MySQL

Tushar B. Kute,
http://tusharkute.com

MySQL Data Types

- A Data Type specifies a particular type of data, like integer, floating points, Boolean, etc.
- It also identifies the possible values for that type, the operations that can be performed on that type, and the way the values of that type are stored.
- In MySQL, each database table has many columns and contains specific data types for each column.

MySQL Data Types

- We can determine the data type in MySQL with the following characteristics:
  - The type of values (fixed or variable) it represents.
  - The storage space it takes is based on whether the values are a fixed-length or variable length.
  - Its values can be indexed or not.
  - How MySQL performs a comparison of values of a particular data type.

MySQL Data Types: Numeric

- MySQL has all essential SQL numeric data types. These data types can include the exact numeric data types (For example, integer, decimal, numeric, etc.), as well as the approximate numeric data types (For example, float, real, and double precision).
- It also supports BIT datatype to store bit values. In MySQL, numeric data types are categories into two types, either signed or unsigned except for bit data type.

MySQL Data Types: Numeric

- TINYINT
  - It is a very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. We can specify a width of up to 4 digits. It takes 1 byte for storage.
- SMALLINT
  - It is a small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. We can specify a width of up to 5 digits. It requires 2 bytes for storage.

MySQL Data Types: Numeric

- MEDIUMINT
  - It is a medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. We can specify a width of up to 9 digits. It requires 3 bytes for storage.
- INT
  - It is a normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. We can specify a width of up to 11 digits. It requires 4 bytes for storage.

MySQL Data Types: Numeric

- BIGINT
  - It is a large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. We can specify a width of up to 20 digits. It requires 8 bytes for storage.
- FLOAT(m,d)
  - It is a floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d). This is not required and will default to 10,2, where 2 is the number of decimals, and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a float type. It requires 2 bytes for storage.

MySQL Data Types: Numeric

- DOUBLE(m,d)
  - It is a double-precision floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d).
  - This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a double. Real is a synonym for double. It requires 8 bytes for storage.

MySQL Data Types: Numeric

- DECIMAL(m,d)
  - An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (m) and the number of decimals (d) is required. Numeric is a synonym for decimal.
- BIT(m)
  - It is used for storing bit values into the table column. Here, M determines the number of bit per value that has a range of 1 to 64.
- BOOL
  - It is used only for the true and false condition. It considered numeric value 1 as true and 0 as false.
- BOOLEAN
  - It is Similar to the BOOL.

MySQL Data Types: Date and Time

- This data type is used to represent temporal values such as date, time, datetime, timestamp, and year.
- Each temporal type contains values, including zero. When we insert the invalid value, MySQL cannot represent it, and then zero value is used.

MySQL Data Types: Date and Time

- YEAR[(2|4)]
  - Year value as 2 digits or 4 digits. The default is 4 digits. It takes 1 byte for storage.
- DATE
  - Values range from '1000-01-01' to '9999-12-31'. Displayed as 'yyyy-mm-dd'. It takes 3 bytes for storage.
- TIME
  - Values range from '-838:59:59' to '838:59:59'. Displayed as 'HH:MM:SS'. It takes 3 bytes plus fractional seconds for storage.

MySQL Data Types: Date and Time

- DATETIME
  - Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Displayed as 'yyyy-mm-dd hh:mm:ss'. It takes 5 bytes plus fractional seconds for storage.
- TIMESTAMP(m)
  - Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC. Displayed as 'YYYY-MM-DD HH:MM:SS'. It takes 4 bytes plus fractional seconds for storage.

MySQL Data Types: String

- The string data type is used to hold plain text and binary data, for example, files, images, etc.
- MySQL can perform searching and comparison of string value based on the pattern matching such as LIKE operator, Regular Expressions, etc.

MySQL Data Types: String

- CHAR(size)
  - It can have a maximum size of 255 characters. Here size is the number of characters to store. Fixed-length strings. Space padded on the right to equal size characters.
- VARCHAR(size)
  - It can have a maximum size of 255 characters. Here size is the number of characters to store. Variable-length string.

MySQL Data Types: String

- TINYTEXT(size)
  - It can have a maximum size of 255 characters. Here size is the number of characters to store.
- TEXT(size)
  - Maximum size of 65,535 characters.   Here size is the number of characters to store.

MySQL Data Types: String

- MEDIUMTEXT(size)
  - It can have a maximum size of 16,777,215 characters. Here size is the number of characters to store.
- LONGTEXT(size)
  - It can have a maximum size of 4GB or 4,294,967,295 characters. Here size is the number of characters to store.
- BINARY(size)
  - It can have a maximum size of 255 characters. Here size is the number of binary characters to store. Fixed-length strings. Space padded on the right to equal size characters.

MySQL Data Types: String

- VARBINARY(size)
  - It can have a maximum size of 255 characters. Here size is the number of characters to store. Variable-length string. (introduced in MySQL 4.1.2)
- ENUM
  - It takes 1 or 2 bytes that depend on the number of enumeration values. An ENUM can have a maximum of 65,535 values.
  - 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.

MySQL Data Types: Binary

- BLOB in MySQL is a data type that can hold a variable amount of data.
- They are categories into four different types based on the maximum length of values can hold.

MySQL Data Types: Binary

- TINYBLOB
  - It can hold a maximum size of 255 bytes.
- BLOB(size)
  - It can hold the maximum size of 65,535 bytes.
- MEDIUMBLOB
  - It can hold the maximum size of 16,777,215 bytes.
- LONGBLOB
  - It can hold the maximum size of 4gb or 4,294,967,295 bytes.

Creating a table

- MySQL allows us to create a table into the database by using the CREATE TABLE command. Following is a generic syntax for creating a MySQL table in the database.

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column_definition1,
    column_definition2,
    ........,
    table_constraints
);
```

Creating a table

- database_name
  - It is the name of a new table. It should be unique in the MySQL database that we have selected. The IF NOT EXIST clause avoids an error when we create a table into the selected database that already exists.
- column_definition
  - It specifies the name of the column along with data types for each column. The columns in table definition are separated by the comma operator. The syntax of column definition is as follows:
  - column_name1 data_type(size) [NULL | NOT NULL]
- table_constraints
  - It specifies the table constraints such as PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, CHECK, etc.

Creating a table

```
CREATE TABLE employee_table(
    id int NOT NULL AUTO_INCREMENT,
    name varchar(45) NOT NULL,
    occupation varchar(35) NOT NULL,
    age int NOT NULL,
    PRIMARY KEY (id)
);
```

Options

- NOT NULL is a field attribute, and it is used because we don't want this field to be NULL. If we try to create a record with a NULL value, then MySQL will raise an error.
- The field attribute AUTO_INCREMENT specifies MySQL to go ahead and add the next available number to the id field. PRIMARY KEY is used to define a column's uniqueness. We can use multiple columns separated by a comma to define a primary key.

Unique Key

- A unique key in MySQL is a single field or combination of fields that ensure all values going to store into the column will be unique. It means a column cannot stores duplicate values.
- For example, the email addresses and roll numbers of students in the "student_info" table or contact number of employees in the "Employee" table should be unique.
- MySQL allows us to use more than one column with UNIQUE constraint in a table. It can accept a null value, but MySQL allowed only one null value per column.
- It ensures the integrity of the column or group of columns to store different values into a table.

Unique Key: Examples

```
CREATE TABLE table_name(
    col1 datatype,
    col2 datatype UNIQUE,
    ...
);
```

Unique Key: Examples

```
CREATE TABLE Student2 (
    Stud_ID int NOT NULL UNIQUE,
    Name varchar(45),
    Email varchar(45),
    Age int,
    City varchar(25)
);
```

Primary Key

- MySQL primary key is a single or combination of the field, which is used to identify each record in a table uniquely.
- If the column contains primary key constraints, then it cannot be null or empty. A table may have duplicate columns, but it can contain only one primary key.
- It always contains unique value into a column.

Primary Key

- When you insert a new row into the table, the primary key column can also use the AUTO_INCREMENT attribute to generate a sequential number for that row automatically.
- MySQL automatically creates an index named "Primary" after defining a primary key into the table. Since it has an associated index, we can say that the primary key makes the query performance fast.

Primary Key

- Following are the rules for the primary key:
  - The primary key column value must be unique.
  - Each table can contain only one primary key.
  - The primary key column cannot be null or empty.
  - MySQL does not allow us to insert a new row with the existing primary key.
  - It is recommended to use INT or BIGINT data type for the primary key column.

Primary Key : Way-1

```
Mysql> CREATE TABLE Login(
  login_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(40),
  password VARCHAR(55),
  email VARCHAR(55)
);
```

Primary Key : Way-2

```
mysql> CREATE TABLE Students (
    Student_ID int,
    Roll_No int,
    Name varchar(45) NOT NULL,
    Age int,
    City varchar(25),
    Primary Key(Student_ID, Roll_No)
  );
```

Foreign Key

- The foreign key is used to link one or more than one table together. It is also known as the referencing key.
- A foreign key matches the primary key field of another table. It means a foreign key field in one table refers to the primary key field of the other table.
- It identifies each row of another table uniquely that maintains the referential integrity in MySQL.

Foreign Key

- A foreign key makes it possible to create a parent-child relationship with the tables.
- In this relationship, the parent table holds the initial column values, and column values of child table reference the parent column values.
- MySQL allows us to define a foreign key constraint on the child table.
- MySQL defines the foreign key in two ways:
  - Using CREATE TABLE Statement
  - Using ALTER TABLE Statement

Foreign Key

- Table: customer

```
CREATE TABLE customer (
  ID INT NOT NULL AUTO_INCREMENT,
  Name varchar(50) NOT NULL,
  City varchar(50) NOT NULL,
  PRIMARY KEY (ID)
);
```

Foreign Key

- Table: contact

```
CREATE TABLE contact (
  ID INT,
  Customer_Id INT,
  Customer_Info varchar(50) NOT NULL,
  Type varchar(50) NOT NULL,
  INDEX par_ind (Customer_Id),
  CONSTRAINT fk_customer FOREIGN KEY (Customer_Id)
  REFERENCES customer(ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

Check

- The check constraint is an integrity constraint that controls the value in a particular column.
- It ensures the inserted or updated value in a column must be matched with the given condition.
- In other words, it determines whether the value associated with the column is valid or not with the given condition.

Check

- Before version 8.0.16, MySQL uses the limited version of this constraint.
- In the previous versions, we can create this constraint, but it does not work. It means its syntax is supported but not works with the database.
- With an earlier version, the CREATE TABLE statement can include a CHECK constraint, but they are parsed and ignored by MySQL.
- We can use it with the below syntax in the previous versions:

```
CHECK (expr)
```

Check

- The following statement creates a new table called vehicle where we specify the check constraint on a column:

```
CREATE TABLE vehicle (
   vehicle_no VARCHAR(18) PRIMARY KEY,
   model_name VARCHAR(45),
   cost_price DECIMAL(10,2 ) NOT NULL CHECK (cost_price >= 0),
   sell_price DECIMAL(10,2) NOT NULL CHECK (sell_price >= 0)
);
```

Select

- The SELECT statement in MySQL is used to fetch data from one or more tables. We can retrieve records of all fields or specified fields that match specified criteria using this statement.

```
SELECT field_name1, field_name 2,... field_nameN
FROM table_name1, table_name2...
[WHERE condition]
[GROUP BY field_name(s)]
[HAVING condition]
[ORDER BY field_name(s)]
[OFFSET M ][LIMIT N];
```

Aggregate Functions

- MySQL's aggregate function is used to perform calculations on multiple values and return the result in a single value like the average of all values, the sum of all values, and maximum & minimum value among certain groups of values.
- We mostly use the aggregate functions with SELECT statements in the data query languages.

Aggregate Functions

- count()
  - It returns the number of rows, including rows with NULL values in a group.
- sum()
  - It returns the total summed values (Non-NULL) in a set.
- avg()
  - It returns the average value of an expression.
- Min()
  - It returns the minimum (lowest) value in a set.

Aggregate Functions

- max( )
  - It returns the maximum (highest) value in a set.
- groutp_concat()
  - It returns a concatenated string.

Sorting

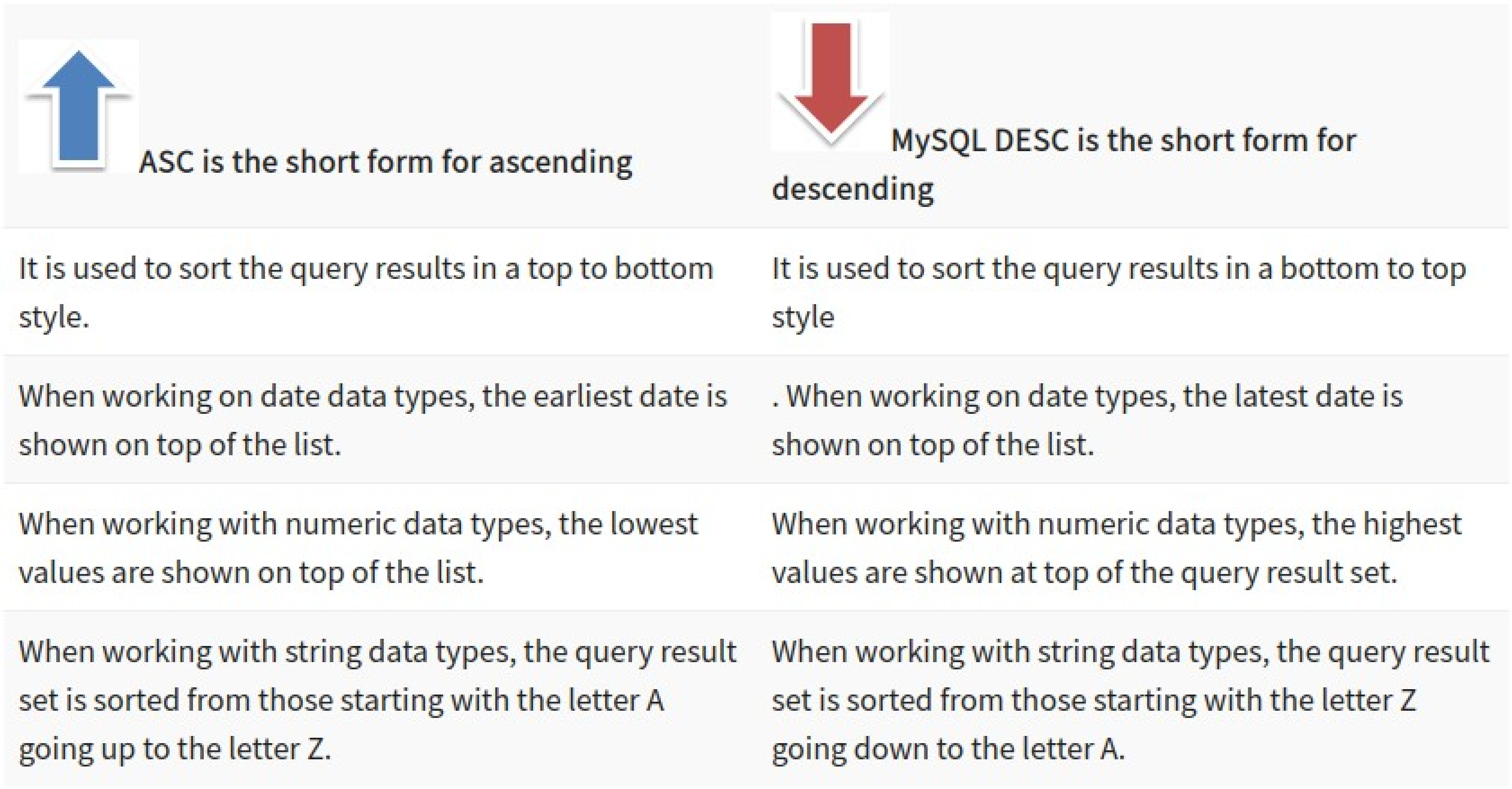
- Using the SELECT command, results were returned in the same order the records were added into the database.
- This is the default sort order. In this section, we will be looking at how we can sort our query results.
- Sorting is simply re-arranging our query results in a specified way. Sorting can be performed on a single column or on more than one column.
- It can be done on number, strings as well as date data types.

Order by

- MySQL ORDER BY is used in conjunction with the SELECT query to sort data in an orderly manner.
- The MySQL ORDER BY clause is used to sort the query result sets in either ascending or descending order.

SELECT statement... [WHERE condition | GROUP BY 'field_name(s)' HAVING condition] ORDER BY 'field_name(s)' [ASC | DESC];

Order by



ASC is the short form for ascending	MySQL DESC is the short form for descending
It is used to sort the query results in a top to bottom style.	It is used to sort the query results in a bottom to top style
When working on date data types, the earliest date is shown on top of the list.	. When working on date types, the latest date is shown on top of the list.
When working with numeric data types, the lowest values are shown on top of the list.	When working with numeric data types, the highest values are shown at top of the query result set.
When working with string data types, the query result set is sorted from those starting with the letter A going up to the letter Z.	When working with string data types, the query result set is sorted from those starting with the letter Z going down to the letter A.

Example:

- SELECT * FROM members ORDER BY date_of_birth DESC;
- SELECT * FROM 'members' ORDER BY 'gender';
- SELECT * FROM 'members' ORDER BY 'gender','date_of_birth' DESC;

The group by clause

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
- That’s what it does, summarizing data from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

The group by clause

- Now that we know what the SQL GROUP BY clause is, let’s look at the syntax for a basic group by query.

  SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING condition];

The group by clause

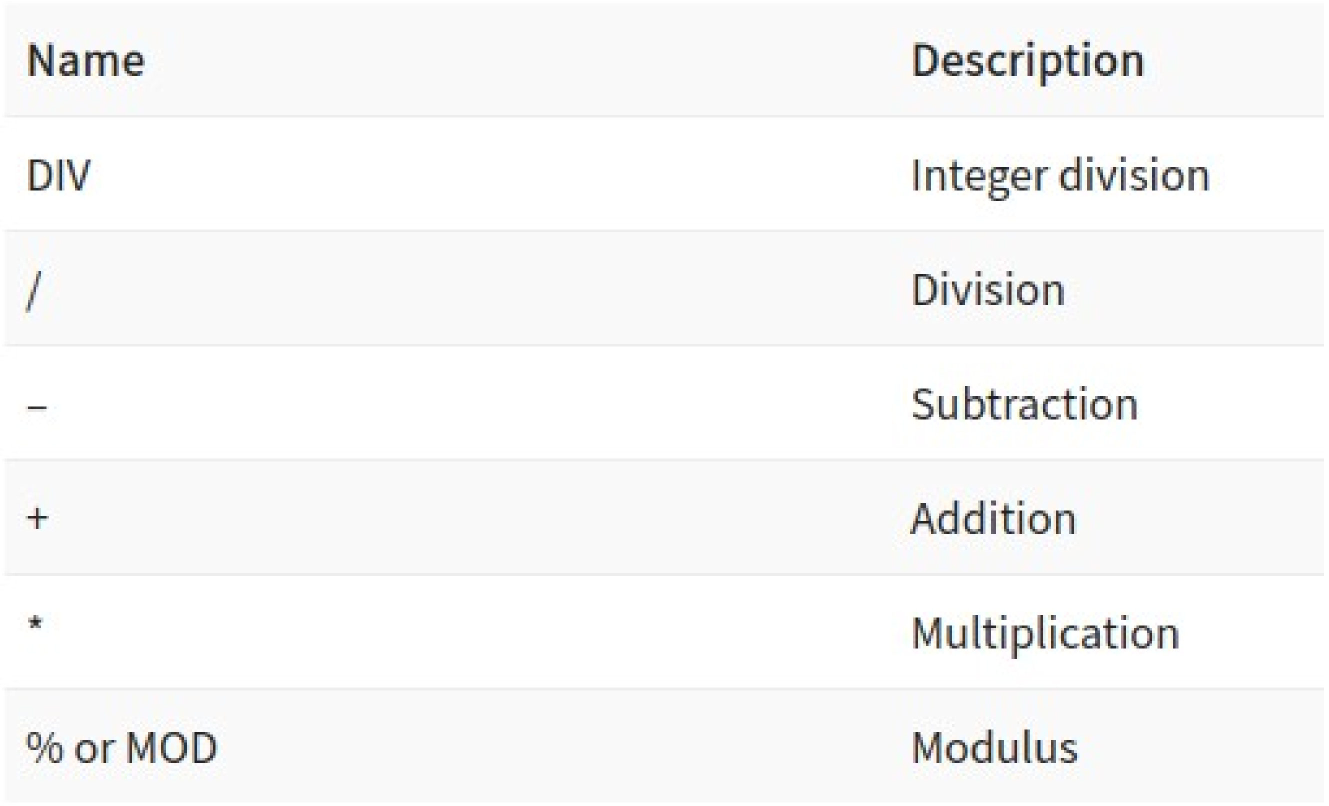
- Suppose we want to get the unique values for genders. We can use a following query –

  SELECT 'gender' FROM 'members' GROUP BY 'gender';

The group by clause

- SELECT 'category_id','year_released' FROM 'movies' GROUP BY 'category_id', 'year_released';

Numeric Functions



Name	Description
DIV	Integer division
/	Division
–	Subtraction
+	Addition
*	Multiplication
% or MOD	Modulus

Numeric Functions

- Integer Division (DIV)

  SELECT 23 DIV 6 ;

  Executing the above script gives us the following results.

  3
- Division operator (/)

  Let's now look at the division operator example. We will modify the DIV example.

  SELECT 23 / 6 ;
- Executing the above script gives us the following results.

  3.8333

Numeric Functions

- Subtraction operator (-)

  Let's now look at the subtraction operator example. We will use the same values as in the previous two examples

  SELECT 23 - 6 ;

  Executing the above script gives us 17
- Addition operator (+)

  Let's now look at the addition operator example. We will modify the previous example.

  SELECT 23 + 6 ;

  Executing the above script gives us 29

Numeric Functions

- Multiplication operator (*)

  Let’s now look at the multiplication operator example. We will use the same values as in the previous examples.

  SELECT 23 * 6 AS 'multiplication_result';

  Executing the above script gives us the following results.

  multiplication_result

  138

Numeric Functions

- Modulo operator (%)
  - The modulo operator divides N by M and gives us the remainder. Let’s now look at the modulo operator example. We will use the same values as in the previous examples.

```
SELECT 23 % 6 ;
```

OR

```
SELECT 23 MOD 6 ;
```

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;
```

Having Clause

```
SELECT emp_name, SUM(working_hours) AS "Tot work hrs"

FROM employees
GROUP BY emp_name
HAVING SUM(working_hours) > 5;
```

The like clause

- 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!

Syntax:

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

- Tip: You can also combine any number of conditions using AND or OR operators.

Examples:

- WHERE CustomerName LIKE 'a%' Finds any values that start with "a"
- WHERE CustomerName LIKE '%a' Finds any values that end with "a"
- WHERE CustomerName LIKE '%or%' Finds any values that have "or" in any position
- WHERE CustomerName LIKE '_r%' Finds any values that have "r" in the second position
- WHERE CustomerName LIKE 'a_%' Finds any values that start with "a" and are at least 2 characters in length
- WHERE CustomerName LIKE 'a__%' Finds any values that start with "a" and are at least 3 characters in length
- WHERE ContactName LIKE 'a%o' Finds any values that start with "a" and ends with "o"

Distinct

- The DISTINCT keyword that allows us to omit duplicates from our results.
- This is achieved by grouping similar values together .

```
SELECT DISTINCT 'movie_id' FROM
'movierentals';
```

Where clause

- WHERE Clause in MySQL is a keyword used to specify the exact criteria of data or rows that will be affected by the specified SQL statement.
- The WHERE clause can be used with SQL statements like INSERT, UPDATE, SELECT, and DELETE to filter records and perform various operations on the data.

Where clause

- The basic syntax for the WHERE clause when used in a MySQL SELECT WHERE statement is as follows.

```
SELECT * FROM tableName WHERE condition;
```

Where clause with AND

- The WHERE condition in MySQL when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.
- Let’s now look at a practical example – Suppose we want to get a list of all the movies in category 2 that were released in 2008, we would use the script shown below is achieve that.

  SELECT * FROM 'movies' WHERE 'category_id' = 2 AND 'year_released' = 2008;

Where clause with OR

- The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.
- The following script gets all the movies in either category 1 or category 2

```
SELECT * FROM 'movies' WHERE 'category_id' = 1 OR 'category_id' = 2;
```

Where clause with IN

- The WHERE in MySQL clause, when used together with the IN keyword only affects the rows whose values matches the list of values provided in the IN keyword.
- The MySQL IN statement helps to reduce number of OR clauses you may have to use.
- The following MySQL WHERE IN query gives rows where membership_number is either 1 , 2 or 3

```
SELECT * FROM 'members' WHERE
'membership_number' IN (1,2,3);
```

Where clause with NOT IN

- The WHERE clause when used together with the NOT IN keyword DOES NOT affects the rows whose values matches the list of values provided in the NOT IN keyword.
- The following query gives rows where membership_number is NOT 1 , 2 or 3

```
SELECT * FROM 'members' WHERE
'membership_number' NOT IN (1,2,3);
```

Where clause with NOT IN

- SELECT * FROM 'members' WHERE 'gender' = 'Female';
- SELECT * FROM 'payments' WHERE 'amount_paid' > 2000;
- SELECT * FROM 'movies' WHERE 'category_id'<> 1;

The 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.

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

The Between operator

- SELECT * FROM Products

  WHERE Price NOT BETWEEN 10 AND 20;

- SELECT * FROM student WHERE name BETWEEN 'Chintoo' AND 'Jenny'

Is NULL

- MySQL IS NULL condition is used to check if there is a NULL value in the expression.
- It is used with SELECT, INSERT, UPDATE and DELETE statement.
- Syntax:
  - expression IS NULL
- Parameter
  - expression: It specifies a value to test if it is NULL value.

Is NULL and Is not NULL

```
SELECT *
FROM officers
WHERE officer_name IS NULL;
```

```
SELECT *
FROM officers
WHERE officer_name IS NOT NULL;
```

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com