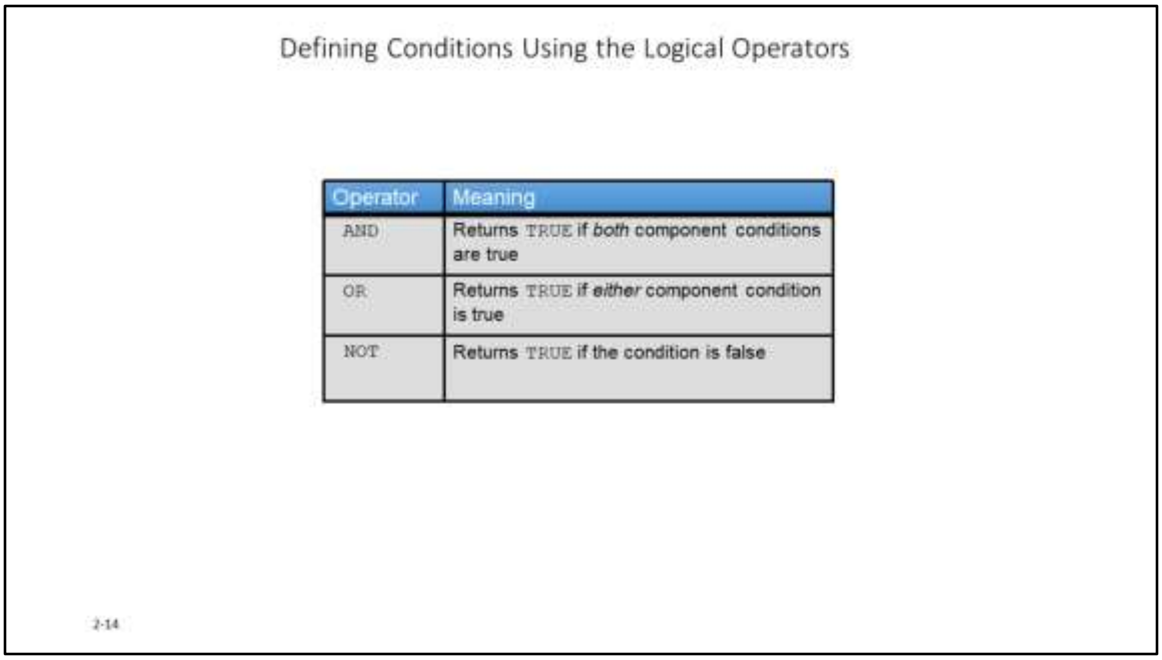
The IS NULL condition tests for nulls. A null value means that the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with =, because a null cannot be equal or unequal to any value. The example in the slide retrieves the last names and managers of all employees who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

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

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



	LAST_NAME	JOB_ID	COMMISSION_PCT
1	Whalen	AD_ASST	(null)
2	Hartstein	MK_MAN	(null)
3	Fay	MK_REP	(null)
4	Higgins	AC_MGR	(null)
5	Gietz	AC_ACCOUNT	(null)



Defining Conditions Using the 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

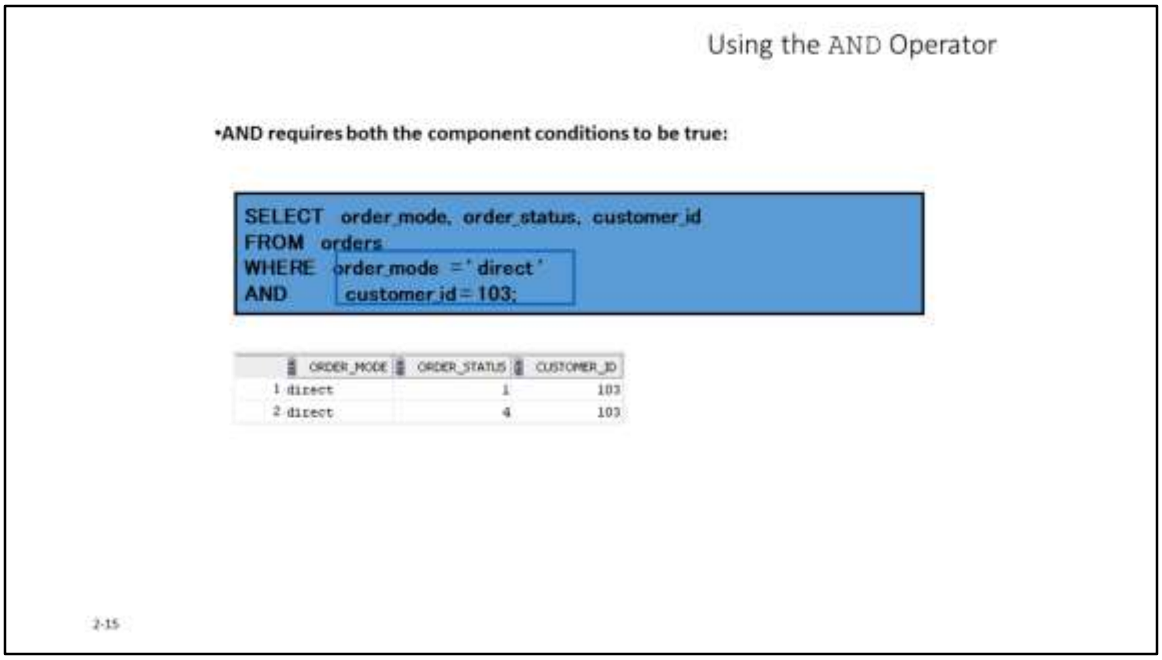
2-14

Defining Conditions Using the Logical Operators

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

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

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



Using the AND Operator

•AND requires both the component conditions to be true:

```
SELECT   order_mode,  order_status,  customer_id
FROM   orders
WHERE  order_mode  = ' direct '
AND      customer_id = 103;
```

```
    ORDER_MODE   ORDER_STATUS   CUSTOMER_ID
1   direct                  1           103
2   direct                  4           103
```

2-15

Using the `AND` Operator

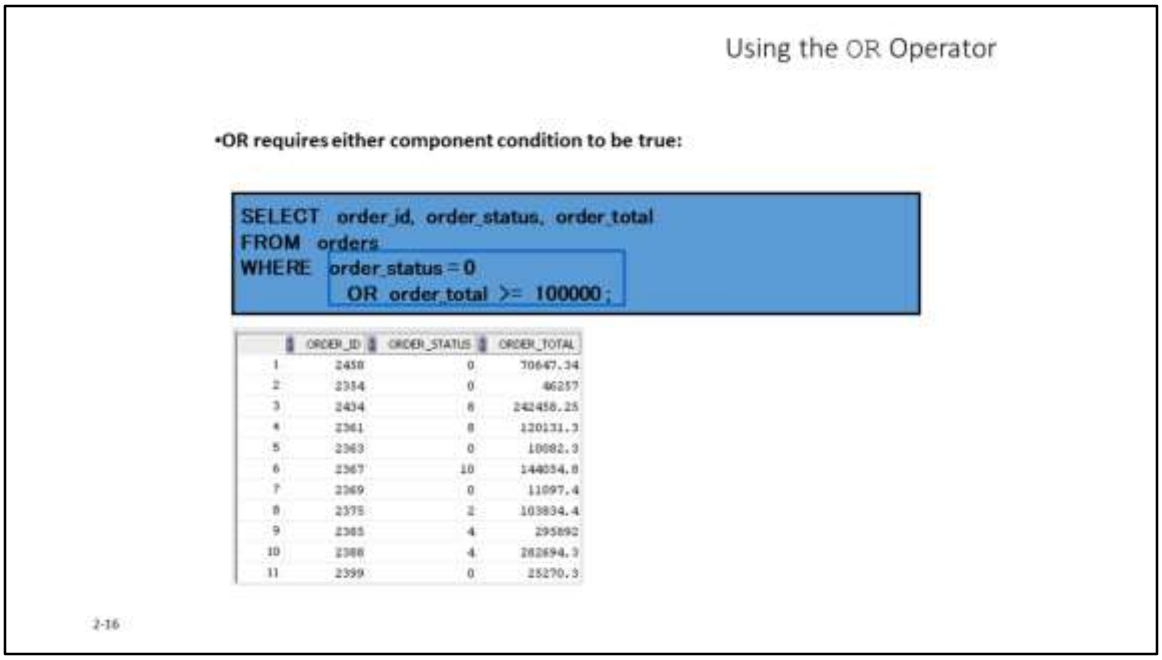
In the example, both the component conditions must be true for any record to be selected. Therefore, only those orders that have their modes as Direct and that have a Customer ID as 103 are selected.

All character searches are case-sensitive.

`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  order_id,  order_status,  order_total
FROM  orders
WHERE  order_status = 0
         OR  order_total  >=  100000 ;
```

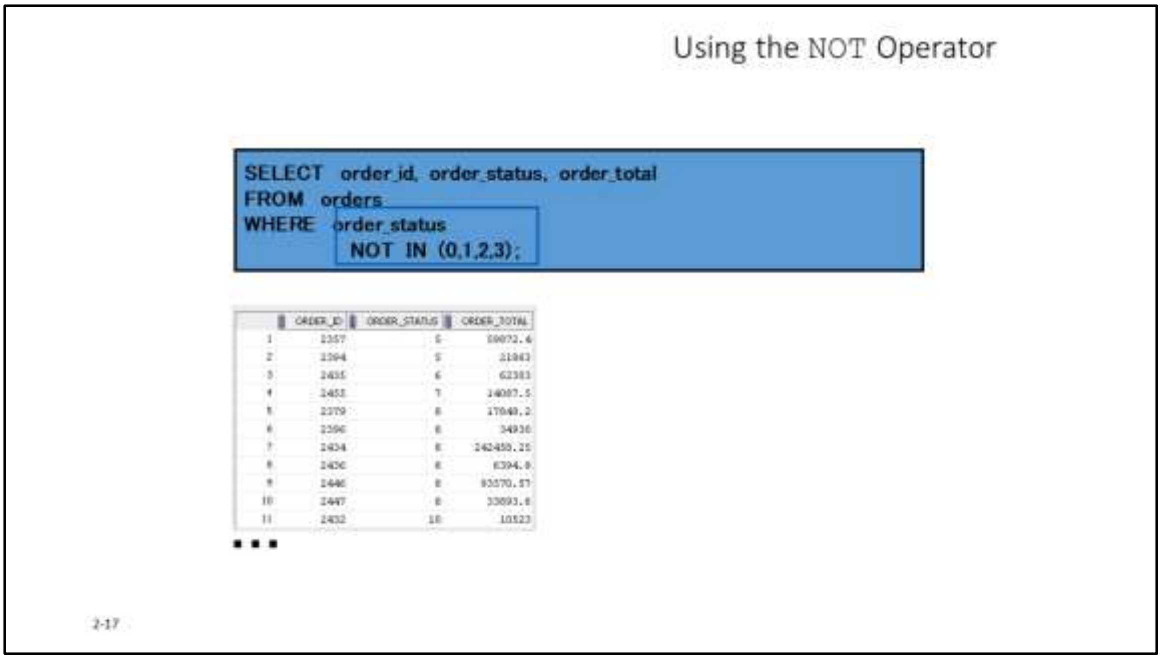
	ORDER_ID	ORDER_STATUS	ORDER_TOTAL
1	2458	0	70647.34
2	2354	0	46257
3	2434	8	242458.25
4	2361	8	120131.3
5	2363	0	10082.3
6	2367	10	144054.8
7	2369	0	11097.4
8	2375	2	103834.4
9	2385	4	295892
10	2388	4	282694.3
11	2399	0	25270.3

2-16

Using the OR Operator

In the example, either component condition can be true for any record to be selected. Therefore, any order that has an order status of 0 *or* has an order_total value of 100000 or more is selected.

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



Using the NOT Operator

```
SELECT  order_id, order_status, order_total
FROM  orders
WHERE  order_status
         NOT  IN  (0,1,2,3);
```

	ORDER_ID	ORDER_STATUS	ORDER_TOTAL
1	2357	5	59872.4
2	2394	5	21863
3	2435	6	62303
4	2453	7	14087.5
5	2379	8	17848.2
6	2396	8	34930
7	2434	8	242450.25
8	2436	8	6394.8
9	2446	8	93570.57
10	2447	8	33893.6
11	2432	10	10523

■ ■ ■

2-17

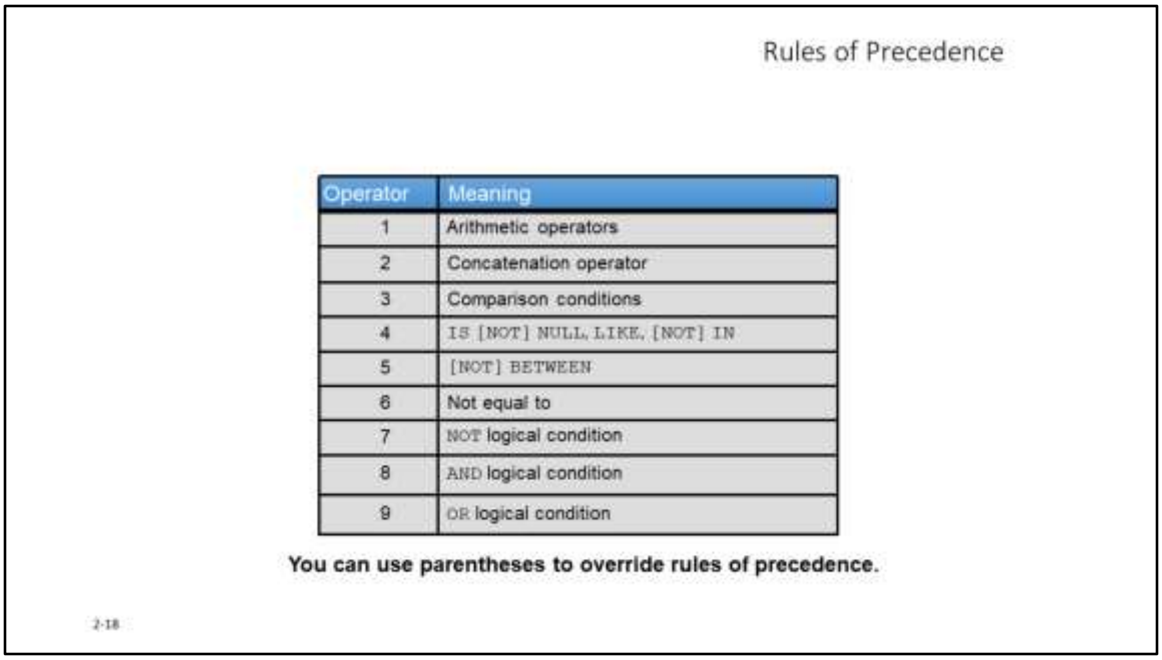
Using the `NOT` Operator

The example in the slide displays the order_id, order_status and order_total of all the orders whose order_status *is not* 0, 1, 2 or 3.

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

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	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.

2-18

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.