### Introduction to SQL

[What is data and information](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12100&course_id=184&preview=on)

[What is Database](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12101&course_id=184&preview=on)

[What is Database Management System](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12102&course_id=184&preview=on)

[What is SQL?](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12103&course_id=184&preview=on)

[Assignment](https://pulse.itvedant.com/index.php/topic/update-preview?id=13117&course_id=184&preview=on)

### DDL & DML

[Data types, expressions, operators](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12104&course_id=184&preview=on)

[Data Definition Language(DDL)](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12105&course_id=184&preview=on)

[Creation of table](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12106&course_id=184&preview=on)

[Dropping a table](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12107&course_id=184&preview=on)

[INSERT statement](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12108&course_id=184&preview=on)

[UPDATE statement](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12109&course_id=184&preview=on)

[DELETE statement](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12110&course_id=184&preview=on)

[Assignment](https://pulse.itvedant.com/index.php/topic/update-preview?id=13156&course_id=184&preview=on)

### DQL

[SELECT statement](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12111&course_id=184&preview=on)

[WHERE clause search condition](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12112&course_id=184&preview=on)

[Arithmetic, Comparison and Logical operator](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12113&course_id=184&preview=on)

[Range operator](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12114&course_id=184&preview=on)

[Assignment](https://pulse.itvedant.com/index.php/topic/update-preview?id=13164&course_id=184&preview=on)

[List operator](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12115&course_id=184&preview=on)

[Like operator](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12116&course_id=184&preview=on)

[Using ORDER BY, DISTINCT and TOP](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12117&course_id=184&preview=on)

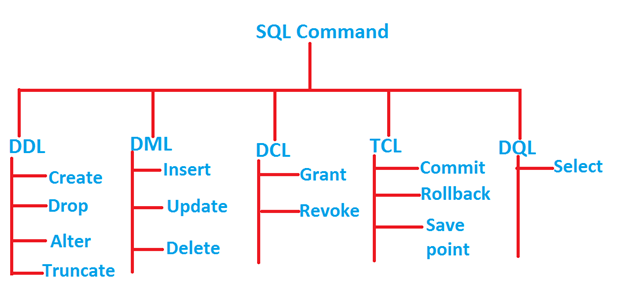
[Using IS NULL and IS NOT NULL](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12118&course_id=184&preview=on)

[Various other clauses](https://pulse.itvedant.com/index.php/topic/preview-subtopic-content?subtopic_id=12119&course_id=184&preview=on)

[Assignment](https://pulse.itvedant.com/index.php/topic/update-preview?id=13161&course_id=184&preview=on)

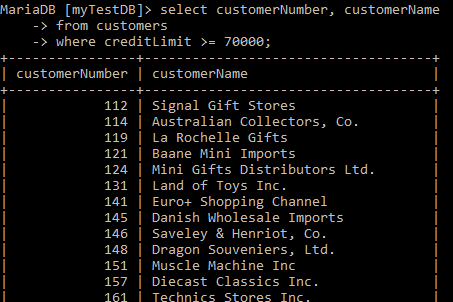
* **Filtering Data:** Here we combine the select statement with conditions to filter data as per requirement. We use WHERE clause to write the condition. Below is the general syntax of WHERE clause:

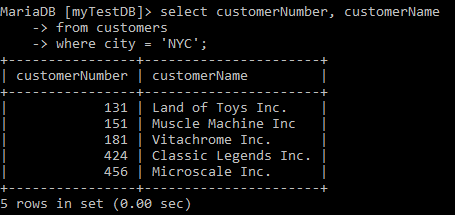
SELECT field1,field2,�FROM tablename WHERE condition



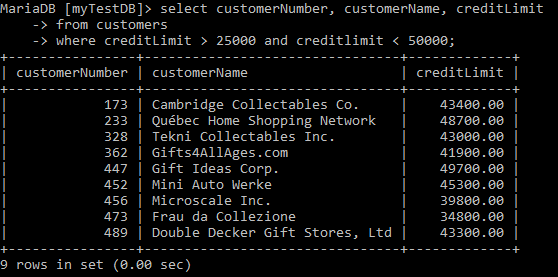
What is data and information What is Database What is Database Management System What is SQL? ALL basics

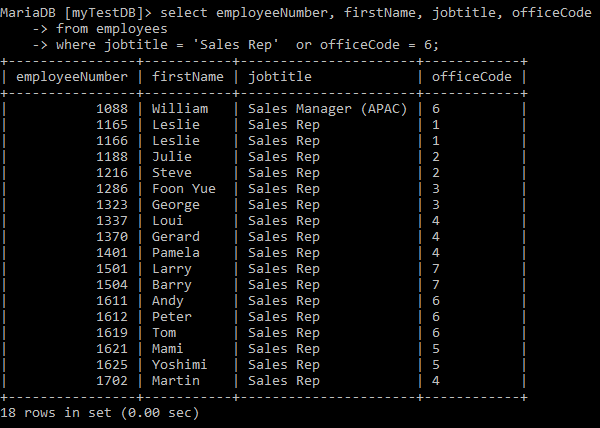
* **WHERE clause with Relational/ Comparison operators:** Following are the relational or comparison operators: greater than (>), less that (<), greater than equal to (>=), less than equal to (<=), equal to (=), not equal to (<> or !=)

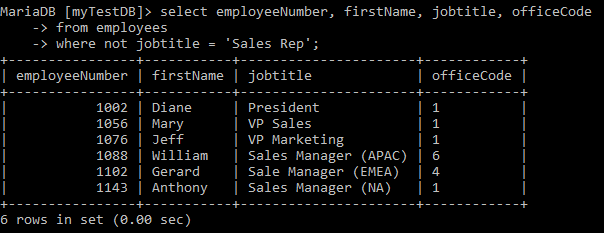




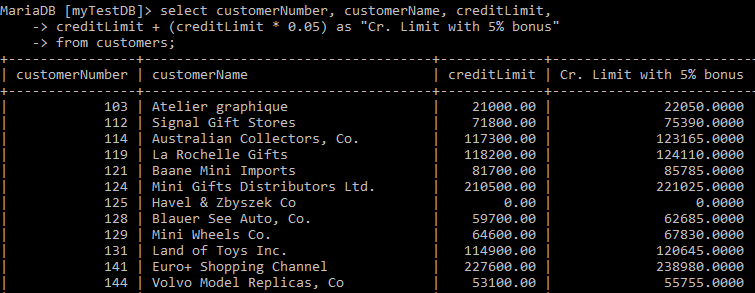
* **WHERE clause with Logical operators:** Following are the logical operators: **AND**: all conditions must evaluate to true **OR**: any one condition can evaluate to true **NOT**: negation.



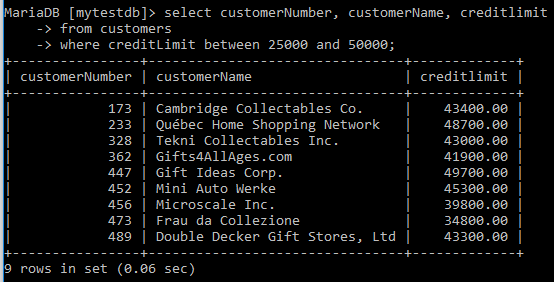




* **Selection with arithmetic operators:** Arithmetic operators (+, -, \*, /) can be used for deriving new values in the result set.

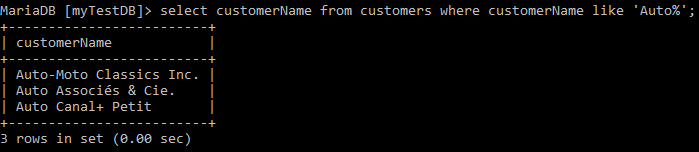


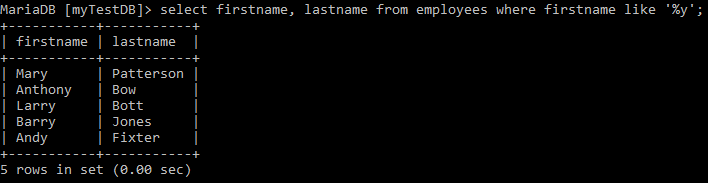
* **WHERE clause with range operator (BETWEEN):** It can be used for range of dates, numeric values. Similar to AND condition but for single field.

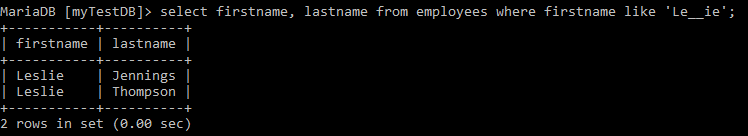


**SELECT with List operator (IN):** It can be used instead of OR condition for a single field and can be used with character or date values.

* **WHERE clause with LIKE operator:** It is used when we want to select rows to display that are similar to another field or constant. It is used with % (percentage) or \_ (underscore) for character data types. % represent many, \_ represent single character.

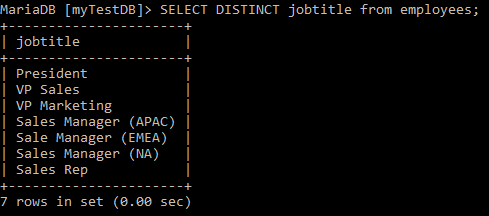






* **?Remove all duplicate values from the records:**

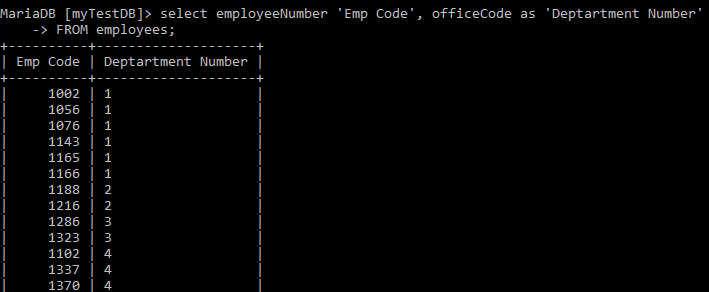
SELECT DISTINCT attribute\_name FROM table\_name;



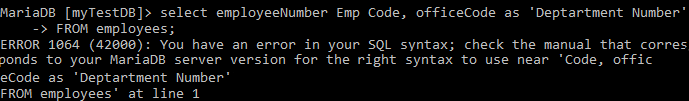
* **Renaming the attribute name:**

SELECT attribute\_name1 AS alias\_name, attribute\_name2 AS alias\_name �.. FROM table\_name;

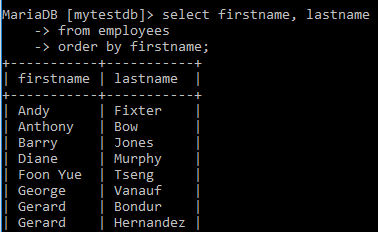
SELECT attribute\_name1 alias\_name, attribute\_name2 alias\_name �.. FROM table\_name;

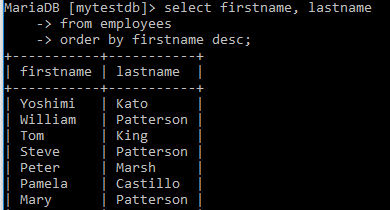


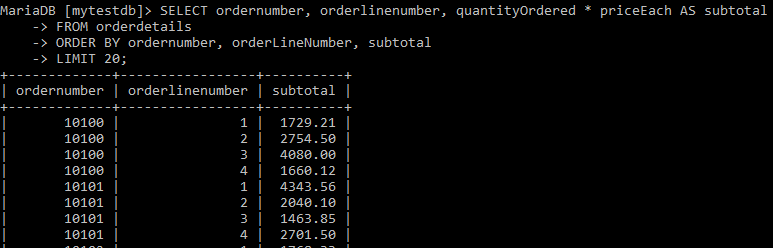
Please note if you are using a space in the alias name, you need to give the alias in single quote (�) or double quote (�) else the following error will be displayed.



* **Sorting Data:** Here we understand how to sort the result set using ORDER BY clause. The custom sort order with the FIELD function.

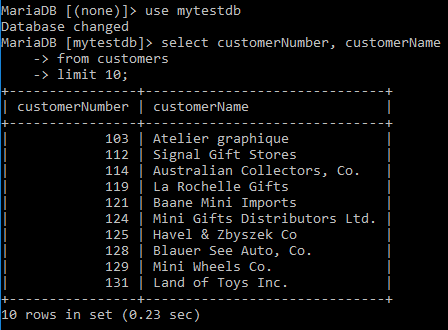




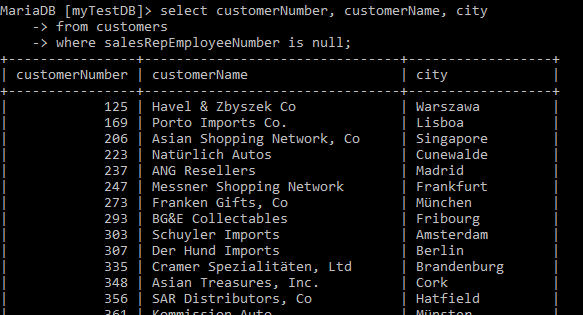


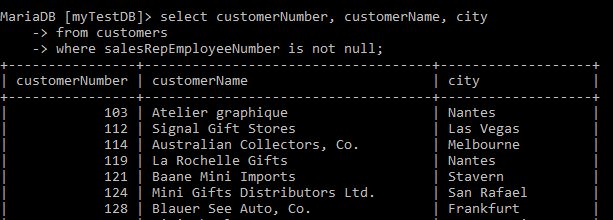
* **SELECT with List operator (IN):** It can be used instead of OR condition for single field and can be used with character or date values.

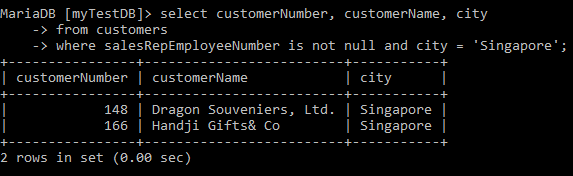
* **SELECT with LIMIT:** This is command is to restrict the number of records in the result set based on the requirement. This is similar to the TOP command in SQL Server.



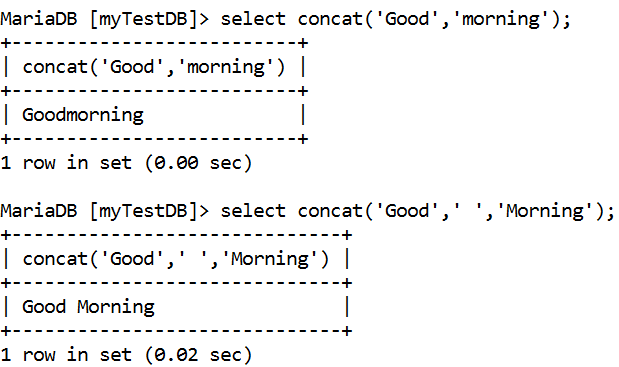
* **WHERE clause with IS NULL operator:** Null is not equal to zero or a blank space.



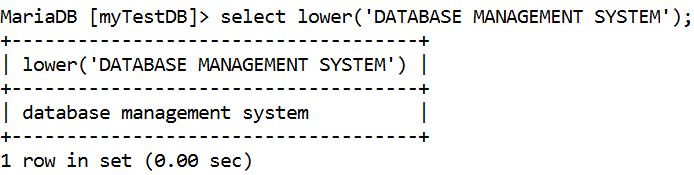




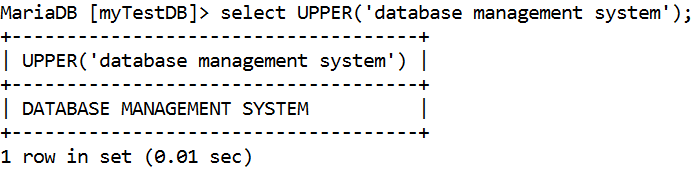
* **String Functions**: Return string values. Return values with varchar or char. If the length of the return values exceeds the limit, the return value is truncated without any error message.
  + **CONCAT(exp1, exp2, � , expn):** Concatenate two or more strings into one.



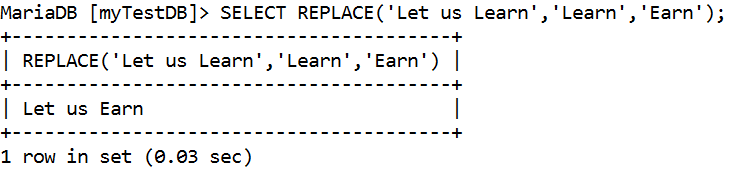
* **LOWER(expr):** converts all the characters to Lowercase



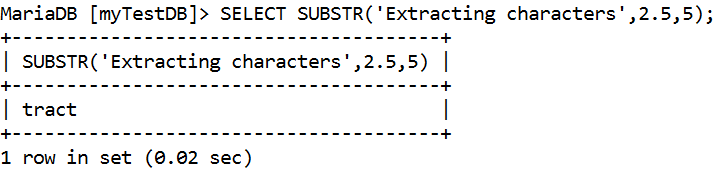
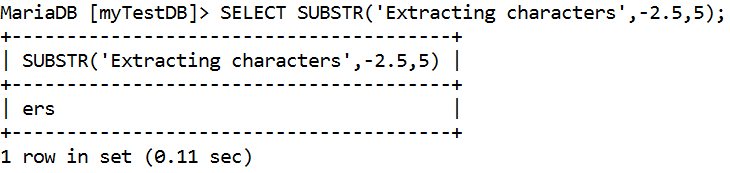
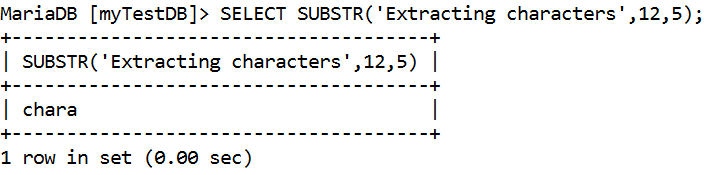
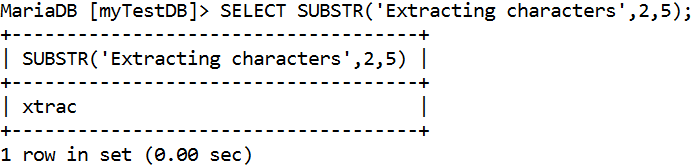
* **UPPER(expr):** converts all the characters to Uppercase



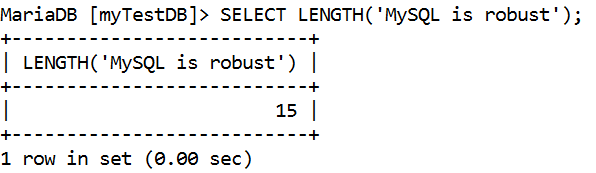
* **REPLACE(expr1,char to be replaced, with what):** Search and replace a substring in a string.



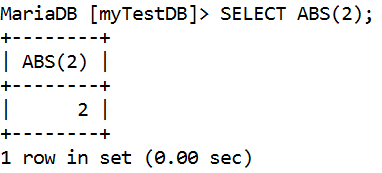
* **SUBSTR(expr1,m,n):** Extracts �n� number of characters from �mth� position. m can be positive or negative. If m is positive, extracting of characters starts from left hand side. If m is negative, extracting of characters starts from right hand side. If m is a floating point, it is converted to integer.

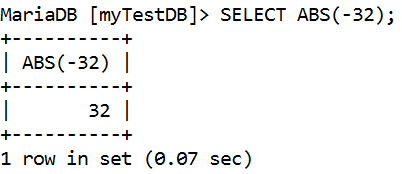


* **LENGTH(expr) and CHAR\_LENGTH(exp):** Returns the length of the string

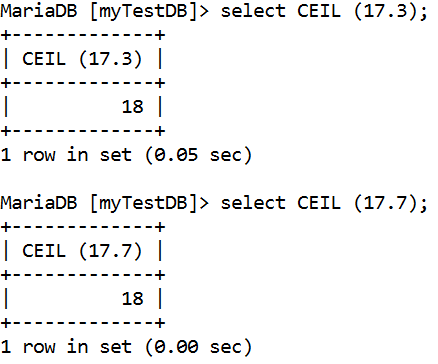


* **Math Functions:** Take numeric values and return numeric values. The values returned are accurate up to 38 decimal digits.
  + **ABS(X):** Returns the absolute value of X.

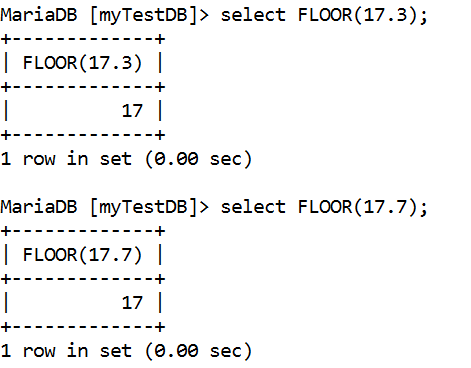




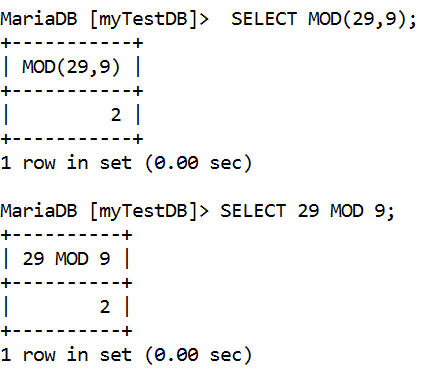
* **CEIL(X):** CEIL() is a synonym for CEILING()

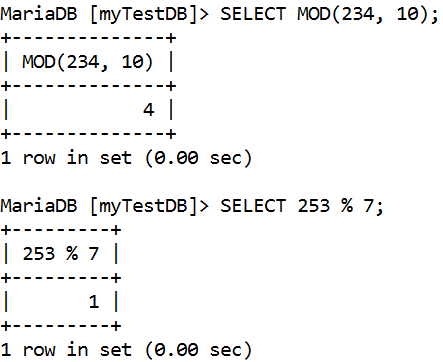


* **FLOOR(X):** Returns the largest integer value not greater than X**.**

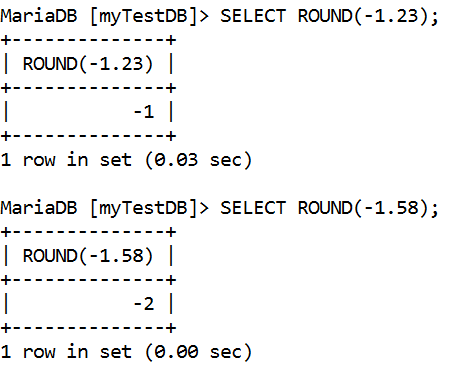


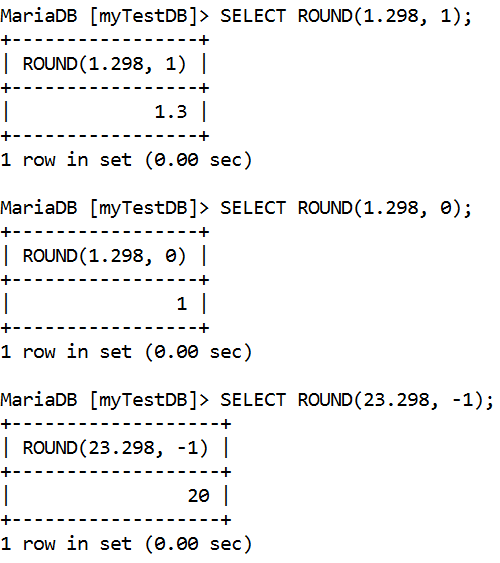
* **MOD(N,M), N % M, N MOD M:** Modulo operation. Returns the remainder of N divided by M.



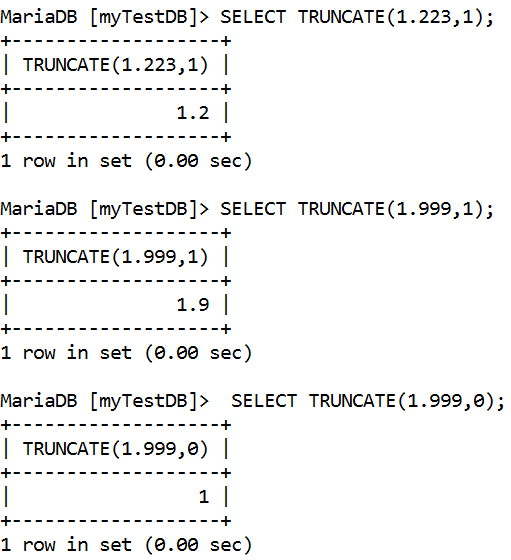


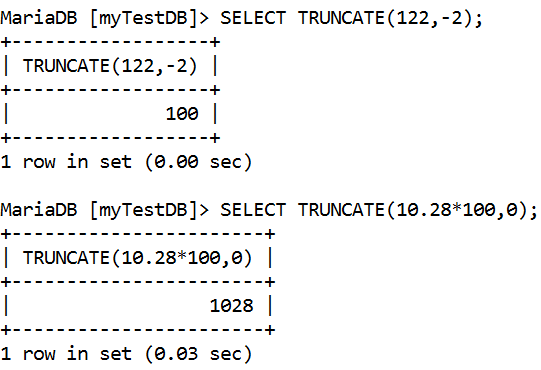
* **ROUND(X), ROUND(X,D):** Rounds the argument X to D decimal places. The rounding algorithm depends on the data type of X. D defaults to 0 if not specified. D can be negative to cause D digits left of the decimal point of the value X to become zero.

****

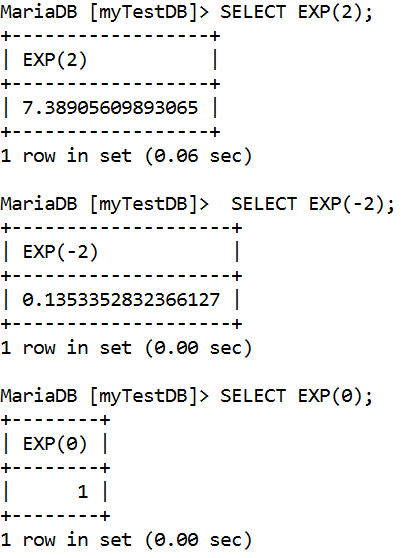
****

* **TRUNCATE(X,D):** Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value X to become zero.

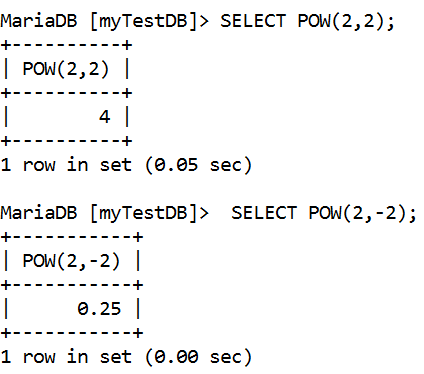




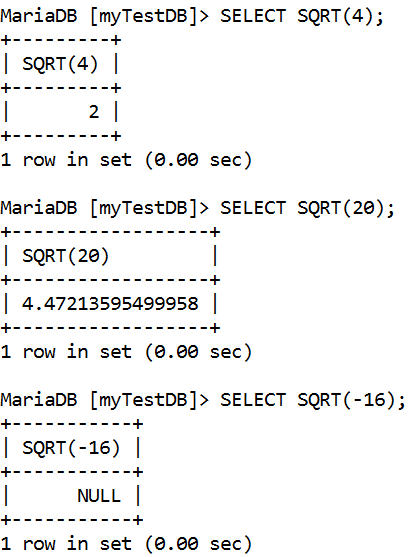
* **EXP(X):** Returns the value of e (the base of natural logarithms) raised to the power of X. The inverse of this function is LOG() (using a single argument only) or LN().



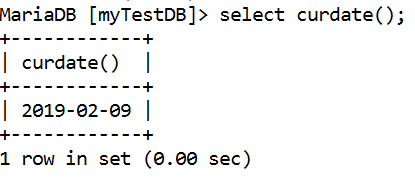
* **POW(X,Y) / POWER(X,Y):** Returns the value of X raised to the power of Y.



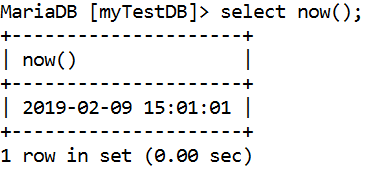
* **SQRT(X):** Returns the square root of a nonnegative number X.



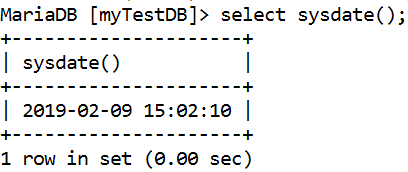
* **Date Functions:** 
  + **CURDATE()**: Returns the current date.

****

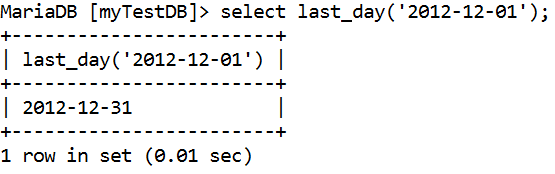
* **NOW()**: Returns the current date and time at which the statement executed.

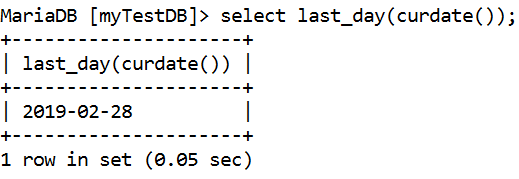


* **SYSDATE()**: Returns the current date.

****

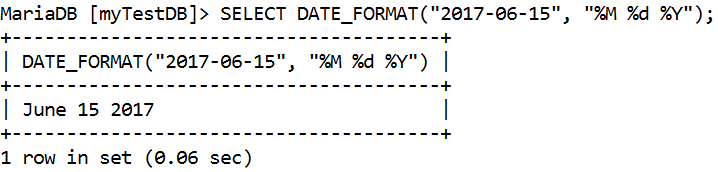
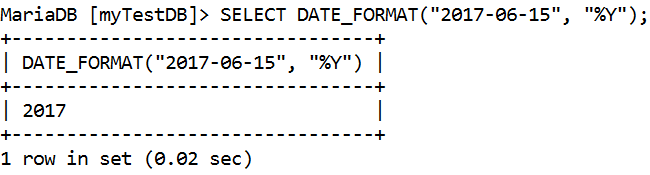
* **Last\_day(date):** returns the last day of the corresponding month for a date or datetime value**.**

****

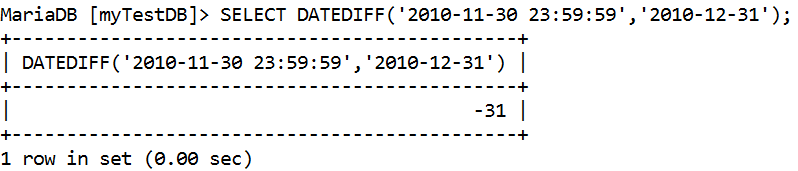
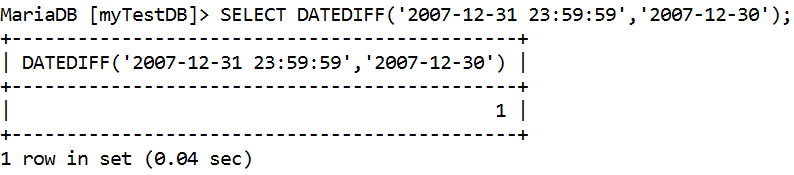
****

* Date\_format(date, format): To format a [date](http://www.mysqltutorial.org/mysql-date/) value to a specific format, you use the DATE\_FORMAT function.

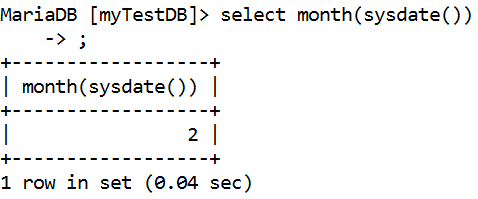
|  |  |
| --- | --- |
| **Format** | **Description** |
| %a | Abbreviated weekday name (Sun to Sat) |
| %b | Abbreviated month name (Jan to Dec) |
| %c | Numeric month name (0 to 12) |
| %D | Day of the month as a numeric value, followed by suffix (1st, 2nd, 3rd, ...) |
| %d | Day of the month as a numeric value (01 to 31) |
| %e | Day of the month as a numeric value (0 to 31) |
| %f | Microseconds (000000 to 999999) |
| %H | Hour (00 to 23) |
| %h | Hour (00 to 12) |
| %I | Hour (00 to 12) |
| %i | Minutes (00 to 59) |
| %j | Day of the year (001 to 366) |
| %k | Hour (0 to 23) |
| %l | Hour (1 to 12) |
| %M | Month name in full (January to December) |
| %m | Month name as a numeric value (00 to 12) |
| %p | AM or PM |
| %r | Time in 12 hour AM or PM format (hh:mm:ss AM/PM) |
| %S | Seconds (00 to 59) |
| %s | Seconds (00 to 59) |
| %T | Time in 24 hour format (hh:mm:ss) |
| %U | Week where Sunday is the first day of the week (00 to 53) |
| %u | Week where Monday is the first day of the week (00 to 53) |
| %V | Week where Sunday is the first day of the week (01 to 53). Used with %X |
| %v | Week where Monday is the first day of the week (01 to 53). Used with %X |
| %W | Weekday name in full (Sunday to Saturday) |
| %w | Day of the week where Sunday=0 and Saturday=6 |
| %X | Year for the week where Sunday is the first day of the week. Used with %V |
| %x | Year for the week where Monday is the first day of the week. Used with %V |
| %Y | Year as a numeric, 4-digit value |
| %y | Year as a numeric, 2-digit value |

****

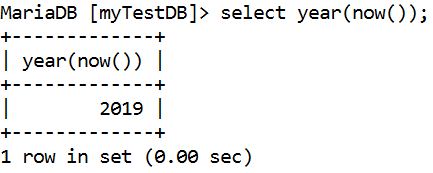
* **DATEDIFF(expr1,expr2):** returns expr1 - expr2 expressed as a value in days from one date to the other. expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.



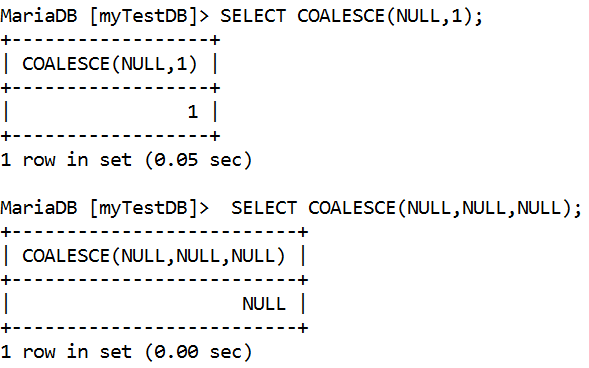
* **MONTH(date):** Returns the month for date, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.



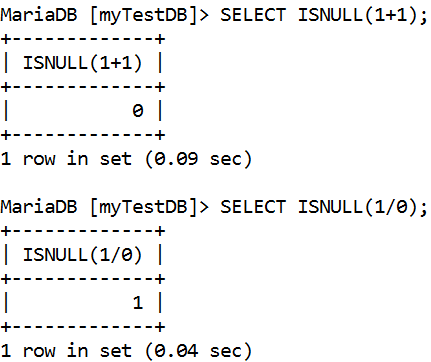
* **YEAR(date):** Returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.



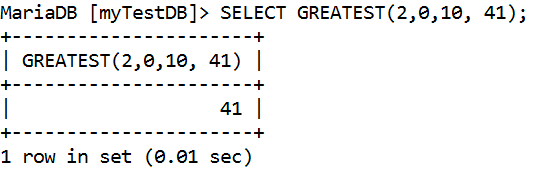
* **Comparison Functions:**
  + **COALESCE(value,...):** Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

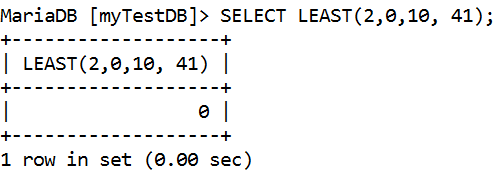


* **ISNULL(expr):** If expr is NULL, ISNULL() returns 1, otherwise it returns 0.

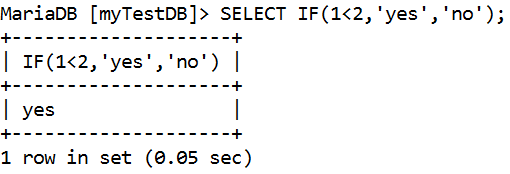


* **GREATEST(value1,value2,...) / LEAST(value1, value2,...):** With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST().

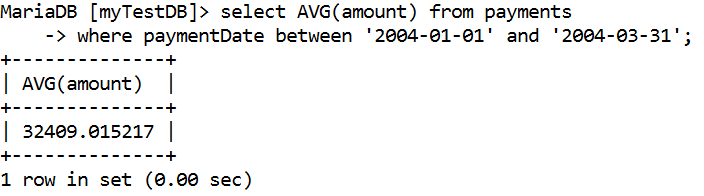




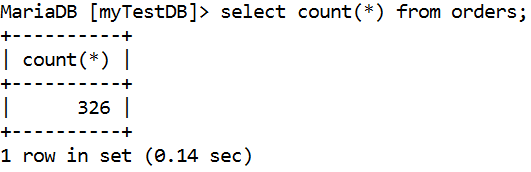
* **IF(expr1,expr2,expr3):** If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL) then IF() returns expr2; otherwise it returns expr3. IF() returns a numeric or string value, depending on the context in which it is used.



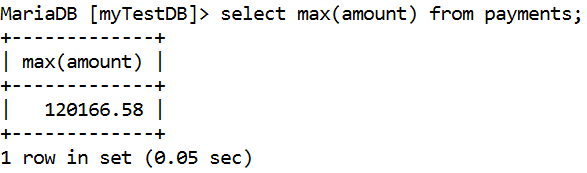
* **Aggregation functions:** The aggregation functions act on set of records or the entire table. Many aggregation functions accept the DISTINCT clause. Aggregation functions are also known as group functions.
  + **AVG([DISTINCT] expr):** Returns the average value of expr. The DISTINCT option can be used to return the average of the distinct values of expr**.**

****

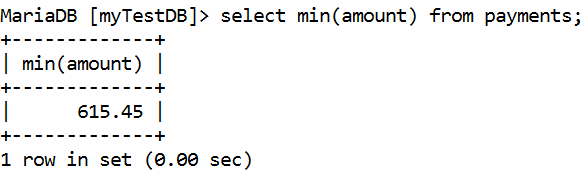
* **COUNT(expr):** Returns a count of the number of non-NULL values of expr in the rows retrieved by a SELECT statement. The result is a BIGINT value.



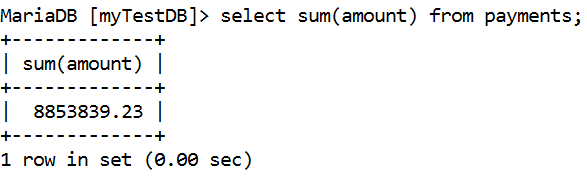
* **MAX([DISTINCT] expr):** Returns the maximum value of expr.



* **MIN([DISTINCT] expr):** Returns the minimum value of expr.



* **SUM([DISTINCT] expr):** Returns the sum of expr. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used to sum only the distinct values of expr.



* **Group by clause:** GROUP BY clause is used to group selected rows and return a single row of summary information based on that group. The group by clause can have following types of expressions:
  + Constants
  + Group functions
  + The function SYSDATE

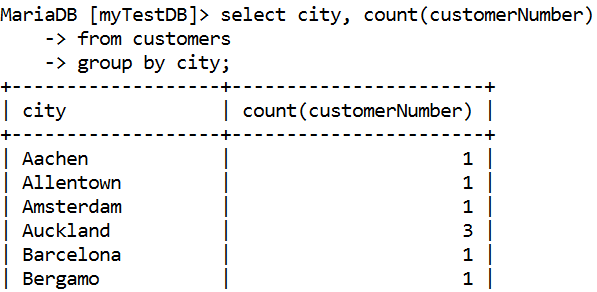
The group by clause can contain not more than 255 expressions.

SELECT fieldname on which you group by,

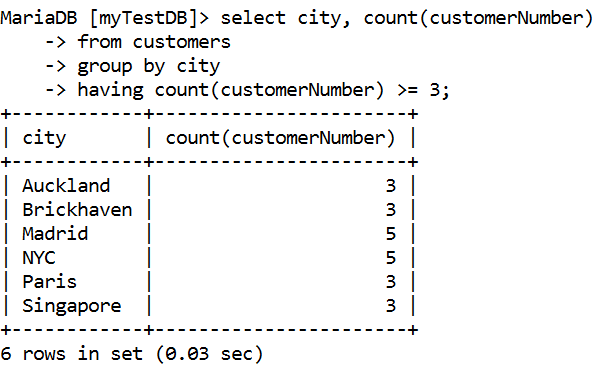
group function(fieldname),�.

FROM tablename WHERE condition

GROUP BY fieldname HAVING condition;



* **When to use Having clause:** 
  + When you need to use a aggregate function in a condition.
  + Eg : Display those departments having minimum salary is less than 5000
  + SELECT deptno, MIN(sal) FROM emp WHERE MIN(sal) < 5000 GROUP BY deptno;
  + Will give an error because where clause cannot have a Group function with GROUP BY clause.
  + SELECT deptno, MIN(salary) FROM emp GROUP BY deptno HAVING MIN(sal)<5000;



**Joins**

A join is a query that combines rows from two or more tables. Database performs a join whenever multiple tables appear in the FROM clause of the query. The select list of the query can select any columns from any of these tables. If any two of these tables have a column name in common, then you must qualify all references to these columns throughout the query with table names to avoid ambiguity.  This can be done by giving an alias name.

**Types of Joins:**

* **Equai Join:** An equijoin is a join which contains an equality operator.

SELECT tab1.fieldname, tab1.fieldname,

tab2.fieldname, tab2.fieldname,

FROM tab1, tab2 WHERE tab1.fieldname = tab2.fieldname;   
Alias can be used in place of table name

Example: SELECT employeeNumber,

CONCAT(firstName, ' ', lastName) AS "Full Name",  
o.officeCode, city FROM employees e, offices o  
WHERE e.officeCode = o.officeCode;

* **Self Join:** It is a join which relates the table to itself.  
  Example: Display the names of all employees who has manager.  
  SELECT Concat(e1.firstName, ' ', e1.lastName, ' works for ', e2.firstName, ' ', e2.lastName)

AS "Employee works for Manager"

FROM employees e1, employees e2

WHERE e1.reportsTo = e2.employeeNumber;

* **Non-equi join:** A join condition using any other operator than �=� is a non-equi join.

Example: Display customer details along with order details between 12-Jan-2003 to 31-Mar-2003.

SELECT c.customerName, o.orderNumber, o.orderDate,

od.quantityOrdered \* od.priceEach as "Total Cost"

FROM customers c, orders o, orderdetails od

WHERE c.customerNumber = o.customerNumber and

o.orderNumber = od.orderNumber and

o.orderDate between '2003-01-12' and '2003-03-31';

* **On Clause:** The ON clause is used to join tables where the column names don�t match in both tables. The join conditions are removed from the filter conditions in the WHERE clause. In other words, the ON clause is similar to the WHERE clause as you can put multiple conditions in the ON clause.  
  SELECT Concat(e.firstName, ' ', e.lastname) as "EmployeeName", c.customerName FROM employees e JOIN customers c  
  ON (e.employeeNumber = c.SalesRepEmployeeNumber);
* **Using Clause:** The columns listed in the USING clause must be present in both of the two tables being joined. The USING clause will be transformed to an ON clause that checks for equality between the named columns in the two tables.

SELECT concat(e.firstname, ' ', e.lastname), o.city

FROM employees e JOIN offices o USING (officeCode);

* **Cross Join:** This join produces a Cartesian product. CROSS JOIN returns the Cartesian product of the sets of rows from the joined tables. Cartesian product is a join of every row of one table to every row of another table. This normally happens when no matching join columns are specified.

SELECT CONCAT(e.firstName, ' ', e.lastname) AS "EmployeeName", c.customerName FROM employees e CROSS JOIN customers c;

* **Outer Join**: An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition. Such rows are not satisfied by a simple join.
  + **Left Outer Join**:

SELECT e.firstName, c.customerName

FROM employees e LEFT OUTER JOIN customers c

ON (e.employeeNumber=c.SalesRepEmployeeNumber);

* **Right Outer Join**:

SELECT e.firstName, c.customerName

FROM employees e RIGHT OUTER JOIN customers c

ON (e.employeeNumber=c.SalesRepEmployeeNumber);

**Subqueries** - A subquery is a SELECT statement that is embedded in a clause of another SELECT statement.

**When is Subquery Used?**

* Whenever you need a result of a particular query to get the necessary result set.
* Subquery is also known as nested queries.

SELECT field list FROM table

WHERE fieldname operator ( SELECT statement);  
**Types of Subqueries:**

* **Single row subqueries:** They are queries that return only one row from the inner select statement.

Example: When we want to find out the employees of a office in which GEORGE is working.

SELECT officeCode FROM employees

WHERE firstname = 'George';

Once it returns the office code (let us say 3) you would then give

SELECT firstName, lastName FROM employees

WHERE officeCode = 3;

This can be done using the subquery as follows:

SELECT firstName, lastName FROM employees

WHERE officeCode = (SELECT officeCode FROM employees

WHERE firstname = 'George');

* **Multiple row subqueries:** They are queries that return more than one row from the inner select statement.

Multiple row operator:

* **IN**: equal to any member in the list

Example: Display all the employees who are in same office as �Tom� or �Martin�.

SELECT firstName, lastName FROM employees

WHERE officeCode IN (SELECT officeCode FROM employees

WHERE firstName IN (�Tom�, �Martin�));

* **ANY**: compare to each value returned by the subquery

Any Operator:

* **< ANY** means less than the maximum.
* **> ANY** means more than the minimum
* **= ANY** is equal to IN

Example: Display all the products whose MSRP is less than any Motorcycles and are not Motorcycles.

SELECT productCode, productName, MSRP, productLine

FROM products

WHERE MSRP > ANY (SELECT MSRP FROM products

WHERE productLine = 'Planes')

AND productLine <> 'Planes';

SELECT productCode, productName, MSRP, productLine

FROM products

WHERE MSRP < ANY (SELECT MSRP FROM products

WHERE productLine = 'Planes')

AND productLine <> 'Planes';

* **ALL**: compare value to every value returned by the subquery.
* ALL operator:
  + **< ALL** means less than the minimum.
  + **> ALL** means more than maximum.

Example: Display all the productNames whose buyPrice is greater than the average buyPrice of all the categories.

SELECT productCode, productName, productLine, buyPrice

FROM products  
WHERE buyPrice > ALL (SELECT AVG(buyPrice) FROM   
products GROUP BY productLine);

Example: Display all the productNames whose buyPrice is less than the average buyPrice of all the categories.

SELECT productCode, productName, productLine, buyPrice

FROM products  
WHERE buyPrice < ALL (SELECT AVG(buyPrice) FROM   
products GROUP BY productLine);

**Guidelines for using a subquery:**

* Enclose subqueries in ().
* Place subqueries on the right side of the comparison operator.
* Do not add an ORDER BY clause to a subquery.
* Use single-row operators with single-row subqueries. (<,>,<=,>=,<>)
* Use multiple-row operators with multiple-row subqueries (IN, ANY, ALL).
* We can write a sub query with in a subquery.

**Multiple Column Subqueries:**

Main query compares to values from a multiple-row and multiple-column subquery. SELECT field1, field2,�.FROM table

WHERE (field1, field2,..) IN

(SELECT field, field,�FROM table WHERE condition);Example: Display all the employees with same job and office as Pamela.

SELECT  firstName, LastName FROm employees

WHERE jobtitle IN (SELECT jobTitle FROM employees

WHERE firstName = 'Pamela')

AND officeCode = (SELECT officeCode FROM employees

WHERE firstName = 'Pamela')

AND firstName <> 'Pamela';

OR

SELECT  firstName, LastName FROm employees

WHERE (jobTitle, officeCode) IN

(SELECT jobTitle, officeCode FROM employees

WHERE firstName = 'Pamela') AND firstName <> 'Pamela';

**Subqueries in DDL and DML**

Example of **DML with subquery**: Increase the credit limit for the customers by 5% whose Sales Representative is Larry.

UPDATE customers

SET creditLimit = creditLimit + (creditLimit \* 0.05)

WHERE salesRepEmployeeNumber =

(SELECT employeeNumber from employees

WHERE firstName = �Larry�);

Example of **DDL with subquery**: Create a duplicate employee table named NemEmp.

CREATE TABLE NewEmp AS SELECT employeeNumber,

CONCAT(firstName, ' ' , lastName) as fullname,

officeCode FROM employees;

**SEARCH CONDITIONS WHERE clause**

Where Clause:

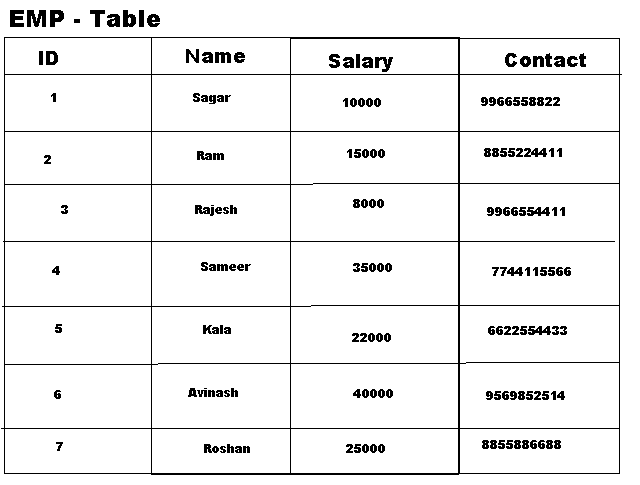
1) The  **Where**clause allows you to specify a search condition for the rows returned by a query.

2) Where Clause is use for to filter the record.

3)The WHERE clause is used to extract only those records that fulfill a specified condition.

Syntax:

select column\_list from tableName where serach\_condition;



**Filter Example:**

**1) Select \* from emp where id=3;**

**2) Select id,name,salary from emp where id=5 and name='Kala';**

**3)select \* from emp where id=5  or id=6 or id=4;**

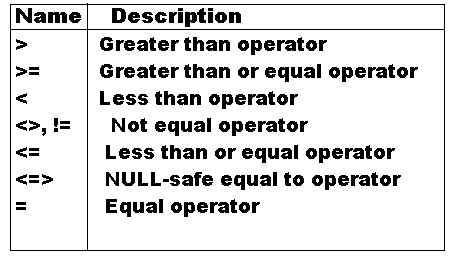
**4) Select id,name,contact,salary from emp where id between 1 and 5**

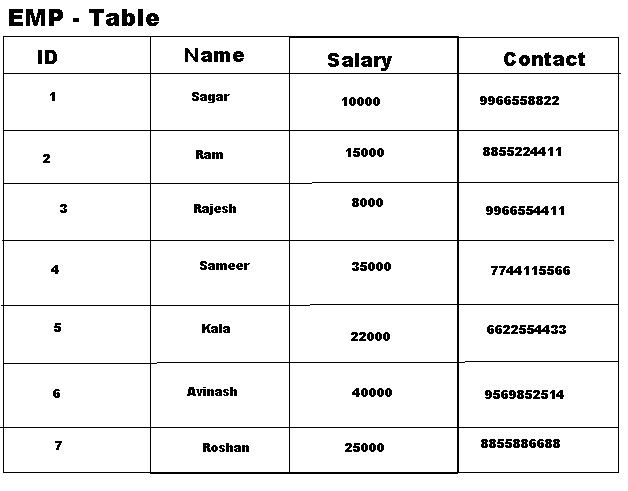
**5) select \* from emp where salary >20000;**

**6) select \* from emp where salary >20000 and salary <40000**

## Comparison operators :

are used in the WHERE clause to determine which records to select. Here is a list of the comparison operators that you can use in MySQL.





1) select \* from emp where id=1 or id=2

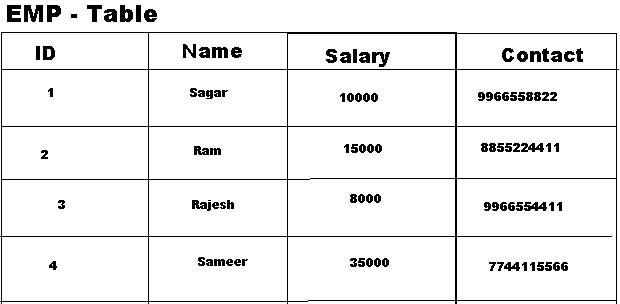
2) select \* from emp where id>1 and id<5

3)select \* from emp where salary>=1000 and  salary<=40000;

# **In Clause:**

1) The function returns 1 if expr is equal to any of the values in the IN list, otherwise, returns 0.

2) It is used to help reduce the need for multiple Or condition in a SELECT, INSERT, UPDATE, or DELETE statement.



1) Select 10 in (12,10,45);

2)select \* from emp where id in (1,2,3,6);

3)select \* from emp where name in("Sameer",'Raj','Rajesh','Kala');

4)select \* from emp where name not in("Sameer",'Raj','Rajesh','Kala');

5)select \* from emp where salary in(10000,8000,40000,22000,35000);