**MySQL datatypes**

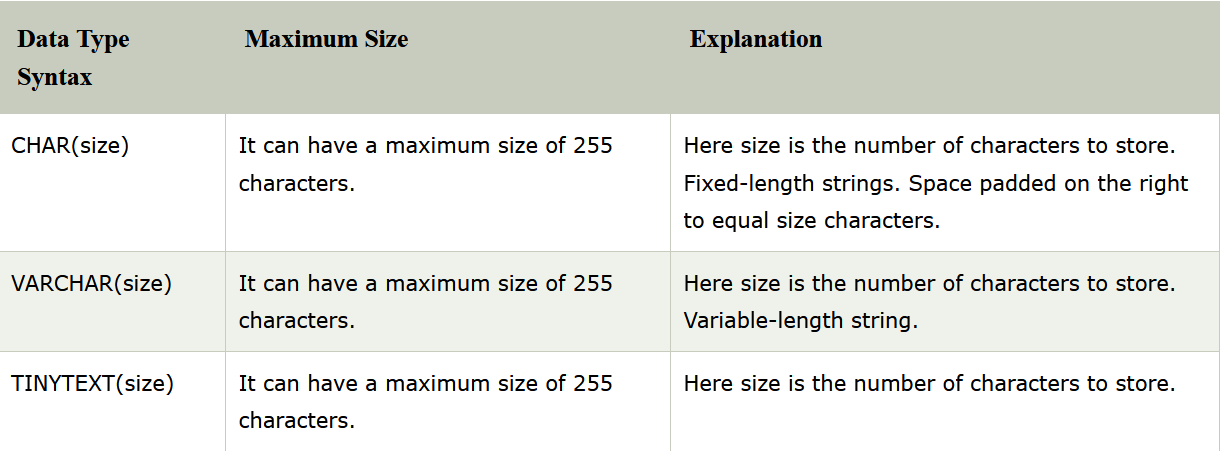
1. BIT
2. BOOLEAN
3. CHAR
4. DATE
5. DATETIME
6. DECIMAL
7. ENUM
8. INT
9. JSON
10. TIME
11. TIMESTAMP
12. VARCHAR

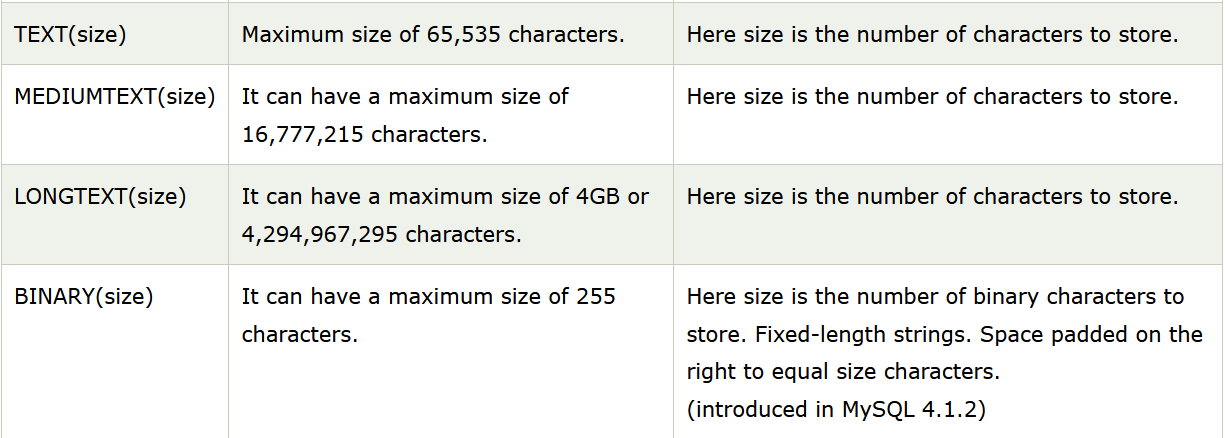
**Boolean data types**

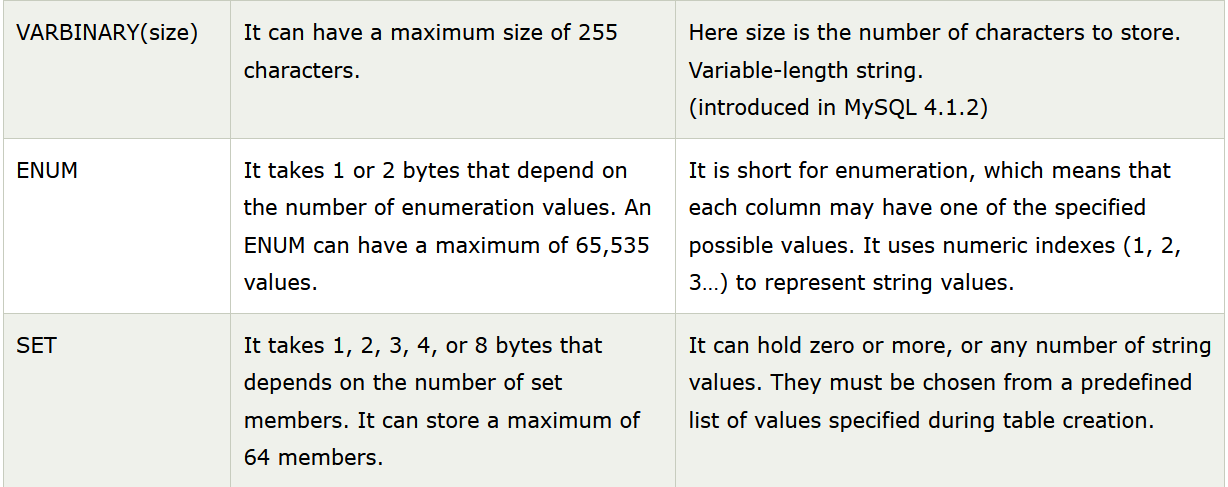
MySQL doesn’t have any specific Boolean datatypes. In MySQL 0 is considered as FALSE/false and non-zero values are considered as TRUE/true.

**String/CHAR/VARCHAR data types**

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







**VARCHAR vs TINYTEXT**

VARCHAR is ISO standard, TINYTEXT is non-standard. TINYTEXT can only have a default value of NULL, where VARCHAR can have any text as a default. TINYTEXT data is stored as a reference that needs to be fetched separately, which may add latency

**Comparing MySQL CHAR values**

* MySQL does not consider trailing spaces when comparing CHAR values using the comparison operator such as =, <>, >, <, etc
* Notice that the LIKE operator does consider the trailing spaces when you do pattern matching with CHAR values

When we use =, <>, >, <, etc operator for comparing or getting values then it check for exact character but if we use LIKE operator then it check for that character, donedn’t care about the trailing spaces.

**MySQL CHAR and UNIQUE index**

If the CHAR column has a UNIQUE index and you insert a value that is different from an existing value in a number of trailing spaces(i.e same value appending with some spaces at end), MySQL will reject the changes because of duplicate-key error.

**MYSQL VARCHAR**

MySQL VARCHAR is the variable-length string whose length can be up to 65,535. MySQL stores a VARCHAR value as a 1-byte or 2-byte length prefix plus actual data.

*If a column requires less than 255 bytes, the length prefix is 1 byte. In case the column requires more than 255 bytes, the length prefix is two length bytes*.

The maximum length (65535), however, is subject to maximum row size (65,535 bytes) and the character set used. It means that the total length of all columns should be less than 65,535 bytes.

**MySQL VARCHAR and spaces**

* MySQL does not pad space when it stores the VARCHAR values. Also, MySQL retains the trailing spaces when it inserts or selects VARCHAR values but trailing spaces will be truncated when string length is more that the specified maximum length.

Let say we have below table:

CREATE TABLE IF NOT EXISTS ajanta.string\_test (

s1 VARCHAR(3) NOT NULL

);

Now we are tying to insert a string of more that specified characters then we will get warnings.

INSERT INTO string\_test(s1)

VALUES('ABC ');

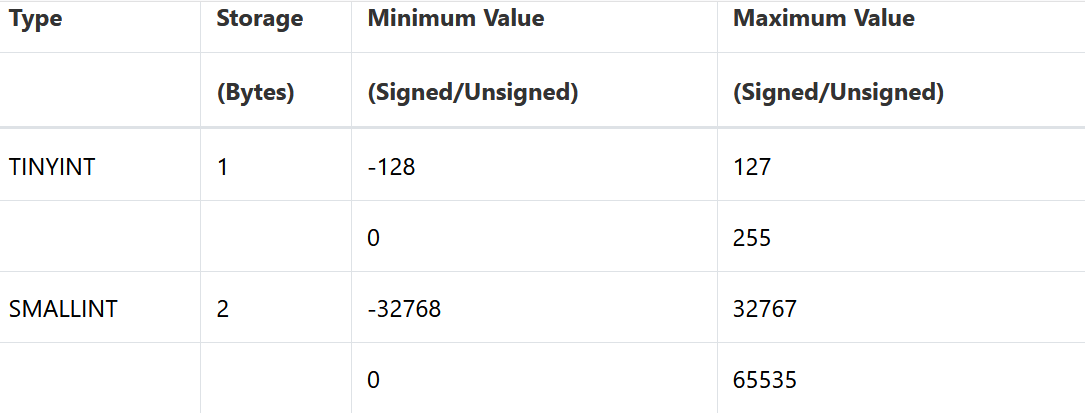
Warning

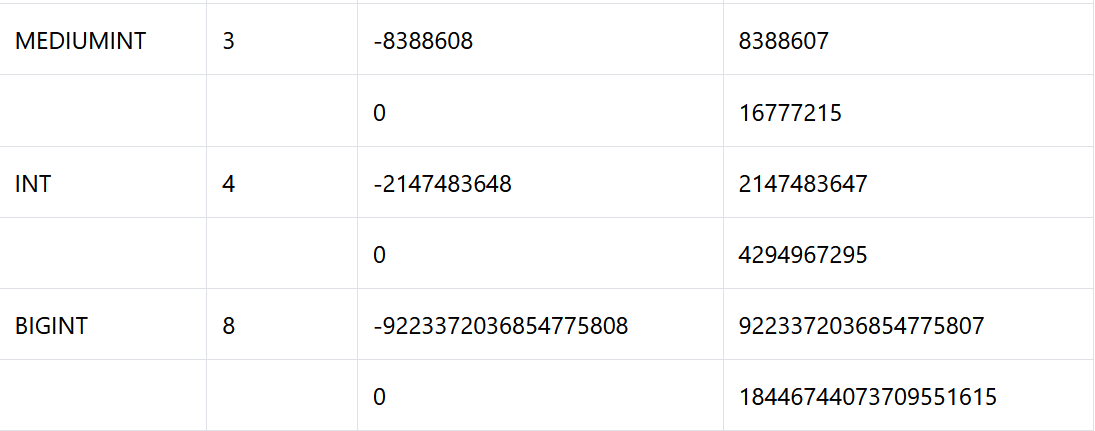
INSERT INTO string\_test(s1)

VALUES('ABC ') 1 row(s) affected, 1 warning(s):

1265 Data truncated for column 's1' at row 1 0.015 sec

**MySQL INT datatypes**





**MySQL DECIMAL data type**

The MySQL DECIMAL data type is used to store exact numeric values in the database. We often use the DECIMAL data type for columns that preserve exact precision

To define a column whose data type is DECIMAL you use the following syntax:

column\_name DECIMAL(P,D);

* P is the precision that represents the number of significant digits. The range of P is 1 to 65.
* D is the scale that that represents the number of digits after the decimal point. The range of D is 0 and 30. MySQL requires that D is less than or equal to (<=) P. default to 0.

**MySQL DATE data type**

MySQL DATE is one of the five temporal data types used for managing date values. MySQL uses yyyy-mm-dd format for storing a date value. This format is fixed and it is not possible to change it.

MySQL uses 3 bytes to store a DATE value. The DATE values range

MySQL Date values with two-digit years

Year values in the range 00-69 are converted to 2000-2069

Year values in the range 70-99 are converted to 1970 – 1999

**Some methods for date in MySQL.**

To get the current system date, you use CURDATE() function.

To get only date part of a DATETIME value, you use the DATE() function.

To format a date value, you use DATE\_FORMAT(date object, format) function (same as python).

**JSON datatypes:**

MySQL supports the native JSON data type since version 5.7.8. The native JSON data type allows you to store JSON documents more efficiently than the JSON text format in the previous versions.

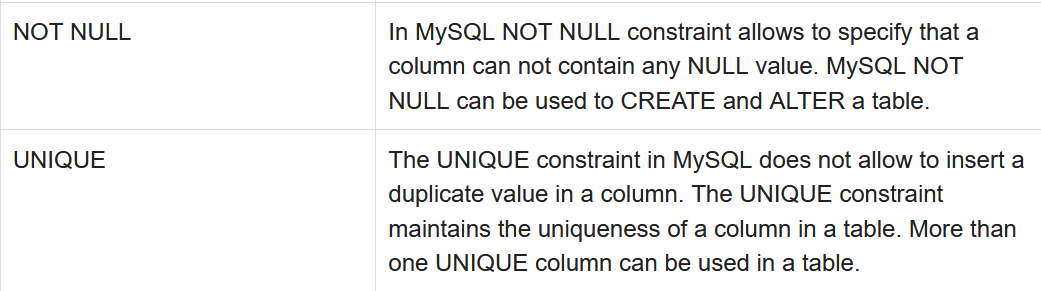
The storage of a JSON document is approximately the same as the storage of LONGBLOB or LONGTEXT data.

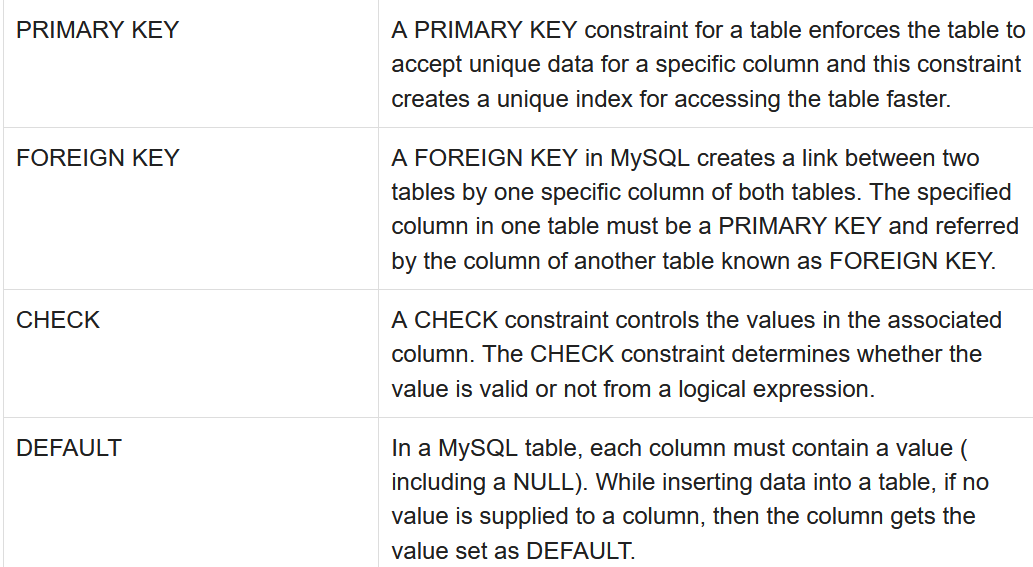
**MySQL constraints**

MySQL CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns.

MySQL CONSTRAINTS are used to limit the type of data that can be inserted into a table.

MySQL CONSTRAINTS can be classified into two types - column level and table level.

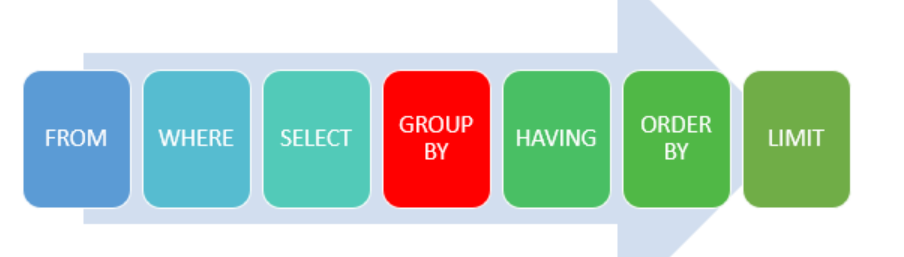




CREATE TABLE [table name]

([column name] [data type]([size]) [column constraint]….

[table constraint] ([[column name]……])……);



**ORDERBY**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

SELECT

select\_list

FROM

table\_name

ORDER BY

column1 [ASC|DESC],

column2 [ASC|DESC],

...;

**MySQL clauses**

There are generally five kinds of SQL Clauses in MySQL Server. They are listed as follows:

* WHERE Clause
* ORDER BY clause
* HAVING Clause
* LIMIT Clause
* GROUP BY Clause

**Group By**

The GROUP BY clause is used to group rows that have the same values in the result set.

**Order by**

The ORDER BY clause is used in SQL for sorting records

**Having clause**

This clause is introduced to apply functions in the query with the WHERE clause. In SQL, the HAVING clause was added because the WHERE clause could not be applied with aggregate functions.

**Limit clause**

The LIMIT clause is used in the SELECT statement to constrain the number of rows to return. The LIMIT clause accepts one or two arguments. The values of both arguments must be zero or positive integers.

SELECT

select\_list

FROM

table\_name

LIMIT row\_count;

row\_count----> maximum number of rows to return.

**LIKE operator**

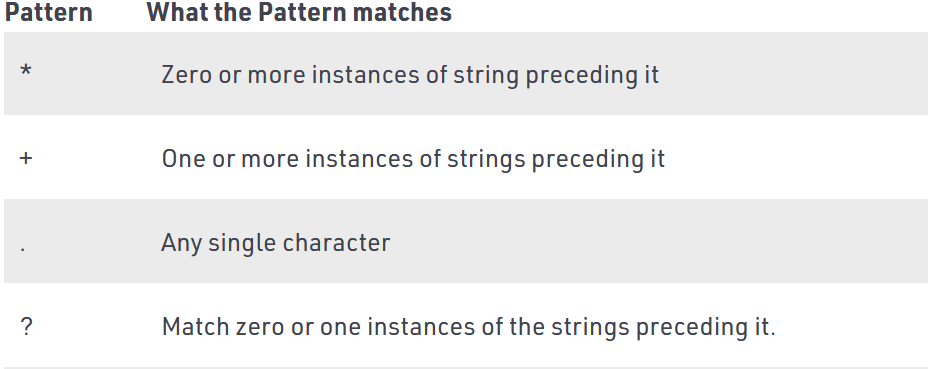
The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

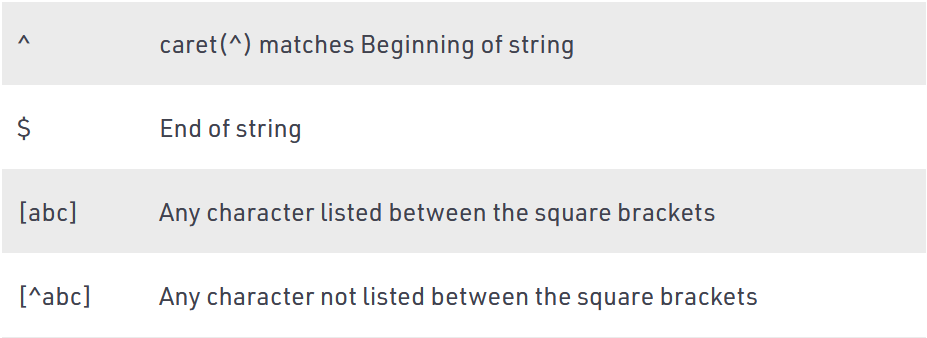
There are two wildcards often used in conjunction with the LIKE operator:

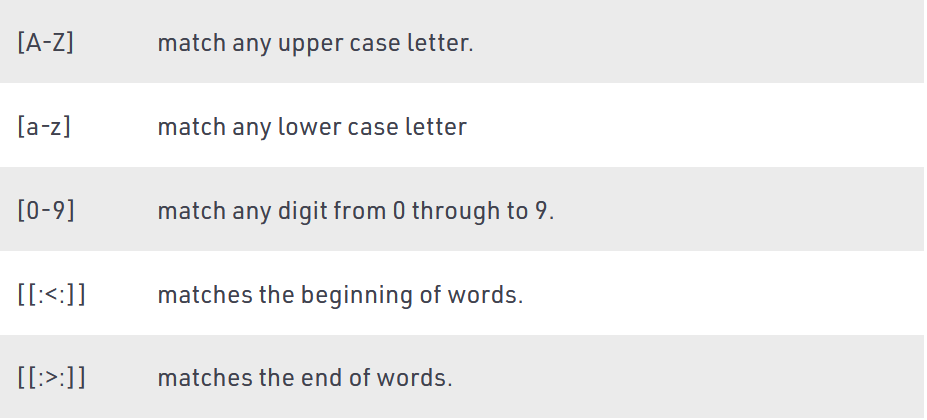
* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

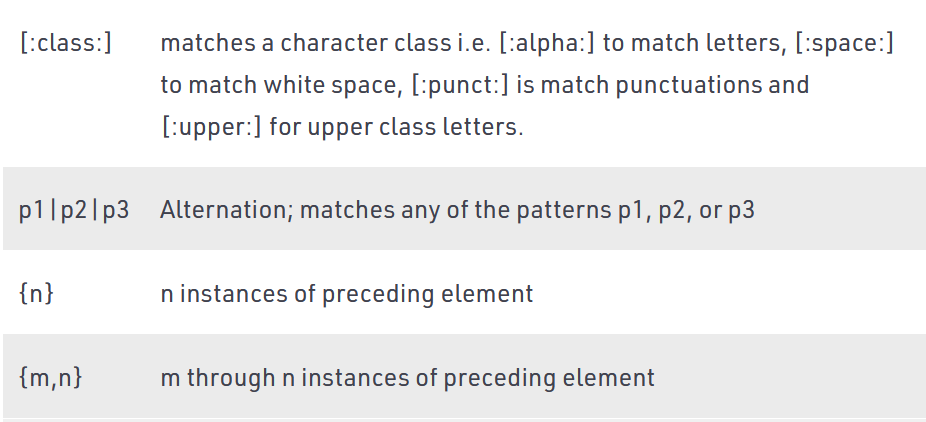
**MySQL Regular expressions (Regexp)**

MySQL supports another type of pattern matching operation based on the regular expressions and the REGEXP operator.









Examples:

**Match beginning of string(^):**

Gives all the names starting with ‘sa’.Example- sam,samarth.

SELECT name FROM student\_tbl WHERE name REGEXP '^sa';

**Match the end of a string($):**

Gives all the names ending with ‘on’.Example – norton,merton

*SELECT name FROM student\_tbl WHERE name REGEXP 'on$';*

**Match zero or one instance of the strings preceding it(?):**

Gives all the titles containing ‘com’.Example – comedy , romantic comedy.

*SELECT title FROM movies\_tbl WHERE title REGEXP 'com?';*

**matches any of the patterns p1, p2, or p3(p1|p2|p3):**

Gives all the names containing ‘be’ or ‘ae’.Example – Abel, Baer.

*SELECT name FROM student\_tbl WHERE name REGEXP 'be|ae' ;*

**Matches any character listed between the square brackets([abc]):**

Gives all the names containing ‘j’ or ‘z’.Example – Lorentz, Rajs.

SELECT name FROM student\_tbl WHERE name REGEXP '[jz]' ;

**Matches any lower case letter between ‘a’ to ‘z’- ([a-z]) ([a-z] and (.)):**

Retrieve all names that contain a letter in the range of ‘b’ and ‘g’, followed by any character, followed by the letter ‘a’.Example – Tobias, sewall.

**Matches any single character(.)**

SELECT name FROM student\_tbl WHERE name REGEXP '[b-g].[a]' ;

**Matches any character not listed between the square brackets.([^abc]):**

Gives all the names not containing ‘j’ or ‘z’. Example – nerton, sewall.

SELECT name FROM student\_tbl WHERE name REGEXP '[^jz]' ;

**Matches the end of words[[:>:]]:**

Gives all the titles ending with character “ack”. Example – Black.

SELECT title FROM movies\_tbl WHERE REGEXP 'ack[[:>:]]';

**Matches the beginning of words[[:<:]]:**

Gives all the titles starting with character “for”. Example – Forgetting Sarah Marshal.

SELECT title FROM movies\_tbl WHERE title REGEXP '[[:<:]]for';

**Matches a character class[:class:]:**

i.e [:lower:]- lowercase character ,[:digit:] – digit characters etc.

Gives all the titles containing alphabetic character only. Example – stranger things, Avengers.

SELECT title FROM movies\_tbl WHERE REGEXP '[:alpha:]' ;

**GROUP BY**

The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group. In other words, it reduces the number of rows in the result set.

We often use the GROUP BY clause with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT

*The GROUP BY clause must appear after the FROM and WHERE clauses*. Following the GROUP BY keywords is a list of comma-separated columns or expressions that you want to use as criteria to group rows.

SELECT

c1, c2,..., cn, aggregate\_function(ci)

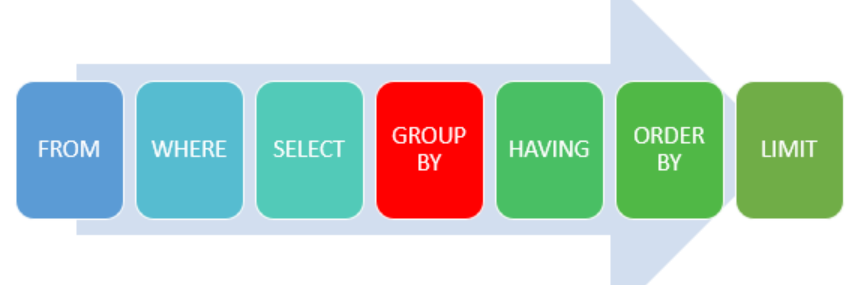
FROM

table

WHERE

where\_conditions

GROUP BY c1 , c2,...,cn;



MySQL evaluates the GROUP BY clause after the FROM, WHERE and SELECT clauses and before the HAVING , ORDER BY and LIMIT clauses.

**MySQL DISTINCT clause**

When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the DISTINCT clause in the SELECT statement.

SELECT DISTINCT

select\_list(s)

FROM

table\_name

**MySQL DISTINCT with multiple columns**

You can use the DISTINCT clause with more than one column. In this case, *MySQL uses the combination of values in these columns to determine the uniqueness of the row in the result set*

SELECT DISTINCT

state, city

FROM

customers

WHERE

state IS NOT NULL

ORDER BY

state,

city;

Above query unique combination of city and state from the customers table.

**DISTINCT clause vs. GROUP BY clause**

If you use the GROUP BY clause in the SELECT statement without using aggregate functions, the GROUP BY clause behaves like the DISTINCT clause.

MySQL Data Manipulation

AND

OR

IN

BETWEEN --- down

IS NULL

TABLE & COLUMN ALIASES ---------- use as keyword for aliasing table to column

jNOINS

INNER JOIN

LEFT JOIN

RIGHT JOING

SELF JOIN

CROSS JOIN

GROUP BY

HAVING

RULLUP

Subquery -------- a query nested within another query

Derived tables

EXIXTS

UNION

MINUES

INTERSECT

INSERT

Insert Multiple ROWS

INSER INTO SELECT

Insert On Duplicate Key update

INSERT IGNORE

UPDATE

UPDATE JOIN

DELETE

DELETE JOIN

ON DELETE CASCADE

REPLACE

**Between Operator**

The BETWEEN operator is a logical operator that allows you to specify whether a value in a range or not. The BETWEEN operator is often used in the WHERE clause of the SELECT, UPDATE, and DELETE statements.

expr [NOT] BETWEEN begin\_expr AND end\_expr; ---- syntax

begin\_expr -- this is starting point of range

end\_expr ---- this is end point of range

e.g—

SELECT

productCode,

productName,

buyPrice

FROM

products

WHERE

buyPrice < 20 OR buyPrice > 100;

**MySQL HAVING clause**

The HAVING clause is used in the SELECT statement to specify filter conditions for a group of rows or aggregates.

SELECT

select\_list

FROM

table\_name

WHERE

search\_condition

GROUP BY

group\_by\_expression

HAVING

group\_condition;

*MySQL evaluates the HAVING clause after the FROM, WHERE, SELECT and GROUP BY clauses and before ORDER BY, and LIMIT clauses.*

**MySQL update statement**

The UPDATE statement updates data in a table. It allows you to change the values in one or more columns of a single row or multiple rows.

UPDATE table\_name

SET

column\_name1 = expr1,

column\_name2 = expr2,

...

[WHERE

condition];

**Example 1: Updating one column data**

Update the email of a employee whose employye id is 1056.

update Employee

SET email="new email.com"

where employee\_id=1056

Example 2 : Multiple column update

Update the last\_name,first\_name of employee whose id if 1056.

update employee

SET first\_name="Mohan",last\_name="hare"

where employee\_id=1056

**MySQL REPLACE statement**

The MySQL REPLACE statement is an extension to the SQL Standard. The MySQL REPLACE statement works as follows:

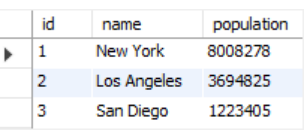
1.) Insert a new row into the table, if a duplicate key error occurs.

2.) If the insertion fails due to a duplicate-key error occurs:

Delete the conflicting row that causes the duplicate key error from the table.

Insert the new row into the table again.

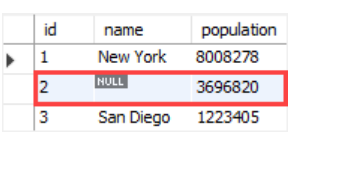
Let say we have below table



REPLACE INTO cities(id,population)

VALUES(2,3696820);

After replace



**MySQL DELETE statement**

To delete data from a table, you use the MySQL DELETE statement. The following illustrates the syntax of the DELETE statement:

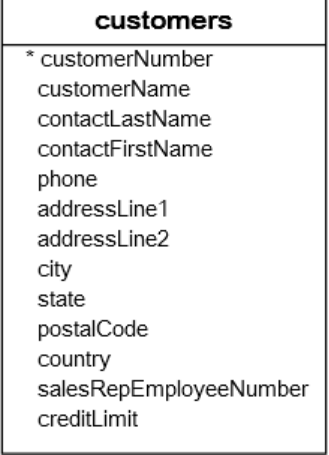
DELETE FROM table\_name

WHERE condition;

* WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table
* Besides deleting data from a table, the DELETE statement returns the number of deleted rows.
* *If don’t want to know how many of rows deleted without any condition then use TRUNCATE TABLE*

Questions

Delete customer from customer table whose country is france, serts them by credit limit in from low to high and delete first 5 customers



delete from customers

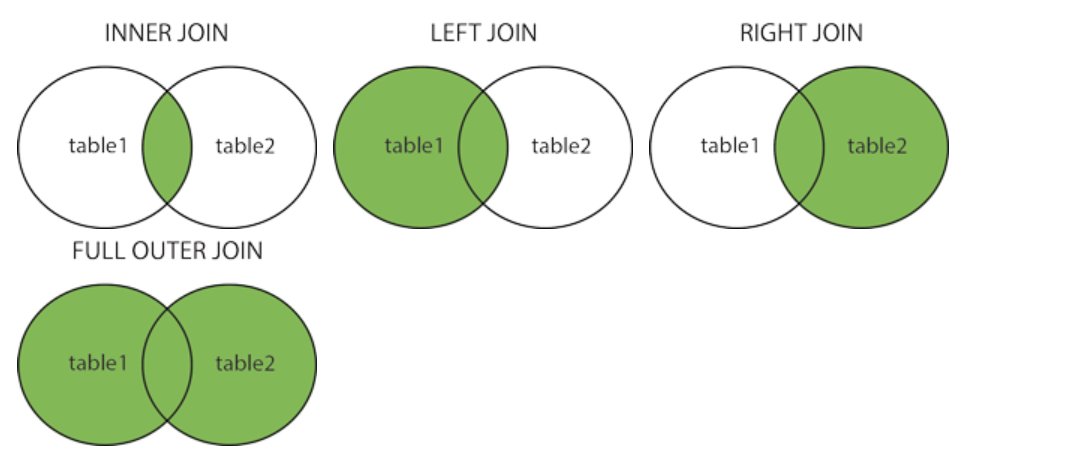
where country="france"

order by credit

limit 5

**JOINS**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.



**INNER join**

The INNER JOIN clause compares each row in the t1 table with every row in the t2 table based on the join condition.

SELECT

select\_list

FROM t1

INNER JOIN t2 ON join\_condition1

INNER JOIN t3 ON join\_condition2

...;

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;

e.g..

SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;\

**LEFT / LEFT OUTERjoin**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

*In some databases it is called as left outer join*

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**Example**

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

**RIGHT join**

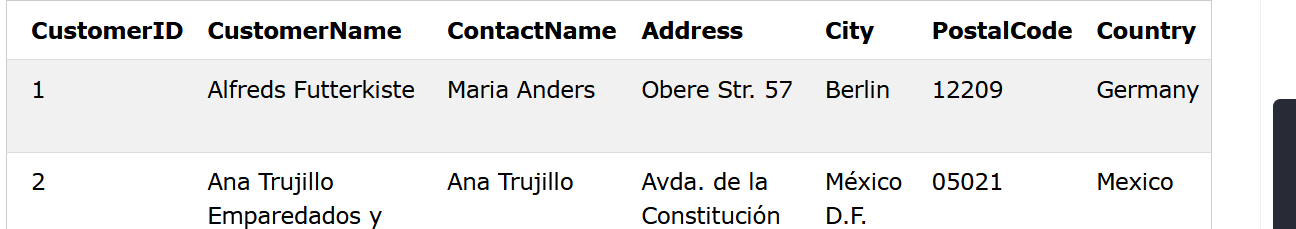
Same as left joins

**Self join**

A self join is a regular join, but the table is joined with itself.

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

Let say we have below table structure



**Question**:- Write a SQL query to find customer from same city.

selet A.customer\_name, B.customer\_name

from customer A , customer B

where A.customername<>B.customername

where A.city=B.city

**CROSS join**

The CROSS JOIN clause returns the Cartesian product of rows from the joined tables.

The result set will include all rows from both tables, where each row is the combination of the row in the first table with the row in the second table. In general, if each table has n and m rows respectively, the result set will have nxm rows.

SELECT \*/column(s) FROM t1

CROSS JOIN t2;

**INSERT INTO**

INSERT INTO table(c1,c2,...)

VALUES (v1,v2,...);

Inserting multiple rows

INSERT INTO table(c1,c2,...)

VALUES

(v11,v12,...),

(v21,v22,...),

...

(vnn,vn2,...);

**INSERT INTO SELECT**

Here instead of value statement we specifies values using select statement

INSERT INTO table\_name(column\_list)

SELECT

select\_list

FROM

another\_table

WHERE

condition;

**INSERT ON DUPLICATE KEY UPDATE**

INSERT INTO table (column\_list)

VALUES (value\_list)

ON DUPLICATE KEY UPDATE

c1 = v1,

c2 = v2,

...;

The only addition to the INSERT statement is the ON DUPLICATE KEY UPDATE clause where you specify a list of column-value-pair assignments in case of duplicate.

Basically, the statement first tries to insert a new row into the table. If a duplicate error occurs, it will update the existing row with the value specified in the ON DUPLICATE KEY UPDATE clause.