DATABASE MANAGEMENT SYSTEMS

(MS SQL)

CHAPTER - 2

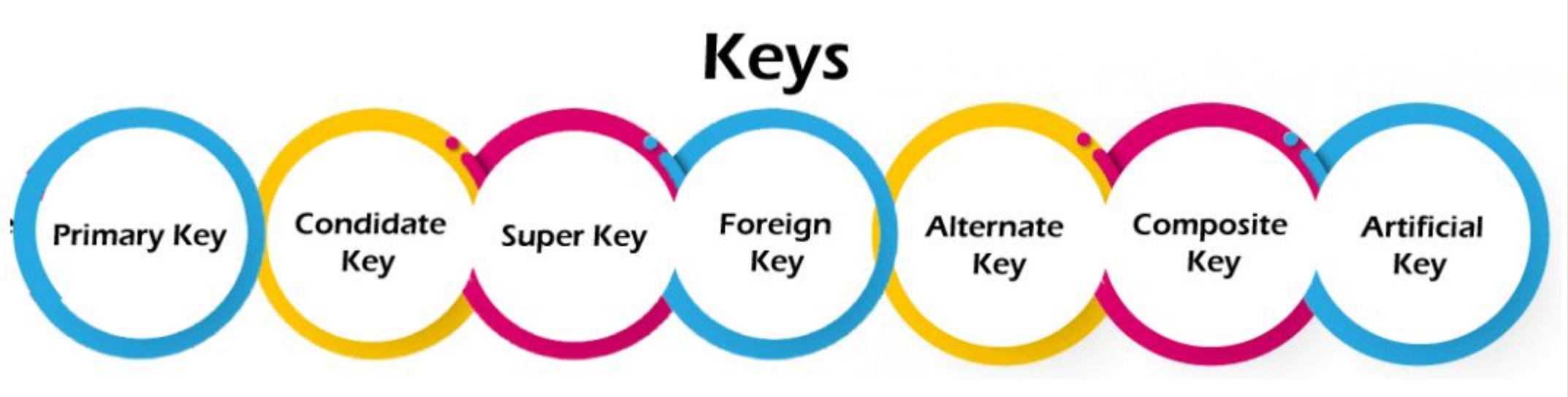
SQL Keys, Operators, Functions

Keys:

- ✔ Keys play an important role in the relational database.
- ✔ It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

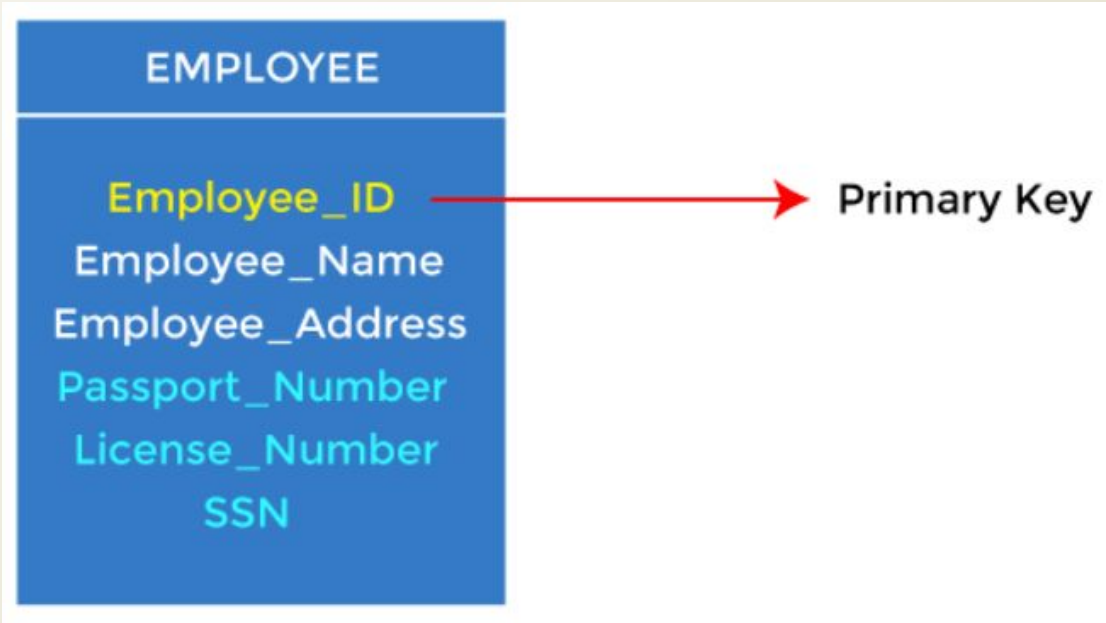
  For example, ID is used as a key in the Student table because it is unique for each student. In the PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

Types of Keys:



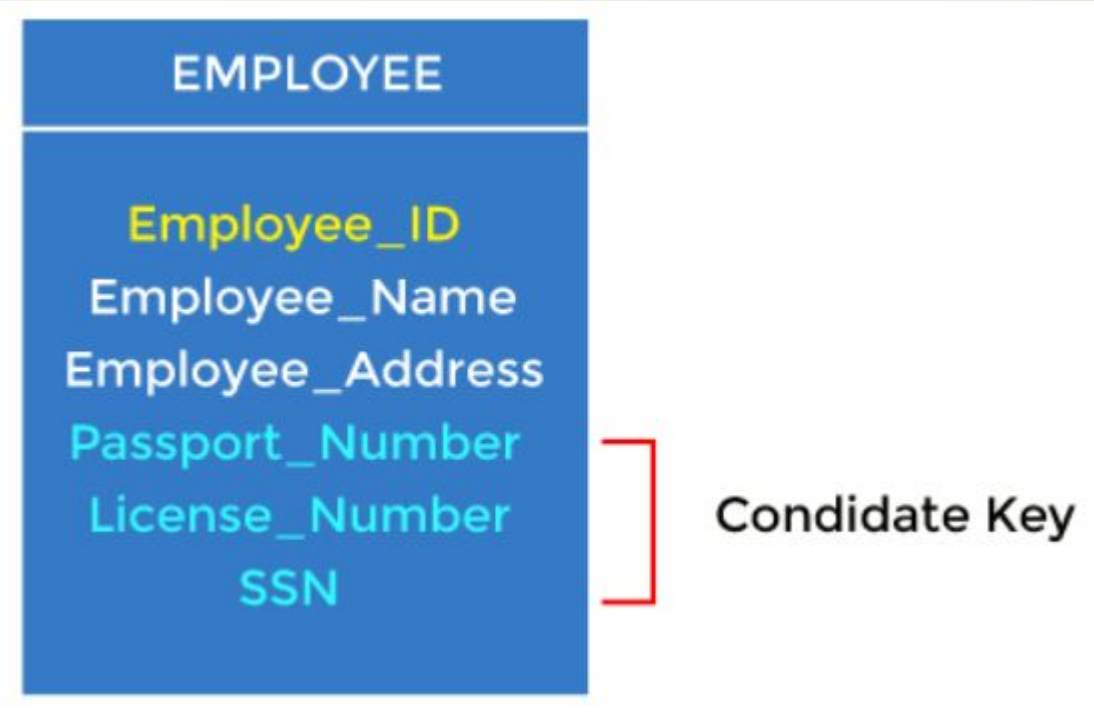
Primary key:

- ✔ It is the first key used to identify one and only one instance of an entity uniquely.
- ✔ An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- ✔ In the EMPLOYEE table, ID can be the primary key since it is unique for each employee.
- ✔ In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- ✔ For each entity, the primary key selection is based on requirements and developers.



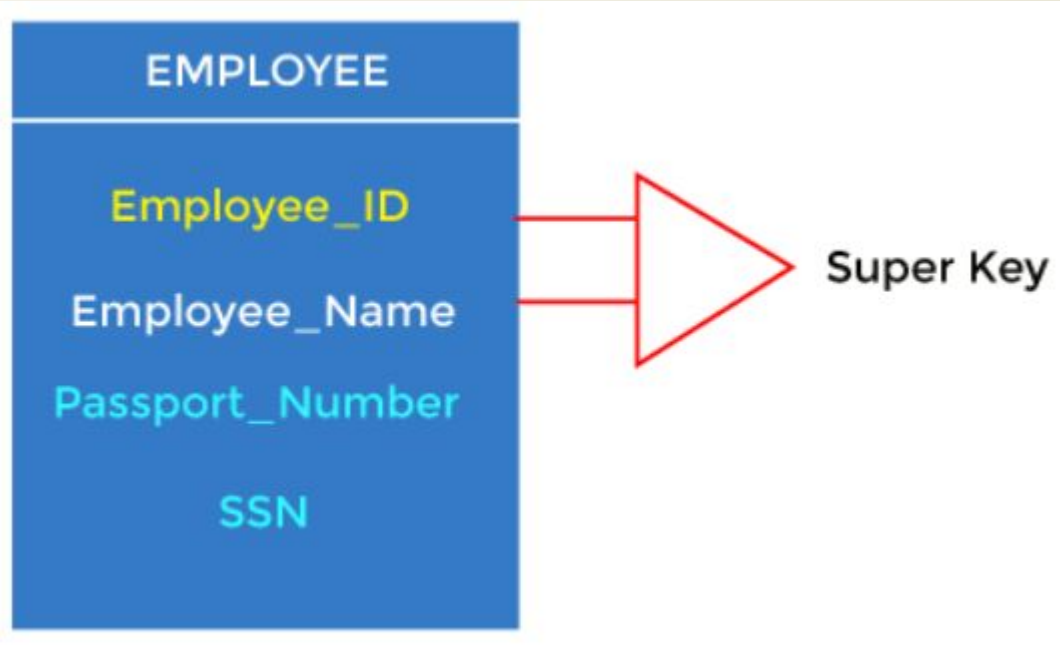
Candidate key:

- ✔ A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- ✔ Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.
- ✔ For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



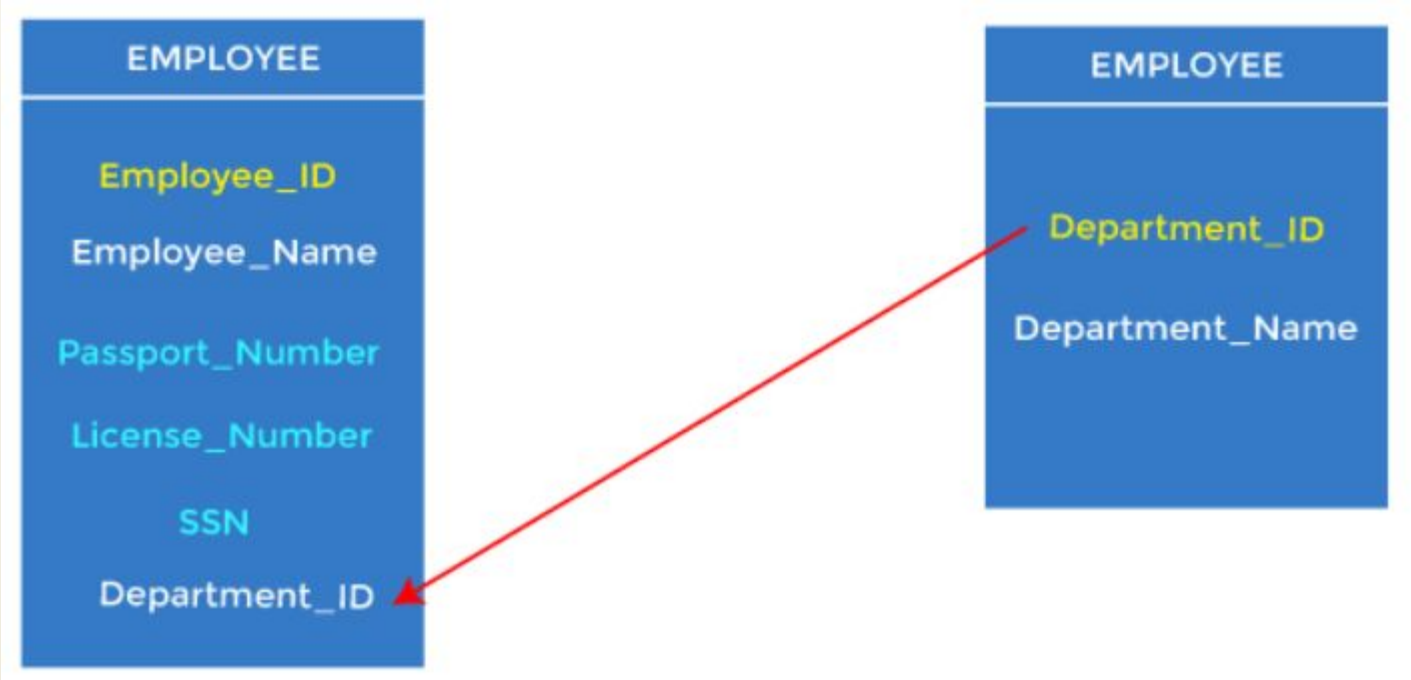
Super Key:

- ✔ Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.
- ✔ For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.
- ✔ The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.



Foreign key:

- ✔ Foreign keys are the column of the table used to point to the primary key of another table.
- ✔ Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- ✔ We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- ✔ In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

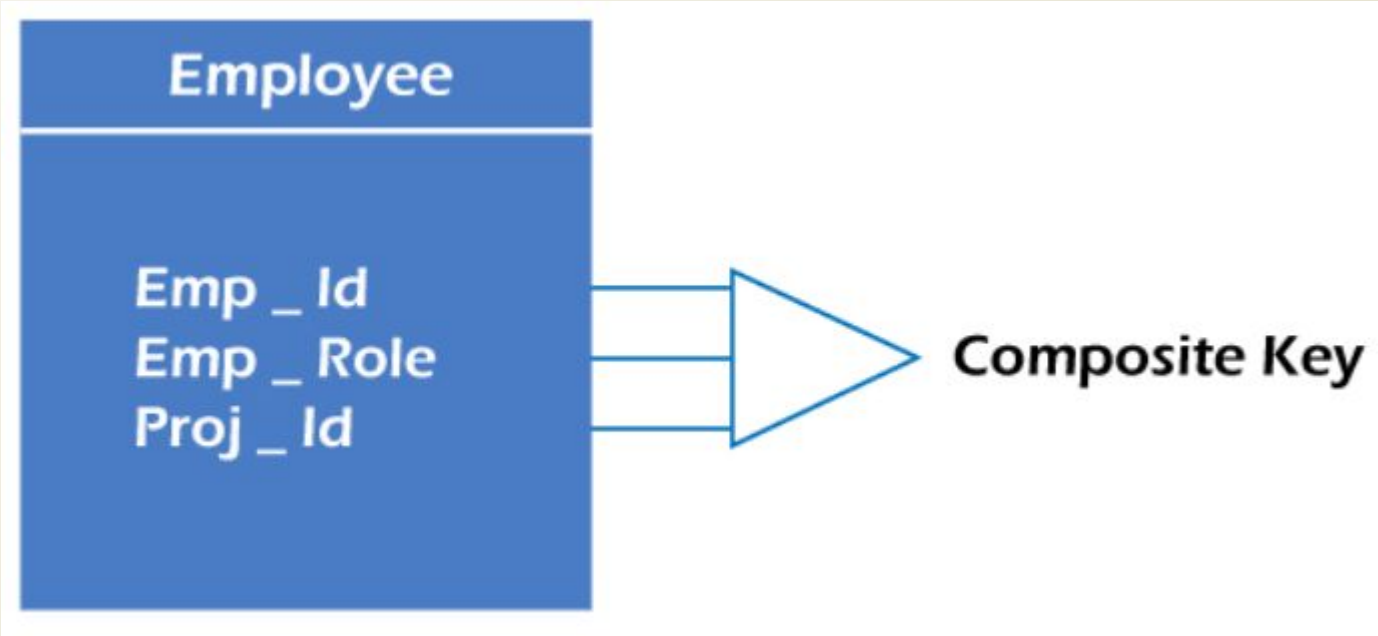


Alternate key:

- ✔ There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation.
- ✔ These attributes or combinations of the attributes are called the candidate keys.
- ✔ One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key.
- ✔ In other words, the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.
- ✔ For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.
- ✔

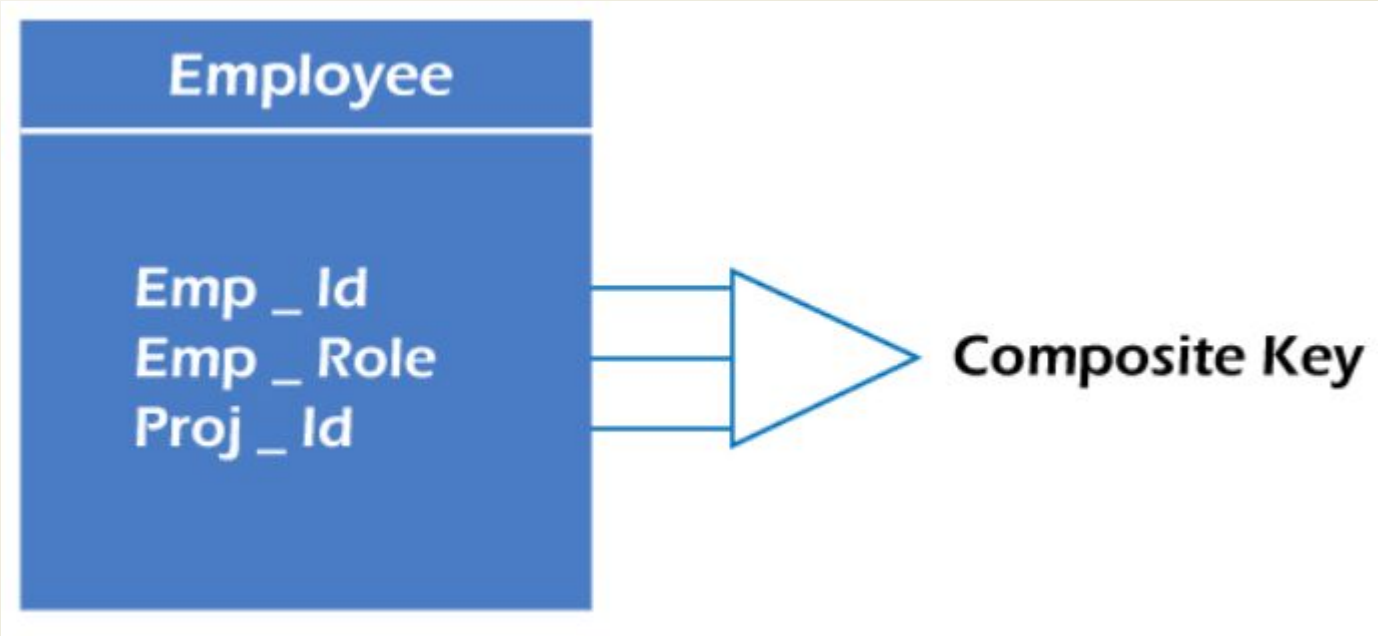
Composite key:

- ✔ Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.
- ✔ For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously.
- ✔ So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



Composite key:

- ✔ Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.
- ✔ For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously.
- ✔ So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



What are Operators in MySQL?

- Operators are used to specifying a condition in a statement in MySQL. Below are the different types of operators used in MySQL.

Arithmetic Operators

- In MySQL, arithmetic operators are used to perform the arithmetic operations as described below.

Operator	Description	Example
+	Addition of two operands	a + b
–	Subtraction of right operand from the left operand	a – b
*	Multiplication of two operands	a * b
/	Division of left operand by the right operand	a / b
%	Modulus – the remainder of the division of left operand by the right	a % b

Operators Let us assume certain values for the below variables as

a = 10 , b = 5

a + b will give the result as 15.

a – b will give the result as 5.

a * b will give the result as 50.

a / b will give the result as 2.

a % b will give the result as 0.

Comparison Operators

The comparison operators in MySql are used to compare values between operands and return true or false according to the condition specified in the statement.

Operator	Description	Example
>	If the value of left operand is greater than that of the value of the right operand, the condition becomes true; if not then false.	a > b
<	If the value of left operand is less than that of a value of the right operand, the condition becomes true; if not then false.	a < b
=	If both the operands have equal value, the condition becomes true; if not then false.	a == b
!=	If both the operands do not have equal value, the condition becomes true; if not then false.	a != y
>=	If the value of left operand is greater than or equal to the right operand, the condition becomes true; if not then false.	a >= b

Comparison Operators

<=	If the value of left operand is less than or equal to the right operand, the condition becomes true; if not then false.	a <= b
!<	If the value of left operand is not less than the value of right operand, the condition becomes true; if not then false.	a !< b
!>	If the value of left operand is not greater than the value of right operand, the condition becomes true; if not then false.	a !> b
<>	If the values of two operands are not equal, the condition becomes true; if not then false.	a <> b

Employee Table:

ID	NAME	AGE	SALARY
4	Sushma	32	35000.00
6	Ritu	23	23000.00
8	Amit	27	30000.00
11	Harish	35	35000.00
18	Pooja	28	29500.00

Employee Table:

```
SELECT * FROM EMPLOYEE WHERE SALARY > 25000;
SELECT * FROM EMPLOYEE WHERE SALARY = 35000;
SELECT * FROM EMPLOYEE WHERE SALARY < 35000;
SELECT * FROM EMPLOYEE WHERE SALARY != 30000;
SELECT * FROM EMPLOYEE WHERE SALARY <> 35000;
```

Logical Operators

BETWEEN	It is used to search within a set of values, by the minimum value and maximum value provided.
EXISTS	It is used to search for the presence of a row in a table which satisfies a certain condition specified in the query.
OR	It is used to combine multiple conditions in a statement by using the WHERE clause.
AND	It allows the existence of multiple conditions in an SQL statement's WHERE clause.
NOT	It reverses the meaning of the logical operator with which it is used. (Examples: NOT EXISTS, NOT BETWEEN, NOT IN, etc.)

Logical Operators

IN	It is used to compare a value in a list of literal values.
ALL	It compares a value to all values in another set of values.
ANY	It compares a value to any value in the list according to the condition specified.
LIKE	It uses wildcard operators to compare a value to similar values.
IS NULL	It compares a value with a NULL value.
UNIQUE	It searches for every row of a specified table for uniqueness (no duplicates).

Logical Operators:

SELECT * FROM EMPLOYEE WHERE AGE <= 25 AND SALARY >= 5000;

SELECT * FROM EMPLOYEE WHERE AGE >= 25 OR SALARY >= 25000;

SELECT * FROM EMPLOYEE WHERE AGE IS NOT NULL;

SELECT * FROM EMPLOYEE WHERE NAME LIKE 'Am%';

SELECT * FROM EMPLOYEE WHERE AGE BETWEEN 25 AND 30;

SELECT NAME FROM EMPLOYEE WHERE EXISTS (SELECT NAME FROM EMPLOYEE WHERE SALARY > 25000);

MySQL 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!

Logical Operators

LIKE Operator	Description
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"

Control Flow Functions: IF()

- IF function is used for validating a function in MySQL.
- The IF function returns a value YES when the given condition evaluates to true and returns a NO value when the condition evaluates to false.
- It returns values either in a string or numeric form depending upon the context in which this function is used.
- Sometimes, this function is known as IF-ELSE and IF THEN ELSE function.

IF (expression 1, expression 2, expression 3)

Expression 1	Required	It is a value, which is used for validation.
Expression 2	Optional	It returns a value when the condition evaluates to true.
Expression 3	Optional	It returns a value when the condition evaluates to false.

Default Return Type:

The return type of IF function can be calculated as follows:

If expression 2 or expression 3 are both strings or produce a string, the result is always a string.

If expression 2 or expression 3 gives a floating-point value, the result is always a floating-point value.

If expression 2 or expression 3 is an integer, the result is always an integer.

Examples:

```
SELECT IF(200>350,'YES','NO');

SELECT IF(251 = 251,' Correct','Wrong');

SELECT IF(STRCMP('Rinky Ponting','Yuvraj Singh')=0, 'Correct', 'Wrong');

SELECT lastname,

IF(age>20,"Mature","Immature")   As Result

FROM student;
```

studentid	firstname	lastname	class	age
1	Rinky	Ponting	12	20
2	Mark	Boucher	11	22
3	Sachin	Tendulkar	10	18
4	Peter	Fleming	10	22
5	Virat	Kohli	12	23
NULL	NULL	NULL	NULL	NULL

IFNULL()

- ✔ The IFNULL function is a part of the MySQL control flow function used for handling NULL values.
- ✔ The IFNULL function accepts two expressions, and if the first expression is not null, it returns the first arguments.
- ✔ If the first expression is null, it returns the second argument.
- ✔ This function returns either string or numeric value, depending on the context where it is used.

IFNULL (Expression1, Expression2)

It returns expression1 when the expression1 is not null. Otherwise, it will return expression2.

IFNULL()

Parameters

Parameter	Requirement	Descriptions
Expression 1	Required	This expression is used to check whether it is NULL or not.
Expression 2	Required	It will return when the expression 1 is NULL.

Example:

```
SELECT IFNULL(0,5);
SELECT IFNULL("Hello", "javaTpoint");
SELECT IFNULL(NULL,5);

CREATE TABLE 'student_contacts' (
'studentid' int unsigned NOT NULL AUTO_INCREMENT,
`contactname` varchar(45) NOT NULL,
`cellphone` varchar(20) DEFAULT NULL,
`homephone` varchar(20) DEFAULT NULL,
);
```

IFNULL()

After inserting the values into the table, execute the following query.

```
SELECT
    contactname, cellphone, homephone
FROM
    student_contacts;
```

It will display the output that contains all rows and columns. Here, we can see that some of the contacts have only a cell phone or home phone number.

studentid	contactname	cellphone	homephone
2	Will Smith	3214356574	NULL
3	Johnsena	NULL	4563480897
4	Peter Joe	NULL	2123157870
5	kelly Bruke	5683128765	NULL
6	Freeda Pinto	4563482354	NULL
NULL	NULL	NULL	NULL

In the above output, we will get all contacts name weather cell phone, and home phone number is available or not. So, in that case, the IFNULL() function plays an important role.

Now, run the following MySQL query. This statement returns the home phone number if the cell phone is NULL.

```
SELECT   contactname, IFNULL(cellphone, homephone) phone
FROM   student_contact;
```

contactname	phone
Will Smith	3214356574
Johnsena	4563480897
Peter Joe	2123157870
kelly Bruke	5683128765
Freeda Pinto	4563482354

Note: You should avoid the use of the IFNULL() function in the WHERE clause because this function reduces the performance of the query.

MySQL NULLIF()

- The NULLIF function accepts two expressions, and if the first expression is equal to the second expression, it returns the NULL. Otherwise, it returns the first expression.

Syntax

NULLIF (Expression1, Expression2)

It returns Null when expression1 is equal to expression2. Otherwise, it will return expression1.

Parameter	Requirement	Descriptions
Expression 1	Required	It specify the first expression for comparison.
Expression 2	Required	It specify the second expression for comparison.

Examples:

SELECT NULLIF("javaTpoint", "javaTpoint");

Output: NULL

SELECT NULLIF("Hello", "404");

Output: Hello

SELECT NULLIF(9,5);

Output: 9

In this example, we are going to understand how NULLIF() function prevents division by zero error. If we run the query "SELECT 1/0", then we get an error output.

SELECT 1/NULLIF(0,0);

Output: NULL

```
CREATE TABLE 'customer' (
  'customer_id' INT UNSIGNED NOT NULL AUTO_INCREMENT,
  'cust_name' VARCHAR(45) NOT NULL,
  'occupation' VARCHAR(45) NOT NULL,
  'income' VARCHAR(15) NOT NULL,
  'qualification' VARCHAR(45) NOT NULL
);
```

```
INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('John Miller', 'Developer', '20000', 'Btech');

INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('Mark Robert', 'Enginneer', '40000', 'Btech');

INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('Reyan Watson', 'Scientists', '60000', 'MSc');

INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('Shane Trump', 'Businessman', '10000', 'MBA');

INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('Adam Obama', 'Manager', '80000', 'MBA');

INSERT INTO 'myproductdb'.'customer' ('cust_name', 'occupation', 'income', 'qualification')
VALUES ('Rincky Ponting', 'Cricketer', '200000', 'Btech');
```

After inserting the values into the table, execute the following query.

```
SELECT * FROM customer;
```

It will give the following table:

customer_id	cust_name	occupation	income	qualification
1	John Miller	Developer	20000	Btech
2	Mark Robert	Enginneer	40000	Btech
3	Reyan Watson	Scientists	60000	MSc
4	Shane Trump	Businessman	10000	MBA
5	Adam Obama	Manager	80000	MBA
6	Rincky Ponting	Cricketer	200000	Btech
NULL	NULL	NULL	NULL	NULL

Now, we are going to use the NULLIF function to check the qualification column value against the Btech. It means if the customer occupation is Btech, it returns NULL. Otherwise, it returns the column value.

```
SELECT cust_name, occupation, qualification,
NULLIF (qualification,"Btech") result
FROM myproductdb.customer;
```

cust_name	occupation	qualification	result
John Miller	Developer	Btech	NULL
Mark Robert	Enginneer	Btech	NULL
Reyan Watson	Scientists	MSc	MSc
Shane Trump	Businessman	MBA	MBA
Adam Obama	Manager	MBA	MBA
Rincky Ponting	Cricketer	Btech	NULL

MySQL CASE Expression

- The CASE expression validates various conditions and returns the result when the first condition is true.
- Once the condition is met, it stops traversing and gives the output.
- If it will not find any condition true, it executes the else block. When the else block is not found, it returns a NULL value.
- The main goal of MySQL CASE statement is to deal with multiple IF statements in the SELECT clause.
- Consider a search_condition in the WHEN clauses, and if it finds, return the result in the corresponding THEN clause. Otherwise, it will return the else clause. If else clause is not specified, it will return a NULL value.

```
CASE
    WHEN [condition] THEN result
    [WHEN [condition] THEN result ...]
    [ELSE result]
END
```

Let us create a table 'students' and perform the CASE statement on this table.

studentid	firstname	lastname	class	age
1	Ricky	Ponting	CS	20
2	Mark	Boucher	EE	22
3	Michael	Clark	CS	18
4	Peter	Fleming	CS	22
5	Virat	Kohli	EC	23
NULL	NULL	NULL	NULL	NULL

In the above table, we can see that the class column contains the short form of the student's department.

```
SELECT studentid, firstname,
 CASE class
    WHEN 'CS' THEN 'Computer Science'
    WHEN 'EC' THEN 'Electronics and Communication'
    ELSE 'Electrical Engineering'
END AS department from students;
```

MySQL 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 Function	Descriptions
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.
average()	It returns the average value of an expression.
min()	It returns the minimum (lowest) value in a set.
max()	It returns the maximum (highest) value in a set.
groutp_concat()	It returns a concatenated string.
first()	It returns the first value of an expression.
last()	It returns the last value of an expression.

Count() Function

- MySQL count() function is used to returns the count of an expression.
- It allows us to count all rows or only some rows of the table that matches a specified condition.
- It is a type of aggregate function whose return type is BIGINT. This function returns 0 if it does not find any matching rows.

- **COUNT(*) Function:** This function uses the SELECT statement to returns the count of rows in a result set. The result set contains all Non-Null, Null, and duplicates rows.
- **COUNT(expression) Function:** This function returns the result set without containing Null rows as the result of an expression.
- **COUNT(distinct expression) Function:** This function returns the count of distinct rows without containing NULL values as the result of the expression.

```
SELECT COUNT (aggregate_expression)
FROM table_name
[WHERE conditions];
```

Examples:

```
mysql> SELECT * FROM employees;
+--------+----------+---------+------------+---------+
| emp_id | emp_name | emp_age | city       | income  |
+--------+----------+---------+------------+---------+
|    101 | Peter    |      32 | Newyork    |  200000 |
|    102 | Mark     |      32 | California |  300000 |
|    103 | Donald   |      40 | Arizona    | 1000000 |
|    104 | Obama    |      35 | Florida    | 5000000 |
|    105 | Linklon  |      32 | Georgia    |  250000 |
|    106 | Kane     |      45 | Alaska     |  450000 |
|    107 | Adam     |      35 | California | 5000000 |
|    108 | Macculam |      40 | Florida    |  350000 |
+--------+----------+---------+------------+---------+
8 rows in set (0.01 sec)
```

```
SELECT COUNT(emp_name) FROM employees;

SELECT COUNT(*) FROM employees WHERE emp_age>32;

SELECT COUNT(DISTINCT emp_age) FROM employees;

SELECT emp_name, city, COUNT(*) FROM employees GROUP BY city;

SELECT emp_name, emp_age, COUNT(*) FROM employees
GROUP BY emp_age
HAVING COUNT(*)>=2
ORDER BY COUNT(*);
```

Sum() function:

- ✔ MySQL sum() function is used to return the total summed value of an expression. It returns NULL if the result set does not have any rows. It is one of the kinds of aggregate functions in MySQL.

Syntax:

SELECT SUM(aggregate_expression)

FROM tables

[WHERE conditions];

```
mysql> SELECT * FROM employees;
+--------+----------+------------+--------------+---------------+
| emp_id | emp_name | occupation | working_date | working_hours |
+--------+----------+------------+--------------+---------------+
|      1 | Joseph   | Business   | 2020-04-10   |            10 |
|      2 | Stephen  | Doctor     | 2020-04-10   |            15 |
|      3 | Mark     | Engineer   | 2020-04-10   |            12 |
|      4 | Peter    | Teacher    | 2020-04-10   |             9 |
|      1 | Joseph   | Business   | 2020-04-12   |            10 |
|      2 | Stephen  | Doctor     | 2020-04-12   |            15 |
|      4 | Peter    | Teacher    | 2020-04-12   |             9 |
|      3 | Mark     | Engineer   | 2020-04-12   |            12 |
|      1 | Joseph   | Business   | 2020-04-14   |            10 |
|      4 | Peter    | Teacher    | 2020-04-14   |             9 |
+--------+----------+------------+--------------+---------------+
10 rows in set (0.00 sec)
```

Examples:

SELECT SUM(working_hours) AS "Total working hours" FROM employees;

SELECT SUM(working_hours) AS "Total working hours" FROM employees WHERE working_hours>=12;

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

```
SELECT emp_id, emp_name, occupation,
SUM(working_hours) Total_working_hours
FROM employees
GROUP BY occupation
HAVING SUM(working_hours)>24;
```

```
SELECT emp_name, occupation,
SUM(DISTINCT working_hours) Total_working_hours
FROM employees
GROUP BY occupation;
```

Avg() function:

- MySQL avg() is an aggregate function used to return the average value of an expression in various records.

```
SELECT AVG(working_hours) Avg_working_hours FROM employees;

SELECT AVG(working_hours) Avg_working_hours FROM employees WHERE working_hours>=12;

SELECT emp_name, occupation, AVG(working_hours) Avg_working_hours FROM employees GROUP BY occupation;

SELECT emp_name, occupation,
AVG(working_hours) Avg_working_hours
FROM employees
GROUP BY occupation
HAVING AVG(working_hours)>9;

SELECT emp_name, occupation,
AVG(DISTINCT working_hours) Avg_working_hours
FROM employees
GROUP BY occupation;
```

MIN() Function

- The MIN() function in MySQL is used to return the minimum value in a set of values from the table. It is an aggregate function that is useful when we need to find the smallest number, selecting the least expensive product, etc.

- SELECT MIN(income) AS Minimum_Income FROM employees;
- SELECT MIN(income) AS Minimum_Income
- FROM employees
- WHERE emp_age >= 32 AND emp_age <= 40;

- Find the minimum income in all rows from the employee table for each emp_age group.

```
SELECT emp_age, MIN(income) AS Minimum_Income
FROM employees
GROUP BY emp_age;
```

MIN() Function

- Returns the minimum income of all employees, grouping them based on their city and returns the result whose MIN(income)>150000.

```
SELECT city, MIN(income) AS Minimum_Income
FROM employees
GROUP BY city
HAVING MIN(income) > 150000;
```

- Removes the duplicate records in the income column of the employee table, group by city, and then returns the minimum value:

```
SELECT emp_name, city, MIN(DISTINCT income) AS Minimum_Income
FROM employees
GROUP BY city;
```

MAX() Function

- The MySQL MAX() function is used to return the maximum value in a set of values of an expression. This aggregate function is useful when we need to find the maximum number, selecting the most expensive product, or getting the largest payment to the customer from your table.
- find the maximum income of the employee available in the table:

```
SELECT MAX(income) AS "Maximum Income" FROM employees;
```

- Finds the maximum income in all rows from the employee table. The WHERE clause specifies all those rows whose emp_age column is greater than 35.

```
SELECT MAX(income) AS "Maximum_Income"
FROM employees
WHERE emp_age > 35;
```

- find the maximum income in all rows from the employee table for each emp_age group.

```
SELECT emp_age, MAX(income) AS "Maximum Income"
FROM employees
GROUP BY emp_age;
```

MAX() Function

- returns the maximum income among all employees, grouping them based on their city and returns the result whose MAX(income) >= 200000.

```
SELECT city, MAX(income) AS "Maximum Income"
FROM employees
GROUP BY city
HAVING MAX(income) >= 200000;
```

- removes the duplicate records in the employee table's income column, group by city, and then returns the maximum value:

```
SELECT city, MAX(DISTINCT income) AS "Maximum Income"
FROM employees
GROUP BY city;
```

```
SELECT * FROM employees WHERE  emp_age = (SELECT MAX(emp_age) FROM employees);
```

- The subquery first finds the maximum age of employees from the table. Then, the main query (outer query) returns the result of age being equal to the maximum age returned from the subquery and other information.

GROUP_CONCAT() Function

- The GROUP_CONCAT() function in MySQL is a type of an aggregate function. This function is used to concatenate string from multiple rows into a single string using various clauses. If the group contains at least one non-null value, it always returns a string value. Otherwise, you will get a null value.

```
GROUP_CONCAT(
  DISTINCT expression
  ORDER BY expression
  SEPARATOR sep  );
```

```
SELECT c1, c2, ....., cN
GROUP_CONCAT (
 [DISTINCT] c_name1
 [ORDER BY]
 [SEPARATOR] )
FROM table_name GROUP BY c_name2;
```

GROUP_CONCAT() Function

emp_id	emp_fname	emp_lname	dept_id	designation
1	David	Miller	2	Engineer
2	Peter	Watson	3	Manager
3	Mark	Boucher	1	Scientist
2	Peter	Watson	3	BDE
1	David	Miller	2	Developer
4	Adam	Warner	4	Receptionist
3	Mark	Boucher	1	Engineer
4	Adam	Warner	4	Clerk

GROUP_CONCAT() Function

SELECT emp_id, emp_fname, emp_lname, dept_id,

GROUP_CONCAT(designation) as "designation" FROM employee group by emp_id;

emp_id	emp_fname	emp_lname	dept_id	designation
1	David	Miller	2	Engineer,Developer
2	Peter	Watson	3	Manager,BDE
3	Mark	Boucher	1	Scientist,Engineer
4	Adam	Warner	4	Receptionist,Clerk

LIMIT Function:

```
SELECT column_name
FROM table_name
LIMIT number;
```

```
SELECT officer_name
FROM officers
LIMIT 1;
```

```
SELECT officer_name
FROM officers
LIMIT 2;
```

```
SELECT column_name
FROM table_name
ORDER BY column_name DESC
LIMIT 1;
```

MySQL String Functions

CONCAT function

- The MySQL CONCAT function takes one or more string arguments and concatenates them into a single string. The CONCAT function requires a minimum of one parameter otherwise it raises an error.

CONCAT(string1,string2, ...);

- The CONCAT function converts all arguments to the string type before concatenating. If any argument is NULL, the CONCAT function returns a NULL value.
-
- Ex:
- SELECT CONCAT('MySQL','CONCAT');
- Result: MYSQLCONCAT

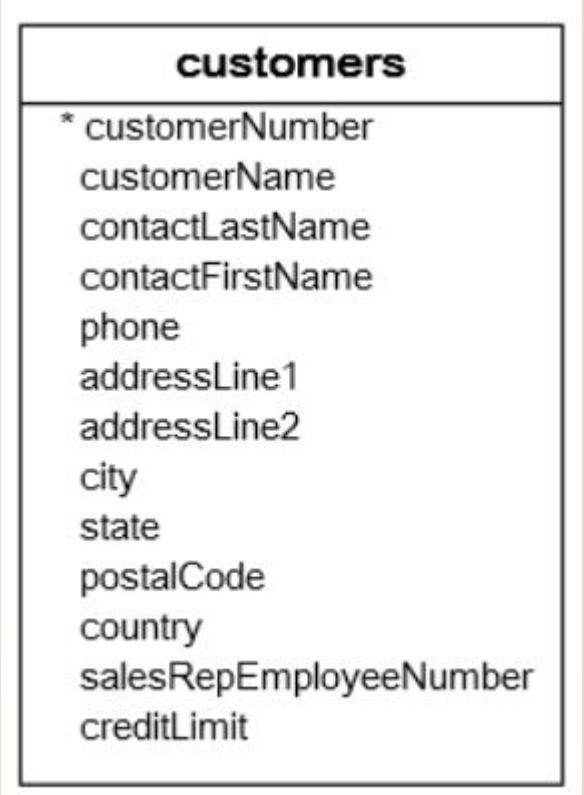
If you add a NULL value, the CONCAT function returns a NULL value as follows:

```
SELECT CONCAT('MySQL',NULL,'CONCAT');
```

Result: NULL

To get the full names of contacts, you use the CONCAT function to concatenate first name, space, last name as the following statement:

```
SELECT
 concat(contactFirstName,' ',contactLastName) Fullname
FROM customers;
```



CONCAT_WS function: Concatenate strings with a separator

MySQL provides a special form of the CONCAT function: CONCAT_WS function. The CONCAT_WS function concatenates two or more string values with a predefined separator.

CONCAT_WS(seperator,string1,string2, ...);

The CONCAT_WS function adds the separator between string arguments and returns a single string with the separator inserted between string arguments.

The following statement concatenates two string values: John and Doe, and separates these two strings by a comma:

SELECT CONCAT_WS(',' , 'John','Doe');

The CONCAT_WS function returns NULL if and only if the first argument, which is the separator, is NULL.

SELECT CONCAT_WS(NULL ,'Jonathan', 'Smith');

Result: NULL

Unlike the CONCAT function, the CONCAT_WS function skips NULL values after the separator argument. In other words, it ignores NULL values.

SELECT CONCAT_WS(',','Jonathan', 'Smith',NULL);

Jonathan,Smith

```
SELECT
   CONCAT_WS(CHAR(13),
         CONCAT_WS(' ', contactLastname, contactFirstname),
         addressLine1,
         addressLine2,
         CONCAT_WS(' ', postalCode, city),
         country,
         CONCAT_WS(CHAR(13), '')) AS Customer_Address
FROM   customers;
```

SELECT CONCAT_WS(NULL ,'Jonathan', 'Smith');

Result: NULL

Unlike the CONCAT function, the CONCAT_WS function skips NULL values after the separator argument. In other words, it ignores NULL values.

SELECT CONCAT_WS(',','Jonathan', 'Smith',NULL);

Jonathan,Smith

```
 SELECT
    CONCAT_WS(CHAR(13),
            CONCAT_WS(' ', contactLastname, contactFirstname),
            addressLine1,
            addressLine2,
            CONCAT_WS(' ', postalCode, city),
            country,
            CONCAT_WS(CHAR(13), '')) AS Customer_Address
FROM   customers;
```

```
Customer_Address
----------------------

Schmitt Carine
54, rue Royale
44000 Nantes
France


King Jean
8489 Strong St.
83030 Las Vegas
USA
```

MySQL INSTR Function

he INSTR function returns the position of the first occurrence of a substring in a string.

If the substring is not found in the str, the INSTR function returns zero (0).

INSTR(str,substr);

The str is the string that you want to search in.

The substr is the substring that you want to search for.

The INSTR function is not case sensitive. It means that it does not matter if you pass the lowercase, uppercase, title case, etc., the results are always the same.

```
SELECT INSTR('MySQL INSTR', 'MySQL');
```
Result: 1

```
SELECT INSTR('MySQL INSTR', 'mysql');
```
Result: 1

```
SELECT INSTR('MySQL INSTR', BINARY 'mysql');
```
Result: 0

```
SELECT
   productName
FROM
   products
WHERE
   INSTR(productname,'Car') > 0;
```

	productName
▶	1911 Ford Town Car
	1999 Indy 500 Monte Carlo SS
	18th Century Vintage Horse Carriage
	1917 Maxwell Touring Car
	1950's Chicago Surface Lines Streetcar
	1962 City of Detroit Streetcar

MySQL Date Time Functions
ADDDATE() Function

- The MYSQL ADDDATE() function is used to add the specified interval to a date value.

Syntax

ADDDATE(date, INTERVAL expr unit);

date is the value representing the date it can be of the type String, DATE (YEAR, MONTH, and DAY), DATETIME (HOURS, MINUTES or, SECONDS) or, TIMESTAMP.

expr is the value representing the interval value.

unit is the interval type represented by the expr value which can be DAY, WEEK, MONTH, QUARTER, YEAR, HOUR, MINUTE, SECOND, MICROSECOND

This function has another syntax as −

ADDDATE(expr, days);

expr − is the expression representing the date.

days − is the interval representing the number of days that is to added to the given date.

Examples:

SELECT ADDDATE('2015-09-05', INTERVAL 20 DAY);

2015-09-25

SELECT ADDDATE('2008-01-02', INTERVAL 4 YEAR);

2012-01-02

SELECT ADDDATE('1995-11-15', 554);

1997-05-22

SELECT ADDDATE('2015-09-05', INTERVAL -20 DAY);

2015-08-16

SELECT ADDDATE('1995-11-15', -554);

1994-05-10

SELECT ADDDATE('2021-03-22', INTERVAL '10.21' MINUTE_SECOND);

2021-03-22 00:10:21

CURDATE() Function

The MYSQL CURDATE() is used to get the current days date.

The resultant value is a string or a numerical value based on the context and, the date returned will be in the 'YYYY-MM-DD' or YYYYMMDD format.

Syntax

CURDATE();

SELECT CURDATE();

2023-02-15

SELECT CURDATE() +0;

20230215

SELECT CURDATE()+2;

20230217

DATEDIFF() Function

The MYSQL DATEDIFF() function accepts two date or, date-time values as parameters, calculates the difference between them (argument1-argument2) and returns the result.

This function returns difference between the given date values in the form of **days**.

This function includes only the date parts of the arguments while calculating the difference.

Syntax

DATEDIFF(expr1, expr2)

SELECT DATEDIFF('2015-09-05', '1989-03-25');

9660

SELECT DATEDIFF('2019-05-25', '2019-05-05');

20

SELECT DATEDIFF('2050-03-25', CURDATE());

10480

YEAR() Function

The MYSQL YEAR() function is used to retrieve and return the year of the given date or, date time expression. This function returns a numerical value ranging from 1000 to 9999.

Syntax: YEAR(date);

Where, date is the date value from which you need to retrieve the year.

SELECT YEAR('2019-05-25');

2019

SELECT YEAR('2008-09-17');

2008

SELECT YEAR('0000-00-07');

0

SELECT YEAR(CURDATE());

2023

MONTH() Function

The MYSQL MONTH() function is used to retrieve and return the MONTH of the given date or, date time expression.

This function returns a numerical value ranging from 1 to 12 representing the month (January to December).

Syntax: MONTH(date);

Where, date is the date value from which you need to retrieve the month.

SELECT MONTH('2019-05-25');

5

SELECT MONTH(1990-11-11);

NULL

SELECT MONTH(CURDATE());

02

DAY() Function

The MYSQL DAY() function is a synonym for DAYOFMONTH() function. It is used to retrieve the day of the month from the given date.

This function returns the numerical value representing the day of the month which will be a number from 1 to 30.

Syntax: DAY(date);

Where, date is the date value from which you need to get the day of the month.

or a non-string value as an argument this function returns NULL.

SELECT DAY('2015-09-05 09:40:45.2300');

5

SELECT DAY(CURDATE());

15

HOUR() Function

The MYSQL HOUR() function is used to retrieve and return the hour of time from the given date time expression.

Syntax: HOUR(time);

Where, time is the time expression from which you need to extract the hour.

SELECT HOUR('00:00:00 09:40:45.2300');

9

SELECT HOUR('00 12:38:48');

12

SELECT HOUR('2015-09-05 22:40:45.2300');

22

SELECT HOUR(NOW());

10

MINUTE() Function

The MYSQL MINUTE() function is used to retrieve and return the minutes in the given time or date time expression.

This returns a numerical value ranging from 0 to 59.

Syntax: MINUTE(time);

Where, time is the time expression from which you need to extract the minute.

```
SELECT MINUTE('00:00:00 09:40:45.2300');
40

SELECT MINUTE('00 12:38:48');
38

SELECT MINUTE(CURRENT_TIMESTAMP);
54
```

SECOND() Function

The MYSQL SECOND() function is used to retrieve and return the seconds value from the given time or date time expression.

Syntax

Following is the syntax of the above function –

```
SECOND(time);
```

Where, time is the time expression from which you need to extract the hour.

```
SELECT SECOND('00:00:00 09:40:45.2300');
```

45

```
SELECT SECOND(NOW());
```

43

DATE() Function

The MYSQL DATE() function is used to retrieve and return the date part of the given date time expression.

Syntax:

DATE(expr);

Where, expr is the date and time expression from which you need to extract the date.

SELECT DATE('2015-09-05 09:40:45.2300');

2015-09-05

SELECT DATE(CURRENT_TIMESTAMP);

2021-07-12

SELECT DATE(NOW());

2023-02-15

CURRENT_TIMESTAMP() Function

The MYSQL CURRENT_TIMESTAMP() function is the synonym for NOW().

It used to get the current date and time value.

The resultant value is a string or a numerical value based on the context and, the value returned will be in the 'YYYY-MM-DD hh:mm:ss' or YYYYMMDDhhmmss format.

Syntax:

CURRENT_TIMESTAMP();

SELECT CURRENT_TIMESTAMP();

2021-07-10 22:11:24

TIMESTAMP() Function

The MYSQL TIMESTAMP() function is converts the date or datetime expression as a datetime value and returns the result in the form o f a string.

Syntax

TIMESTAMP(expr), TIMESTAMP(expr1,expr2)

Where, expr is the date-time or the time expression from which you need to get the timestamp.

SELECT TIMESTAMP('2012:11:01');

2012-11-01 00:00:00

SELECT TIMESTAMP('2015-09-05 09:40:45.2300');

2015-09-05 09:40:45.2300

SELECT TIMESTAMP('1986:06:26', '12:45:38');

1986-06-26 12:45:38

SELECT TIMESTAMP(CURRENT_TIMESTAMP, '12:12:12');

2021-07-15 11:24:15

SELECT TIMESTAMP('1986:06:26', CURTIME());

SELECT TIMESTAMP(NOW());

2021-07-14 23:12:26

TIMESTAMPDIFF() Function

The MYSQL TIMESTAMPDIFF() function accepts two datetime or date expressions as parameters, calculates the difference between them and returns the result. One of the arguments can be date and the other a datetime expression.

Syntax

TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)

SELECT TIMESTAMPDIFF(DAY, '1898-03-22', '2019-05-02');

44235

SELECT TIMESTAMPDIFF(MONTH, '1898-03-22', '2019-05-02');

1453

SELECT TIMESTAMPDIFF(WEEK, '1898-03-22', '2019-05-02');

6319