COMP810

Lab 2 – SQL operators, restricting and sorting data

SQL operators

By the end of this session, you will be able to use strings, operators, and *order by* clause in a SQL statement:

- Character strings
- Comparison operators
- Rules of precedence
- Sorting and ordering

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-YY.

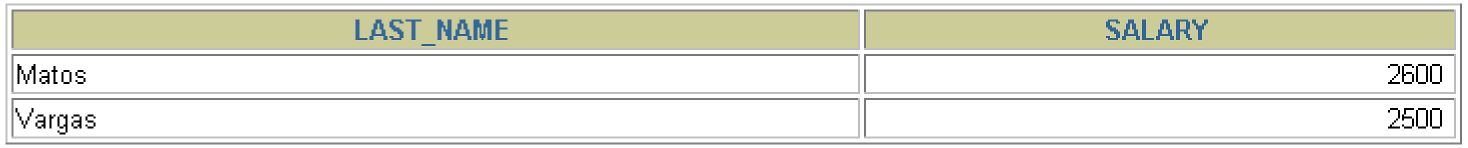
```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = $\boxed{\text{'Whalen'}}$ ;
```

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<> or !=	Not equal to

Using the Comparison Operators

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



LAST_NAME	SALARY
Matos	2600
Vargas	2500

Other Comparison Operators

Operator	Meaning
BETWEEN ...AND...	Between two values (inclusive)
IN(list)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

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

Values specified with the BETWEEN operator are inclusive!

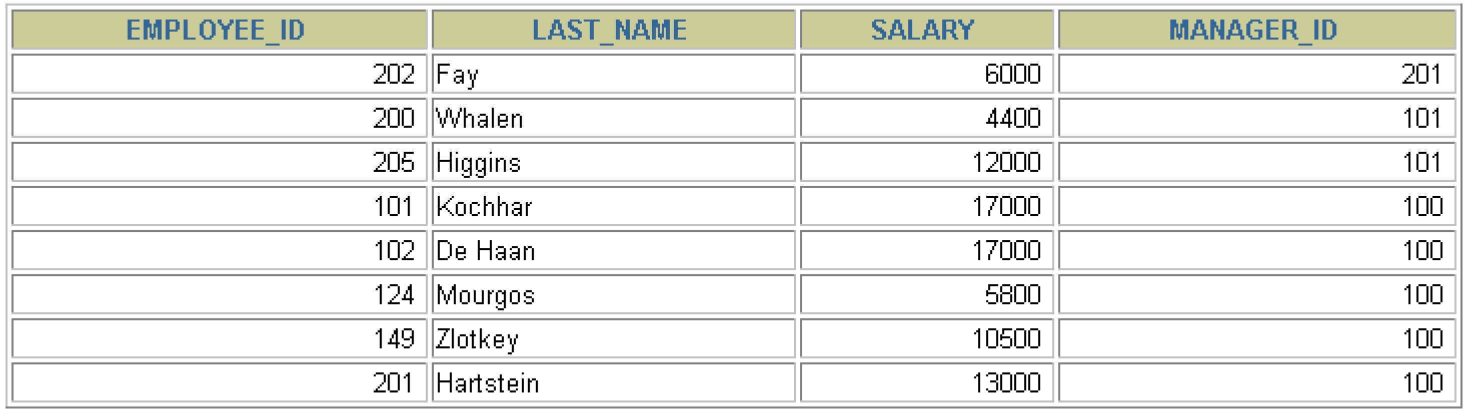
LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Use NOT BETWEEN to do the reverse operation. NOT BETWEEN is exclusive – the lower and upper limits are not included in the output. Try it!

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
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

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.

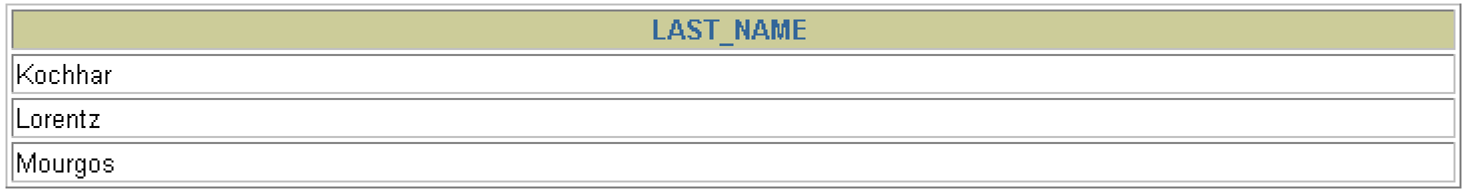
_ denotes one character.

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

Using the LIKE Operator

– You can combine pattern-matching characters:

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



LAST_NAME
Kochhar
Lorentz
Mourgos

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

Using the LIKE Operator

If you really want to search for special characters such as (actual percent sign '%' or underscore '_'characters with the LIKE operator, you can do this with the ESCAPE option of the LIKE operator, as demonstrated below:

```
SELECT empno, begindate, comments
FROM history
WHERE comments LIKE '%0\%%' ESCAPE '\';
```

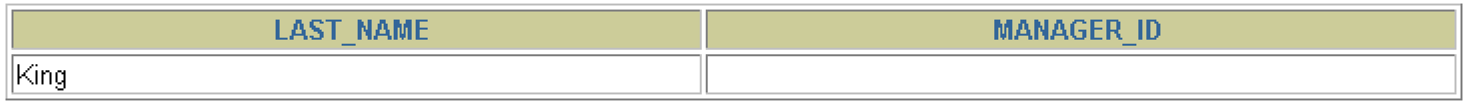
EMPNO	BEGINDATE	COMMENTS
7566	01-JUN-2002	From accounting to human resources; 0% salary change
7788	15-APR-2015	Transfer to human resources; 0% salary raise

The WHERE clause is used to filter the result set to include only those comments that include a textual reference to 0% in the COMMENTS column of the HISTORY table. The backslash (\) suppresses the special meaning of the second percent sign in the search string.

Using the IS NULL Operator

Test for nulls with the `IS NULL` operator.

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



LAST_NAME	MANAGER_ID
King	

The IS NULL operator tests for values that are null. A null value means the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with (=) because a null value cannot be equal or unequal to any value.

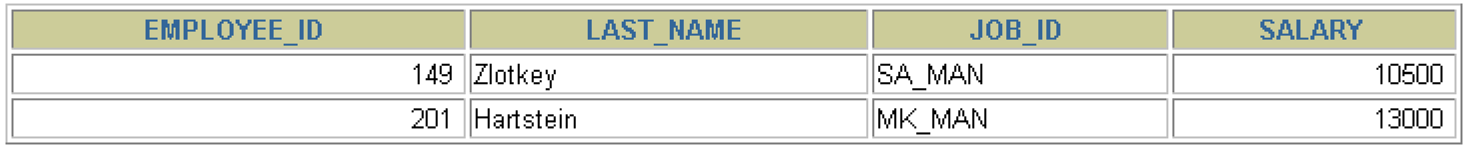
Logical Operators

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 following condition is FALSE

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%' ;
```

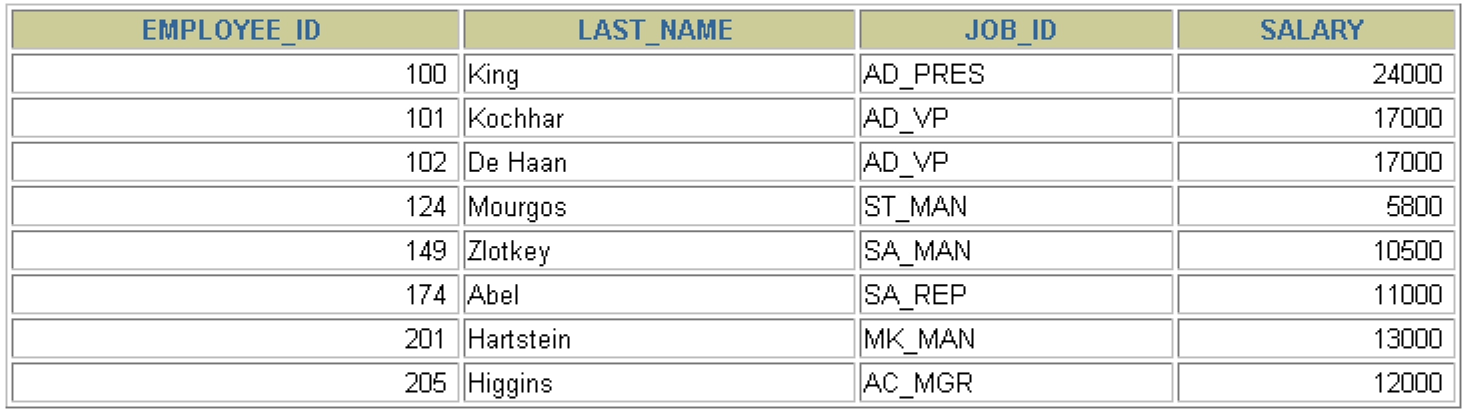


EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

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
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

```
8 rows selected.
```

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
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

Rules of Precedence

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

Note: Override rules of precedence by using parentheses.

Rules of Precedence

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

1

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

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

2

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

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 ;
```

Sorting in Descending Order

– Sorting in descending order:

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

1

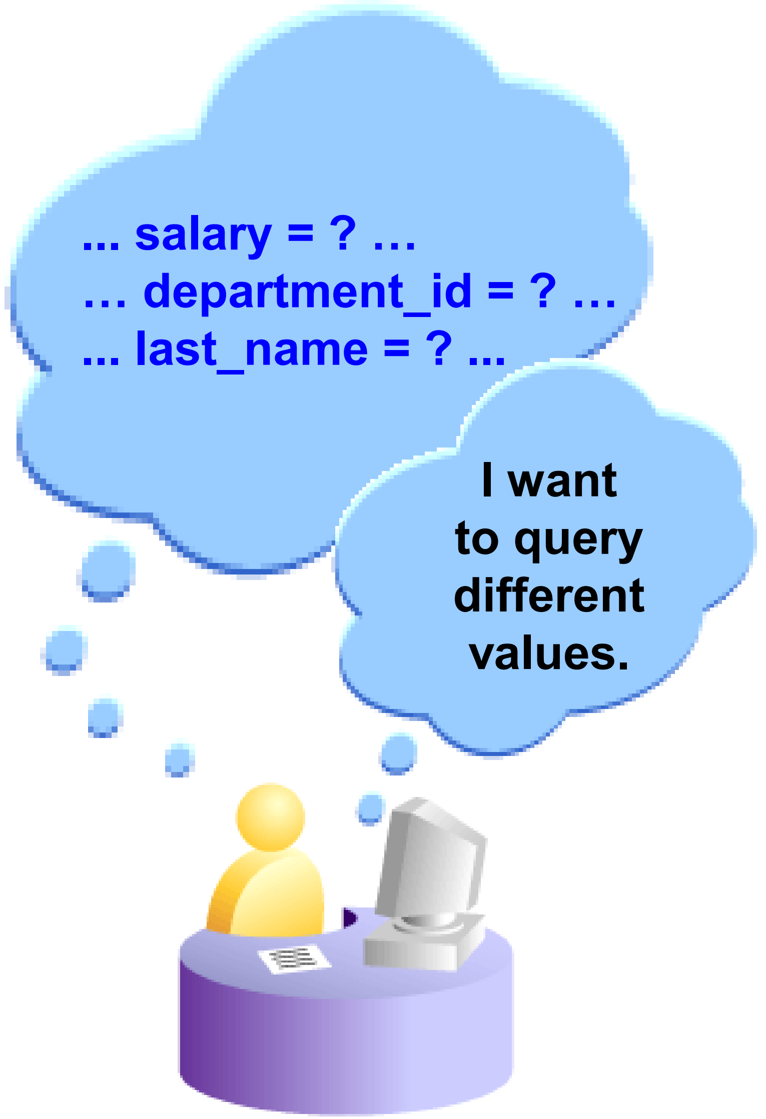
```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

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

3

Substitution Variables



users can be prompted to supply their own values to restrict the range of data returned by using substitution variables. Substitution variables (&) can be embedded 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 (`&`)
- Use substitution variables to supplement the following:
  - `WHERE` conditions
  - `ORDER BY` clauses
  - Column expressions
  - Table names
  - Entire `SELECT` statements

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 ;
```

Lab Activities

- Complete SQL lab exercise