For B.E. B.C.A. M.C.A. M.C.M. B.S.C. Polythenic

MySQL®

by Aditya Choudhari Sir

CCIT

Keeping Pace with Technology

An ISO 9001 : 2008 Certified Company

website: www.ccitindia.com

Rajapeth: 0721-2563615 GadgeNagar: 0721-2552289

SQL

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, PostgreSQL and SQL Server use SQL as their standard database language.

SQL is a language specifically designed with databases in mind. SQL enables people to create databases, add new data to them, maintain the data in them, and retrieve selected parts of the data. Developed in the 1970s at IBM, SQL has grown and advanced over the years to become the industry standard. It is governed by a formal standard maintained by the International Standards Organization (ISO).

Various kinds of databases exist, each adhering to a different model of how the data in the database is organized.

SQL was originally developed to operate on data in databases that follow the Relational model. Recently, the international SQL standard has incorporated part of the object model, resulting in hybrid structures called object-relational databases. In this chapter, I discuss data storage, devote a section to how the relational model compares with other major models, and provide a look at the important features of relational databases. Before I talk about SQL, however, I want to nail down what I mean by the term database. Its meaning has changed, just as computers have changed the way people record and maintain information.

SQL

Stuctured Query Language

SQL is used in

MySQL

MySQL is the most popular Open Source Relational SQL Database Management System. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. This tutorial will give you a quick start to MySQL and make you comfortable with MySQL programming.

SQL Server

SQL Server is a relational database management system (RDBMS) developed and marketed by Microsoft. As a database server, the primary function of the SQL Server is to store and retrieve data used by other applications.

MS Access

Microsoft Access is a Database Management System offered by Microsoft. It uses the Microsoft Jet Database Engine and comes as a part of the Microsoft Office suite of application. Microsoft Access offers the functionality of a database and the programming capabilities to create easy to navigate screens (forms).

Oracle

Oracle database is a relational database management system. It is known as Oracle database, OracleDB or simply Oracle. It is produced and marketed by Oracle Corporation. Oracle database is the first database designed for enterprise grid computing. The enterprise grid computing provides the most flexible and cost effective way to manage information and applications.

PostgreSQL

PostgreSQL (pronounced as post-gress-Q-L) is an open source relational database management system (DBMS) developed by a worldwide team of volunteers. PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge.

SQLite

SQLite is a open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC, ODBC etc.

MySQL

MySQL is owned by MySQL AB. The "AB" is an acronym for the Swedish word "aktiebolag" or "stock company," which translates to the English (US) term "incorporated." What began as a capital venture to build an open source relational database system has become a credible alternative to the commercial database system market. MySQL was created by a Swedish company, MySQL AB, founded by David Axmark, Allan Larsson and Michael "Monty" Widenius. The first version of MySQL appeared on 23 May 1995. It was initially created for personal usage from mSQL based on the low-level language ISAM, which the creators considered too slow and inflexible. They created a new SQL interface, while keeping the same API as mSQL.

MySQL is a relational database management system designed for use in client/server architectures. MySQL can also be used as an embedded database library. Of course, if you have used MySQL before, you are familiar with its capabilities and no doubt have decided to choose MySQL for some or all of your database needs.

MySQL has become the worlds most popular and most successful open source database system. This popularity is due in large part to its reliability, performance, and ease of use. There are over 8 million installations of MySQL products worldwide. MySQL AB's success can be attributed to a sound core values statement: "To make superior data management software available and affordable to all." This core values statement is manifested by MySQL AB's key business objectives—to make its database system products

- The world best and most widely used
- Easy to use
- Continuously improved while maintaining speed and data integrity
- Fun and easy to extend and evolve
- Free from defects

Database

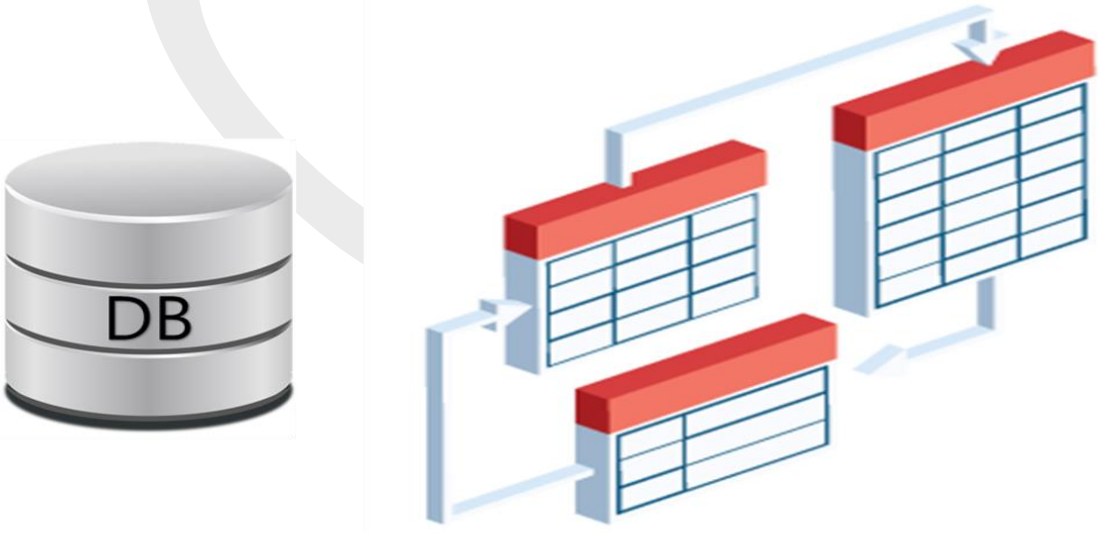
The term database has fallen into loose use lately, losing much of its original meaning. To some people, a database is any collection of data items (phone books, laundry lists, parchment scrolls . . . whatever). Other people define the term more strictly.

In this book, I define a database as a self-describing collection of integrated records. And yes, that does imply computer technology, complete with programming languages such as SQL.

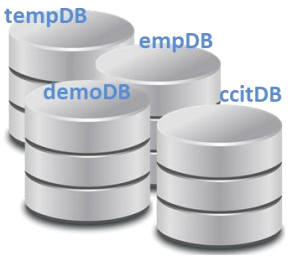
A record is a representation of some physical or conceptual object. Say, for example, that you want to keep track of a business' s customers. You assign a record for each customer. Each record has multiple attributes, such as name, address, and telephone number. Individual names, addresses, and so on are the data.

A database consists of both data and metadata. Metadata is the data that describes the data' s structure within a database. If you know how your data is arranged, then you can retrieve it. Because the database contains a description of its own structure, it' s self-describing. The database is integrated because it includes not only data items but also the relationships among data items.

The database stores metadata in an area called the data dictionary, which describes the tables, columns, indexes, constraints, and other items that make up the database.



Database Commands



Creating Database

Syntax : **CREATE DATABASE <database_name>;**

For example : create database ccitdb **;**

```
|----------------|
| Databases      |
|----------------|
| tempDB         |
| empDB          |
| demoDB         |
| ccitDB         |
|----------------|
```

Showing Databases

Syntax: **SHOW DATABASES;**

Selecting database to work with

Syntax: **USE <database_name>;**

For example : use ccitdb ;

Removing Database

Syntax : **DROP DATABASE <database_name>;**

For example : drop database empDB ;

To see all the tables in the Database.

Syntax : **SHOW TABLES;**

```
|----------------|
| Tables         |
|----------------|
| Person         |
| empolyee       |
| Students       |
| products       |
|----------------|
```

To see table's field formats.

Syntax : **DESCRIBE <table_name>;**

For example : describe students ;

```
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| Emp_no    | int(11)     | NO   | PRI | NULL    | auto_increment |
| Name      | varchar(100)| NO   |     | NULL    |                |
| Email     | varchar(254)| NO   |     | NULL    |                |
| Join_Date | date        | NO   |     | NULL    |                |
| job_id    | int(11)     | NO   | MUL | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Database Table

A database table looks a lot like a spreadsheet table: a two-dimensional array made up of rows and columns. You can create a table by using the SQL CREATE TABLE command. Within the command, you specify the name and data type of each column. After you create a table, you can start loading it with data. (Loading data is a DML, not a DDL, function.)

- The database table is where all the data in a database is stored.
- Each table is made up of rows and columns.
- Each row represents a record.
- Each row in a table is uniquely identified by a primary key. This can be by one or more sets of column values. In most scenarios it is a single column, such as ID.

Creating Tables

A table can be created by specifying information about all columns of table and rules (constraints).

Syntax:

```
CREATE TABLE  <table_name>
        (
            colname datatype[(size)] [column_Constraint],
              ..........................................,
              ..........................................,
            [table Constraint]
         ) ;
```

For example:

```
Create table students
 (.
 id integer primary key ,
 name varchar(30),
 gender char(1) ,
 branch varchar(10)
);
```

```
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| id        | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| gender    | varchar(1)  | YES  |     | NULL    |
| branch    | varchar(10) | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

Data Type

MySQL supports SQL data types in several categories: numeric types, date and time types, string (character and byte) types, spatial types, and the JSON data type.

String

CHAR(size)

A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

VARCHAR(size)

A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.

TEXT

A field with a maximum length of 65,535 characters.

TINYTEXT(size)

Maximum size of 255 characters.

MEDIUMTEXT(size)

Maximum size of 16,777,215 characters.

LONGTEXT

A field with a maximum length of 4,294,967,295 characters.

Binary Data

BLOB

(Binary Large OBjects). max 65,535 bytes of data.

LONGBLOB

(Binary Large OBjects). max 4,294,967,295 bytes of data.

Special Types

ENUM

It is a list of Items. When defining an ENUM, you are creating a list of items from which the value must be selected For ex: gender ENUM('M' , 'F')

JSON

It is used to stored data in JSON (JavaScript Object Notation) documents.

Numeric

NUMERIC data can have a fractional component in addition to its integer component. You can specify both the precision and the scale of NUMERIC data. (Precision, remember, is the maximum number of significant digits possible.)

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION).

INT(size)

A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.

BIGINT(size)

A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits..

TINYINT(size)

A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.

SMALLINT(size)

A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.

MEDIUMINT(size)

A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.

FLOAT(size,d)

A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D).

DOUBLE(size,d)

A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

DECIMAL(size,d)

An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

Date and Time Types

The MySQL date and time datatypes are as follows

DATE

A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.

DATETIME

A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

TIME

Stores the time in a HH:MM:SS format.

YEAR(M)

Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

For example:

Create table emp

Empno numeric

Name text

Job text

Salary numeric

JoinDate date

```
Create Table emp
(
empno integer primary key,
 name varchar (30) ,
 job varchar (10) ,
 salary integer ,
joindate date
) ;
```

```
+----------+-------------+------+-----+---------+
| Field    | Type        | Null | Key | Default |
+----------+-------------+------+-----+---------+
| empno    | int(11)     | NO   | PRI | NULL    |
| name     | varchar(30) | YES  |     | NULL    |
| job      | varchar(10) | YES  |     | NULL    |
| salary   | int(11)     | YES  |     | NULL    |
| joindate | date        | YES  |     | NULL    |
+----------+-------------+------+-----+---------+
```

For example:

Create table person

fName text

lName text

Gender (m/f)

hobbies text

Birthdate date

Photo image

```
Create Table person
(
fname varchar (30),
lname varchar (30),
gender enum('m','f'),
hobbies text(500) ,
birthdate date ,
photo BLOB(50000)
) ;
```

```
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| fname     | varchar(30) | NO   | PRI | NULL    |
| lname     | varchar(30) | YES  |     | NULL    |
| gender    | enum(m,f)   | YES  |     | NULL    |
| hobbies   | text        | YES  |     | NULL    |
| birthdate | date        | YES  |     | NULL    |
| photo     | BLOB(50000) | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

INSERT INTO

To insert data into a MySQL table, you would need to use the SQL INSERT INTO command. You can insert data into the MySQL table. MySQL INSERT statement is used to insert data in MySQL table within the database. We can insert single or multiple records using a single query in MySQL.

Syntax:

```
INSERT INTO table_name  VALUES (value1, value2, ...)
```

To Insert All Values of a Row

For example:

```
insert into students values (2351,'Raj Agrawal' ,'M','CS' ) ;
```

```
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
```

To Insert Values into selected columns of a Row

Note : columns whose values are not specified are set as NULL

Syntax:

```
INSERT INTO tname(col1, col2,...) VALUES (val1, val2, ...) ;
```

For example:

```
insert into students(name, id, gender) values( 'Raj Agrawal ' , 2351
,'M') ;
```

```
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |NULL    |
+-------+-------------+-------+--------+
```

To Insert Values from another table

Syntax:

```
INSERT INTO tname(col1, col2,...)  select statement ;
```

For example:

```
insert into studlist(sid , sname ) select id, name from students where
branch='CS' ;
```

```
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+

+-------+-------------+---------+
| sid   | SName       | Joindate|
+-------+-------------+---------+
| 4117  | Amit Jain   | NULL    |
| 3012  | Mona Mantri | NULL    |
| 2351  | Raj Agrawal | NULL    |
+-------+-------------+---------+
```

Update Statement

MySQL UPDATE statement is used to update data of the MySQL table within the database. In real life scenario, records are changed over the period of time. So, we need to make changes in the values of the tables also. To do so, we need to use the UPDATE statement.

The UPDATE statement is used with the SET, and WHERE clauses. The SET clause is used to change the values of the specified column. We can update single or multiple columns at a time. The WHERE clause is used to specify the condition, but it is optional.

Syntax:

```
UPDATE table_name
        SET column1 = value1, column2 = value2....
        [WHERE <condition>];
```

To update single record

For example:

```
Update employee set job='SrClerk' , Salary=25400 where empno=4117 ;
```

```
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 4117  | Amit Jain   | Clerk     | 18500  |
| 6543  | Mona Mantri | Clerk     | 18500  |
| 5332  | Raj Agrawal | Programmer| 31900  |
+-------+-------------+-----------+--------+
After update:
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 4117  | Amit Jain   | SrClerk   | 25400  |
| 6543  | Mona Mantri | Clerk     | 18500  |
| 5332  | Raj Agrawal | Programmer| 31900  |
+-------+-------------+-----------+--------+
```

To update multiple records

For example:

```
Update employee set  Salary=Salary+500 where job='Clerk' ;
```

```
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 4117  | Amit Jain   | Clerk     | 18500  |
| 6543  | Mona Mantri | Clerk     | 18500  |
| 5332  | Raj Agrawal | Programmer| 31900  |
| 6124  | Amar Rathi  | Clerk     | 18500  |
+-------+-------------+-----------+--------+
After update:
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 4117  | Amit Jain   | Clerk     | 19000  |
| 6543  | Mona Mantri | Clerk     | 19000  |
| 5332  | Raj Agrawal | Programmer| 31900  |
| 6124  | Amar Rathi  | Clerk     | 19000  |
+-------+-------------+-----------+--------+
```

To update All records

For example:

```
Update employee set  Salary=Salary+500;
```

```
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 54700  |
| 4117  | Amit Jain   | Clerk    | 18500  |
| 6543  | Mona Mantri | Clerk    | 18500  |
+-------+-------------+----------+--------+
After update:
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 55200  |
| 4117  | Amit Jain   | Clerk    | 19000  |
| 6543  | Mona Mantri | Clerk    | 19000  |
+-------+-------------+----------+--------+
```

Delete Statement

MySQL DELETE statement is used to delete data from the MySQL table within the database. By using delete statement, we can delete records on the basis of conditions.

Syntax:

```
DELETE from tname [WHERE <condition>];
```

To delete single record

For example:

```
Delete from employee  where empno=4117 ;
```

```
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 4117  | Amit Jain   | Clerk     | 18500  |
| 6543  | Mona Mantri | Clerk     | 18500  |
| 5332  | Raj Agrawal | Programmer| 31900  |
| 6124  | Amar Rathi  | Clerk     | 18500  |
+-------+-------------+-----------+--------+
After delete:
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 54700  |
| 6543  | Mona Mantri | Clerk     | 19000  |
| 5332  | Raj Agrawal | Programmer| 31900  |
| 6124  | Amar Rathi  | Clerk     | 19000  |
+-------+-------------+-----------+--------+
```

To delete multiple records

For example:

```
delete from employee where job='Clerk';
```

```
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 54700  |
| 4117  | Amit Jain   | Clerk    | 18500  |
| 6543  | Mona Mantri | Clerk    | 18500  |
| 5332  | Raj Agrawal | Progrmmer| 31900  |
| 6124  | Amar Rathi  | Clerk    | 18500  |
+-------+-------------+----------+--------+
After delete:
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 54700  |
| 5332  | Raj Agrawal | Programmer| 31900 |
+-------+-------------+----------+--------+
```

To delete All records

For example:

```
Delete from employee ;
```

```
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 54700  |
| 4117  | Amit Jain   | Clerk    | 18500  |
| 6543  | Mona Mantri | Clerk    | 18500  |
| 5332  | Raj Agrawal | Programmer| 31900 |
| 6124  | Amar Rathi  | Clerk    | 18500  |
+-------+-------------+----------+--------+
After delete:
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
```

Select Statement

The SELECT statement allows you to read data from one or more tables. It is used to fetch data from the MySQL da You can fetch one or more fields in a single SELECT command.

- You can specify star (*) to retrieve all columns.
- WHERE condition is used to filter records.
- GROUP BY is used to put together records that have the same field values.
- HAVING condition is used to specify criteria when working using the GROUP BY keyword.
- ORDER BY is used to specify the sort order of the result set.
- OFFSET indicates from where SELECT will start returning records.
- LIMIT indicates the number of records.tabase.

Syntax:

```
SELECT column1, column2...
        FROM table1, table2,..
        [ WHERE < conditions > ]
        [ GROUP BY column1, column2.. ]
        [ HAVING < conditions > ]
        [ ORDER BY column1, column2.. ]
        [ OFFSET <m> ]
        [ LIMIT <n> ];
```

SQL Arithmetic Operators

Operator	Description
+	Addition of two operands
-	Subtraction of right operand from the left operand
*	Multiply
/	Divide
%	Modulo

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

SQL Logical Operators

Operator	Description
AND	TRUE if all the conditional expressions separated by AND is TRUE.
BETWEEN	TRUE if the operand is inside the series of comparisons.
IN	TRUE if the operand matches one of a list of expressions.
LIKE	TRUE if the operand equals a pattern.
NOT	Shows a record if the condition(s) is FALSE.
OR	TRUE if any of the conditions separated by OR is valid
IS NULL	It compares a value with a NULL value

For example:

Find list of students

```
mysql> select * from students;

+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
```

For example:

Find name of students

```
mysql> select * from students;
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+

mysql> select name from students;
+-------------+
| Name        |
+-------------+
| Amit Jain   |
| Gopal Pandey|
| Mona Mantri |
| Raj Agrawal |
+-------------+
```

For example:

Find list of male students

```
mysql> select * from students;
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+

mysql> Select * from students where gender='M';
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
```

For example:

Find list of male students from branch cs

```
mysql> select * from students;
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+

mysql> Select * from students where gender='M';
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
```

For example:

Find list of male students from branch cs.

```
mysql> select * from students;
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 5555  | Gopal Pandey| M     |IT      |
| 3012  | Mona Mantri | F     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
mysql> Select * from students where gender='M';
+-------+-------------+-------+--------+
| ID    | Name        | Gender| Branch |
+-------+-------------+-------+--------+
| 4117  | Amit Jain   | M     |CS      |
| 2351  | Raj Agrawal | M     |CS      |
+-------+-------------+-------+--------+
```

BETWEEN

The BETWEEN operator is used to search for values that are within given the minimum value and the maximum value.

Syntax:

SELECT <column>,... FROM <table_name> colname BETWEEN minval and maxval;

For example:

Find list of students whose fees is in range 2000 to 3000

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
mysql> select * from students  where fees >=2000 and fees<= 3000
mysql> select * from students  where fees between 2000 and 3000
    For example:
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
+-------+-------------+-------+--------+----------+
```

For example:

Find list of students who have joined in year 2017

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
mysql> select * from students  where joindate between '2017-1-1' and '2017-12-
31';
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
```

IN

The IN operator is used to compare a value to a list of values that have been specified.

Syntax:

```
SELECT <column>,... FROM <table_name> column IN (value1,value2,. . . );
```

For example:

Find list of students who have joined for c++ or java course

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
mysql> select * from students where or course=‘C++’ or course=‘java’;
                      -----------or-----------
mysql> select * from students where course in ('C++','Java');
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
+-------+-------------+-------+--------+----------+
```

IS NULL

The IS NULL operator is used to compare a value with a NULL value.

Syntax:

```
SELECT <column>,... FROM <table_name> column IS NULL;
```

For example:

Find list of students whose course is not confirmed

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| NULL  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
mysql> select * from students where course is null ;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 5555  | Gopal Pandey| NULL  |2500    |25-2-2017 |
+-------+-------------+-------+--------+----------+
```

For example:

Find list of students whose course is confirmed

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 5555  | Gopal Pandey| NULL  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
mysql> select * from students where course is not null ;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Amit Jain   | C++   |2000    |22-5-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
```

LIKE

The LIKE operator is used to compare a value to similar values using wildcard operators i.e. for pattern matching.

Where:

- **%** any sequence of chars
- **_** any one char

Syntax:

```
SELECT <column>,... FROM <table_name> column LIKE <value>;
```

For example:

```
mysql> select * from students;
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Raj Jain    | C++   |2000    |22-5-2017 |
| 5555  | Rohit Pandey| NULL  |2500    |25-2-2017 |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |
| 2351  | Raja Dubey  |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+

mysql> select * from students where name like 'R%';
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Raj Jain    | C++   |2000    |22-5-2017 |
| 5555  | Rohit Pandey| NULL  |2500    |25-2-2017 |
| 2351  | Raja Dubey  |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+

mysql> select * from students where name like '_a%'
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 4117  | Raj Jain    | C++   |2000    |22-5-2017 |
| 2351  | Raja Dubey  |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+

mysql> select * from students where name like '%ey'
+-------+-------------+-------+--------+----------+
| ID    | Name        | Course| Fees   |JoinDate  |
+-------+-------------+-------+--------+----------+
| 5555  | Rohit Pandey| NULL  |2500    |25-2-2017 |
| 2351  | Raja Dubey  |Asp.net|3500    |15-7-2017 |
+-------+-------------+-------+--------+----------+
```

DISTINCT

The DISTINCT operator searches every row of a specified table for uniqueness (no duplicates).

Note: it is used in columns list.

Syntax:

```
SELECT DISTINCT <column>, <column>... FROM <table_name>;
```

For example:

List cities from where students have joined ccit

```
mysql> select * from students;
+-------+------------+-------+--------+----------+-----------+
| ID    | Name       | Course| Fees   |JoinDate  |city       |
+-------+------------+-------+--------+----------+-----------+
| 4117  | Amit Jain  | C++   |2000    |22-5-2017 |Amravati   |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |Akola      |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |Amravati   |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |Pune       |
+-------+------------+-------+--------+----------+-----------+
mysql> select DISTINCT city from students;
+----------+
| City     |
+----------+
| Amravati |
| Akola    |
| Pune     |
+----------+
```

For example:

List cities from where students have joined ccit for different courses.

```
mysql> select * from students;
+-------+------------+-------+--------+----------+-----------+
| ID    | Name       | Course| Fees   |JoinDate  |city       |
+-------+------------+-------+--------+----------+-----------+
| 4117  | Amit Jain  | C++   |2000    |22-5-2017 |Amravati   |
| 5555  | Gopal Pandey| Java  |2500    |25-2-2017 |Akola      |
| 3012  | Mona Mantri | C++   |2000    |11-6-2018 |Amravati   |
| 2351  | Raj Agrawal |Asp.net|3500    |15-7-2017 |Pune       |
+-------+------------+-------+--------+----------+-----------+
mysql> select DISTINCT course , city from students;;
+---------+----------+
|Course   | City     |
+---------+----------+
|C++      | Amravati |
|Java     | Akola    |
|Asp.net  | Pune     |
+---------+----------+
```

Aggregate Functions

An aggregate function performs a calculation on a set of values and returns a single value. An aggregate function performs a calculation on multiple values and returns a single value like the sum of all values, maximum and minimum among certain groups of values.

All Group functions ignore NULL values.

You can use the IFNULL function to force group functions to include NULL values

SUM(exp)	Returns the total sum
MIN(exp)	Returns the lowest value
MAX(exp)	Returns the highest value
AVG(exp)	Returns the average value
COUNT(*)	Returns the number of records in a table
COUNT(column)	Returns the number of values (NULL values will not be counted)
COUNT(DISTINCT column)	Returns the number of distinct values

For example:

```
mysql> select * from employee;
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 50000  |
| 4117  | Amit Jain   | Clerk    | 20000  |
| 6543  | Raj Rathi   | NULL     | 30000  |
| 5332  | Mona Mantri | Progrmmer| 40000  |
| 6124  | Raj Agrawal | Clerk    | 20000  |
+-------+-------------+----------+--------+

mysql> Select sum(salary) from employee;
+------------+
| sum(salary)|
+------------+
| 160000     |
+------------+
```

```
mysql> Select avg(salary) from employee ;
+------------+
| sum(salary)|
+------------+
| 32000.0000 |
+------------+

mysql> select max(salary) from employee;
+------------+
| max(salary)|
+------------+
| 50000      |
+------------+

mysql> Select min(salary) from employee ;
+------------+
| min(salary)|
+------------+
| 20000      |
+------------+

mysql> Select count(*) from employee ;
+------------+
| count(*)   |
+------------+
| 5          |
+------------+

mysql> Select count(job) from employee ;
+------------+
| count(job) |
+------------+
| 4          |
+------------+

mysql> Select count(distinct job) from employee ;
+--------------------+
| count(distinct job)|
+--------------------+
| 3                  |
+--------------------+
```

Group By Clause

The GROUP BY clause is used to group values from a column, and, if you wish, perform calculations on that column. You can use COUNT, SUM, AVG, etc., i.e. group functions on the grouped column.

Syntax:

```
SELECT   column1, column2,…., aggregate_function(ci)
      FROM   table
      [ WHERE <condition> ]
      [ GROUP BY <column 1 , column 2,...> ] ;
```

For example:

Find sum of salary paid to each job

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+-------------+-----------+--------+

mysql> Select job, sum(salary) from employee group by job;
+------------+------------+
| job        |sum(salary) |
+------------+------------+
| Manager    |50000       |
| Clerk      |60000       |
| Programmer |75000       |
+------------+------------+
```

For example:

Count total employees for each job

```
mysql> select * from employee;
+-------+------------+----------+--------+
| ID    | Name       | job      | Salary |
+-------+------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 50000  |
| 4117  | Amit Jain  | Clerk    | 20000  |
| 6543  | Raj Rathi  | Clerk    | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk    | 20000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+------------+----------+--------+
mysql> Select job, count(*) from employee group by job;
+------------+--------+
| job        |count(*)|
+------------+--------+
| Manager    |1       |
| Clerk      |3       |
| Programmer |2       |
+------------+--------+
```

For example:

Find sum of salary paid to clerks , programmers, drivers

```
mysql> select * from employee;
+-------+------------+----------+--------+
| ID    | Name       | job      | Salary |
+-------+------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 50000  |
| 4117  | Amit Jain  | Clerk    | 20000  |
| 6543  | Raj Rathi  | Clerk    | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk    | 20000  |
| 6124  | Raja kumar  | programmer| 20000  |
| 6124  | Ram Agrawal | Driver   | 15000  |
+-------+------------+----------+--------+

mysql> Select job, sum(salary) from employee where job in
('Clerk','Driver','Programmer' ) group by job;;
+------------+----------+
| job        |sum(salary)|
+------------+----------+
| Driver     |150000    |
| Clerk      |600000    |
| Programmer |750000    |
+------------+----------+
```

Having Clause

The HAVING Clause enables you to specify conditions that filter which group results appear in the results. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

```
SELECT column1, column2
            FROM table1, table2
            [ WHERE < conditions > ]
            [ GROUP BY column1, column2
            [ HAVING < conditions > ] ];
```

For example:

Find list of jobs where sum of salary paid is greater than or equals to 60000

```
mysql> select * from employee;
+-------+-------------+----------+--------+
| ID    | Name        | job      | Salary |
+-------+-------------+----------+--------+
| 3012  | Gopal Pandey| Manager  | 50000  |
| 4117  | Amit Jain   | Clerk    | 20000  |
| 6543  | Raj Rathi   | Clerk    | 20000  |
| 5332  | Mona Mantri | Programmer| 40000 |
| 6124  | Raj Agrawal | Clerk    | 20000  |
| 6124  | Raja Kumar  | programmer| 20000 |
+-------+-------------+----------+--------+

mysql>Select job,sum(salary) from employee group by job having sum(salary)>=
60000;
+------------+------------+
| job        |sum(salary) |
+------------+------------+
| Clerk      |60000       |
| Programmer |75000       |
+------------+------------+
```

For example:

List job where Count of total employees for a job is greater than 1

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+-------------+-----------+--------+
mysql> Select job, count(*) from employee group by job having
count(*)>1;
+------------+--------+
| job        |count(*)|
+------------+--------+
| Clerk      |3       |
| Programmer |2       |
+------------+--------+
```

For example:

Find list of jobs where male employes count is greater than 1

```
mysql> select * from employee;
+-------+-------------+-----------+--------+--------+
| ID    | Name        | job       | Salary | Gender |
+-------+-------------+-----------+--------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  | M      |
| 4117  | Amit Jain   | Clerk     | 20000  | M      |
| 6543  | Mona Mantri | Clerk     | 20000  | F      |
| 5332  | Raj Rathi   | Programmer| 40000  | M      |
| 6124  | Raj Agrawal | Clerk     | 20000  | M      |
| 6124  | Raja kumar  | programmer| 20000  | M      |
| 6124  | Ram Agrawal | programmer| 15000  | M      |
+-------+-------------+-----------+--------+--------+

mysql> Select job, count(*) from employee where gender='M' group by job
having count(*)>1 ;
+------------+-----------+
| job        |count(*)   |
+------------+-----------+
| Clerk      |2          |
| Programmer |3          |
+------------+-----------+
```

Order By Clause

The 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.

Syntax:

```
SELECT column1, column2, ...
    FROM table_name
    ORDER BY column1, column2, ... ASC|DESC;
```

For example:

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+-------------+-----------+--------+

mysql> select * from employee order by name;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 4117  | Amit Jain   | Clerk     | 20000  |
| 3012  | Gopal Pandey| Manager   | 50000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+-------------+-----------+--------+
```

For example:

```
mysql>select * from employee order by name desc;;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 6124  | Raja Kumar  | programmer| 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
+-------+-------------+-----------+--------+
mysql> select * from emp order by job,name;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 3012  | Gopal Pandey| Manager   | 50000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raja Kumar  | programmer| 20000  |
+-------+-------------+-----------+--------+

mysql> select * from emp order by job desc,name;

+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raja Kumar  | programmer| 20000  |
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
+-------+-------------+-----------+--------+
```

Limit Clause

MySQL provides a LIMIT clause that is used to specify the number of records to return. 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.

The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1

Syntax:

```
SELECT column1, column2, ...
    FROM table_name
    LIMIT <value> OFFSET <value>;
```

For example:

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6122  | Raja Kumar  | programmer| 20000  |
| 3013  | Gopal Pandey| clerk     | 20000  |
| 5335  | Mona Mantri | Manager   | 50000  |
+-------+-------------+-----------+--------+

mysql> select * from employee limit 4 ;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
+-------+-------------+-----------+--------+
```

For example:

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6122  | Raja Kumar  | programmer| 20000  |
| 3013  | Gopal Pandey| clerk     | 20000  |
| 5335  | Mona Mantri | Manager   | 50000  |
+-------+-------------+-----------+--------+

mysql> select * from employee limit 4 offset 3;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6122  | Raja Kumar  | programmer| 20000  |
| 3013  | Gopal Pandey| clerk     | 20000  |
+-------+-------------+-----------+--------+
```

ALTER Command

MySQL ALTER command can be used to add columns to an existing table, drop a column from a table, rename an existing column, and change the data type of a column. Below are the syntax used for the different operations which can be performed using Alter command.

To change structure of table.

To adding a new column

Syntax:

```
ALTER TABLE table_name ADD COLUMN column_name datatype;
```

To renaming a column of an existing table

Syntax

```
ALTER TABLE table_name change old_column_name new_column_name datatype;
```

To dropping a column

Syntax

```
ALTER TABLE table_name DROP column_name;
```

To Modify a Column

Syntax

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

For example:

Add joindate column of type date to employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> ALTER TABLE employee ADD COLUMN joindate date;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
| joindate  | date        | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

For example:

Change joindate column to birthdate column of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
| joindate  | date        | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> ALTER TABLE employee CHANGE joindate birthdate date;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
| birthdate | date        | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

For example:

Remove column joindate of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
| joindate  | date        | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> ALTER TABLE employee DROP joindate;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

For example:

change column salary from integer to big integer of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> ALTER TABLE employee modify salary bigint;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | bigint(20)  | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

Constraints

MySQL Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table while performing insert , update operations. This ensures the accuracy and reliability of the data in the database.

Constraints are of 2 types

1. **Column constraints**

   The column level constraints are applied only to one column

   For ex:

   Gender must be 'm' or 'f'

2. **Table constraints**

   The table level constraints are applied to the whole table.

   For ex:

   If gender is 'm' then age>=21 or

   If gender if 'f' then age>=18

Constraints can be set while creating table or by using alter table command after table has been created.

To add Constraint

Syntax:

```
ALTER TABLE table_name ADD CONSTRAINT  [column_name ] Constraint [details];
```

To remove Constraint

Syntax:

```
ALTER TABLE table_name DROP CONSTRAINT column_name;
```

Primary Key

MySQL A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values. A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

For example: column constraint

```
create table students
( id integer primary key,
  name varchar(30),
  gender char(1),
  branch varchar(10)) ;
```

For example: table constraint

```
create table students
( id integer,
  name varchar(30),
  gender char(1),
  branch varchar(10),
  primary key(id)) ;
```

For example: composite key

```
create table students
( rollno integer,
  name varchar(30),
  gender char(1),
  branch varchar(10),
  primary key(rollno,branch,year)) ;
```

For example: if table exists

```
Alter table students add primary key (id);
```

Note : for this table must contains all not null and unique values in that column.

For example: to remove primary key

```
Alter table students drop primary key;
```

For example:

Add primary key to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   |     | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table employee add primary key (empno);
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

For example:

remove primary key to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table employee drop primary key;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   |     | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

Not NULL

It indicates that NULL values are not allowed for the column.

For example: column constraint

```
create table students
( id integer primary key,
  name varchar(30) Not NULL,
  gender char(1),
  branch varchar(10)) ;
```

For example: if table exists

```
Alter table students modify name varchar(30) Not NULL;
```

Note : for this table must contains all not null values in that column.

For example: to remove Not Null

```
Alter table students modify name varchar(30) NULL;
```

For example:

Add primary key to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   |  PRI| NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table employee modify name varchar(30) NOT NULL;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | NO   |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

Unique

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. Unique Columns can contain null values but if values are specified then they must be unique.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

For example: column constraint

```
create table students
( id integer unique,
  name varchar(30),
  gender char(1),
  branch varchar(10)) ;
```

For example: table constraint

```
create table students
( id integer,
  name varchar(30),
  gender char(1),
  branch varchar(10),
  unique(id)) ;
```

For example: composite key

```
create table students
( rollno integer,
  name varchar(30),
  gender char(1),
  branch varchar(10),
  unique (rollno,branch,year)) ;
```

For example: if table exists

```
Alter table students add constraint Unique (id );
```

For example: to remove unique

```
Alter table students drop index id;
```

For example:

Add unique to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table employee add unique(empno);
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  | UNI | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

For example:

remove primary key to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  | UNI | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table employee drop index name;
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

DEFAULT

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

For example: column constraint

```
Create table students
( id integer primary key,
  name varchar(30),
  gender char(1) default 'M',
  branch varchar(10));
```

For example: if table exists

```
Alter table students modify gender char(1) default 'M';
```

For example: to remove default

```
Alter table students alter column gender drop default;
```

For example:

Add primary key to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | NULL    |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
mysql> Alter table students modify job varchar(10) default 'clerk';
mysql> desc employee;
+-----------+-------------+------+-----+---------+
| Field     | Type        | Null | Key | Default |
+-----------+-------------+------+-----+---------+
| empno     | int(11)     | NO   | PRI | NULL    |
| name      | varchar(30) | YES  |     | NULL    |
| job       | varchar(10) | YES  |     | clerk   |
| salary    | int(11)     | YES  |     | NULL    |
+-----------+-------------+------+-----+---------+
```

CHECK Constraint

The CHECK constraint is used to limit the value that can be placed in a column.

For example: column constraint

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
     Age int CHECK (Age>=18));
```

For example: if table exists

```
ALTER TABLE Persons ADD CHECK (Age>=18);
```

For example: to remove Not Null

```
ALTER TABLE Persons DROP CHECK;
```

AUTO INCREMENT

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field.

For example: column constraint

```
CREATE TABLE Persons
( Personid int  PRIMARY KEY  AUTO_INCREMENT,
Name varchar(25) );
```

For example: to start increment from a specified value

```
CREATE TABLE Persons
( Personid int  PRIMARY KEY   AUTO_INCREMENT,
Name varchar(25) )AUTO_INCREMENT=1000;
```

For example: if table exists with Auto_Increment col then start value can be set

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

For example:

Add primary key auto_increment to empno of employee table

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| empno     | int(11)     | NO   | PRI | NULL    |                |
| name      | varchar(30) | YES  |     | NULL    |                |
| job       | varchar(10) | YES  |     | NULL    |                |
| salary    | int(11)     | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
mysql> Alter table employee AUTO_INCREMENT;
mysql> desc employee;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| empno     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(30) | YES  |     | NULL    |                |
| job       | varchar(10) | YES  |     | NULL    |                |
| salary    | int(11)     | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

For example: specify foreign key while creating table

```
create table employee
    ( empno int primary key,
      name varchar(20),
      job varchar(10),
      deptno int,
      FOREIGN KEY (deptno) REFERENCES dept(deptno) );
```

For example: adding foreign key after creating table

```
ALTER TABLE employee ADD FOREIGN KEY (deptno) REFERENCES dept(deptno);
```

Add foreign key to job of employee table

```
mysql> desc jobs;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| job       | varchar(10) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
mysql> desc employee;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| empno     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(30) | YES  |     | NULL    |                |
| job       | int(11)     | YES  |     | NULL    |                |
| salary    | int(11)     | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
mysql> ALTER TABLE employee ADD FOREIGN KEY (job) REFERENCES jobs(id);
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| empno     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(30) | YES  |     | NULL    |                |
| job       | int(11)     | YES  | MUL | NULL    |                |
| salary    | int(11)     | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Column aliases

Instead of displaying column names / expressions in our query we can use column aliases. This will make column names more readable.

Syntax:

```
select column_name1, column_name as column_aliases from table_name;
```

Add foreign key to job of employee table

```
mysql> select * from jobs;
+-----------+--------+-------+-------+---------+
| job       | basic  | hra   | ta    | da      |
+-----------+--------+-------+-------+---------+
| CEO       | 75000  | 25000 | 15000 | 15000   |
| clerk     | 15000  | 5000  | 3000  | 2000    |
| manager   | 40000  | 15000 | 5000  | 5000    |
| developer | 25000  | 10000 | 5000  | 5000    |
| driver    | 15000  | 4000  | 2000  | 2000    |
+-----------+--------+-------+-------+---------+
mysql> select job ,basic+hra+ta+da  from jobs;
+-----------+-----------------+
| job       | basic+hra+ta+da |
+-----------+-----------------+
| CEO       | 130000          |
| clerk     | 25000           |
| manager   | 65000           |
| developer | 45000           |
| driver    | 20000           |
+-----------+-----------------+
mysql> select job ,basic+hra+ta+da as salary  from jobs;
+-----------+---------+
| job       | salary  |
+-----------+---------+
| CEO       | 130000  |
| clerk     | 25000   |
| manager   | 65000   |
| developer | 45000   |
| driver    | 20000   |
+-----------+---------+
```

SET Operations in SQL

Set operators are used to join the results of two (or more) SELECT statements. Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

MySQL Supports:

- UNION
- UNION ALL

UNION

It is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same data type
- Have them in the same order
- But they need not have to be in the same length.

Syntax:

```
select column_name1, column_name2… from table_name
  UNION
select column_name1, column_name2… from table_name;
```

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 6122  | Raja Kumar  | programmer| 20000  |
| 3013  | Gopal Pandey| clerk     | 20000  |
| 5335  | Mona Mantri | Manager   | 50000  |
+-------+-------------+-----------+--------+
mysql> select * from customer;
+-------+-------------+-----------+
| ID    | Name        | City      |
+-------+-------------+-----------+
| 3432  | Sumit Pandey| Amravati  |
| 4134  | Arjun Jain  | Akola     |
| 3455  | karan Rathi | Pune      |
| 5675  | Mona Mantri | Amravati  |
| 5675  | Amit jain   | Amravati  |
+-------+-------------+-----------+
mysql> select Name from employee union select Name from customer;
+-------------+
| Name        |
+-------------+
| Gopal Pandey|
| Amit Jain   |
| Raj Rathi   |
| Mona Mantri |
| Raj Agrawal |
| Raja Kumar  |
| Gopal Pandey|
| Mona Mantri |
| Sumit Pandey|
| Arjun Jain  |
| karan Rathi |
+-------------+
```

UNION ALL

It is used to combine the results of two SELECT statements including duplicate rows. The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax:

```
select column_name1, column_name2… from table_name
  UNION ALL
select column_name1, column_name2… from table_name;
```

```
mysql> select * from employee;
+-------+-------------+-----------+--------+
| ID    | Name        | job       | Salary |
+-------+-------------+-----------+--------+
| 3012  | Gopal Pandey| Manager   | 50000  |
| 4117  | Amit Jain   | Clerk     | 20000  |
| 6543  | Raj Rathi   | Clerk     | 20000  |
| 5332  | Mona Mantri | Programmer| 40000  |
| 6124  | Raj Agrawal | Clerk     | 20000  |
| 3013  | Gopal Pandey| clerk     | 20000  |
| 5335  | Mona Mantri | Manager   | 50000  |
+-------+-------------+-----------+--------+
mysql> select * from customer;
+-------+-------------+-----------+
| ID    | Name        | City      |
+-------+-------------+-----------+
| 3432  | Sumit Pandey| Amravati  |
| 4134  | Arjun Jain  | Akola     |
| 3455  | karan Rathi | Pune      |
| 5675  | Mona Mantri | Amravati  |
| 5675  | Amit jain   | Amravati  |
+-------+-------------+-----------+
mysql> select Name from employee union all select Name from customer;
+-------------+
| Name        |
+-------------+
| Gopal Pandey|
| Amit Jain   |
| Raj Rathi   |
| Mona Mantri |
| Raj Agrawal |
| Gopal Pandey|
| Mona Mantri |
| Sumit Pandey|
| Arjun Jain  |
| karan Rathi |
| Mona Mantri |
| Amit jain   |
+-------------+
```

JOINS

MySQL Joins plays an important role when we have to join two tables together based on one or more common values shared by two tables. A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

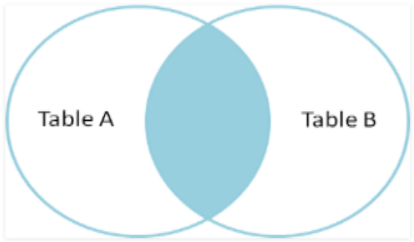
Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN

INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

In an inner join, we only select the data which is common in both the tables. (ie, part 3 here) In order to make it more precise, all the records from both the tables matching up the condition mentioned with the join are picked in this join.



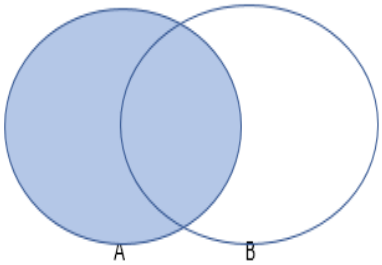
Syntax:

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
mysql> select * from employee;
+-------+----------------+-----------+--------+--------+
| ID    | Name           | job       | Salary | deptno |
+-------+----------------+-----------+--------+--------+
| 1001  | Amit Jain      | Manager   | 59005  | 10     |
| 1005  | Sumit Rai      | Manager   | 59005  | 20     |
| 1006  | Rajev Ranjan   | Clerk     | 21605  | 20     |
| 1007  | Rohan Joshi    | Clerk     | 15500  | 20     |
| 1008  | Roshan Agrawal | Clerk     | 35500  | 20     |
| 1009  | Radha Rathi    | Manager   | 48000  | 30     |
| 1011  | Chetan Deshmukh| Developer | 34500  | 20     |
| 1011  | Manjiri Kaste  | Clerk     | 22300  | 20     |
| 1011  | Amita Agrawal  | NULL      | NULL   | NULL   |
| 1011  | Gopal Pandey   | NULL      | NULL   | NULL   |
+-------+----------------+-----------+--------+--------+
mysql> select * from dept;
+-------+------------+-----------+
| deptno| Dname      | loc       |
+-------+------------+-----------+
| 10    | Accounts   | Mumbai    |
| 20    | Marketing  | Delhi     |
| 30    | Production | Banglore  |
| 40    | Research   | Chennai   |
+-------+------------+-----------+
mysql> select empno, name, dname from employee inner join dept on emp.deptno
= dept.deptno;
+-------+----------------+-----------+
| ID    | Name           | dname     |
+-------+----------------+-----------+
| 1001  | Amit Jain      | Accounts  |
| 1005  | Sumit Rai      | Marketing |
| 1006  | Rajev Ranjan   | Marketing |
| 1007  | Rohan Joshi    | Marketing |
| 1008  | Roshan Agrawal | Marketing |
| 1009  | Radha Rathi    | Production|
| 1011  | Chetan Deshmukh| Marketing |
+-------+----------------+-----------+
```

LEFT JOIN

In The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL. In left join, we select all the data from the left table and from the right table only select the data set which matches up with the condition mentioned with the join



Syntax:

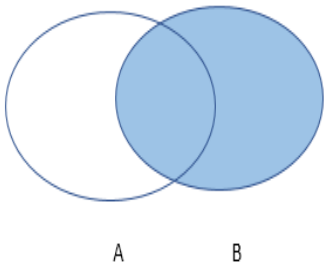
```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
mysql> select * from employee;
+-------+---------------+-----------+--------+--------+
| ID    | Name          | job       | Salary | deptno |
+-------+---------------+-----------+--------+--------+
| 1001  | Amit Jain     | Manager   | 59005  | 10     |
| 1005  | Sumit Rai     | Manager   | 59005  | 20     |
| 1006  | Rajev Ranjan  | Clerk     | 21605  | 20     |
| 1007  | Rohan Joshi   | Clerk     | 15500  | 20     |
| 1011  | Chetan Deshmukh| Developer | 34500  | 20     |
| 1011  | Amita Agrawal | NULL      | NULL   | NULL   |
| 1011  | Gopal Pandey  | NULL      | NULL   | NULL   |
+-------+---------------+-----------+--------+--------+
mysql> select * from dept;
+-------+------------+-----------+
| deptno| Dname      | loc       |
+-------+------------+-----------+
| 10    | Accounts   | Mumbai    |
| 20    | Marketing  | Delhi     |
| 30    | Production | Banglore  |
| 40    | Research   | Chennai   |
+-------+------------+-----------+
mysql> select empno, name, dname from emp left join dept on emp.deptno =
dept.deptno
+-------+---------------+-----------+
| ID    | Name          | dname     |
+-------+---------------+-----------+
| 1001  | Amit Jain     | Accounts  |
| 1005  | Sumit Rai     | Marketing |
| 1006  | Rajev Ranjan  | Marketing |
| 1007  | Rohan Joshi   | Marketing |
| 1011  | Chetan Deshmukh| Marketing |
| 1011  | Amita Agrawal | NULL      |
| 1011  | Gopal Pandey  | NULL      |
+-------+---------------+-----------+
```

RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table.

If there is no matching in both tables, it will return NULL. In right join, we select all the data from the right table and from the left table only select the data set which matches up with the condition mentioned with the join.



Syntax:

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

```
mysql> select * from employee;
+-------+----------------+-----------+--------+--------+
| ID    | Name           | job       | Salary | deptno |
+-------+----------------+-----------+--------+--------+
| 1001  | Amit Jain      | Manager   | 59005  | 10     |
| 1005  | Sumit Rai      | Manager   | 59005  | 20     |
| 1006  | Rajev Ranjan   | Clerk     | 21605  | 20     |
| 1007  | Rohan Joshi    | Clerk     | 15500  | 20     |
| 1011  | Chetan Deshmukh| Developer | 34500  | 20     |
| 1011  | Amita Agrawal  | NULL      | NULL   | NULL   |
| 1011  | Gopal Pandey   | NULL      | NULL   | NULL   |
+-------+----------------+-----------+--------+--------+
mysql> select * from dept;
+-------+------------+-----------+
| deptno| Dname      | loc       |
+-------+------------+-----------+
| 10    | Accounts   | Mumbai    |
| 20    | Marketing  | Delhi     |
| 30    | Production | Banglore  |
| 40    | Research   | Chennai   |
+-------+------------+-----------+
mysql> select empno, name, dname from emp left join dept on emp.deptno =
dept.deptno
+-------+----------------+-----------+
| ID    | Name           | dname     |
+-------+----------------+-----------+
| 1001  | Amit Jain      | Accounts  |
| 1005  | Sumit Rai      | Marketing |
| 1006  | Rajev Ranjan   | Marketing |
| 1007  | Rohan Joshi    | Marketing |
| 1011  | Chetan Deshmukh| Marketing |
| NULL  | NULL           | Production|
| NULL  | NULL           | Research  |
+-------+----------------+-----------+
```

Simple Join

We can join multiple tables by specifying table list and appropriate condition..

Syntax:

```
Select col1,col2, . . .
From table1,table2, . .
Where <condition>
```

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
|  1011 | G.Gandhi     | clerk     |     20 |   1006 |
|  1012 | A.Raja       | manager   |     40 |   1000 |
|  1013 | M.Joshi      | manager   |     10 |   1000 |
+-------+--------------+-----------+--------+--------+

mysql> select * from dept;
+--------+------------+--------+
| deptno | name       | loc    |
+--------+------------+--------+
|     10 | Production | Nagpur |
|     20 | Marketing  | Mumbai |
|     30 | Accounts   | Nasik  |
|     40 | Research   | Pune   |
+--------+------------+--------+
```

```
mysql> select * from jobs;
+-----------+-------+-------+-------+-------+
| job       | basic | hra   | ta    | da    |
+-----------+-------+-------+-------+-------+
| CEO       | 75000 | 25000 | 15000 | 15000 |
| clerk     | 15000 |  5000 |  3000 |  2000 |
| developer | 25000 | 10000 |  5000 |  5000 |
| driver    | 12000 |  4000 |  2000 |  2000 |
| manager   | 40000 | 15000 |  5000 |  5000 |
+-----------+-------+-------+-------+-------+

mysql> select emp.name,emp.job,jobs.basic,dept.name from emp,jobs,dept where
emp.deptno=dept.deptno and emp.job=jobs.job;

+--------------+-----------+-------+------------+
| name         | job       | basic | name       |
+--------------+-----------+-------+------------+
| Amit Jain    | clerk     | 15000 | Production |
| Gopal Pandey | clerk     | 15000 | Marketing  |
| Mona Mantri  | clerk     | 15000 | Accounts   |
| Raja Rathi   | clerk     | 15000 | Research   |
| M.Rao        | manager   | 40000 | Production |
| G.Joshi      | manager   | 40000 | Marketing  |
| A.Agrawal    | manager   | 40000 | Accounts   |
| A.Patil      | developer | 25000 | Production |
| M.Pandey     | developer | 25000 | Production |
| R.Wankhade   | developer | 25000 | Production |
| G.Gandhi     | clerk     | 15000 | Marketing  |
| A.Raja       | manager   | 40000 | Research   |
| M.Joshi      | manager   | 40000 | Production |
+--------------+-----------+-------+------------+
```

Table aliases

If table names are lengthy then we can use table aliases instead of table names in our query.

Syntax:

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1 as tablealias, table1 as tablealias;
```

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
+-------+--------------+-----------+--------+--------+
mysql> select * from dept;
+--------+------------+--------+
| deptno | name       | loc    |
+--------+------------+--------+
|     10 | Production | Nagpur |
|     20 | Marketing  | Mumbai |
|     30 | Accounts   | Nasik  |
|     40 | Research   | Pune   |
+--------+------------+--------+
mysql>select e.name,e.job,d.name from emp e,dept d where e.deptno=d.deptno;
+--------------+-----------+------------+
| name         | job       | name       |
+--------------+-----------+------------+
| Amit Jain    | clerk     | Production |
| Gopal Pandey | clerk     | Marketing  |
| Mona Mantri  | clerk     | Accounts   |
| Raja Rathi   | clerk     | Research   |
| M.Rao        | manager   | Production |
| G.Joshi      | manager   | Marketing  |
| A.Agrawal    | manager   | Accounts   |
| A.Patil      | developer | Production |
| M.Pandey     | developer | Production |
| R.Wankhade   | developer | Production |
+--------------+-----------+------------+
```

Self Joins

The SQL SELF JOIN is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement.

Syntax:

```
SELECT column_name(s) FROM table1 T1, table1 T2 WHERE condition;
```

```
mysql> select * from employee;
+-------+-------------+----------+--------+--------+
| empno | name        | job      | deptno | bossid |
+-------+-------------+----------+--------+--------+
|  1000 | R.R.Sharma  | CEO      |   NULL |   NULL |
|  1001 | Amit Jain   | clerk    |     10 |   1005 |
|  1002 | Gopal Pandey| clerk    |     20 |   1006 |
|  1003 | Mona Mantri | clerk    |     30 |   1007 |
|  1004 | Raja Rathi  | clerk    |     40 |   1012 |
|  1005 | M.Rao       | manager  |     10 |   1000 |
|  1006 | G.Joshi     | manager  |     20 |   1000 |
|  1007 | A.Agrawal   | manager  |     30 |   1000 |
|  1008 | A.Patil     | developer|     10 |   1013 |
|  1009 | M.Pandey    | developer|     10 |   1013 |
|  1010 | R.Wankhade  | developer|     10 |   1005 |
+-------+-------------+----------+--------+--------+
mysql> select e.name,e.job,b.name as boss,b.job from emp e,emp b where
e.bossid=b.empno;
+-------------+----------+------------+---------+
| name        | job      | boss       | job     |
+-------------+----------+------------+---------+
| Amit Jain   | clerk    | M.Rao      | manager |
| Gopal Pandey| clerk    | G.Joshi    | manager |
| Mona Mantri | clerk    | A.Agrawal  | manager |
| Raja Rathi  | clerk    | A.Raja     | manager |
| M.Rao       | manager  | R.R.Sharma | CEO     |
| G.Joshi     | manager  | R.R.Sharma | CEO     |
| A.Agrawal   | manager  | R.R.Sharma | CEO     |
| A.Patil     | developer| M.Joshi    | manager |
| M.Pandey    | developer| M.Joshi    | manager |
| R.Wankhade  | developer| M.Rao      | manager |
| G.Gandhi    | clerk    | G.Joshi    | manager |
| A.Raja      | manager  | R.R.Sharma | CEO     |
| M.Joshi     | manager  | R.R.Sharma | CEO     |
+-------------+----------+------------+---------+
```

Views

MySQL view is nothing but a virtual table of the database. The view contains fields like a real table, but those fields are from one or more tables in the database which is executed by running a bunch of MySQL queries. We can perform operations like WHERE and JOIN clauses in the virtual tables. On the other hand, VIEW is nothing but SELECT queries.

To Create View

Syntax:

```
CREATE VIEW view_name AS SELECT column1, column2,… FROM table;
```

To Drop View

Syntax:

```
DROP VIEW view_name;
```

```
mysql> select * from employee;
+----+-------------+--------+-----------------+------+
| id | Name        | salary | email           | job  |
+----+-------------+--------+-----------------+------+
|  1 | Amit Jain   |  25000 | amit@gmail.com  |    1 |
|  2 | Sumit Kumar |  35000 | sumit@gmail.com |    3 |
|  3 | Gopal pande |  30000 | gopal@gmail.com |    2 |
|  5 | Arjun Sharm |  50000 | arjun@yahoo.com |    2 |
|  6 | Kapil Kumar |  40000 | kapil@yahoo.com |    3 |
|  7 | Harry Rai   |  40000 | harry@gmail.com |    3 |
+----+-------------+--------+-----------------+------+

mysql> select * from jobs;
+----+------------+
| id | job        |
+----+------------+
|  1 | clerk      |
|  2 | manager    |
|  3 | programmer |
+----+------------+
```

```
mysql> create view emp_jobs as select emp.Name,jobs.job,salary from emp,jobs
where emp.job=jobs.id;

mysql> show tables;
+------------------+
| Tables_in_ccitdb |
+------------------+
| emp              |
| emp_job          |
| jobs             |
+------------------+

mysql> select * from emp_jobs;
+-------------+------------+--------+
| Name        | job        | salary |
+-------------+------------+--------+
| Amit Jain   | clerk      |  25000 |
| Gopal pande | manager    |  30000 |
| Arjun Sharm | manager    |  50000 |
| Sumit Kumar | accountant |  35000 |
| Kapil Kumar | programmer |  40000 |
| Harry Rai   | programmer |  40000 |
+-------------+------------+--------+

mysql> drop view emp_jobs;

mysql> show tables;
+------------------+
| Tables_in_ccitdb |
+------------------+
| emp              |
| jobs             |
+------------------+
```

SUB-Queries

A MySQL subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. In addition, a subquery can be nested inside another subquery. A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

A subquery is a SQL query nested inside a larger query.
A subquery may occur in:

- A SELECT clause
- A FROM clause
- A WHERE clause

The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query.

Syntax:

```
select collist...
     from tablelist...
     where exp Operator ( select statement....);
```

Note: Where operators can be >, <, = etc.

The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.

SQL Logical Operators

Operator	Description
EXISTS	It is used to search for the presence of a row in a table which satisfies a certain condition specified in the query.
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.

For example:

Find list of employees whose salary is greater than average salary of employees.

```
mysql> select * from employee;
+-------+---------------+----------+--------+--------+
| ID    | Name          | job      | Salary | deptno |
+-------+---------------+----------+--------+--------+
| 1005  | Sumit Rai     | Manager  | 59005  | 20     |
| 1006  | Rajev Ranjan  | Clerk    | 21605  | 20     |
| 1007  | Rohan Joshi   | Clerk    | 15500  | 20     |
| 1008  | Roshan Agrawal| Clerk    | 35500  | 20     |
| 1009  | Radha Rathi   | Manager  | 48000  | 30     |
| 1011  | Chetan Deshmukh| Developer| 34500 | 20     |
| 1011  | Manjiri Kaste | Clerk    | 22300  | 20     |
+-------+---------------+----------+--------+--------+
mysql> select * from emp where salary > (select avg(salary) from emp);
+-------+---------------+----------+--------+--------+
| ID    | Name          | job      | Salary | deptno |
+-------+---------------+----------+--------+--------+
| 1005  | Sumit Rai     | Manager  | 59005  | 20     |
| 1008  | Roshan Agrawal| Clerk    | 35500  | 20     |
| 1009  | Radha Rathi   | Manager  | 48000  | 30     |
| 1011  | Chetan Deshmukh| Developer| 34500 | 20     |
+-------+---------------+----------+--------+--------+
```

Note : a subquery can be used in select , insert, update delete statements .

For example:

Delete all employees whose salary is greater than average salary of clerks.

```
mysql> select * from employee;
+-------+---------------+----------+--------+--------+
| ID    | Name          | job      | Salary | deptno |
+-------+---------------+----------+--------+--------+
| 1005  | Sumit Rai     | Manager  | 59005  | 20     |
| 1006  | Rajev Ranjan  | Clerk    | 21605  | 20     |
| 1007  | Rohan Joshi   | Clerk    | 15500  | 20     |
| 1008  | Roshan Agrawal| Clerk    | 35500  | 20     |
| 1009  | Radha Rathi   | Manager  | 48000  | 30     |
| 1011  | Chetan Deshmukh| Developer| 34500 | 20     |
| 1011  | Manjiri Kaste | Clerk    | 22300  | 20     |
+-------+---------------+----------+--------+--------+
mysql> Delete from emp where salary > (select avg(salary) from emp where
job='Clerk');

+-------+---------------+----------+--------+--------+
| ID    | Name          | job      | Salary | deptno |
+-------+---------------+----------+--------+--------+
```

For example:

Find list of employees from the city where there is company office.

```
mysql> select * from employee;
+-------+----------------+----------+--------+--------+----------+
| ID    | Name           | job      | Salary | deptno | city     |
+-------+----------------+----------+--------+--------+----------+
| 1005  | Sumit Rai      | Manager  | 59005  | 20     | Mumbai   |
| 1006  | Rajev Ranjan   | Clerk    | 21605  | 20     | Amravati |
| 1007  | Rohan Joshi    | Clerk    | 15500  | 20     | Amravati |
| 1008  | Roshan Agrawal | Clerk    | 35500  | 20     | Delhi    |
| 1009  | Radha Rathi    | Manager  | 48000  | 30     | Mumbai   |
| 1011  | Chetan Deshmukh| Developer| 34500  | 20     | Delhi    |
| 1011  | Manjiri Kaste  | Clerk    | 22300  | 20     | Amravati |
| 1020  | Raja Rathi     | Clerk    | 20300  | NULL   | Delhi    |
+-------+----------------+----------+--------+--------+----------+

mysql> select * from dept;
+--------+-----------+----------+
| deptno | name      | loc      |
+--------+-----------+----------+
|     10 | Accounts  | Mumbai   |
|     20 | Marketing | Delhi    |
|     30 | Production| Banglore |
|     40 | Research  | Pune     |
+--------+-----------+----------+

mysql> select * from emp where city in (select loc from dept );
+-------+----------------+----------+--------+--------+----------+
| ID    | Name           | job      | Salary | deptno | city     |
+-------+----------------+----------+--------+--------+----------+
| 1005  | Sumit Rai      | Manager  | 59005  | 20     | Mumbai   |
| 1008  | Roshan Agrawal | Clerk    | 35500  | 20     | Delhi    |
| 1009  | Radha Rathi    | Manager  | 48000  | 30     | Mumbai   |
| 1011  | Chetan Deshmukh| Developer| 34500  | 20     | Delhi    |
| 1020  | Raja Rathi     | Clerk    | 20300  | NULL   | Delhi    |
+-------+----------------+----------+--------+--------+----------+
```

ANY / SOME Operator

The ANY operator returns true if any of the subquery values meet the condition. Some operator is alias for any.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
(SELECT column_name FROM table_name WHERE condition);
```

For example:

Find list of employees whose salary is greater than salary of any developer.

```
mysql> select * from employee;
+-------+---------------+-----------+--------+--------+----------+
| ID    | Name          | job       | Salary | deptno | city     |
+-------+---------------+-----------+--------+--------+----------+
| 1005  | Sumit Rai     | Manager   | 59005  | 20     | Mumbai   |
| 1006  | Rajev Ranjan  | Clerk     | 21605  | 20     | Amravati |
| 1007  | Rohan Joshi   | Clerk     | 15500  | 20     | Amravati |
| 1008  | Roshan Agrawal| Clerk     | 35500  | 20     | Delhi    |
| 1009  | Radha Rathi   | Manager   | 48000  | 30     | Mumbai   |
| 1011  | Chetan Deshmukh| Developer | 34500 | 20     | Delhi    |
| 1011  | Amita Agrawal | Developer | 42300  | 20     | Amravati |
| 1020  | Raja Rathi    | Clerk     | 20300  | NULL   | Delhi    |
+-------+---------------+-----------+--------+--------+----------+
mysql> select name,job,salary from emp
where salary>=any (select salary from emp where job='Developer');
+---------------+-----------+--------+
| Name          | job       | Salary |
+---------------+-----------+--------+
| Sumit Rai     | Manager   | 59005  |
| Radha Rathi   | Manager   | 48000  |
| Chetan Deshmukh| Developer | 34500 |
| Raja Rathi    | Developer | 42000  |
+---------------+-----------+--------+
```

ALL Operator

This operator returns true if all of the subquery values meet the condition.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
(SELECT column_name FROM table_name WHERE condition);
```

For example:

Find list of employees whose salary is greater than salary of all developers.

```
mysql> select * from employee;
+-------+----------------+-----------+--------+--------+-----------+
| ID    | Name           | job       | Salary | deptno | city      |
+-------+----------------+-----------+--------+--------+-----------+
| 1005  | Sumit Rai      | Manager   | 59005  | 20     | Mumbai    |
| 1006  | Rajev Ranjan   | Clerk     | 21605  | 20     | Amravati  |
| 1007  | Rohan Joshi    | Clerk     | 15500  | 20     | Amravati  |
| 1008  | Roshan Agrawal | Clerk     | 35500  | 20     | Delhi     |
| 1009  | Radha Rathi    | Manager   | 48000  | 30     | Mumbai    |
| 1011  | Chetan Deshmukh| Developer | 34500  | 20     | Delhi     |
| 1011  | Amita Agrawal  | Developer | 42300  | 20     | Amravati  |
| 1020  | Raja Rathi     | Clerk     | 20300  | NULL   | Delhi     |
+-------+----------------+-----------+--------+--------+-----------+

mysql> select name,job,salary from emp
where salary>all(select salary from emp where job='Developer') ;
+----------------+-----------+--------+
| Name           | job       | Salary |
+----------------+-----------+--------+
| Sumit Rai      | Manager   | 59005  |
| Radha Rathi    | Manager   | 48000  |
+----------------+-----------+--------+
```

Exists Operator

It is used to test for the existence of rows returned by the subquery. If the subquery returns at least one row, the EXISTS operator returns true, otherwise, it returns false.

Syntax:

```
SELECT  col list . .
FROM table list . .
WHERE  [NOT] EXISTS ( subquery ) ;
```

For example:

Find list of managers if there is a CEO in company.

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
+-------+--------------+-----------+--------+--------+
mysql> select * from emp where job='Manager' and exists(select * from emp
where job='CEO');
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
+-------+--------------+-----------+--------+--------+
```

Select Clause–Sub Query

A subquery can also be found in the SELECT clause.

For example:

Find list employees with their boss name.

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
+-------+--------------+-----------+--------+--------+
mysql> select name,(select name from emp where empno=e.bossid) boss from emp
e;
+--------------+------------+
| name         | boss       |
+--------------+------------+
| Amit Jain    | M.Rao      |
| Gopal Pandey | G.Joshi    |
| Mona Mantri  | A.Agrawal  |
| Raja Rathi   | A.Raja     |
| M.Rao        | R.R.Sharma |
| G.Joshi      | R.R.Sharma |
| A.Agrawal    | R.R.Sharma |
| A.Patil      | M.Joshi    |
| M.Pandey     | M.Joshi    |
| R.Wankhade   | M.Rao      |
| G.Gandhi     | G.Joshi    |
| A.Raja       | R.R.Sharma |
| M.Joshi      | R.R.Sharma |
+--------------+------------+
```

From Clause–Sub Query

When a subquery starts at the FROM clause, the result set is referred to as a derived table /materialized subquery.

Note: In this case table alias is compulsory.

For example:

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
+-------+--------------+-----------+--------+--------+

mysql> select name from ( select * from emp where job='Clerk') as tempemp;
+--------------+
| name         |
+--------------+
| Amit Jain    |
| Gopal Pandey |
| Raja Rathi   |
| G.Gandhi     |
+--------------+
```

INDEXES

A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records. While creating index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.

An index is a data structure that improves the speed of data retrieval on a table. This can be internally achieved at the cost of additional writes and storage to maintain it. The query optimizer may use indexes to quickly locate data without having to scan every row in a table for a given query.

Note: index are automatically created when we add primary key or unique clause.

To create Index:

Syntax:

```
CREATE INDEX idx_name ON tname (column_list);
```

For ex:

create index idxdept on emp(deptno);

create index idxdeptjob on emp(deptno,job);

To show index list on table:

Syntax:

```
SHOW INDEX FROM tname;
```

For ex:

show index from emp;

To remove index:

Syntax:

```
DROP INDEX idxname ON tname;
```

For ex:

drop index idxdept on emp;

For example:

```
mysql> select * from employee;
+-------+--------------+-----------+--------+--------+
| empno | name         | job       | deptno | bossid |
+-------+--------------+-----------+--------+--------+
|  1000 | R.R.Sharma   | CEO       |   NULL |   NULL |
|  1001 | Amit Jain    | clerk     |     10 |   1005 |
|  1002 | Gopal Pandey | clerk     |     20 |   1006 |
|  1003 | Mona Mantri  | clerk     |     30 |   1007 |
|  1004 | Raja Rathi   | clerk     |     40 |   1012 |
|  1005 | M.Rao        | manager   |     10 |   1000 |
|  1006 | G.Joshi      | manager   |     20 |   1000 |
|  1007 | A.Agrawal    | manager   |     30 |   1000 |
|  1008 | A.Patil      | developer |     10 |   1013 |
|  1009 | M.Pandey     | developer |     10 |   1013 |
|  1010 | R.Wankhade   | developer |     10 |   1005 |
+-------+--------------+-----------+--------+--------+

mysql> CREATE INDEX idx_emp ON employee(empno,name);
```

Normalization

Normalization in general terms is the technique of organizing the data into the database in order to reduce the insertion, deletion and updating anomaly and to remove data redundancy. This process divides the larger tables into smaller ones and links them with each other through relationships of the primary and foreign keys. Duplicate and unnormalized data not only consumes extra memory but makes it difficult to manage the table while insertion, deletion, and updating of tables as the number of data increases. Therefore it is very important to normalize the tables before designing the database of any application.

It is the processes of reducing the redundancy of data in the table and also improving the data integrity. So why is this required? without Normalization in SQL, we may face many issues such as

- Insertion anomaly: It occurs when we cannot insert data to the table without the presence of another attribute.
- Update anomaly: It is a data inconsistency that results from data redundancy and a partial update of data.
- Deletion Anomaly: It occurs when certain attributes are lost because of the deletion of other attributes.

In brief, normalization is a way of organizing the data in the database. Normalization entails organizing the columns and tables of a database to ensure that their dependencies are properly enforced by database integrity It usually divides a large table into smaller ones, so it is more efficient. In 1970 the First Normal Form was defined by Edgar F. Codd and eventually, other Normal Forms were defined.

One question that arises in between is, what does SQL have to do with Normalization. Well SQL is the language that is used to interact with the database. To initiate any interaction the data present in the database has to be of Normalized Form. Else we cannot proceed further as it results in anomalies.

Normalization in SQL will enhance the distribution of data. Now let' s understand each and every Normal Form with examples constraints.

1st Normal Form (1NF)

In this Normal Form, we tackle the problem of atomicity. Here atomicity means values in the table should not be further divided. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.

Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

Course	Content
Programmer	Java,C++
web	HTML,PHP,ASP

In the above table, we can clearly see that the Content column has more than one values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.

Course	Content
Programmer	Java
Programmer	C++
web	HTML
web	PHP
web	ASP

By this, we have achieved atomicity and also each and every column have unique values.

2nd Normal Form (2NF)

The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate key determines a non-prime attribute.

The entity should be considered already in 1NF, and all attributes within the entity should depend solely on the unique identifier of the entity.

Rules

- It should be in the First Normal form.
- And, it should not have Partial Dependency.

Student_id	Course	Teacher
1001	Java	A.Jain
1002	C++	S.Rao
1003	Java	A.Jain
1004	C	G.Pandey
1005	Python	M.Mantri

This table has a composite primary key Student_id, Course. The non-key attribute is Teacher. In this case, Teacher only depends on Course, which is only part of the primary key. Therefore, this table does not satisfy the second Normal Form.

Student_id	Course
1001	Java
1002	C++
1003	Java
1004	C
1005	Python

Course	Teacher
Java	A.Jain
C++	S.Rao
C	G.Pandey
Python	M.Mantri

3rd Normal Form (3NF)

The same rule applies as before i.e, the table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes. That means non-prime attributes (which doesn' t form a candidate key) should not be dependent on other non-prime attributes in a given table. So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z (where it is not the case that Y → X).

Rules

- It is in the Second Normal form.
- And, it doesn't have Transitive Dependency.

Student_id	Student_Name	Subject_id	city	Subject
1001	Amit Jain	1802	Amravati	C
1002	Sumit Kumar	1803	Pune	C++
1003	Gopal Pandey	1802	Nagpur	C
1004	Mona Mantri	1801	Pune	Java
1005	Raja Rathi	1801	Amravati	Java

In the above table, Student ID determines Subject ID, and Subject ID determines Subject. Therefore, Student ID determines Subject via Subject ID. This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

Student_id	Student_Name	Subject_id	city
1001	Amit Jain	1802	Amravati
1002	Sumit Kumar	1803	Pune
1003	Gopal Pandey	1802	Nagpur
1004	Mona Mantri	1801	Pune
1005	Raja Rathi	1801	Amravati

Subject_id	Subject
1802	C
1803	C++
1802	C
1801	Java
1801	Java

Boyce Codd Normal Form (BCNF)

This is also known as 3.5 NF. Its the higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomalies which were not dealt with 3NF. Before proceeding to BCNF the table has to satisfy 3rd Normal Form. In BCNF if every functional dependency A → B, then A has to be the Super Key of that particular table.

Rules

- R must be in 3rd Normal Form
- For each functional dependency (X → Y), X should be a super Key.

Student_id	Subject	Professor
1001	C	Prof.Amit
1002	C++	Prof.Goapl
1003	Java	Prof.Mona
1004	C++	Prof.Goapl
1005	Java	Prof.Amit

One student can enroll for multiple subjects. There can be multiple professors teaching one subject And, For each subject, a professor is assigned to the student. As you can see Student ID, and Subject form the primary key, which means the Subject column is a prime attribute. But, there is one more dependency, Professor → Subject. And while Subject is a prime attribute, Professor is a non-prime attribute, which is not allowed by BCNF. Now in order to satisfy the BCNF, we will be dividing the table into two parts. One table will hold Student ID which already exists and newly created column Professor ID.

Student_id	Professor_ID
1001	PR1701
1002	PR1702
1003	PR1703
1004	PR1702
1005	PR1704

Professor_ID	Subject	Professor
PR1701	C	Prof.Amit
PR1702	C++	Prof.Goapl
PR1703	Java	Prof.Mona
PR1704	Java	Prof.Amit

SQL Function

MySQL comes bundled with a number of built in functions. Built in functions are simply functions come already implemented in the MySQL server. These functions allow us to perform different types of manipulations on the data. The built in functions can be basically categorized into the following most used categories.

- Strings functions - operate on string data types
- Numeric functions - operate on numeric data types
- Date functions - operate on date data types

Strings Function

Function	Description
UPPER(str)	Converts a string to upper-case
UCASE(str)	Converts a string to upper-case
LOWER(str)	Converts a string to lower-case
LCASE(str)	Converts a string to lower-case
LENGTH(str)	Returns the length of a string (in bytes)
CHAR_LENGTH(str)	Returns the length of a string (in characters)
REVERSE(str)	Reverses a string and returns the result
SPACE(n)	Returns a string of the specified number of space characters
REPEAT(str,count)	Repeats a string as many times as specified
SUBSTRING(string, start, length)	Extracts a substring from a string (starting at any position)
MID(str, startindex,len)	Extracts a substring from a string (starting at any position)
LEFT(string, number)	Extracts a number of characters from a string (starting from left)
RIGHT(str ,len)	Returns the rightmost len characters from the string
SUBSTRING_INDEX(str,delim,count)	Returns the substring from string before count occurrences of the delimiter delim.
REPLACE(str, fromstr,tostr)	Returns the string by replacing fromstr with tostr.
RPAD(str , len , padstr)	Returns the string right-padded with the string padstr to a length of len characters.
LPAD(str , len , padstr)	Returns the string str, left-padded with the string padstr to a length of len characters.
LTRIM(str)	Returns the string str with leading space characters removed.
RTRIM(str)	Returns the string str with trailing space characters removed.

TRIM(str)	Returns the string str by removing leading and trailing black spaces.
SOUNDEX(str)	Returns a phonetic representation of str.
INSTR(str1, str2)	Returns the position of the first occurrence of a string in another string
LOCATE(substr, string, start)	Returns the position of the first occurrence of a substring in a string
INSERT(string, position, number, string2)	Inserts a string within a string at the specified position and for a certain number of characters

Numeric Function

Function	Description
ABS(number)	Returns the absolute value of a number
AVG(expression)	Returns the average value of an expression
CEIL(number)	Returns the smallest integer value that is >= to a number
COUNT(expression)	Returns the number of records returned by a select query
DEGREES(number)	Converts a value in radians to degrees
EXP(number)	Returns e raised to the power of a specified number
FLOOR(number)	Returns the largest integer value that is <= to a number
GREATEST(arg1, arg2, arg3, ...)	Returns the greatest value of the list of arguments
LEAST(arg1, arg2, arg3, ...)	Returns the smallest value of the list of arguments
MAX(expression)	Returns the maximum value in a set of values
MIN(expression)	Returns the minimum value in a set of values
MOD(x, y)	Returns the remainder of a number divided by another number
PI()	Returns the value of PI
POW(x, y)	Returns the value of a number raised to the power of another number
ROUND(number, decimals)	Rounds a number to a specified number of decimal places
SIGN(number)	Returns the sign of a number
SQRT(number)	Returns the square root of a number
SUM(expression)	Calculates the sum of a set of values
TRUNCATE(number, decimals)	Truncates a number to the specified number of decimal places

DATE Function

Function	Description
ADDDATE(date, days)	Adds a time/date interval to a date and then returns the date
ADDTIME(datetime, addtime)	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE()	Returns the current date
CURRENT_DATE()	Returns the current date
CURTIME()	Returns the current time
CURRENT_TIME()	Returns the current time
NOW()	Returns the current date and time
CURRENT_TIMESTAMP()	Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS'
DATE(expr)	Extracts the date part of the date or datetime expr.
TIME(expr)	Extracts the time part of the time or datetime expr.
DATEDIFF(date1, date2)	It returns date1 – date2 expressed as a value in days.
DAY(date)	Returns the day of the month for date, in the range 1 to 31.
MONTH(date)	Returns the month for date, in the range 1 to 12.
MONTHNAME(date)	Returns the full name of the month for date.
YEAR(date)	Returns the year for date, in 4 digit.
QUARTER(date)	Returns the quarter of the year for date, in the range 1 to 4.
LAST_DAY(date)	Takes a date or datetime value and returns the corresponding value for the last day of the month.
DAYOFWEEK(date)	Returns the weekday index for date (1 = Sunday, 2 = Monday, ., 7 = Saturday).
DAYNAME(date)	Returns the name of the weekday for date.