جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

PEARSON BTEC International Standards Verifier

ICT Program

Dr. Amany AbdElSamea Saeed

Lec. 3 Data Types, Clauses, and Operators

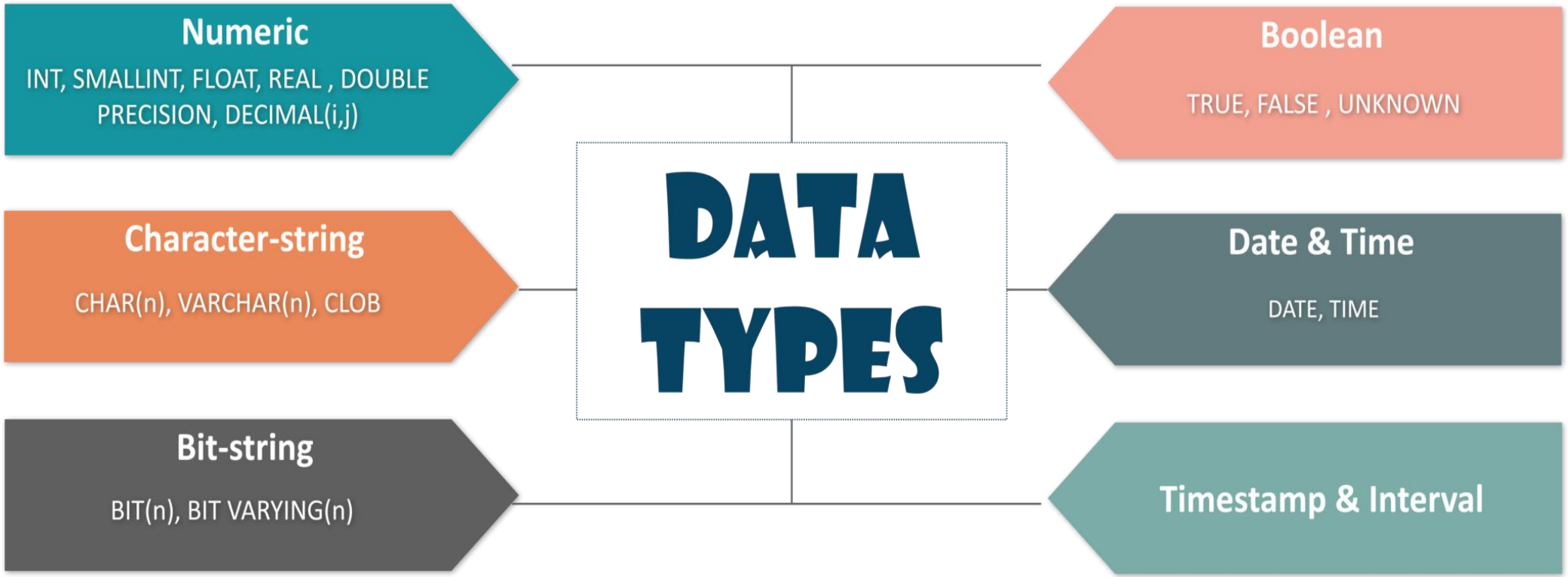
March 2024

Outline

- Data Types
- Clauses
- Operators
- Arithmetic Operators
- Comparison Operators
- Logical Operators

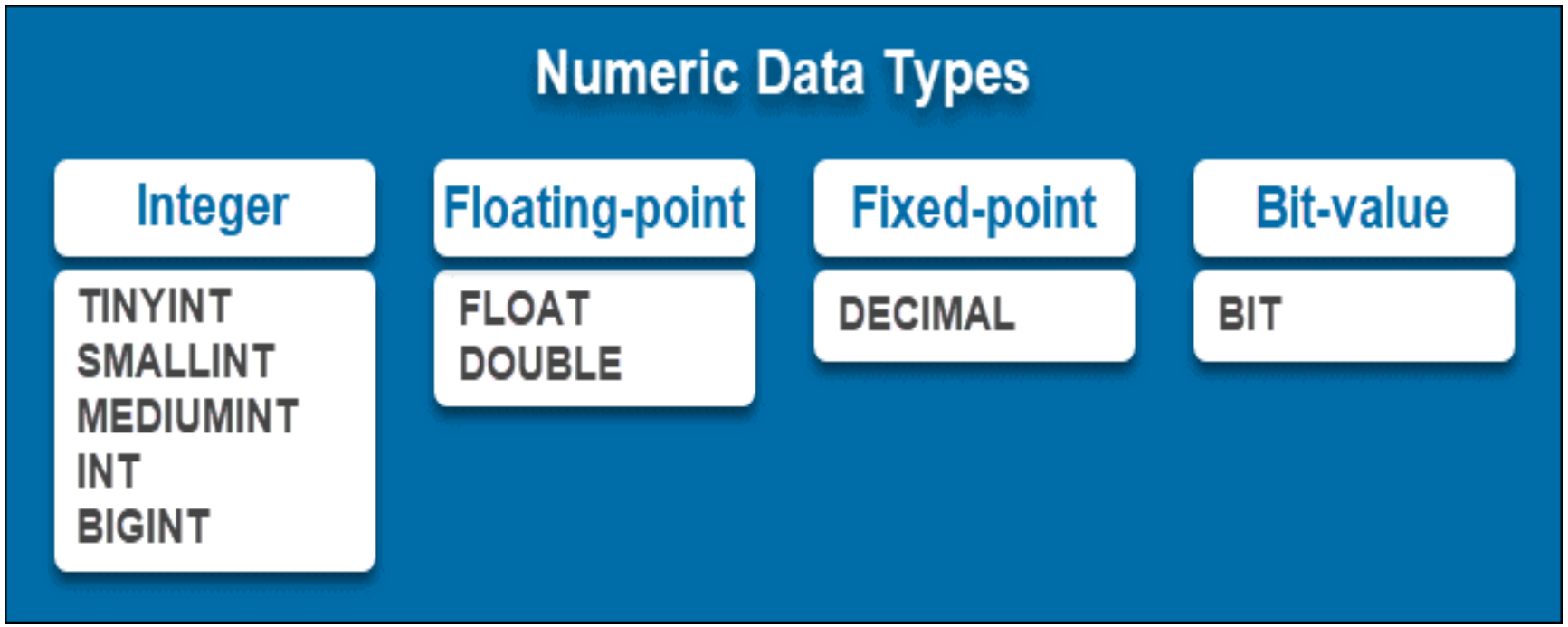
Data Types

- A name and a data type define each column in a database table. The specified data type tells MySQL what kind of values it will store, how much space they require, and what type of operations it can perform with this type of data.



Numeric Data Types

- **Numeric –** This data type includes integers of various sizes, floating-point(real) of various precisions and formatted numbers, decimal and bit value data type.



Integer Types

- Integer data types are used for whole numbers (integers). They include both positive and negative values. However, they do not handle fractional numbers.
- Integer types are signed or unsigned. They are further subdivided based on their size, differing by their length and range.

	Bytes	Range (unsigned)	Range (signed)
TINYINT	1	from 0 to 255	from -128 to 127
SMALLINT	2	from 0 to 65535	from -32768 to 32767
MEDIUMINT	3	from 0 to 16777215	from -8388608 to 8388607
INT	4	from 0 to 4294967295	from -2147483648 to 2147483647
BIGINT	8	from 0 to 18446744073709551615	from -9223372036854775808 to 9223372036854775807

Floating-Point Types

- Floating-point numeric data types are rational numbers used for representing approximate values. Use floating-point data types for high-precision calculations.
- Floating-point types include:
  - **FLOAT** represents single-precision values that use 4 bytes and include up to 6 or 7 significant digits.
  - **DOUBLE** represents double-precision values that use 8 bytes and include up to 15 or 16 significant digits

	Bytes	Range (unsigned)	Range (signed)
FLOAT	4	from 1.175494351E-38 to 3.402823466E+38	from -3.402823466E+38 to -1.175494351E-38
DOUBLE	8	from 0 and 2.225073858507201 4E-308 to 1.797693134862315 7E+308	from -1.797693134862315 7E+ 308 to -2.225073858507201 4E-308

Fixed Point Types

- To store exact numeric values, use the fixed-point data type
  - **DECIMAL**

- The basic syntax is **DECIMAL(P,D)**
  - where **P** stands for **precision** (the number of significant digits) and
  - **D** stands for **scale** (the number of digits after the decimal point

- The maximum number of digits for precision is **65**, while the maximum value for scale is 30

- If you do not define the precision and scale, the column uses default values. By default, the values for P,D are **10,0**

Bit-Value Types

- The **BIT** data type stores binary values. When creating a column that will store such values, you define the number of bit values ranging **from 1 to 64**

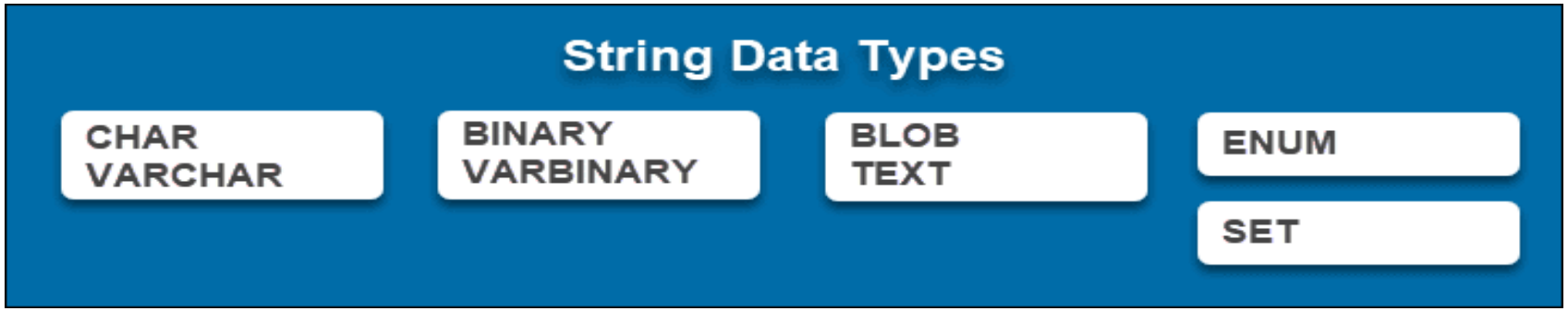
- The syntax for this MySQL data type is **BIT(N)**. If you do not specify **N**, the default value is1

Bit-Value Types

- The **BIT** data type stores binary values. When creating a column that will store such values, you define the number of bit values ranging **from 1 to 64**

- The syntax for this MySQL data type is **BIT(N)**. If you do not specify **N**, the default value is1

String Data Types



- **CHAR** and **VARCHAR** are data types used to store non-binary strings. The main difference between the two is how they store data.

- **CHAR** stores fixed-length strings up to **255** characters. When creating a **CHAR** column, you specify the length using the **CHAR(N)** syntax. **N** is the number of characters you want to take up. If you do not define the length, it uses the default value**1**
- **VARCHAR** stores variable-length strings. They have a maximum limit, but the length is not fixed and varies depending on the data.

BLOB and TEXT

- **BLOB** handles **B**inary **L**arge **O**bjects (that is, large sets of binary data such as images, audio or PDF files)

- There are 4 kinds of BLOB data types to use, depending on the size your data requires:
  - **TINYBLOB** (0 – 255; 255 bytes)
  - **BLOB** (0 – 65,535; 16 KB)
  - **MEDIUMBLOB** (0 – 16,777,215; 16 MB)
  - **LONGBLOB** (0 – 4,294,967,295; 4 GB)

- **TEXT** data types are for storing longer strings of text. According to the amount of data required, there is:
  - **TINYTEXT** (0 – 255; 255 bytes)
  - **TEXT** (0 – 65,535; 16 KB)
  - **MEDIUMTEXT** (0 – 16,777,215; 16 MB)
  - **LONGTEXT** (0 – 4,294,967,295; 4 GB)

ENUM Data Types

- It is short for enumeration, which means that each column may have one of the specified possible values. It uses numeric indexes (1, 2, 3…) to represent string values.
- An ENUM is a string object whose value is decided from a set of permitted literals(Values) that are explicitly defined at the time of column creation.
- ENUM syntax for columns:

```
CREATE TABLE table_name (
  col...
  col ENUM ('value_1','value_2','value_3', ....),
  col...
);
```

DATETIME and TIMESTAMP

- To store date and time values, use either **DATETIME** or **TIMESTAMP** .
- Both data types store information in the **YYYY-MM-DD HH:MM:SS** format. It includes the **y**ear, **m**onth, **d**ay, **h**our, **m**inutes, and **s**econds.
- The main difference between the two is their range:
  - **DATETIME** values range from **1000-01-01 00:00:00** to .**9999-12-31 23:59:59**
  - **TIMESTAMP** values range from **1970-01-01 00:00:01** to **2038-01-19 03:14:07**

Date, Time, and Year

DATE

- **DATE** is used for storing date values in the format **YYYY-MM-DD** (year, month, date)
- The data type supports the range **1000-01-01 to .9999-12-31**

TIME

- **TIME** is used to store time values as **HH-MM-SS** (hours, minutes, seconds) or **HHH-MM-SS** .
- The data type supports the range **1000-01-01 to 9999-12-31**

YEAR

YEAR stores year values in the format **YYYY** .
It supports values within the range .**1901-2155**

MYSQL Clauses

MySQL WHERE Clause

- MySQL WHERE Clause is used with SELECT, INSERT, UPDATE and DELETE clause to filter the results. It specifies a specific position where you have to do the operation.
- Syntax:

```
WHERE conditions;
```

- conditions: It specifies the conditions that must be fulfilled for records to be selected.

```
mysql> SELECT*FROM officers;
+------------+--------------+-----------+
| officer_id | officer_name | address   |
+------------+--------------+-----------+
|          1 | Ajeet        | Mau       |
|          2 | Deepika      | Lucknow   |
|          3 | Vimal        | Faizabad  |
|          4 | Rahul        | Lucknow   |
+------------+--------------+-----------+
4 rows in set (0.00 sec)

mysql>
```

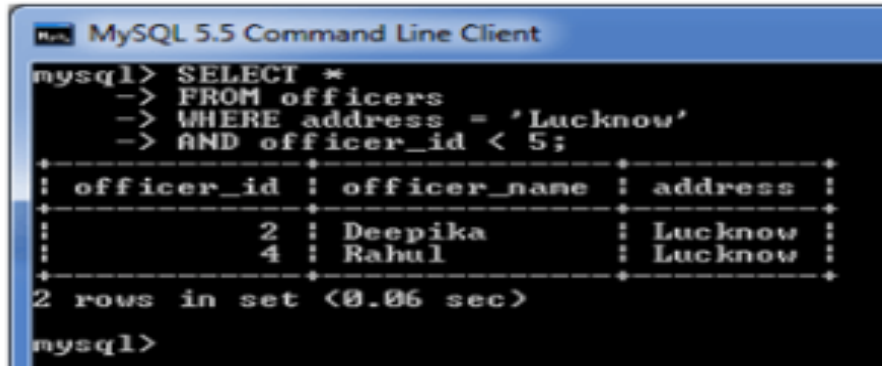
```
mysql> SELECT *
    -> FROM officers
    -> WHERE address = 'Mau';
+------------+--------------+-----------+
| officer_id | officer_name | address   |
+------------+--------------+-----------+
|          1 | Ajeet        | Mau       |
+------------+--------------+-----------+
1 row in set (0.00 sec)

mysql> _
```

MySQL WHERE Clause with And and OR

Execute the following query:

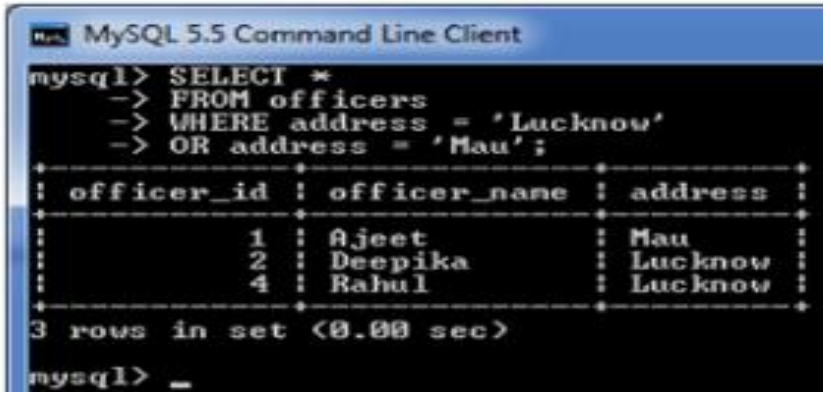
```
SELECT *
FROM officers
WHERE address = 'Lucknow'
AND officer_id < 5;
```

Output:




```
SELECT *
FROM officers
WHERE address = 'Lucknow'
OR address = 'Mau';
```

Output:



MySQL Distinct Clause

- MySQL DISTINCT clause is used to remove duplicate records from the table and fetch only the unique records. The DISTINCT clause is only used with the SELECT statement.
- **Syntax:**

```
SELECT DISTINCT expressions
FROM tables
[WHERE conditions];
```

```
mysql> SELECT * FROM officers;
+------------+--------------+-----------+
| officer_id | officer_name | address   |
+------------+--------------+-----------+
|          1 | Ajeet        | Mau       |
|          2 | Deepika      | Lucknow   |
|          3 | Vimal        | Faizabad  |
|          4 | Rahul        | Lucknow   |
+------------+--------------+-----------+
4 rows in set (0.00 sec)

mysql> SELECT DISTINCT address
    -> FROM officers;
+-----------+
| address   |
+-----------+
| Mau       |
| Lucknow   |
| Faizabad  |
+-----------+
3 rows in set (0.00 sec)

mysql>
```

- MySQL DISTINCT Clause with single expression
- If you use a single expression then the MySQL DISTINCT clause will return a single field with unique records (no duplicate record).
- MySQL DISTINCT Clause with multiple expressions

```
mysql> SELECT DISTINCT officer_name, address
    -> FROM officers;
+--------------+-----------+
| officer_name | address   |
+--------------+-----------+
| Ajeet        | Mau       |
| Deepika      | Lucknow   |
| Vimal        | Faizabad  |
| Rahul        | Lucknow   |
+--------------+-----------+
4 rows in set (0.00 sec)

mysql>
```

MySQL FROM Clause

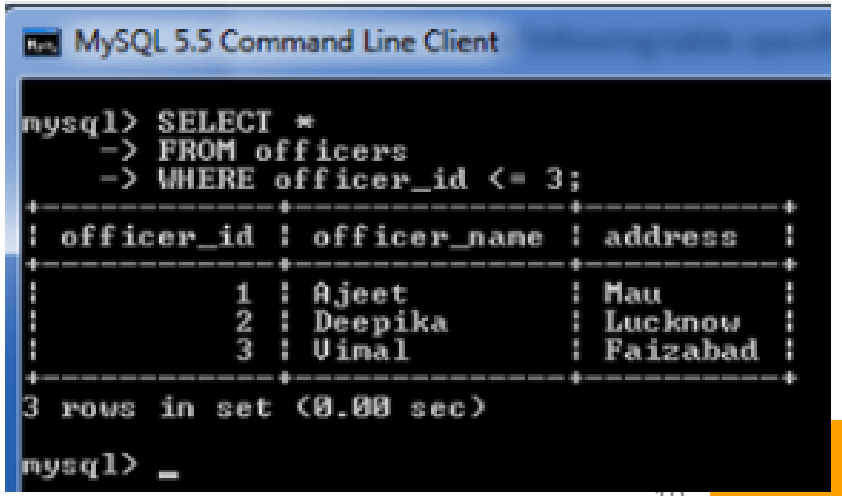
- The MySQL FROM Clause is used to select some records from a table. It can also be used to retrieve records from multiple tables using JOIN condition.

- **Syntax:**

```
FROM table1
[ { INNER JOIN | LEFT [OUTER] JOIN| RIGHT [OUTER] JOIN } table2
ON table1.column1 = table2.column1 ]
```

- MySQL FROM Clause: Retrieve data from one table

```
SELECT *
FROM officers
WHERE officer_id <= 3;
```



MySQL ORDER BY Clause

- The MYSQL ORDER BY Clause is used to sort the records in ascending or descending order. If you use MySQL ORDER BY clause without specifying the ASC and DESC modifier then by default you will get the result in ascending order.
-

```
SELECT expressions
FROM tables
[WHERE conditions]
ORDER BY expression [ ASC | DESC ];
```

  - **ASC:** It is optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).
  - **DESC:** It is also optional. It sorts the result set in descending order by expression.

MySQL ORDER BY Clause CONT.,

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name ASC;
```

Output:

```
MySQL 5.5 Command Line Client
mysql> SELECT *
    -> FROM officers
    -> WHERE address = 'Lucknow'
    -> ORDER BY officer_name ASC;
+------------+--------------+---------+
| officer_id | officer_name | address |
+------------+--------------+---------+
|          2 | Deepika      | Lucknow |
|          4 | Rahul        | Lucknow |
+------------+--------------+---------+
2 rows in set (0.00 sec)
```

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name DESC;
```

```
MySQL 5.5 Command Line Client
mysql> SELECT *
    -> FROM officers
    -> WHERE address = 'Lucknow'
    -> ORDER BY officer_name DESC;
+------------+--------------+---------+
| officer_id | officer_name | address |
+------------+--------------+---------+
|          4 | Rahul        | Lucknow |
|          2 | Deepika      | Lucknow |
+------------+--------------+---------+
2 rows in set (0.00 sec)
```

MySQL GROUP BY Clause

- The MYSQL GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.
- You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.
- **Syntax:**

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n;
```

MySQL GROUP BY Clause

- (i) MySQL GROUP BY Clause with COUNT function

```
SELECT address, COUNT(*)
FROM   officers
GROUP BY address;
```

```
mysql> SELECT address, COUNT(*)
    -> FROM   officers
    -> GROUP BY address;
+-----------+-----------+
| address   | COUNT(*)  |
+-----------+-----------+
| Faizabad  |         1 |
| Lucknow   |         2 |
| Mau       |         1 |
+-----------+-----------+
3 rows in set (0.04 sec)
```

- (ii) MySQL GROUP BY Clause with SUM function

```
SELECT emp_name, SUM(working_hours) AS "Total working hours"
FROM employees
GROUP BY emp_name;
```

```
mysql> SELECT emp_name, SUM(working_hours) AS "Total working hours"
    -> FROM employees
    -> GROUP BY emp_name;
+----------+---------------------+
| emp_name | Total working hours |
+----------+---------------------+
| Ajeet    |                  36 |
| Ayan     |                  20 |
| Milan    |                  27 |
| Ruchi    |                  12 |
+----------+---------------------+
4 rows in set (0.00 sec)
```

- Simply, it can also be used with MIN, MAX and AVG functions

MySQL HAVING Clause

- MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.
- **Syntax:**

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n
HAVING condition;
```

```
SELECT emp_name, SUM(working_hours) AS "Total working hours"
FROM employees
GROUP BY emp_name
HAVING SUM(working_hours) > 5;
```

```
mysql> SELECT emp_name, SUM(working_hours) AS "Total working hours"
    -> FROM employees
    -> GROUP BY emp_name
    -> HAVING SUM(working_hours) > 5;
+----------+----------------------+
| emp_name | Total working hours  |
+----------+----------------------+
| Ajeet    |                   36 |
| Ayan     |                   20 |
| Milan    |                   27 |
| Ruchi    |                   12 |
+----------+----------------------+
4 rows in set (0.00 sec)
```

- HAVING Clause with SUM function
- Simply, it can also be used with COUNT, MIN, MAX and AVG functions.

MYSQL Operators

Operators

- An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Operators used to negate conditions

Arithmetic operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

Examples

Example

```
SQL> select 10+ 20;
```

Output

```
+---------+
| 10+ 20 |
+---------+
|      30 |
+---------+
1 row in set (0.00 sec)
```

Example

```
SQL> select 12 %  5;
```

Output

```
+----------+
| 12 %  5 |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

Comparison Operators

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=

Examples

Example

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY > 5000;
```

Output

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
3 rows in set (0.00 sec)
```

Example

```
SQL>  SELECT * FROM CUSTOMERS WHERE SALARY != 2000;
```

Output

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
5 rows in set (0.00 sec)
```

Logical Operators

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list as per the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

AND (&&)Operators

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Examples

Consider the CUSTOMERS table having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result −

ID	NAME	SALARY
6	Komal	4500.00
7	Muffy	10000.00

OR (| |) Operator

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Examples

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

NOT (!) Operator

The NOT operator displays a record if the condition(s) is NOT TRUE.

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

```
mysql> select * from student where not (studid=1);
                                (or)
mysql> select * from student where ! (studid=1);
+--------+-------+-------+-----------------+---------+
| studid | name  | marks | address         | phone   |
+--------+-------+-------+-----------------+---------+
|      2 | david |   100 | welling street  |  547896 |
|      4 | jack  |    82 | welling street  | 2436821 |
|      5 | anne  |   100 | downing street  | 2634821 |
|      6 | steve |    75 | downing street  | 2874698 |
|      7 | anne  |    80 | edinburgh       | 2569843 |
|      8 | mille |    98 | victoria street | 1236547 |
+--------+-------+-------+-----------------+---------+
6 rows in set (0.00 sec)
```

You can also combine the AND, OR and NOT operators.

LIKE Operator

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

MySQL Wildcard Characters

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200.
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position.
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions.
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length.
WHERE SALARY LIKE '%2'	Finds any values that end with 2.
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3.
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3.

Examples

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%';
```

This would produce the following result −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00

IN Operator

The `IN` operator allows you to specify multiple values in a `WHERE` clause.

The `IN` operator is a shorthand for multiple `OR` conditions.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Example

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

```
| officeCode | city          | phone           | country |
+------------+---------------+-----------------+---------+
| 1          | San Francisco | +1 650 219 4782 | USA     |
| 2          | Boston        | +1 215 837 0825 | USA     |
| 3          | NYC           | +1 212 555 3000 | USA     |
| 4          | Paris         | +33 14 723 4404 | France  |
+------------+---------------+-----------------+---------+
4 rows in set (0.01 sec)
```

BETWEEN Operator

The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.

The `BETWEEN` operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

NOT BETWEEN

To negate the `BETWEEN` operator, you use the `NOT` operator:

```
value NOT BETWEEN low AND high
```

Example

```
SELECT
    productCode,
    productName,
    buyPrice
FROM
    products
WHERE
    buyPrice BETWEEN 90 AND 100;
```

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

IS NULL and IS NOT NULL Operator

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the `IS NULL` and `IS NOT NULL` operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Example

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
    FROM CUSTOMERS
    WHERE SALARY IS NOT NULL;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

Example

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NULL;
```

This would produce the following result –

ID	NAME	AGE	ADDRESS	SALARY
6	Komal	22	MP	
7	Muffy	24	Indore	


جامعة
القاهرة الجديدة
التكنولوجية
NCT
NEW CAIRO
TECHNOLOGICAL
UNIVERSITY

Thank you