LAB 04

BASIC QUERIES (PART 2)

❖ **Main contents**

- How to use some operators such as IN, BETWEEN, UNION, LIKE, ORDER BY;
- Derived Attribute

1. IN operator

IN operator allows selecting the value from a set of values. The syntax used is as follows:

```
SELECT list of column
FROM table name
WHERE column IN ("value 1","value 2"…)
```

The columns in the `WHERE` clause do not need to appear in the selected column list, but it must be a column in the table. If the list has more than one value, each item is separated by a comma. Besides, the `NOT` operator may be used with the `IN` operator for negative purposes.

Let's take a look at some examples:

Suppose if we want to find all offices located in USA and France, we can execute the following query:

```
SELECT officeCode, city, phone
FROM offices
WHERE country = 'USA' OR country = 'France';
```

In this case, we can use `IN` instead of the above query:

```
SELECT officeCode, city, phone
FROM offices
WHERE country IN ('USA','France');
```

Result:

officeCode	city	phone
1	San Francisco	+1 650 219 4782
2	Boston	+1 215 837 0825
3	NYC	+1 212 555 3000
4	Paris	+33 14 723 4404

To obtain all offices note located in the USA and France, we can use `NOT IN` as follows:

```
SELECT officeCode, city, phone
FROM offices
WHERE country NOT IN ('USA','France');
```

Result:

officeCode	city	phone
5	Tokyo	+81 33 224 5000
6	Sydney	+61 2 9264 2451
7	London	+44 20 7877 2041

2. BETWEEN operator

`BETWEEN` operator allows taking values within a specific range. It must be used in the `WHERE` clause. The following illustrates the syntax:

```
SELECT column_list
FROM table_name
WHERE column_1 BETWEEN lower_range AND upper_range;
```

MySQL returns all records in which the column_1 value is in the lower_rage and upper_range ranges. The equivalent query to get the same result is:

```
SELECT column_list
FROM table_name
WHERE column_1 >= lower_range AND column_1 <= upper_range;
```

Example:

Assuming we want to find all products priced between $ 90 and $ 100, we can execute the following query:

```
SELECT productCode, ProductName, buyPrice
FROM products
WHERE buyPrice BETWEEN 90 AND 100
ORDER BY buyPrice DESC;
```

Result:

productCode	ProductName	buyPrice
S10_1949	1952 Alpine Renault 1300	98.58
S24_3856	1956 Porsche 356A Coupe	98.3
S12_1108	2001 Ferrari Enzo	95.59
S12_1099	1968 Ford Mustang	95.34
S18_1984	1995 Honda Civic	93.89
S18_4027	1970 Triumph Spitfire	91.92
S10_4698	2003 Harley-Davidson Eagle Drag Bike	91.02

To find all records that are not in a range, we use `NOT BETWEEN`. For example, to find all products with purchase prices outside the 20 and 100 ranges, we could use the following query:

```
SELECT productCode, ProductName, buyPrice
FROM products
WHERE buyPrice NOT BETWEEN 20 AND 100;
```

Result:

productCode	ProductName	buyPrice
S10_4962	1962 LanciaA Delta 16V	103.42
S18_2238	1998 Chrysler Plymouth Prowler	101.51
S24_2840	1958 Chevy Corvette Limited Edition	15.91
S24_2972	1982 Lamborghini Diablo	16.24

The above query is equivalent to the following query:

```
SELECT productCode, ProductName, buyPrice
FROM products
WHERE buyPrice < 20 OR buyPrice > 100
ORDER BY buyPrice DESC;
```

Result:

productCode	ProductName	buyPrice
S10_4962	1962 LanciaA Delta 16V	103.42
S18_2238	1998 Chrysler Plymouth Prowler	101.51
S24_2972	1982 Lamborghini Diablo	16.24
S24_2840	1958 Chevy Corvette Limited Edition	15.91

3. LIKE operator

`LIKE` operator allows searching for information based on character comparisons ('*like*'). `LIKE` is often used with the `SELECT` statement in the `WHERE` clause. MySQL provides two wildcards for constructing patterns, which is % and _.

- The percentage (%) wildcard matches any string of zero or more characters.
- The underscore (_) wildcard matches any single character.

Example: Suppose we want to find employees whose first name starts with the letter 'a', it can be done as follows:

```
SELECT employeeNumber, lastName, firstName
 FROM employees
 WHERE firstName LIKE 'a%';
```

Result:

employeeNumber	lastName	firstName
1143	Bow	Anthony
1611	Fixter	Andy

MySQL scans the entire employees table to find all employees whose first names start with the letter 'a' and are followed by any number of characters.

Example: Find all employees whose last name end with the string 'on', we can execute the following query:

```
SELECT employeeNumber, lastName, firstName
 FROM employees
 WHERE lastName LIKE '%on';
```

Result:

employeeNumber	lastName	firstName
1056	Patterson	Mary
1088	Patterson	William
1166	Thompson	Leslie
1216	Patterson	Steve

If we only know that the search string is embedded inside in the middle of a string, you can set the beginning and the end of the pattern with the percentage wildcard (%) to find all possibilities.

Example: Get all the employees whose last names contain “on”, we can execute the following query:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName LIKE '%on%';
```

Result:

employeeNumber	lastName	firstName
1056	Patterson	Mary
1088	Patterson	William
1102	Bondur	Gerard
1166	Thompson	Leslie
1216	Patterson	Steve
1337	Bondur	Loui
1504	Jones	Barry

The MySQL allows you to combine the NOT operator with the LIKE operator to find a string that does not match a specific pattern.
Example: Get employees whose last names don’t start with the character 'B', execute as follows:

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastName NOT LIKE 'B%';
```

Result:

employeeNumber	lastName	firstName
1002	Murphy	Diane
1056	Patterson	Mary
1076	Firrelli	Jeff
1088	Patterson	William
1165	Jennings	Leslie
1166	Thompson	Leslie
1188	Firrelli	Julie
1216	Patterson	Steve
1286	Tseng	Foon Yue
1323	Vanauf	George
1370	Hernandez	Gerard
1401	Castillo	Pamela
1504	Jones	Barry
1611	Fixter	Andy
1612	Marsh	Peter

Note: When searching for partial strings in MySQL with LIKE you will match case-insensitive (not case-sensitive) by default, so 'b%' and 'B%' are the same.

Sometimes the pattern, which you want to match, contains wildcard character e.g., 10%, _20, etc. In this case, you can use the ESCAPE clause to specify the escape character so that MySQL will interpret the wildcard character as a literal character. If you don't specify the escape character explicitly, the backslash character \ is the default escape character.

Example: Find products whose code contains the string '_20', then execute the query as follows:

```
SELECT productCode, productName
FROM products
WHERE productCode LIKE '%\_20%';
```

Result:

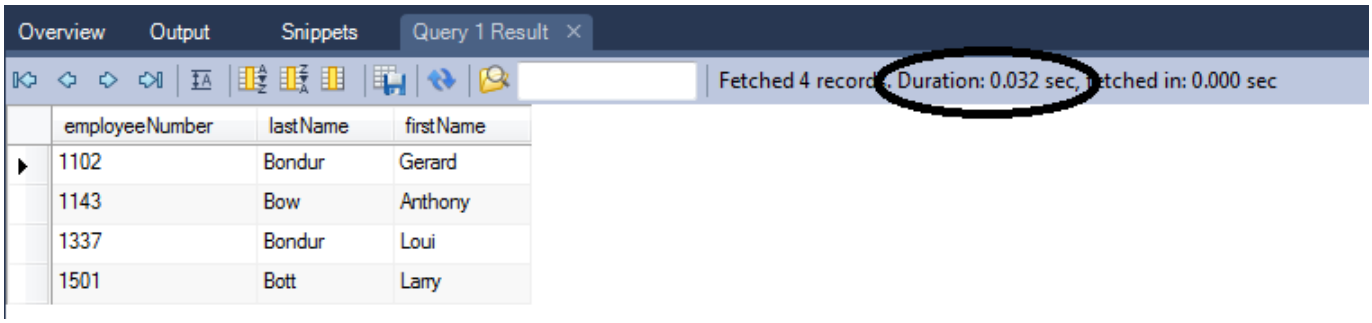
productCode	productName
S10_2016	1996 Moto Guzzi 1100i
S24_2000	1960 BSA Gold Star DBD34
S24_2011	18th century schooner
S24_2022	1938 Cadillac V-16 Presidential Limousine
S700_2047	HMS Bounty

`LIKE` provides a convenient way to find records whose columns contain strings that match the search pattern. However, because the `LIKE` implementation is to scan the entire table to find all matching records so it does not allow the database engine to use the index to search quickly. When the data in the table is large enough, `LIKE` performance will be degraded. In some cases, this problem can be avoided by using other techniques to achieve equivalent results. Or use MySQL's FULLTEXT indexing method (will discuss this technique in the course of MySQL Database Management System).

Example: if looking for all the staff have first names beginning with a string specified using the `LEFT()` function like the following query:

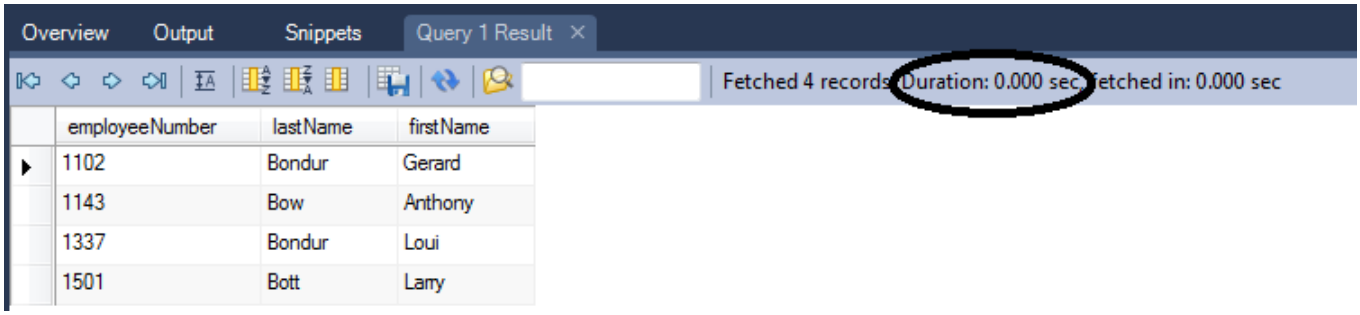
```
SET @str = 'b';
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE LEFT(lastname, length(@str)) = @str;
```

Result:



The result of this query is equivalent to the following query, but the performance of the following clause is much better because we can use the index on the *lastname* column.

```
SELECT employeeNumber, lastName, firstName
FROM employees
WHERE lastname LIKE 'b%';
```



4. Derived Attribute

SQL provides the ability to create Derived Attribute in the result table by using operators and functions based on available attributes. Column names of Derived Attribute depend on systems, but alias can be assigned as column names.

The following example creates an derived attribute column named ***lineTotal***, which is the result of the multiplication between `priceEach` and `quantityOrdered` properties.

```
SELECT orderNumber, (priceEach*quantityOrdered) as lineTotal
FROM orderdetails;
```

orderNumber	lineTotal
10100	4080
10100	2754.5
10100	1660.12
10100	1729.21
10101	2701.5
10101	4343.56
10101	1463.8500000000001
10101	2040.1000000000001
10102	3726.45
10102	1768.3300000000002
10103	5571.8
10103	5026.14
10103	3284.28
10103	3307.5
10103	1283.48
10103	2489.13

5. Sort the results with ORDER BY

`ORDER BY` clause allows you to sort the results in one or more columns of the query results in ascending or descending order. To sort results in ascending order, use `ASC`; descending is `DESC`. By default, `ORDER BY` will sort the results in ascending order.

Example: To sort the employee list by *name* and *job title*, the following query can be executed:

```
SELECT FirstName, LastName, jobtitle
FROM Employees
ORDER BY firstname ASC, jobtitle DESC;
```

FirstName	LastName	jobtitle
Andy	Fixter	Sales Rep
Anthony	Bow	Sales Manager (NA)
Barry	Jones	Sales Rep
Diane	Murphy	President
Foon Yue	Tseng	Sales Rep
George	Vanauf	Sales Rep
Gerard	Hernandez	Sales Rep
Gerard	Bondur	Sale Manager (EMEA)
Jeff	Firrelli	VP Marketing
Julie	Firrelli	Sales Rep
Larry	Bott	Sales Rep
Leslie	Jennings	Sales Rep
Leslie	Thompson	Sales Rep
Loui	Bondur	Sales Rep
Mami	Nishi	Sales Rep

Or, we can provide information about product names in ascending order of inventory by querying as follows:

```
SELECT productName
FROM Products
ORDER BY quantityInStock;
```

In the statement above, the `ASC` keyword is not used, because by default, the results will be sorted in ascending order. The result of the statement in the following figure.

productName
1960 BSA Gold Star DBD34
1968 Ford Mustang
1928 Ford Phaeton Deluxe
1997 BMW F650 ST
Pont Yacht
1911 Ford Town Car
1928 Mercedes-Benz SSK
F/A 18 Hornet 1/72
2002 Yamaha YZR M1
The Mayflower
1996 Peterbilt 379 Stake Bed with Outrigger
P-51-D Mustang
1970 Chevy Chevelle SS 454
Diamond T620 Semi-Skirted Tanker
1969 Ford Falcon

If it is not specified if sorting is executed in ascending or descending order, MySQL will default to the data sorting in ascending order.

6. Combine the results with UNION operator

UNION operator allows combining two or more result sets from multiple tables together. The syntax of using MySQL UNION is as follows:

```
SELECT statement
UNION [DISTINCT | ALL]
SELECT statement
UNION [DISTINCT | ALL]
…
```

To use UNION, there are a few guidelines to follow:

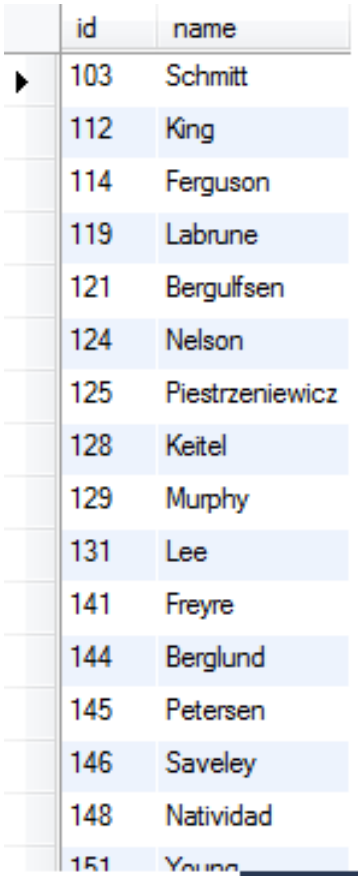
- The number of columns in each SELECT statement must be the same.
- The data types of the column in the column list of the SELECT statement must be the same or at least convertible to each other.

By default, UNION MySQL removes all duplicate rows from the results even if the DISTINCT keyword is not used after the UNION keyword.

If using UNION ALL, the duplicate rows remain in the final result set. This should only be used in cases where either you want to keep duplicates of rows, or make sure that there are no duplicate rows in the result set.

Example: To combine information about customers and employees into a result set, use the following query:

```
SELECT customerNumber id, contactLastname name
FROM customers
UNION
SELECT employeeNumber id, firstname name
FROM employees;
```



id	name
103	Schmitt
112	King
114	Ferguson
119	Labrune
121	Bergulfsen
124	Nelson
125	Piestrzeniewicz
128	Keitel
129	Murphy
131	Lee
141	Freyre
144	Berglund
145	Petersen
146	Saveley
148	Natividad
151	Young

When using `ORDER BY` to sort results with `UNION`, it must be placed at the last position in the `SELECT` statement.

Example: Assuming a combination of employee and customer information, then want to sort the results by ***name*** and ***ID*** in ascending order

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber, firstname
FROM employees)
ORDER BY contactLastname, customerNumber;
```

customerNumber	contactLastname
249	Accorti
481	Altagar,G M
307	Andersen
1611	Andy
1143	Anthony
465	Anton
187	Ashworth
204	Barajas
1504	Barry
462	Benitez
240	Bennett
144	Berglund
121	Bergulfsen
172	Bertrand
321	Brown
324	Brown

If the column names are not the same in the two `SELECT` statements of the `UNION` operation, which names will be displayed at the output if we do not use aliases for each column in the `SELECT` clause. The answer is that MySQL will use the column names of the first `SELECT` statement as the column names in the output.

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber, firstname
FROM employees)
ORDER BY contactLastname, customerNumber;
```

The result of the mathematical operation between the two result sets from the customers and employees data table.

customerNumber	contactLastname
249	Accorti
481	Altagar,G M
307	Andersen
1611	Andy
1143	Anthony
465	Anton
187	Ashworth
204	Barajas
1504	Barry
462	Benitez
240	Bennett

MySQL also provides another option to sort the result set based on the column position in the ORDER BY clause as follows:

```
(SELECT customerNumber, contactLastname
FROM customers)
UNION
(SELECT employeeNumber, firstname
FROM employees)
ORDER BY 2, 1;
```

customerNumber	contactLastname
249	Accorti
481	Altagar,G M
307	Andersen
1611	Andy
1143	Anthony
465	Anton
187	Ashworth
204	Barajas
1504	Barry
462	Benitez
240	Bennett
144	Berglund
121	Bergulfsen
172	Bertrand

❖ Practical exercises

1. Use the IN operator to find customers living in Nantes and Lyon. Write another way using the OR operator.

2. Use BETWEEN to find orders shipped between January 10, 2003 and March 10, 2003. Write another way using the AND operator.

3. Use LIKE to get product lines that its name contain the word ‘CARS’.

4. Get 10 products with the largest quantity in stock.