**SQL Tutorial for Beginners: Learn SQL in 7 Days**

**Training Summary**

Databases can be found in almost all software applications. SQL is the standard language to query a database. This course will teach you database design. Also, it teaches you basic to advanced SQL.

**What Should I Know?**

The course is designed for SQL beginners. No prior DB experience is required.

**Database Fundamentals**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/introduction-to-database-sql.html) | What is a database? What is MySQL? |
| [**Tutorial**](https://www.guru99.com/introduction-to-mysql-workbench.html) | Introduction to MySQL & MySQL workbench |

**Database Design**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/database-design.html) | Introduction To Database Design |
| [**Tutorial**](https://www.guru99.com/database-normalization.html) | Learn Database Normalization with the help of a case study |
| [**Tutorial**](https://www.guru99.com/er-modeling.html) | Learn ER Modeling with a Case Study |

**SQL Basics**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/how-to-create-a-database.html) | How To Create A Database & MySQL DataTypes |
| [**Tutorial**](https://www.guru99.com/select-statement.html) | How to use SELECT in MySQL |
| [**Tutorial**](https://www.guru99.com/where-clause.html) | How to query data using Where clause in MySQL |
| [**Tutorial**](https://www.guru99.com/insert-into.html) | How to use Insert Into in MySQL |
| [**Tutorial**](https://www.guru99.com/delete-and-update.html) | How to Delete & Update data in MySQL |

**Data Sorting**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/order-by-desc-and-asc.html) | How sorting is done in MySQL using ORDER BY, DESC and ASC |
| [**Tutorial**](https://www.guru99.com/group-by.html) | How to use Group By in MySQL |
| [**Tutorial**](https://www.guru99.com/wildcards.html) | How to use Wildcards in MySQL |
| [**Tutorial**](https://www.guru99.com/regular-expressions.html) | Using Regular Expressions & Wild Cards in MySQL |

**Functions**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/functions.html) | Ultimate guide to Functions used in PHP |
| [**Tutorial**](https://www.guru99.com/aggregate-functions.html) | How to use Aggregate Function in MySQL |

**Must Know Stuff!**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/null.html) | All about Null value & Keyword in MySQL |
| [**Tutorial**](https://www.guru99.com/auto-increment.html) | How to use Auto Increment in MySQL |
| [**Tutorial**](https://www.guru99.com/alter-drop-rename.html) | How to use Alter, Drop & Rename function in MySQL |
| [**Tutorial**](https://www.guru99.com/limit.html) | How to use Limit keyword in MySQL |

**Most Dreaded Topics!**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/sub-queries.html) | Using Sub-Queries in MySQL |
| [**Tutorial**](https://www.guru99.com/joins.html) | How to use Joins in MySQL |
| [**Tutorial**](https://www.guru99.com/unions.html) | How to use Unions in MySQL |
| [**Tutorial**](https://www.guru99.com/views.html) | How to use Views in MySQL |
| [**Tutorial**](https://www.guru99.com/indexes.html) | How to use Index in MySQL |

**What Next!**

|  |  |
| --- | --- |
| [**Tutorial**](https://www.guru99.com/php-practical-example.html) | Creating an Application using MySQL |
| [**Tutorial**](https://www.guru99.com/mysql-certification-guide.html) | Oracle MySQL 5.6 Certification with Study Guide |
| [**Tutorial**](https://www.guru99.com/sql-vs-mysql.html) | SQL vs MySQL: What's the Difference? |
| [**Tutorial**](https://www.guru99.com/top-20-sql-management-tools.html) | 20 Best SQL Management Tools |
| [**Tutorial**](https://www.guru99.com/sql-interview-questions-answers.html) | Top 50 SQL Interview Questions & Answers |
| [**Tutorial**](https://www.guru99.com/sql-tutorial-pdf.html) | SQL Tutorial for Beginners PDF |

**What is a DBMS?**

A database management system (DBMS) is a software used to store and manage data. It guarantees the quality, durability, and confidentiality of information. The most popular type of DBMS are Relational Database Management Systems, or RDBMSs. Here, the database consists of a structured set of tables and each row of a table is a record.

**What is SQL?**

Structured Query Language (SQL) is the standard language for data manipulation in a DBMS. In in simple words its used to talk to the data in a DBMS. Following are types of SQL Statements

1. Data Definition Language (DDL) allows you to create objects like Schemas, Tables in the database
2. Data Control Language (DCL) allows you to manipulate and manage access rights on database objects
3. Data Manipulation Language (DML) is used for searching, inserting, updating, and deleting data, which will be partially covered in this programming tutorial.

**What is Query?**

A Query is a set of instruction given to the database management system. It tells any database what information you would like to get from the database. For example, to fetch the student name from the database table STUDENT, you can write the SQL Query like this:

SELECT Student\_name from STUDENT;

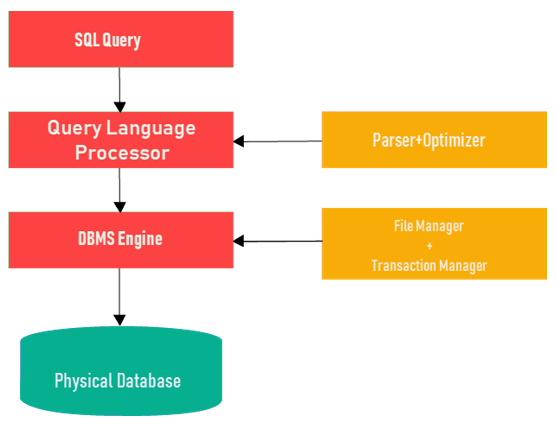
**SQL Process**

When you want to execute an SQL command for any DBMS system, you need to find the best method to carry out your request, and SQL engine determines how to interpret that specific task.

Important components included in this SQL process are:

* SQL Query Engine
* Optimization Engines
* Query Dispatcher
* Classic Query Engine

A classic query engine allows you to manage all the non-SQL queries.



SQL Process

**SQL Optimization**

Knowing how to make queries is not too difficult, but you need to really learn and understand how data storage works, and how queries are read in order to optimize SQL performance. Optimizations are based on two key factors:

1. Making the right choices when defining the database structure
2. Applying the most appropriate methods to read the data.

**What will you learn in this course?**

This course is designed for anyone planning to work with databases, especially in the roles of system administrators and application developers. The tutorials help beginners learn the basic SQL commands, including SELECT, INSERT INTO, UPDATE, DELETE FROM, and more. Each SQL command comes with clear and concise examples.

In addition to the list of SQL commands, the tutorial presents flashcards with SQL functions, such as AVG(), COUNT(), and MAX(). Along with these, quizzes help validate your basic knowledge of the language.

The tutorial can help you handle various aspects of the SQL programming language.

DATA BASE FUNDAMENTALS

**What is Database? What is SQL?**

**What You Will Learn:** [hide]

* [What is Data?](https://www.guru99.com/introduction-to-database-sql.html#2)
* [What is a database?](https://www.guru99.com/introduction-to-database-sql.html#3)
* [What is a Database Management System?](https://www.guru99.com/introduction-to-database-sql.html#4)
* [Types of DBMS](https://www.guru99.com/introduction-to-database-sql.html#5)
* [What is SQL?](https://www.guru99.com/introduction-to-database-sql.html#6)
* [What is NoSQL ?](https://www.guru99.com/introduction-to-database-sql.html#7)
* [Why it makes sense to learn SQL after NOSQL ?](https://www.guru99.com/introduction-to-database-sql.html#8)

Before we learn about database , let's understand -

**What is Data?**

In simple words data can be facts related to any object in consideration.

For example your name, age, height, weight, etc are some data related to you.

A picture , image , file , pdf etc can also be considered data.

**What is a Database?**

Database is a systematic collection of data. Databases support storage and  manipulation of data. Databases make data management easy. Let's discuss few examples.

An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.

Your electricity service provider is obviously using a database to manage billing , client related issues, to handle fault data, etc.

Let's also consider the facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more.

We can provide countless number of examples for usage of databases .

## What is a Database Management System (DBMS)?

Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of  data .

It also helps to control access to the  database.

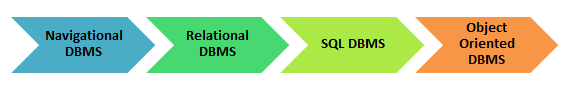
Database Management Systems are not a new concept and as such had been first implemented in 1960s.

Charles Bachmen's [Integrated Data Store](http://en.wikipedia.org/wiki/Integrated_Data_Store) (IDS) is said to be the first DBMS in history.

With time database technologies evolved a lot while usage and expected functionalities of databases have been increased immensely.

## Types of DBMS

Let's see how the DBMS family got evolved with the time. Following diagram shows the evolution of DBMS categories.



There are 4 major types of DBMS. Let's look into them in detail.

* **Hierarchical** - this type of DBMS employs the "parent-child" relationship of storing data. This type of DBMS is rarely used nowadays. Its structure is like a tree with nodes representing records and branches representing fields. The windows registry used in Windows XP is an example of a hierarchical database. Configuration settings are stored as tree structures with nodes.
* **Network DBMS** - this type of DBMS supports many-to many relations. This usually results in complex database structures.  RDM Server is an example of a database management system that implements the network model.
* **Relational DBMS** - this type of DBMS defines database relationships in form of tables, also known as relations. Unlike network DBMS, RDBMS does not support many to many relationships.Relational DBMS usually have pre-defined data types that they can support. This is the most popular DBMS type in the market. Examples of relational database management systems include MySQL, Oracle, and Microsoft SQL Server database.
* **Object Oriented Relation DBMS** - this type supports storage of new data types. The data to be stored is in form of objects. The objects to be stored in the database have attributes (i.e. gender, ager) and methods that define what to do with the data. PostgreSQL is an example of an object oriented relational DBMS.

**What is SQL?**

SQL stands for **Structured Query language,** pronounced as "S-Q-L" or sometimes as "See-Quel". SQL is the standard language for dealing with Relational Databases. SQL can be used to insert, search, update and delete database records. SQL can do lots of other operations including optimizing and maintenance of databases. Relational databases like MySQL Database, Oracle, Ms SQL server, Sybase, etc uses SQL.



**How to use sql syntaxes?**

SQL Example

SELECT \* FROM Members WHERE Age > 30

SQL syntaxes used in these databases are almost similar, except the fact that some are using few different syntaxes and even proprietary SQL syntaxes.

**What is NoSQL ?**

NoSQL is an upcoming category of Database Management Systems. Its main characteristic is its non-adherence to Relational Database Concepts. NOSQL means "Not only SQL".

Concept of NoSQL databases grew with internet giants such as Google, Facebook, Amazon etc who deal with gigantic volumes of data.

When you use relational database for massive volumes of data , the system starts getting slow in terms of response time.

To overcome this , we could of course "scale up" our systems by upgrading our existing hardware.

The alternative to the above problem would be to distribute our database load on multiple hosts as the load increases.

This is known as "scaling out".

 NOSQL database are **non-relational databases** that scale out better than relational databases and are designed with web applications in mind.

They do not use SQL to query the data and do not follow strict schemas like relational models.With NoSQL, ACID (Atomicity, Consistency, Isolation, Durability) features are not guaranteed always

**Why it makes sense to learn SQL after NOSQL ?**

With the advantages of NOSQL databases outlined above that scale out better than relational models, you might be thinking **why one would still want to learn about SQL database?**

Well, **NOSQL databases** are sort of highly specialized systems and have their special usage and limitations. NOSQL suit more for those who handles huge volumes of data. The vast majority, use relational databases and associated tools.

Relational databases have the following advantages over NOSQL databases;

* SQL(relational) databases have a mature data storage and management model . This is crucial for enterprise users.
* SQL databases support the notion of views which allow users to only see data that they are authorized to view. The data that they are not authorized to see is kept hidden from them.
* SQL databases support stored procedure sql which allow database developers to implement part of the business logic into the database.
* SQL databases have better security models compared to NoSQL databases.

The world has not deviated from use of relational databases. There is **growing** a demand for professionals who can handle relational databases. Thus learning databases and SQL still holds merit.

|  |  |
| --- | --- |
|  | **Summary**   * DBMS stands for Database Management System. * We have four major types of DBMSs namely Hierarchical, Network, Relational, Object Oriented * The most widely used DBMS is the relational model that saves data in table formats. It uses SQL as the standard query language * SQL language is used to Sql query a database * The database approach has many advantages when it comes to storing data compared to the traditional flat file based systems |

**MySQL Workbench Tutorial & MySQL Introduction**

**What is MySQL?**

MySQL is an open source relational database.

MySQL is cross platform which means it runs on a number of different platforms such as Windows, Linux, and Mac OS etc.

**In this tutorial, you will learn-**

* [What is MySQL?](https://www.guru99.com/introduction-to-mysql-workbench.html#1)
* [Why use MySQL?](https://www.guru99.com/introduction-to-mysql-workbench.html#2)
* [Introducing MySQL Workbench](https://www.guru99.com/introduction-to-mysql-workbench.html#3)
* [MySQL workbench- Modeling and Design tool](https://www.guru99.com/introduction-to-mysql-workbench.html#4)
* [MySQL workbench - SQL development tool](https://www.guru99.com/introduction-to-mysql-workbench.html#5)
* [MySQL workbench - Administration tool](https://www.guru99.com/introduction-to-mysql-workbench.html#6)
* [Install MySQL workbench Guide](https://www.guru99.com/introduction-to-mysql-workbench.html#7)

**Why use MySQL?**

There are a number of relational database management systems on the market.

Examples of relational databases include Microsoft SQL Server, Microsoft Access, Oracle, DB2 etc.

One may ask why we would choose MySQL over the other database management systems.

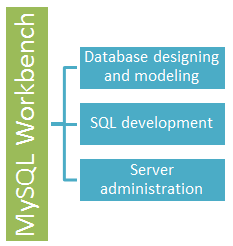
The answer to this question depends on a number of factors.

**Let's look at the strengths of MySQL compared to over relational databases such as SQL Server**-

* MySQL supports multiple storage engines each with its own specifications while other systems like SQL server only support a single storage engine. In order to appreciate this statement, let's look at two of the storage engines supported by MySQL.
* InnoDB: - its default storage engine provided with MySQL as of version 5.5. InnoDB supports foreign keys for referential integrity and also supports ACID-standard transactions.
* MyISAM: - it was the default storage engine for MySQL prior to version 5.5. MyISAM lacks support for transactions. Its advantages over InnoDB include simplicity and high performance.
* MySQL has high performance compared to other relation database systems. This is due to its simplicity in design and support for multiple-storage engines.
* Cost effective, it's relatively cheaper in terms of cost when compared to other relational databases. In fact, the community edition is free. The commercial edition has a licensing fee which is also cost effective compared to licensing fees for products such as Microsoft SQL Server.
* Cross platform - MySQL works on many platforms which means it can be deployed on most machines. Other systems such as MS SQL Server only run on the windows platform.

In order to interact with MySQL, you will need a **server access tool**that can communicate with MySQL server. MySQL supports multiple user connections.

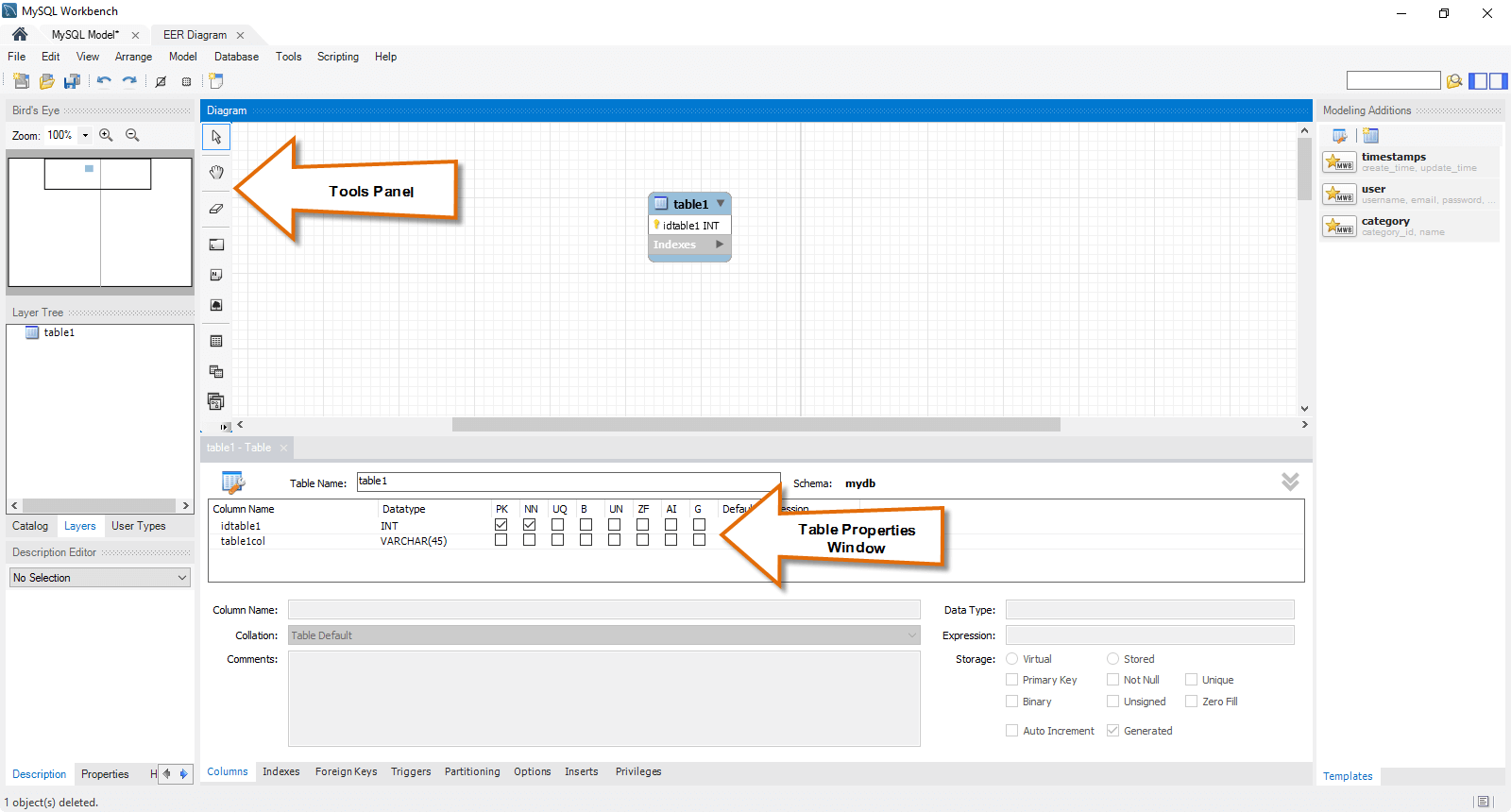
**Introducing MySQL Workbench**

MySQLWorkbench is a **Visual database designing and modeling** access tool for MySQL server relational database. It facilitates creation of new physical data models and modification of existing MySQL databases with reverse/forward engineering and change management functions.

**Getting Started MySQL workbench- Modeling and Design tool**

* Models are at the core of most valid and high performance databases. MySQLworkbench has tools that allow developers and database administrators visually create physical database design models that can be easily translated into MySQL databases using forward engineering.
* MySQL workbench supports creation of multiple models in the same environment.
* It supports all objects such as tables, views, stored procedures, triggers,  etc. that make up a database.
* MySQL workbench has a built in model validating utility that reports any issues that might be found to the data modeler.
* It also allows for different modeling notations and can be extended by using LUA a scripting language.

The figure shown below shows the modeling window for MySQLWorkbench.

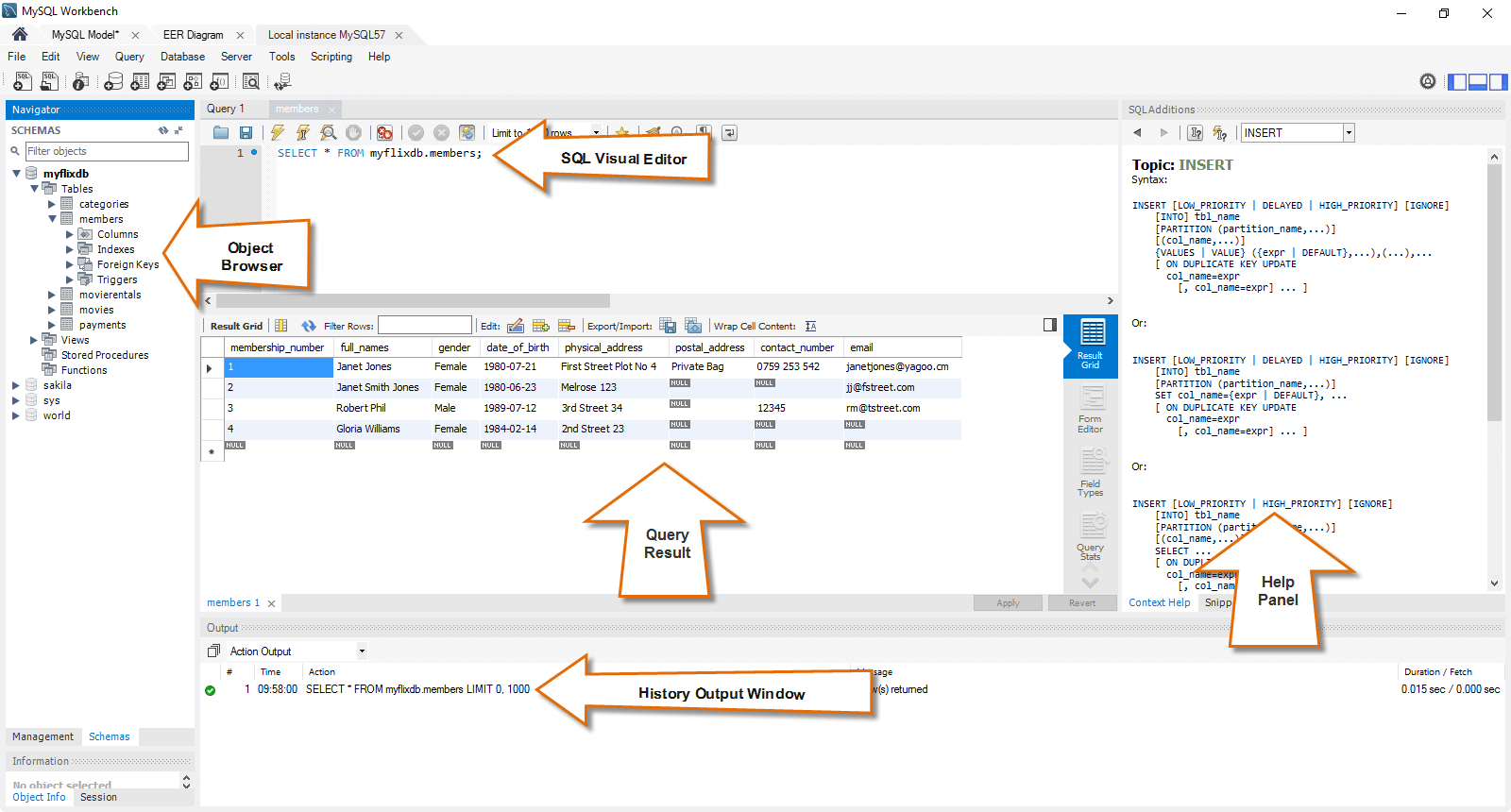


**MySQL workbench - SQL development tool**

Structured Query Language (SQL) allows us to manipulate our relational databases. SQL is at the heart of all relational databases.

* MySQLworkbench, has built in SQL visual editor.
* The Visual SQL editor allows developers to build, edit and run queries against MySQL server databases. It has utilities for viewing data and exporting it.
* Its syntax color highlighters help developers easily write and debug SQL statements.
* Multiple queries can be run and results automatically displayed in different tabs.
* The queries are also saved in the history panel for later retrieval and running.

The figure shown below shows the SQL development window for MySQL Workbench.

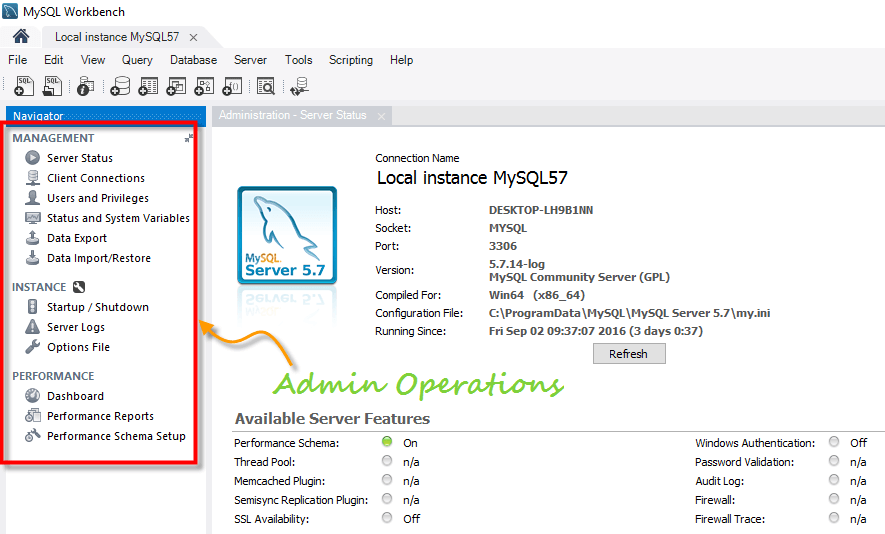


**MySQL workbench - Administration tool**

Server administration plays a critical role in securing the data of the company. The major issues concerning server administration are users' management, server configuration, server logs and many more. Workbench MySQL has the following features that simplify the process of MySQL server administration;

* **User administration** - visual utility for managing users that lets database administrators easily add new and remove existing users if need arises, grant and drop privileges and view user profiles.
* **Server configuration** - allows for advanced configuration of the server and fine tuning for optimal performance.
* **Database backup and restorations** - visual tool for exporting/importing MySQL dump files. MySQL dump files contain SQL scripts for creating databases, tables, views, stored procedures and insertion of data.
* **Server logs** - visual tool for viewing MySQL server logs. The logs include error logs, binary logs and InnodDB logs. These logs come in handy when performing diagnosis on the server. The figure shown below shows the modeling window for MySQL Workbench.

The figure shown below shows  the Admin panel for Workbench MySQL .



## Install MySQL workbench Guide(for Windows)

This is a 2 step process

1)      Install [MySQL Community Server](http://dev.mysql.com/downloads/mysql/)

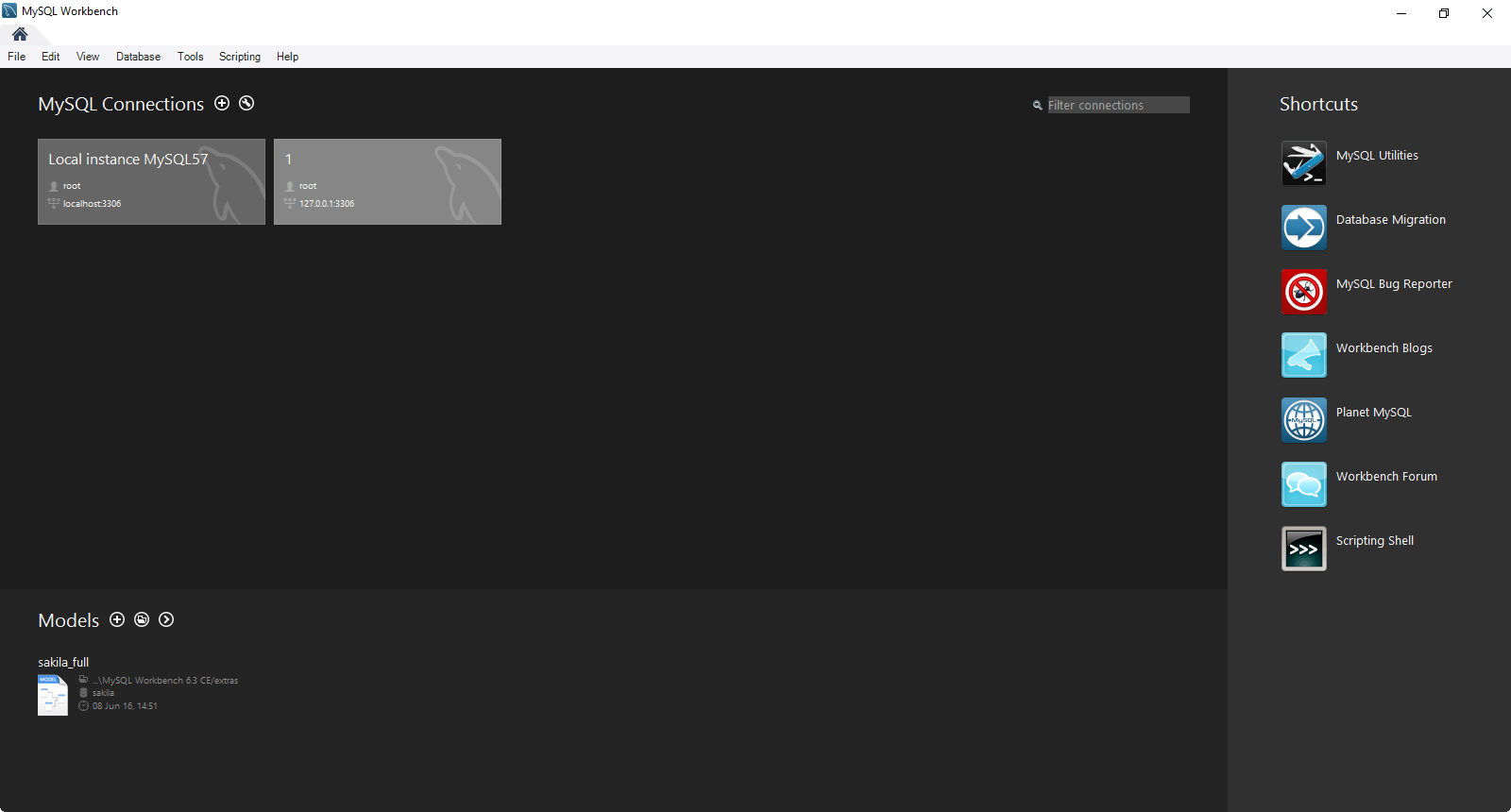
2)      Install [MySQL workbench](http://dev.mysql.com/downloads/workbench/). - You can install the workbench using a zip file or an msi installer (recommended)

Note: You will require Administrator or Power User Privileges to perform installation.

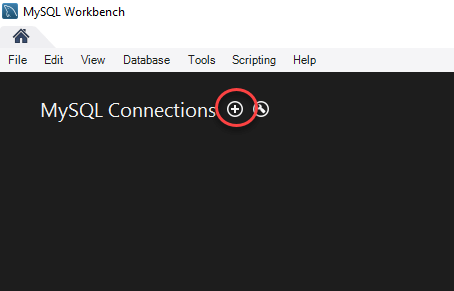
**Getting Started**

Once you have finished installing above you need to set up MySQL Workbench as shown below -

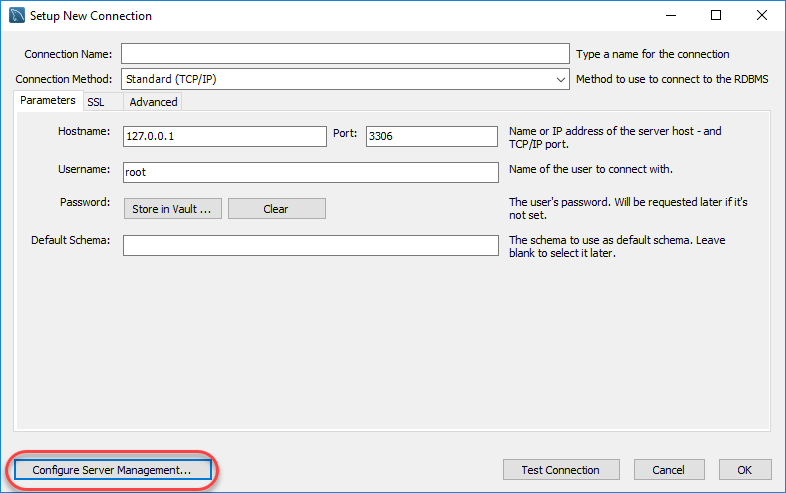
**1.**First step is launching the Workbench MySQL . What you see is called **Home Window**



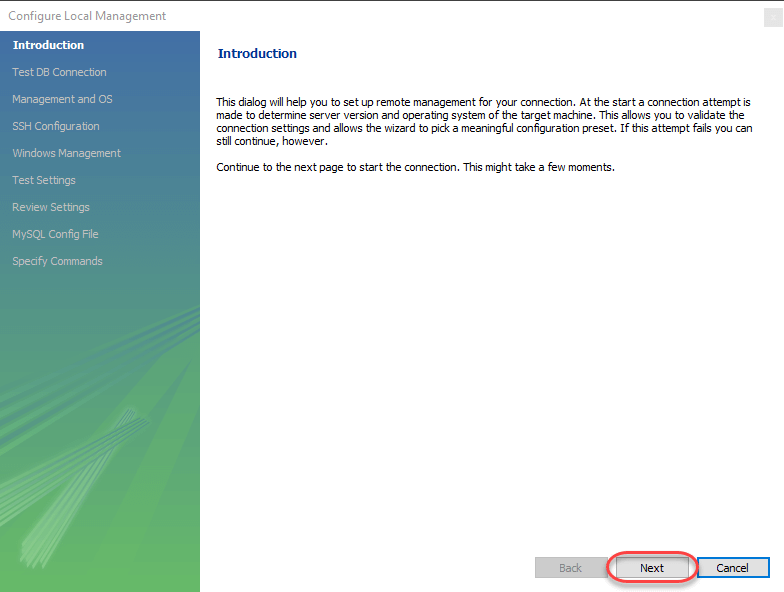
2. Next you need to create your MySQL Server Connection which contains details about target database server including how to connect to it. Click **" +  "** in MySQL Workbench Home Window. This will open **Setup New Connection**. Wizard



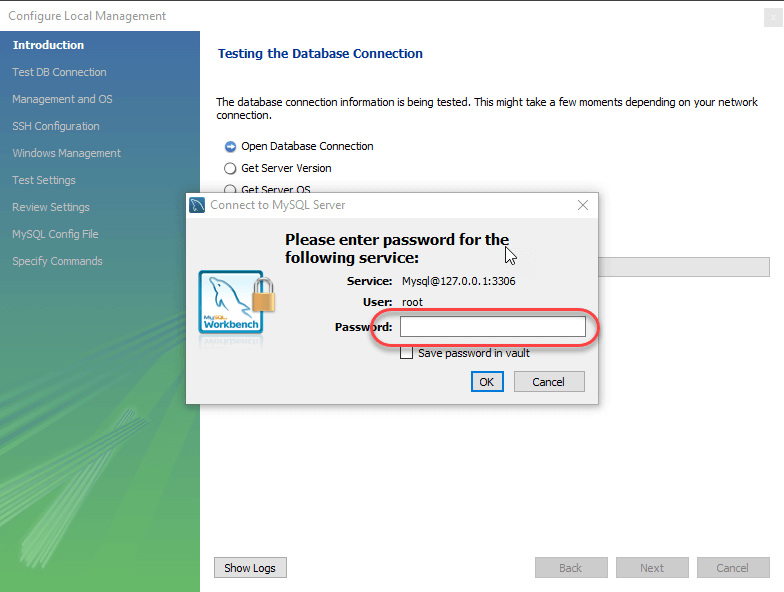
1. As a beginner you can create a connection for a locally installed server. Click **Configure Server Management**button in **Setup New Connection** window to check the cofiguration of the MySQL server.



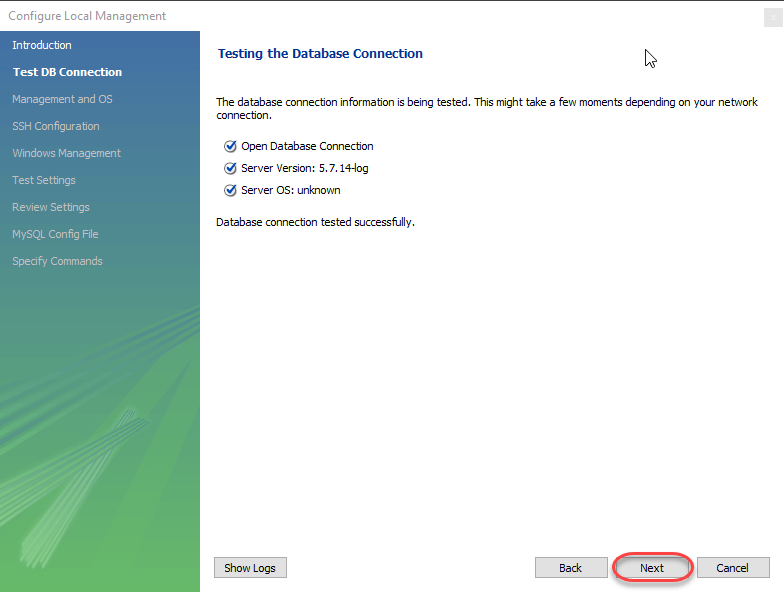
4. A new window opens named **Configure Local Management** . Click Next button to continue.

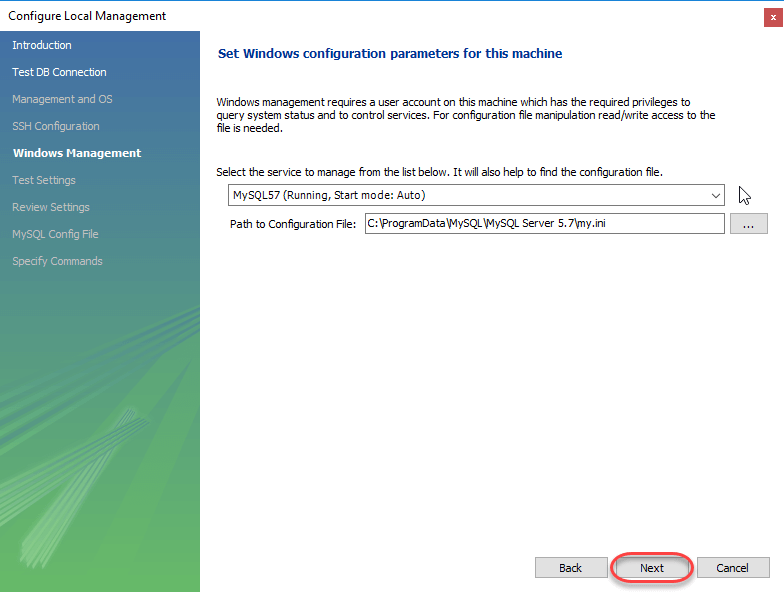


1. Next the Wizard will test connections to database. If test fails, go back and correct database connection parameters.5. Next it will open a pop up window asking your root password to test your connection with the local mysql server instance. The password is the one you set during installation of MySQL Workbench. Enter your password and press **OK**

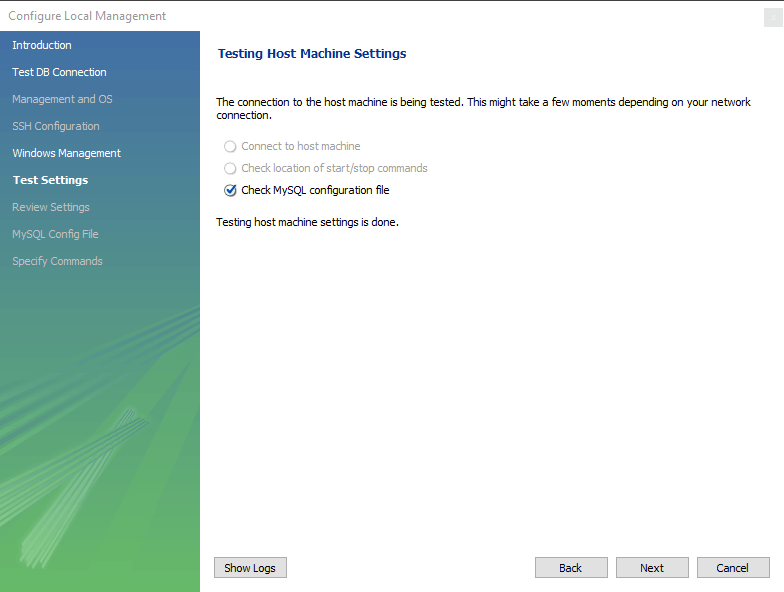


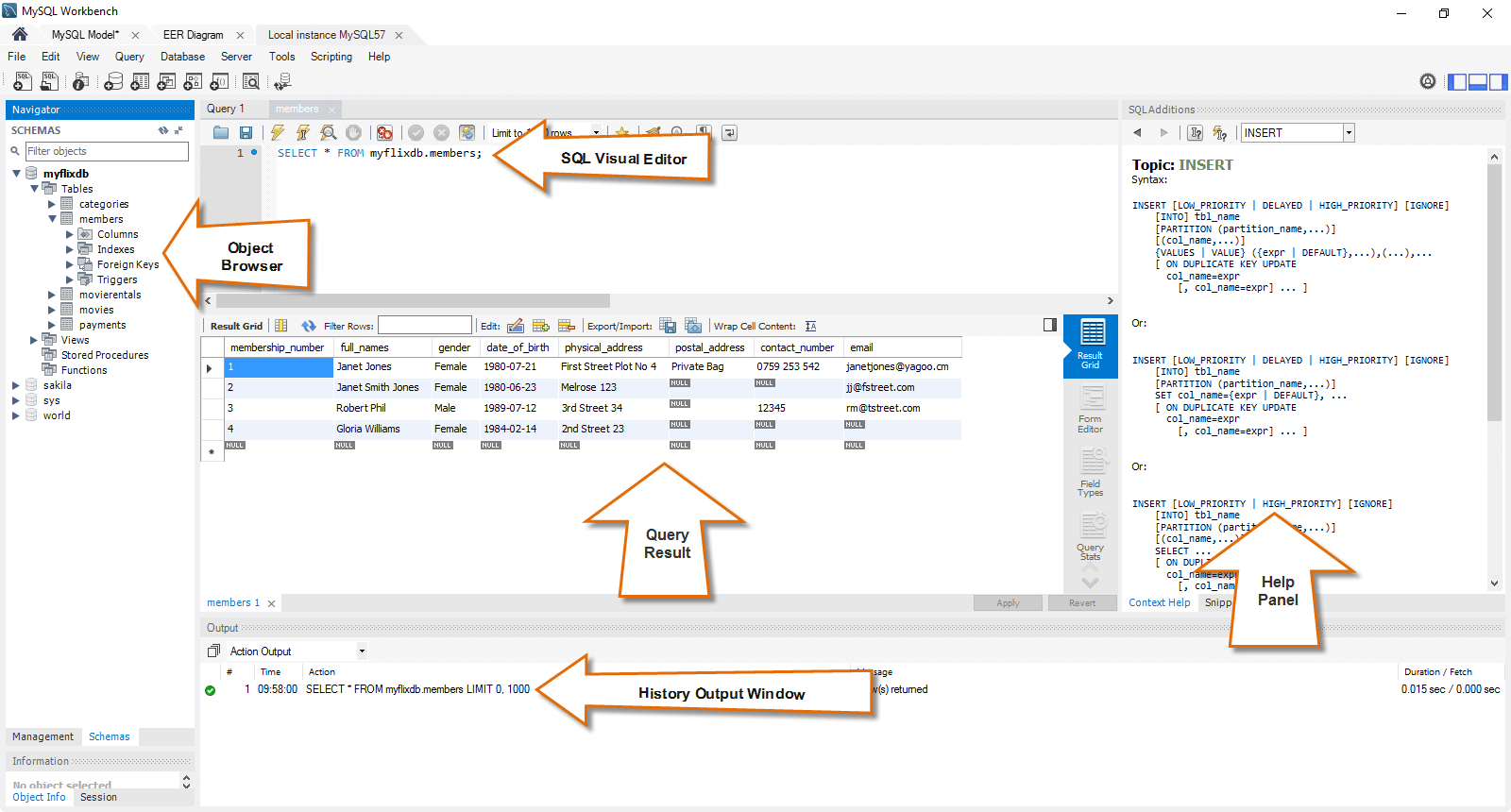
1. Next the Wizard will test connections to database. If test fails, go back and correct database connection parameters. Else if all tests are sucessful click Next to continue.

7. After that a new wizard will open about Local Service Management - It lets you switch between multiple mysql severs installed on one machines. As a beginner you can bypass this and click **Next** to continue.

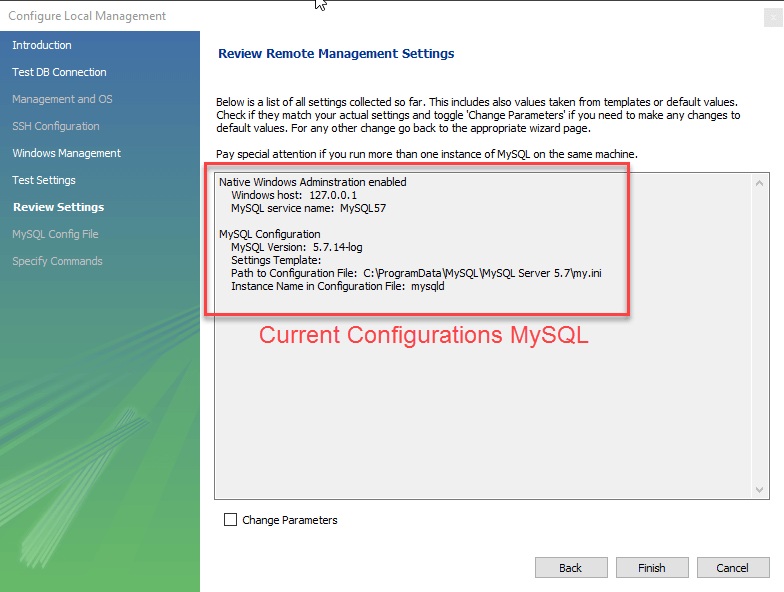


8. The Wizard will then check ability to access MySQL Server Configuration File, and test start/stop commands.

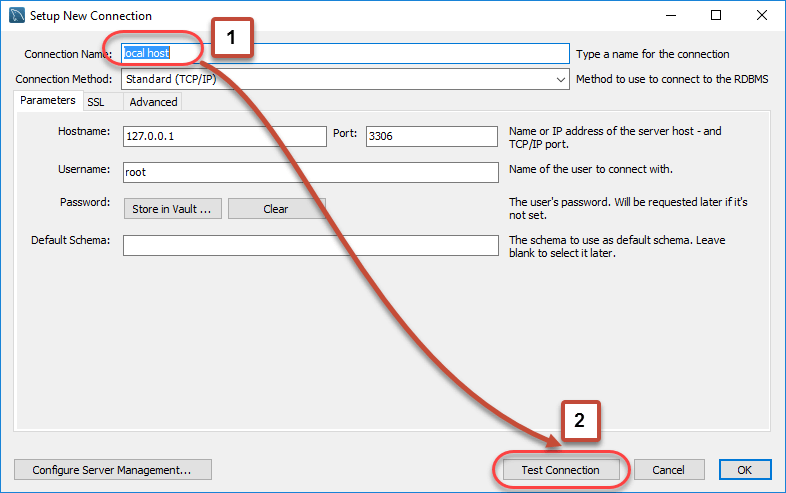




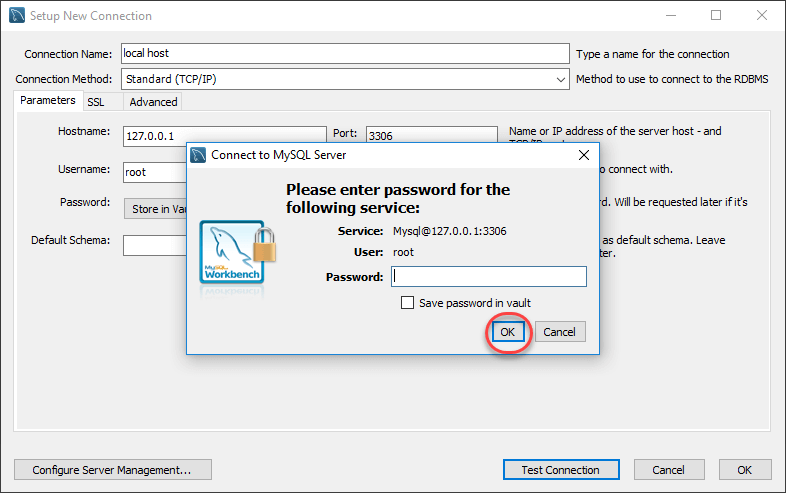
9. Next you can review current configurations. After reviewing the configurations, Click Finish to finsh server configuration



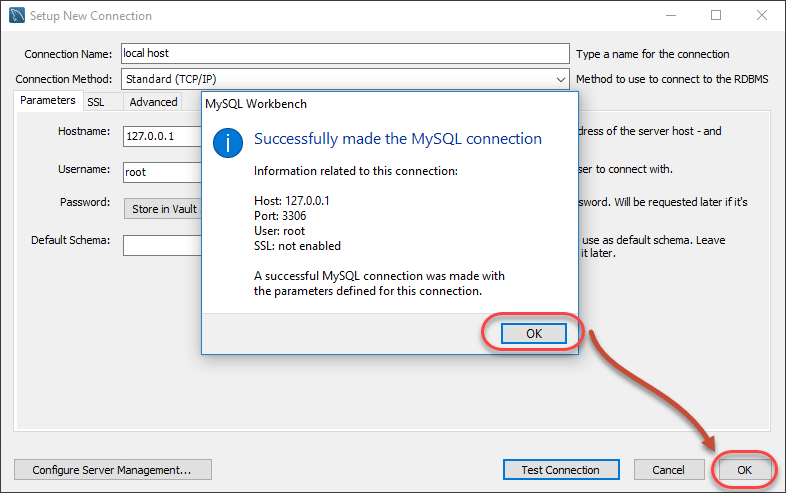
10. Next Step is to setup a connection, which can be used to connect to server. If you have not created a connection already, you can use the default values given. Click on Test Connection [ 2 ] after entering the Connection Name [ 1 ].



11. A new dialog box will open asking you password to root/selected user. If your MySQL root user has a password, you can enter that using Store in Vault feature. Click OK.



If the entered password for the user is correct then the following screen will show. Click on **both OK** buttons and you will be good to go.



A new instance is shown in the homepage



Double click and start querying.

**Summary**

* MySQL is an open source relational database that is cross platform.
* MySQL supports multiple storage engines which greatly improve the server performance tuning and flexibility. Prior to version 5.5, the default storage engine was MyISAM which lacked support for transactions, as of version 5.5; the default storage engine is InnoDB which supports transactions and foreign keys.
* MySQL server can be administered using a number of server access mysql tools which include both commercial and open source products. Popular examples include;
* **phpMyAdmin** - cross platform web based open source server access tool
* **SQLYog** - targeted at the windows platform, desktop commercial server access tool
* **MySQL** **workbench** - cross platform open source server access tool.
* MySQL workbench is an integrated development environment for MySQL server. It has utilities for database modeling and designing, SQL development and server administration.

**Database Design Tutorial: Learn Data Modeling**

**What is Database Design?**

**Database Design** is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

 The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

 The physical data design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).

**In this tutorial, you will learn-**

* [Why Database Design is Important ?](https://www.guru99.com/database-design.html#2)
* [Database development life cycle](https://www.guru99.com/database-design.html#3)
* [Requirements analysis](https://www.guru99.com/database-design.html#4)
* [Database designing](https://www.guru99.com/database-design.html#5)
* [Implementation](https://www.guru99.com/database-design.html#6)
* [Types of Database Techniques](https://www.guru99.com/database-design.html#7)

**Why Database Design is Important ?**

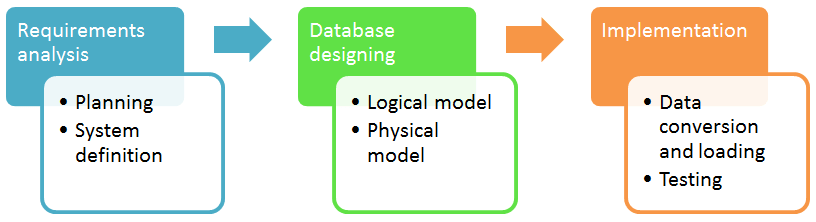
It helps produce  database systems

1. That meet the requirements of the users
2. Have high performance.

Database designing is crucial to **high performance** database system.

Note , the genius of a database is in its design . Data operations using SQL is relatively simple

**Database development life cycle**



The database development life cycle has a number of stages that are followed when developing database systems.

The steps in the development life cycle do not necessary have to be followed religiously in a sequential manner.

On small database systems, the database system development life cycle is usually very simple and does not involve a lot of steps.

In order to fully appreciate the above diagram, let's look at the individual components listed in each step.

**Requirements analysis**

* **Planning** - This stages concerns with planning of entire Database Development Life Cycle  It  takes into consideration the Information Systems strategy of the organization.
* **System definition** - This stage defines the scope and boundaries of the proposed database system.

**Database designing**

* **Logical model** - This stage is concerned with developing a database model based on requirements. The entire design is on paper without any physical implementations or specific DBMS considerations.
* **Physical model** - This stage implements the logical model of the database taking into account the DBMS and physical implementation factors.

**Implementation**

* **Data conversion and loading** - this stage is concerned with importing and converting data from the old system into the new database.
* **Testing** - this stage is concerned with the identification of errors  in the newly implemented system .It checks the database against requirement specifications.

**Two Types of Database Techniques**

1. **Normalization**
2. **ER Modeling**

Let's study them one by one

**What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples**

**What is Normalization?**

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

It divides larger tables to smaller tables and links them using relationships.

In this tutorial, you will learn-

* [Database Normal Forms](https://www.guru99.com/database-normalization.html#9)
* [1NF Rules](https://www.guru99.com/database-normalization.html#2)
* [What is a KEY?](https://www.guru99.com/database-normalization.html#3)
* [What is Composite Key](https://www.guru99.com/database-normalization.html#4)
* [2NF Rules](https://www.guru99.com/database-normalization.html#10)
* [Database - Foreign Key](https://www.guru99.com/database-normalization.html#5)
* [What are transitive functional dependencies?](https://www.guru99.com/database-normalization.html#6)
* [3NF Rules](https://www.guru99.com/database-normalization.html#7)
* [Boyce-Codd Normal Form (BCNF)](https://www.guru99.com/database-normalization.html#8)

The inventor of the relational model Edgar Codd proposed the theory of normalization with the introduction of First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined with Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Theory of Data Normalization in SQL is still being developed further. For example, there are discussions even on 6th Normal Form. **However, in most practical applications, normalization achieves its best in 3rd Normal Form**. The evolution of Normalization theories is illustrated below-

[What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples](https://www.guru99.com/images/NormalizationProcess(1).png)

**Database Normalization Examples -**

Assume a video library maintains a database of movies rented out. Without any normalization, all information is stored in one table as shown below.

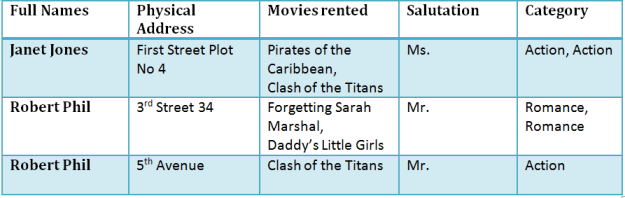


TABLE 1

Here you see **Movies Rented column has multiple values**.

**Database Normal Forms**

Now let's move into 1st Normal Forms

**1NF (First Normal Form) Rules**

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

The above table in 1NF-

**1NF Example**

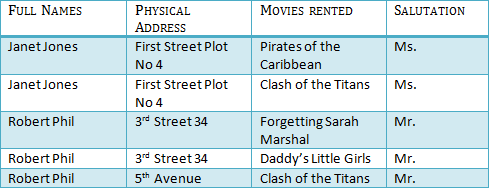


TABLE 1: In 1NFForm

Before we proceed let's understand a few things --

**What is a KEY?**

A KEY is a value used to identify a record in a table uniquely. A KEY could be a single column or combination of multiple columns

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

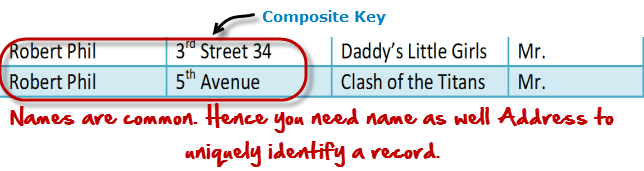
What is a Primary Key?

|  |  |
| --- | --- |
| [What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples](https://www.guru99.com/images/PrimaryKey.png) | A primary is a single column value used to identify a database record uniquely.  It has following attributes   * A primary key cannot be NULL * A primary key value must be unique * The primary key values should rarely be changed * The primary key must be given a value when a new record is inserted. |

**What is Composite Key?**

A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name Robert Phil, but they live in different places.



Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Let's move into second normal form 2NF

**2NF (Second Normal Form) Rules**

* Rule 1- Be in 1NF
* Rule 2- Single Column Primary Key

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

TABLE 1



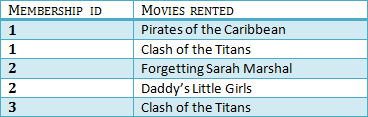
TABLE 2

We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

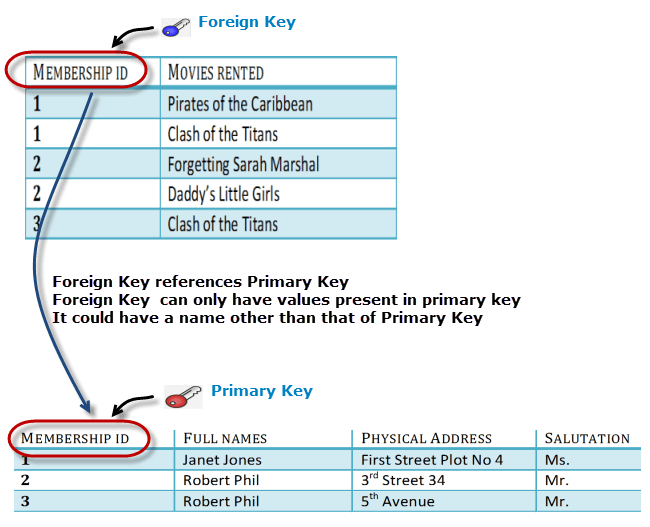
We have introduced a new column called Membership\_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

## Database - Foreign Key

In Table 2, Membership\_ID is the Foreign Key



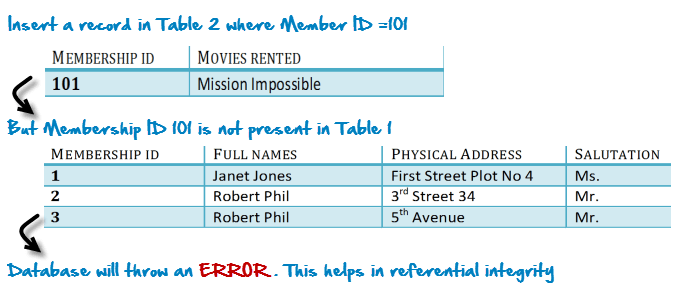
|  |  |
| --- | --- |
| [What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples](https://www.guru99.com/images/ForeignKey.png) | Foreign Key references the primary key of another Table! It helps connect your Tables   * A foreign key can have a different name from its primary key * It ensures rows in one table have corresponding rows in another * Unlike the Primary key, they do not have to be unique. Most often they aren't * Foreign keys can be null even though primary keys can not |



#### Why do you need a foreign key?

Suppose, a novice inserts a record in Table B such as

You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.



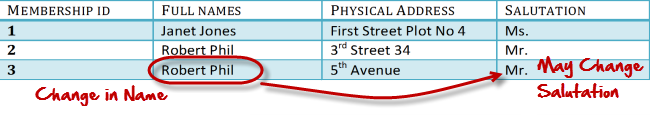
The above problem can be overcome by declaring membership id  from Table2  as foreign key of membership id from Table1

Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

## What are transitive functional dependencies?

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.

Let's move into 3NF

**3NF (Third Normal Form) Rules**

* Rule 1- Be in 2NF
* Rule 2- Has no transitive functional dependencies

To move our 2NF table into 3NF, we again need to again divide our table.

**3NF Example**





TABLE 1



TABLE 2



TABLE 3

We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3

Now our little example is at a level that cannot further be decomposed to attain higher forms of normalization. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases.  However, we will be discussing next levels of normalizations in brief in the following.

**Boyce-Codd Normal Form (BCNF)**

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate**Key.

Sometimes is BCNF is also referred as **3.5 Normal Form.**

**4NF (Fourth Normal Form) Rules**

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

**5NF (Fifth Normal Form) Rules**

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

**6NF (Sixth Normal Form) Proposed**

6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future...

That's all to Normalization!!!

**Summary**

* Database designing is critical to the successful implementation of a database management system that meets the data requirements of an enterprise system.
* Normalization helps produce database systems that are cost-effective and have better security models.
* Functional dependencies are a very important component of the normalize data process
* Most database systems are normalized database up to the third normal forms.
* A primary key uniquely identifies are record in a Table and cannot be null
* A foreign key helps connect table and references a primary key

**What is ER Modeling? Learn with Example**

**What is ER Modeling?**

**Entity Relationship Modeling**(ER Modeling) is a graphical approach to database design. It uses Entity/Relationship to represent real world objects.

An **Entity**is a thing or object in real world that is distinguishable from surrounding environment. For example each employee of an organization is a separate entity. Following are some of major characteristics of entities.

* An entity has a set of properties.
* Entity properties can  have values.

**In this tutorial, you will learn-**

* [What is ER Modeling?](https://www.guru99.com/er-modeling.html#1)
* [Enhanced Entity Relationship (EER) Model](https://www.guru99.com/er-modeling.html#2)
* [Why use ER Model?](https://www.guru99.com/er-modeling.html#3)
* [Entities in the "MyFlix" library](https://www.guru99.com/er-modeling.html#4)
* [Defining the relationships among entities](https://www.guru99.com/er-modeling.html#5)

Let's consider our first example again. An employee of an organization is an entity. If "Peter" is a programmer (an **employee**) at Microsoft, he can have **attributes (**properties) like name, age, weight, height, etc. It is obvious that those do hold values relevant to him.

Each attribute can have **Values**. In most cases single attribute have one value. But it is possible for attributes have **multiple values** also.  For example Peter's age has a single value. But his "phone numbers" property can have multiple values.

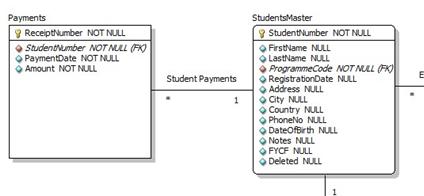
Entities can have **relationships**with each other. Let's consider a simplest example. Assume that each Microsoft Programmer is given a Computer. It is clear that that **Peter's Computer**is also an entity. Peter is using that computer and the same computer is used by Peter. In other words there is a mutual relationship among Peter and his computer.

In **Entity Relationship Modeling,**we model entities, their attributes and relationships among entities.

**Enhanced Entity Relationship (EER) Model**

Enhanced Entity Relationship (EER) Model is a high level data model which provides extensions to original **Entity Relationship**(ER) model. EER Models supports more details design. EER Modeling emerged as a solution for modeling highly complex databases.

**EER uses UML notation.**UML is the acronym for Unified Modeling Language; it is a general purpose modeling language used when designing object oriented systems. Entities are represented as class diagrams. Relationships are represented as associations between entities. The diagram shown below illustrates an ER diagram using the UML notation.



**Why use ER Model?**

Now you may think why use ER modeling when we can simply create the database and all of its objects without ER modeling? One of the challenges faced when designing database is the fact that designers, developers and end-users tend to view data and its usage differently. If this situation is left unchecked, we can end up producing a database system that does not meet the requirements of the users.

Communication tools understood by all stakeholders(technical as well non-technical users) are critical in producing database systems that meet the requirements of the users. ER models are examples of such tools.

ER diagrams also increase user productivity as they can be easily translated into relational tables.

Case Study: ER diagram for "MyFlix" Video Library

Let's now work with the MyFlix Video Library database system to help understand the concept of ER diagrams. We will using this database for all hand-on in the remainder of this tutorials

MyFlix is a business entity that rents out movies to its members. MyFlix has been storing its records manually.  The management now wants to move to a DBMS

Let's look at the steps to develop EER diagram for this database-

1. Identify the entities and determine the relationships that exist among them.
2. Each entity, attribute and relationship, should have appropriate names that can be easily understood by the non-technical people as well.
3. Relationships should not be connected directly to eachother. Relationships should connect entities.
4. Each attribute in a given entity should have a unique name.

**Entities in the "MyFlix" library**

The entities to be included in our ER diagram are;

* **Members** - this entity will hold member information.
* **Movies** - this entity will hold information regarding movies
* **Categories** - this entity will hold information that places movies into different categories such as "Drama", "Action", and "Epic" etc.
* **Movie Rentals** - this entity will hold information that about movies rented out to members.
* **Payments** - this entity will hold information about the payments made by members.

**Defining the relationships among entities**

**Members and movies**

The following holds true regarding the interactions between the two entities.

* A member can rent a more than movie in a given period.
* A movie can be rented by more than one member in a given period.

From the above scenario, we can see that the nature of the relationship is many-to-many. **Relational databases do not support many-to-many relationships. We need to introduce a junction entity**. This is the role that the MovieRentals entity plays. It has a one-to-many relationship with the members table and another one-to-many relationship with movies table.

**Movies and categories entities**

The following holds true about movies and categories.

* A movie can only belong to one category but a category can have more than one movie.

We can deduce from this that the nature of the relation between categories and movies table is one-to-many.

**Members and payments entities**

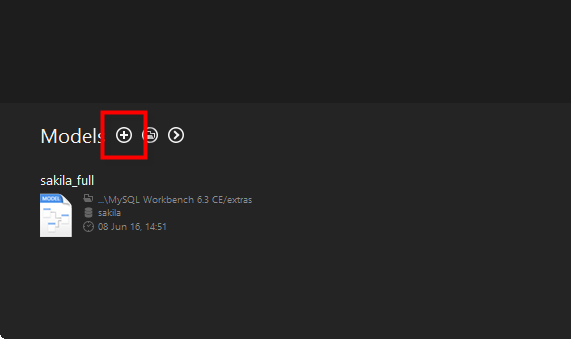
The following holds true about members and payments

* A member can only have one account but can make a number of payments.

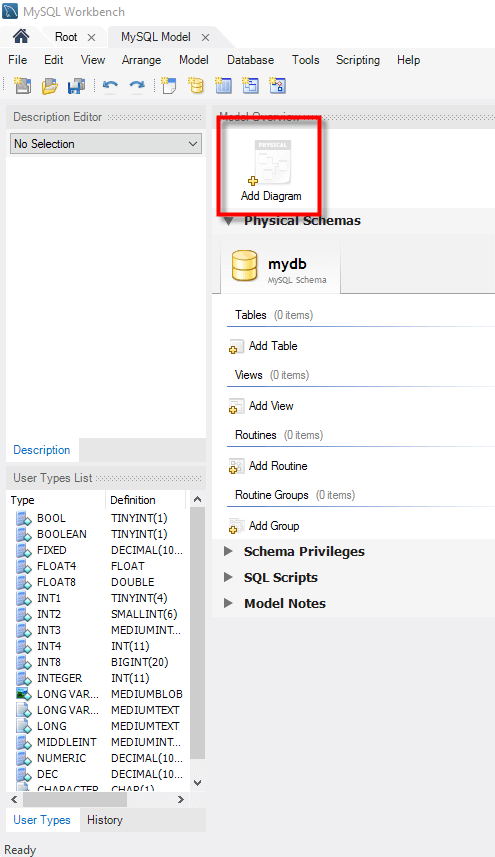
We can deduce from this that the nature of the relationship between members and payments entities is one-to-many.

Now lets create EER model using MySQL Workbench

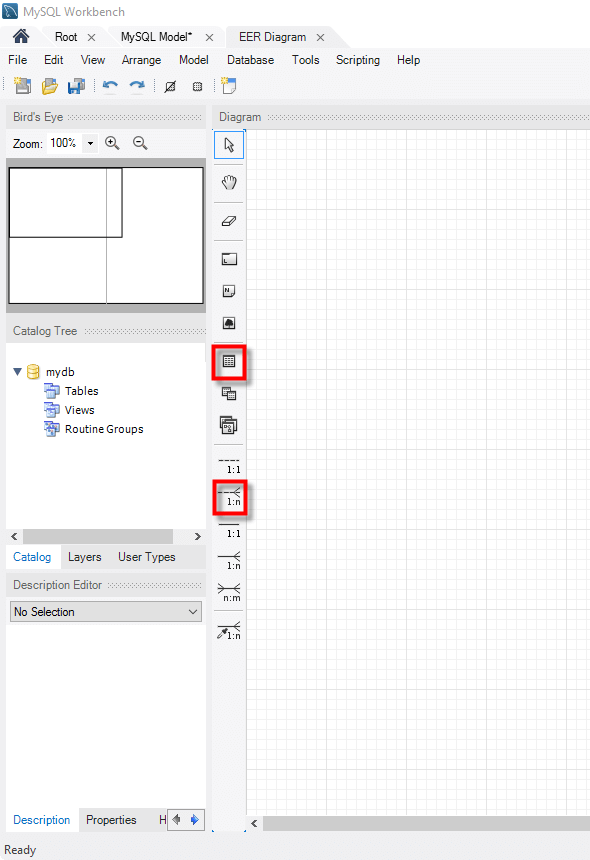
In the MySQL workbench , Click - "+" Button



Double click on Add Diagram button to open the workspace for ER diagrams.



Following window appears



Let's look at the two objects that we will work with.

* [What is ER Modeling? Learn with Example](https://www.guru99.com/images/table_tool.png)The table object allows us to create entities and define the attributes associated with the particular entity.
* [What is ER Modeling? Learn with Example](https://www.guru99.com/images/relationship_tool.png)The place relationship button allows us to define relationships between entities.

The **members'** entity will have the following attributes

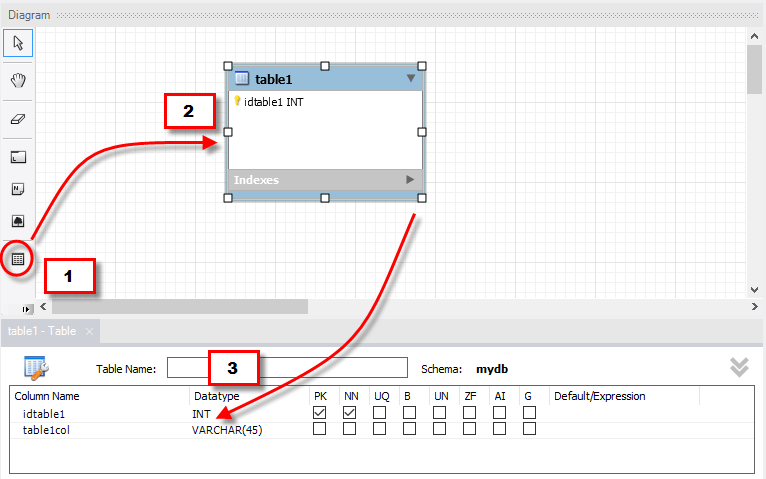
* Membership number
* Full names
* Gender
* Date of birth
* Physical address
* Postal address

**Let's now create the members table**

1.Drag the table object from the tools panel

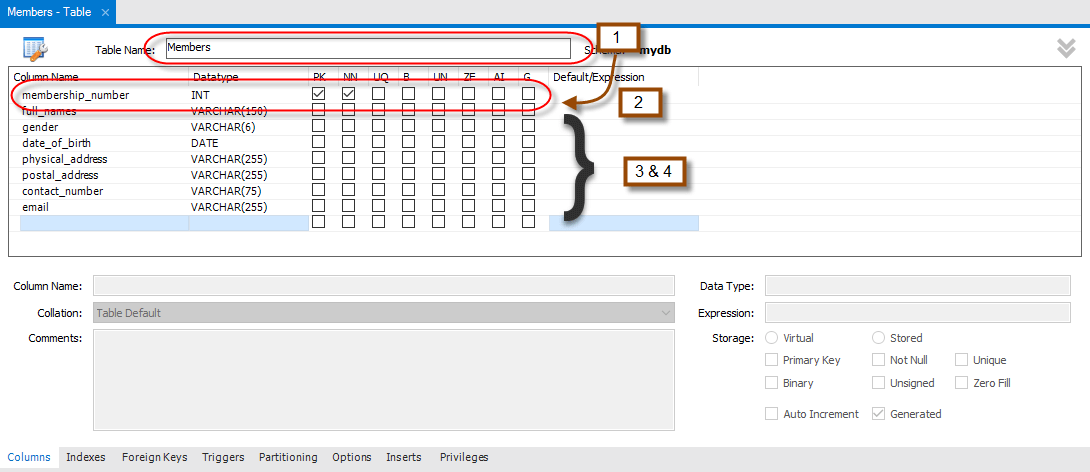
2.Drop it in the workspace area. An entity named table 1 appears

3.Double click on it. The properties window shown below appears

Next ,

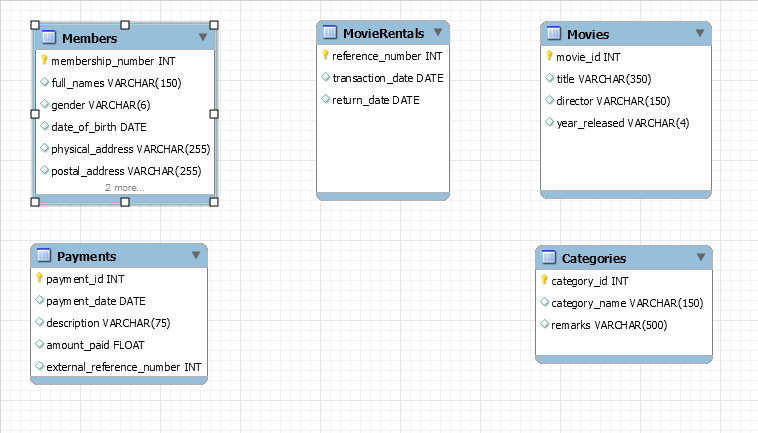
1. Change table 1 to Members
2. Edit the default idtable1 to membership\_number
3. Click on the next line to add the next field
4. Do the same for all the attributes identified in members' entity.

Your properties window should now look like this.

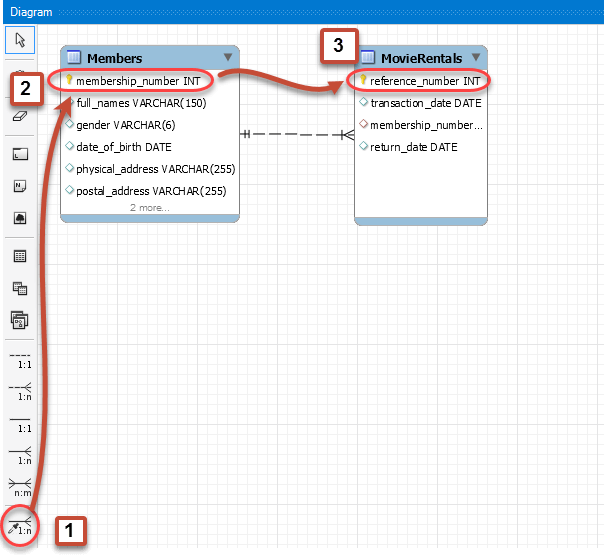


Repeat the above steps for all the identified entities.

Your diagram workspace should now look like the one shown below.

**Lets create relationship between Members and Movie Rentals**

1. Select the place relationship using existing columns too
2. Click on membership\_number in the Members table
3. Click on reference\_number in the MovieRentals table



### Repeat above steps for other relationships. Your ER diagram should now look like this -https://www.guru99.com/images/myflix_erd.pngSummary

* ER Diagrams play a very important role in the database designing process. They serve as a non-technical communication tool for technical and non-technical people.
* Entities represent real world things; they can be conceptual as a sales order or physical such as a customer.
* All entities must be given unique names.
* ER models also allow the database designers to identify and define the relations that exist among entities.

The entire ER Model is attached below. You can simple import it in MySQL Workbench

**MySQL Create Database, Tables, Data Types**

**Steps for Create Database Mysql**

**Create Database in two ways**

1) By executing a simple SQL query

2) By using forward engineering in MySQL Workbench

**In this tutorial, you will learn-**

* [Create Database](https://www.guru99.com/how-to-create-a-database.html#1)
* [Creating Tables MySQL](https://www.guru99.com/how-to-create-a-database.html#2)
* [Data types](https://www.guru99.com/how-to-create-a-database.html#3)
* [MySQL workbench ER diagram forward Engineering](https://www.guru99.com/how-to-create-a-database.html#4)

As SQL beginner , let's look into the query method first.

**Create Database**

CREATE DATABASE is the SQL command for creating a database.

Imagine you need to create a database with name "movies". You can do it by executing following SQL command.

CREATE DATABASE movies;

**Note: you can also use the command CREATE SCHEMA instead of CREATE DATABASE**

Now let's improve our SQL query adding more parameters and specifications.

**IF NOT EXISTS**

A single MySQL server could have multiple databases. If you are not the only one accessing the same MySQL server or if you have to deal with multiple databases there is a probability of attempting to create a new database with name of an existing database .  **IF NOT EXISTS**let you to instruct MySQL server to check the existence of a database with a similar name prior to creating database.

When **IF NOT EXISTS**is used database is created only if given name does not conflict with an existing database's name. Without the use of **IF NOT EXISTS**MySQL throws an error.

CREATE DATABASEIF NOT EXISTS movies;

**Collation and Character Set**

**Collation** is set of **rules** **used in comparison.**Many people use MySQL to store data other than English. Data is stored in MySQL using a specific character set. The character set can be defined at different levels viz, server , database , table and columns.

You need to select the rules of collation which in turn depend on the character set chosen.

For instance, the Latin1 character set uses the latin1\_swedish\_ci collation which is the Swedish case insensitive order.

CREATE DATABASE IF NOT EXISTS movies CHARACTER SET latin1 COLLATE latin1\_swedish\_ci

The best practice while using local languages like Arabic , Chinese etc is to select Unicode (utf-8) character set which has several collations or just stick to default collation utf8-general-ci.

You can find the list of all collations and character sets here [here](http://dev.mysql.com/doc/refman/5.5/en/charset-charsets.html)

You can see list of existing databases by running following SQL command.

SHOW DATABASES

**Creating Tables MySQL**



Tables can be created using **CREATE TABLE**statement and it actually has the following syntax.

CREATE TABLE [IF NOT EXISTS] `TableName` (`fieldname` dataType [optional parameters]) ENGINE = storage Engine;

**HERE**

* "CREATE TABLE" is the one responsible for the creation of the table in the database.
* "[IF NOT EXISTS]" is optional and only create the table if no matching table name is found.
* "`fieldName`" is the name of the field and "data Type" defines the nature of the data to be stored in the field.
* "[optional parameters]" additional information about a field such as "  AUTO\_INCREMENT" , NOT NULL etc

**Create Table Example:-**

CREATE TABLE IF NOT EXISTS `MyFlixDB`.`Members` (

`membership\_number` INT AUTOINCREMENT ,

`full\_names` VARCHAR(150) NOT NULL ,

`gender` VARCHAR(6) ,

`date\_of\_birth` DATE ,

`physical\_address` VARCHAR(255) ,

`postal\_address` VARCHAR(255) ,

`contact\_number` VARCHAR(75) ,

`email` VARCHAR(255) ,

PRIMARY KEY (`membership\_number`) )

ENGINE = InnoDB;

Now let's see what the MySQL's data types are. You can use any of them depending on your need. You should always try to not to underestimate or overestimate potential range of data when creating a database.

**DATA TYPES**

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has **3** main categories of data types namely

1. Numeric,
2. Text
3. Date/time.

**Numeric Data types**

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

|  |  |
| --- | --- |
| TINYINT( ) | -128 to 127 normal 0 to 255 UNSIGNED. |
| SMALLINT( ) | -32768 to 32767 normal 0 to 65535 UNSIGNED. |
| MEDIUMINT( ) | -8388608 to 8388607 normal 0 to 16777215 UNSIGNED. |
| INT( ) | -2147483648 to 2147483647 normal 0 to 4294967295 UNSIGNED. |
| BIGINT( ) | -9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED. |
| FLOAT | A small approximate number with a floating decimal point. |
| DOUBLE( , ) | A large number with a floating decimal point. |
| DECIMAL( , ) | A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values. |

**Text Data Types**

As data type category name implies these are used to store text values. Always make sure you length of your textual data do not exceed maximum lengths.

|  |  |
| --- | --- |
| CHAR( ) | A fixed section from 0 to 255 characters long. |
| VARCHAR( ) | A variable section from 0 to 255 characters long. |
| TINYTEXT | A string with a maximum length of 255 characters. |
| TEXT | A string with a maximum length of 65535 characters. |
| BLOB | A string with a maximum length of 65535 characters. |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters. |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters. |
| LONGTEXT | A string with a maximum length of 4294967295 characters. |
| LONGBLOB | A string with a maximum length of 4294967295 characters. |

**Date / Time**

|  |  |
| --- | --- |
| DATE | YYYY-MM-DD |
| DATETIME | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS |
| TIME | HH:MM:SS |

Apart from above there are some other data types in MySQL.

|  |  |
| --- | --- |
| ENUM | To store text value chosen from a list of predefined text values |
| SET | This is also used for storing text values chosen from a list of predefined text values. It can have multiple values. |
| BOOL | Synonym for TINYINT(1), used to store Boolean values |
| BINARY | Similar to CHAR, difference is texts are stored in binary format. |
| VARBINARY | Similar to VARCHAR, difference is texts are stored in binary format. |

Now let's see a sample SQL query for creating a table which has data of all data types. Study it and identify how each data type is defined.

CREATE TABLE`all\_data\_types` (

`varchar` VARCHAR( 20 ) ,

`tinyint` TINYINT ,

`text` TEXT ,

`date` DATE ,

`smallint` SMALLINT ,

`mediumint` MEDIUMINT ,

`int` INT ,

`bigint` BIGINT ,

`float` FLOAT( 10, 2 ) ,

`double` DOUBLE ,

`decimal` DECIMAL( 10, 2 ) ,

`datetime` DATETIME ,

`timestamp` TIMESTAMP ,

`time` TIME ,

`year` YEAR ,

`char` CHAR( 10 ) ,

`tinyblob` TINYBLOB ,

`tinytext` TINYTEXT ,

`blob` BLOB ,

`mediumblob` MEDIUMBLOB ,

`mediumtext` MEDIUMTEXT ,

`longblob` LONGBLOB ,

`longtext` LONGTEXT ,

`enum` ENUM( '1', '2', '3' ) ,

`set` SET( '1', '2', '3' ) ,

`bool` BOOL ,

`binary` BINARY( 20 ) ,

`varbinary` VARBINARY( 20 )

) ENGINE= MYISAM ;

Best practices

* Use upper case letters for SQL keywords i.e. "DROP SCHEMA IF EXISTS `MyFlixDB`;"
* End all your SQL commands using semi colons.
* Avoid using spaces in schema, table and field names. Use underscores instead to separate schema, table or field names.

**MySQL workbench ER diagram forward engineering**

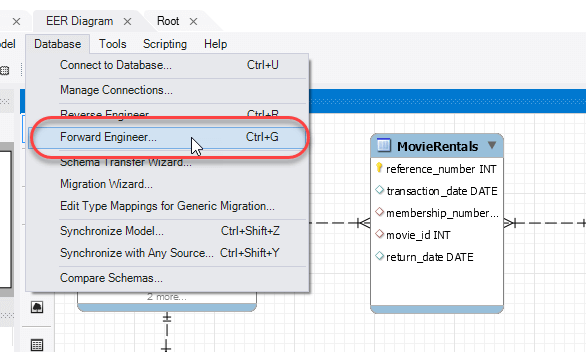
MySQL workbench has utilities that support forward engineering. **Forward engineering** is a technical term is to describe the process of **translating a logical model into a physical implement automatically**.

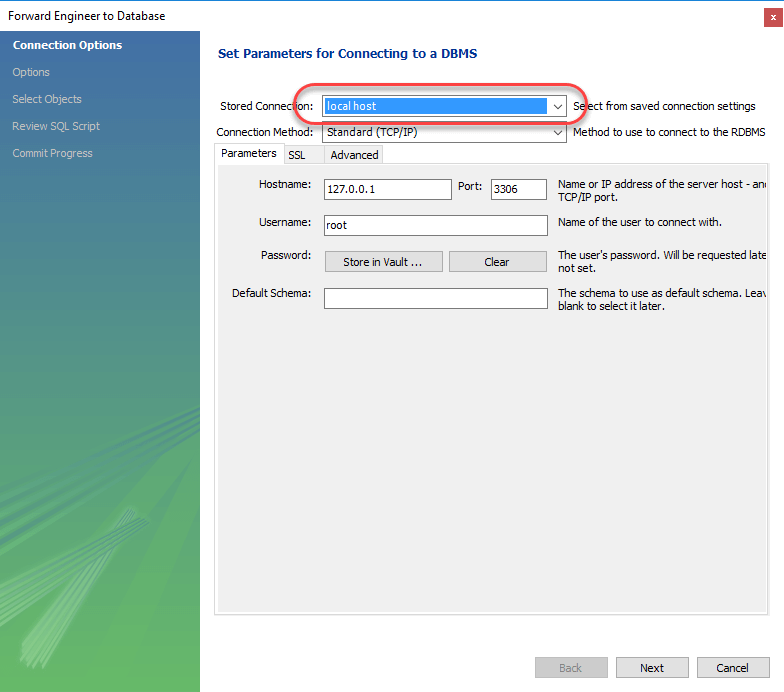
We created an ER diagram on our ER modeling tutorial. We will now use that ER model to generate the SQL scripts that will create our database.

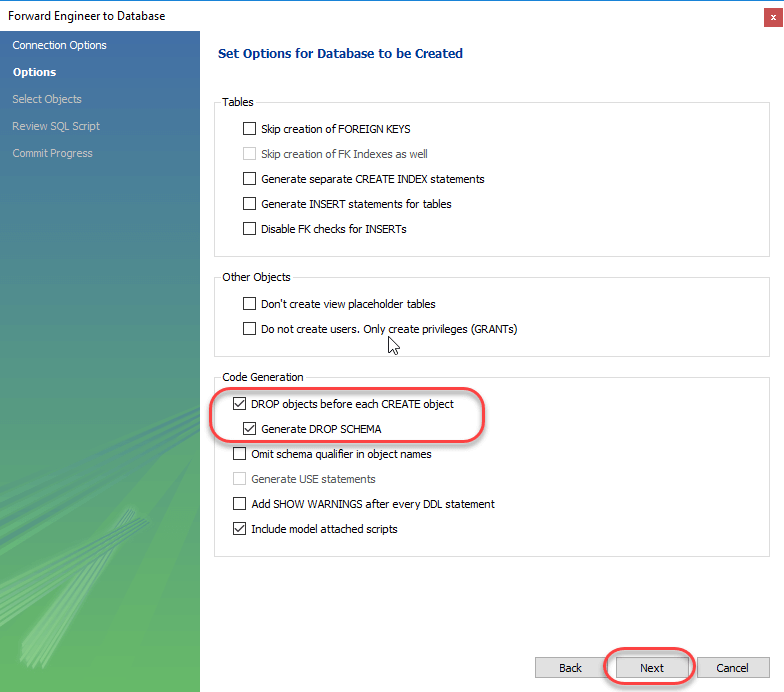
**Creating the MyFlix database from the MyFlix ER model**

1.       Open the ER model of MyFlix database that you created in earlier tutorial.

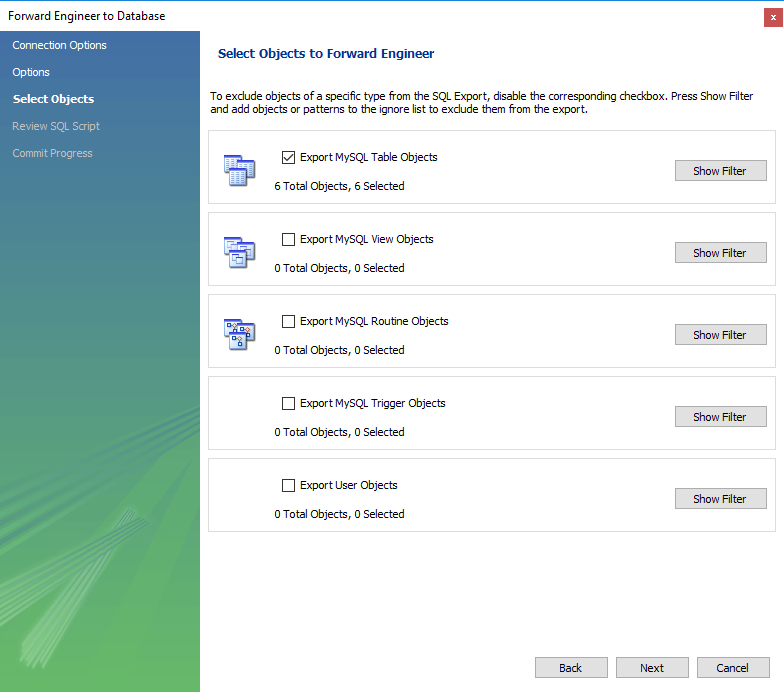
2.       Click on the database menu. Select forward engineer



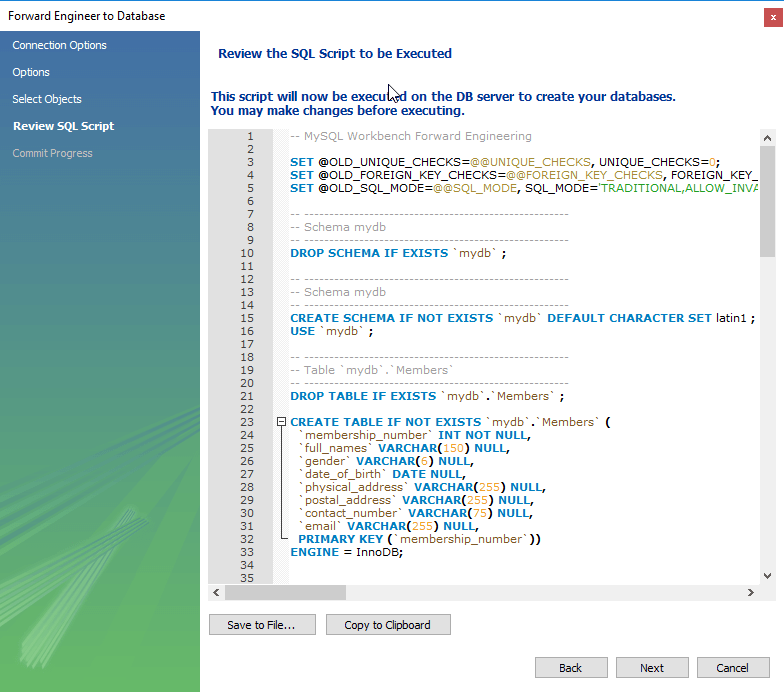
3.       The next window,  allows you to connect to an instance of MySQL server. Click on the stored connection drop down list and select local host. Click Execute4.       Select the options shown below in the wizard that appears. Click next



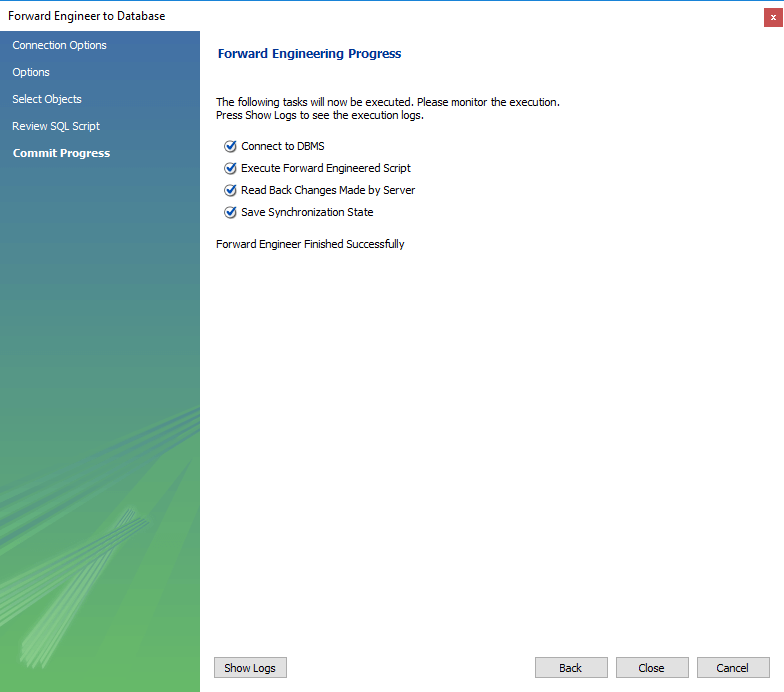
5.The next screen shows the summary of objects in our EER diagram. Our MyFlix DB has 5 tables.  Keep the selections default and click Next.



6..       The window shown below appears. This window allows you to preview the SQL script to create our database. We can save the scripts to a \*.sql" file or copy the scripts to the clipboard. Click on next button



7.The window shown below appears after successfully creating the database on the selected MySQL server instance.



**Summary**

* Creating a database involves translating the logical database design model into the physical database.
* MySQL supports a number of data types for numeric, dates and strings values.
* CREATE DATABASE command is used to create a database
* CREATE TABLE command is used to create tables in a database
* MySQL workbench supports forward engineering which involves automatically generating SQL scripts from the logical database model that can be executed to create the physical database

The Database along with Dummy Data is attached. We will be using this DB for all our further tutorials. Simple import the DB in MySQL Workbench to get started

**MySQL SELECT Statement with Examples**

Databases store data for later retrieval. Ever wondered how that is achieved? It's the **SELECT** SQL command that does the job.

That's what it's all about, retrieving data from the database tables. It's part of the **data manipulation language** that is responsible for **query the data from the database**.

[](https://www.guru99.com/images/select_confused.png)

**SQL SELECT statement syntax**

It is the most frequently used SQL command and has the following general syntax

SELECT [DISTINCT|ALL ] { \* | [fieldExpression [AS newName]} FROM tableName [alias] [WHERE condition][GROUP BY fieldName(s)] [HAVING condition] ORDER BY fieldName(s)

**HERE**

* **SELECT** is the SQL keyword that lets the database know that you want to retrieve data.
* **[DISTINCT | ALL]** are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
* **{\*| [fieldExpression [AS newName]}** at least one part must be specified, "\*" selected all the fields from the specified table name, fieldExpression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
* **FROM** tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.
* **WHERE** condition is optional, it can be used to specify criteria in the result set returned from the query.
* **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.

\*

The Star symbol is used to select all the columns in table. An example of a simple SELECT statement looks like the one shown below.

SELECT \* FROM `members`;

The above statement selects all the fields from the members table. The semi-colon is a statement terminate. It's not mandatory but is considered a good practice to end your statements like that.

**Practical examples**

[**Click to download**](https://drive.google.com/uc?export=download&id=0B_vqvT0ovzHccjhtdGlrZ0MtZ0k)the myflix DB used for practical examples.

You can learn to import the .sql file into MySQL WorkBench

The Examples are performed on the following two tables

Table 1: **members** table

| **membership\_ number** | **full\_names** | **gender** | **date\_of\_ birth** | **physical\_ address** | **postal\_ address** | **contct\_ number** | **email** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |

Table 2: **movies** table

| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| --- | --- | --- | --- | --- |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 3 | X-Men | NULL | 2008 | NULL |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |
| 6 | Angels and Demons | NULL | 2007 | 6 |
| 7 | Davinci Code | NULL | 2007 | 6 |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 16 | 67% Guilty | NULL | 2012 | NULL |

**Getting members listing**

Let's suppose that we want to get a list of all the registered library members from our database, we would use the script shown below to do that.

SELECT \* FROM `members`;

Executing the above script in MySQL workbench produces the following results.

| **membership\_ number** | **full\_names** | **gender** | **date\_of\_ birth** | **physical\_ address** | **postal\_ address** | **contct\_ number** | **email** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |

Our above query has returned all the rows and columns from the members table.

Let's say we are only interested in getting only the full\_names, gender, physical\_address and email fields only. The following script would help us to achieve this.

SELECT `full\_names`,`gender`,`physical\_address`, `email` FROM `members`;

Executing the above script in MySQL workbench produces the following results.

| **full\_names** | **gender** | **physical\_address** | **email** |
| --- | --- | --- | --- |
| Janet Jones | Female | First Street Plot No 4 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| Janet Smith Jones | Female | Melrose 123 | [jj@fstreet.com](mailto:jj@fstreet.com) |
| Robert Phil | Male | 3rd Street 34 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| Gloria Williams | Female | 2nd Street 23 | NULL |

**Getting movies listing**

Remember in our above discussion that we mention expressions been used in SELECT statements. Let's say we want to get a list of movie from our database. We want to have the movie title and the name of the movie director in one field. The name of the movie director should be in brackets. We also want to get the year that the movie was released. The following script helps us do that.

SELECT Concat(`title`, ' (', `director`, ')') , `year\_released` FROM `movies`;

**HERE**

* The Concat () MySQL function is used join the columns values together.
* The line "Concat (`title`, ' (', `director`, ')')  gets the title, adds an opening bracket followed by the name of the director then adds the closing bracket.

String portions are separated using commas in the Concat () function.

Executing the above script in MySQL workbench produces the following result set.

| **Concat(`title`, ' (', `director`, ')')** | **year\_released** |
| --- | --- |
| Pirates of the Caribean 4 ( Rob Marshall) | 2011 |
| Forgetting Sarah Marshal (Nicholas Stoller) | 2008 |
| NULL | 2008 |
| Code Name Black (Edgar Jimz) | 2010 |
| NULL | 2007 |
| NULL | 2007 |
| NULL | 2007 |
| Honey mooners (John Schultz) | 2005 |
| NULL | 2012 |

**Alias field names**

The above example returned the Concatenation code as the field name for our results. Suppose we want  to use a more descriptive field name in our result set. We would use the column alias name to achieve that. The following is the basic syntax for the column alias name

SELECT `column\_name|value|expression` [AS] `alias\_name`;

**HERE**

* ***"SELECT ` column\_name|value|expression `"*** is the regular SELECT statement which can be a column name, value or expression.
* ***"[AS]"*** is the optional keyword before the alias name that denotes the expression, value or field name will be returned as.
* ***"`alias\_name`"*** is the alias name that we want to return in our result set as the field name.

The above query with a more meaningful column name

SELECT Concat(`title`, ' (', `director`, ')') AS 'Concat', `year\_released` FROM `movies`;

We get the following result

| **Concat** | **year\_released** |
| --- | --- |
| Pirates of the Caribean 4 ( Rob Marshall) | 2011 |
| Forgetting Sarah Marshal (Nicholas Stoller) | 2008 |
| NULL | 2008 |
| Code Name Black (Edgar Jimz) | 2010 |
| NULL | 2007 |
| NULL | 2007 |
| NULL | 2007 |
| Honey mooners (John Schultz) | 2005 |
| NULL | 2012 |

**Getting members listing showing the year of birth**

Suppose we want to get a list of all the members showing the membership number, full names and year of birth, we can use the LEFT string function to extract the year of birth from the date of birth field. The script shown below helps us to do that.

SELECT `membership\_number`,`full\_names`,LEFT(`date\_of\_birth`,4) AS `year\_of\_birth` FROM members;

**HERE**

* ***"LEFT(`date\_of\_birth`,4)"*** the **LEFT string function** accepts the date of birth as the parameter and only returns 4 characters from the left.
* ***"AS `year\_of\_birth`"*** is the **column alias name** that will be returned in our results. Note the ***AS keyword is optional***, you can leave it out and the query will still work.

Executing the above query in MySQL workbench against the myflixdb gives us the results shown below.

| **membership\_number** | **full\_names** | **year\_of\_birth** |
| --- | --- | --- |
| 1 | Janet Jones | 1980 |
| 2 | Janet Smith Jones | 1980 |
| 3 | Robert Phil | 1989 |
| 4 | Gloria Williams | 1984 |

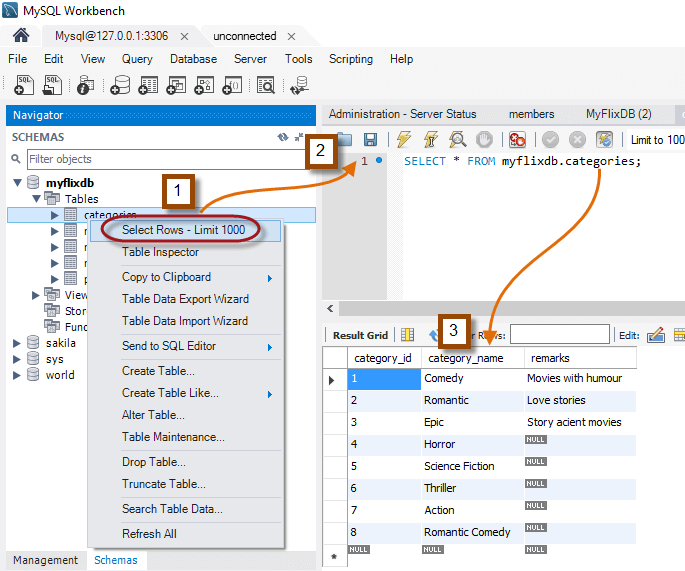
**SQL using MySQL Workbench**

We are now going to use MySQL workbench to generate the script that will display all the field names from our categories table.

1.       Right Click on the Categories Table. Click on "Select Rows - Limit 1000"

2.       MySQL workbench will automatically create a SQL query and paste in the editor.

3.       Query Results will be show



Notice that we didn't write the SELECT statement ourselves. MySQL workbench generated it for us.

### Why use the SELECT SQL command when we have MySQL Workbench?

Now, you might be thinking why learn the SQL SELECT command to query data from the database when you can simply use a tool such as MySQL workbench's to get the same results without knowledge of the SQL language. Of course that is possible, but **learning how to use the SELECT command**gives you more **flexibility** and **control** over your **SQL SELECT statements**.

MySQL workbench falls in the category of "**Query by Example**" QBE tools. It's intended to help generate SQL statements faster to increase the user productivity.

Learning the SQL SELECT command can enable you to create **complex queries** that cannot be easily generated using Query by Example utilities such as MySQL workbench.

To improve productivity you can **generate the code using MySQL workbench**then **customize** it to **meet your requirements**. This can only happen if you understand how the SQL statements work!

**Summary**

* The SQL SELECT keyword is used to query data from the database and it's the most commonly used command.
* The simplest form has the syntax "SELECT \* FROM tableName;"
* Expressions can also be used in the select statement . Example "SELECT quantity + price FROM Sales"
* The SQL SELECT command can also have other optional parameters such as WHERE, GROUP BY, HAVING, ORDER BY. They will be discussed later.
* MySQL workbench can help develop SQL statements, execute them and produce the output result in the same window.

# MySQL WHERE Clause with Examples - AND, OR, IN, NOT IN

### What is the WHERE Clause?

We looked at how to query data from a database using the SELECT statement in the previous tutorial. The SELECT statement returned all the results from the queried database table.

They are however, times when we want to restrict the query results to a specified condition. The SQL WHERE clause comes in handy in such situations.



#### WHERE clause Syntax

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

SELECT \* FROM tableName WHERE condition;

**HERE**

* **"SELECT \* FROM tableName"** is the standard SELECT statement
* **"WHERE"** is the keyword that restricts our select query result set and **"condition"** is the filter to be applied on the results. The filter could be a range, single value or sub query.

Let's now look at a **practical example**.

Suppose we want to get a member's personal details from members table given the membership number 1, we would use the following script to achieve that.

SELECT \* FROM `members` WHERE `membership\_number` = 1;

Executing the above script in MySQL workbench on the "myflixdb" would produce the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |

### WHERE clause combined with - *****AND*****LOGICAL Operator

The WHERE clause when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.

Let's now look at a practical example - Suppose we want to get a list of all the movies in category 2 that were released in 2008, we would use the script shown below is achieve that.

SELECT \* FROM `movies` WHERE `category\_id` = 2 AND `year\_released` = 2008;

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |

### WHERE clause combined with - *****OR*****LOGICAL Operator

The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.

The following script gets all the movies in either category 1 or category 2

SELECT \* FROM `movies` WHERE `category\_id` = 1 OR `category\_id` = 2;

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |

### WHERE clause combined with - *****IN*****Keyword

The WHERE clause when used together with the IN keyword only affects the rows whose values matches the list of values provided in the IN keyword. IN helps reduces number of OR clauses you may have to use

The following query gives rows where membership\_number is either 1 , 2 or 3

SELECT \* FROM `members` WHERE `membership\_number` IN (1,2,3);

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |

### WHERE clause combined with - NOT IN Keyword

The  WHERE clause when used together with the NOT IN keyword  DOES NOT affects the rows whose values matches the list of values provided in the NOT IN keyword.

The following query gives rows where membership\_number is NOT  1 , 2 or 3

SELECT \* FROM `members` WHERE `membership\_number` NOT IN (1,2,3);

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |

### WHERE clause combined with - ****COMPARISON OPERATORS****

The less than (), equal to (=), not equal to () comparison operators can be  used with the Where clause

### *****=*****Equal To

The following script gets all the female members from the members table using the equal to comparison operator.

SELECT \* FROM `members` WHERE `gender` = 'Female';

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |

### ****>****Greater than

The following script gets all the payments that are greater than 2,000 from the payments table.

SELECT \* FROM `payments` WHERE `amount\_paid` > 2000;

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **payment\_id** | **membership\_number** | **payment\_date** | **description** | **amount\_paid** | **external\_reference\_number** |
| 1 | 1 | 23-07-2012 | Movie rental payment | 2500 | 11 |
| 3 | 3 | 30-07-2012 | Movie rental payment | 6000 | NULL |

### < > Not Equal To

The following script gets all the movies whose category id is not 1.

SELECT \* FROM `movies` WHERE `category\_id`<> 1;

Executing the above script in MySQL workbench against the "myflixdb" produces the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |
| 6 | Angels and Demons | NULL | 2007 | 6 |
| 7 | Davinci Code | NULL | 2007 | 6 |
| 9 | Honey mooners | John Schultz | 2005 | 8 |

### Summary

* The SQL WHERE clause is used to restrict the number of rows affected by a SELECT, UPDATE or DELETE query.
* The WHERE clause can be used in conjunction with logical operators such as AND and OR, comparison operators such as ,= etc.
* When used with the AND logical operator, all the criteria must be met.
* When used with the OR logical operator, any of the criteria must be met.
* The key word IN is used to select rows matching a list of values.

Brain Teaser

Let's suppose that we want to get a list of rented movies that have not been returned on time 25/06/2012. We can use the WHERE clause together with the less than comparison operator and AND logical operator to achieve that.

SELECT \* FROM `movierentals` WHERE `return\_date` < '2012-06-25' AND movie\_returned = 0;

Executing the above script in MySQL workbench gives the following results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **reference\_number** | **transaction\_date** | **return\_date** | **membership\_number** | **movie\_id** | **movie\_returned** |
| 14 | 21-06-2012 | 24-06-2012 | 2 | 2 | 0 |

**MySQL query INSERT INTO Table with Examples**

**What is INSERT INTO?**

The main goal of database systems is to store data in the tables. The data is usually supplied by application programs that run on top of the database. Towards that end, SQL has the INSERT command that is used to store data into a table. The **INSERT command creates a new row** in the table to store data.

**Basic syntax**

Let's look at the basic syntax of the SQL INSERT command shown below.

INSERT INTO `table\_name`(column\_1,column\_2,...) VALUES (value\_1,value\_2,...);

**HERE**

* **INSERT INTO `table\_name`** is the command that tells MySQL server to add new row into a table named `table\_name`.
* **(column\_1,column\_2,...)**specifies the columns to be updated in the  new row
* **VALUES (value\_1,value\_2,...)** specifies the values to be added into the new row

When supplying the data values to be inserted into the new table, the following should be considered while dealing with different data types.

* **String data types** - all the string values should be enclosed in single quotes.
* **Numeric data types** - all numeric values should be supplied directly without enclosing them in single or double quotes.
* **Date data types** - enclose date values in single quotes in the format 'YYYY-MM-DD'.

**Example:**

Suppose that we have the following list of new library members that need to be added into the database.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Full names** | **Date of Birth** | **gender** | **Physical address** | **Postal address** | **Contact number** | **Email Address** |
| Leonard Hofstadter |  | Male | Woodcrest |  | 0845738767 |  |
| Sheldon Cooper |  | Male | Woodcrest |  | 0976736763 |  |
| Rajesh Koothrappali |  | Male | Fairview |  | 0938867763 |  |
| Leslie Winkle | 14/02/1984 | Male |  |  | 0987636553 |  |
| Howard Wolowitz | 24/08/1981 | Male | South Park | P.O. Box 4563 | 0987786553 | [lwolowitz@email.me](mailto:lwolowitz@email.me) |

Lets' INSERT data one by one. We will start with Leonard Hofstadter. We will treat the contact number as a numeric data type and not enclose the number in single quotes.

INSERT INTO `members` (`full\_names`,`gender`,`physical\_address`,`contact\_number`) VALUES ('Leonard Hofstadter','Male','Woodcrest',0845738767);

Executing the above script drops the 0 from Leonard's contact number. This is because the value will be treated as a numeric value and the zero (0) at the beginning is dropped since it's not significant.

In order to avoid such problems, the value must be enclosed in single quotes as shown below -

INSERT INTO `members` (`full\_names`,`gender`,`physical\_address`,`contact\_number`) VALUES ('Sheldon Cooper','Male','Woodcrest', '0976736763');

 In the above case , zero(0) will not be dropped

**Changing the order of the columns has no effect on the INSERT query as long as the correct values have been mapped to the correct columns.**

 The query shown below demonstrates the above point.

INSERT INTO `members` (`contact\_number`,`gender`,`full\_names`,`physical\_address`)VALUES ('0938867763','Male','Rajesh Koothrappali','Woodcrest');

The above queries skipped the date of birth column, **by default MySQL will insert NULL values in columns that are skipped in the INSERT query.**

 Let's now insert the record for Leslie which has the date of birth supplied. **The date value should be enclosed in single quotes using the format 'YYYY-MM-DD'.**

INSERT INTO `members` (`full\_names`,`date\_of\_birth`,`gender`,`physical\_address`,`contact\_number`) VALUES ('Leslie Winkle','1984-02-14','Male','Woodcrest', '0987636553');

All of the above queries specified the columns and mapped them to values in the insert statement.**If we are supplying values for ALL the columns in the table, then we can omit the columns from the insert query.**

Example:-

INSERT INTO `members` VALUES (9,'Howard Wolowitz','Male','1981-08-24','SouthPark','P.O. Box 4563', '0987786553', 'lwolowitz[at]email.me');

Let's now use the SELECT statement to view all the rows in the members table.SELECT \* FROM `members`;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 | [lwolowitz@email.me](mailto:lwolowitz@email.me) |

Notice the contact number for Leonard Hofstadter has dropped the zero (0) from the contact number. The other contact numbers have not dropped the zero (0) at the beginning.

**Inserting into  a Table from another Table**

The INSERT command can also be used to insert data into a table from another table. The basic syntax is as shown below.

INSERT INTO table\_1 SELECT \* FROM table\_2;

Let's now look at a practical example, we will create a dummy table for movie categories for demonstration purposes. We will call the new categories table categories\_archive. The script shown below creates the table.

CREATE TABLE `categories\_archive` ( `category\_id` int(11) AUTO\_INCREMENT, `category\_name` varchar(150) DEFAULT NULL, `remarks` varchar(500) DEFAULT NULL, PRIMARY KEY (`category\_id`) )

Execute the above script to create the table.

Let's now insert all the rows from the categories table into the categories archive table. The script shown below helps us to achieve that.

INSERT INTO `categories\_archive` SELECT \* FROM `categories`;

**Executing the above script inserts all the rows from the categories table into the categories archive table. Note the table structures will have to be the same for the script to work. A more robust script is one that maps the column names in the insert table to the ones in the table containing the data**.

The query shown below demonstrates its usage.

INSERT INTO `categories\_archive`(category\_id,category\_name,remarks) SELECT category\_id,category\_name,remarks FROM `categories`;

Executing the SELECT query

SELECT \* FROM `categories\_archive`

gives the following results shown below.

|  |  |  |
| --- | --- | --- |
| **category\_id** | **category\_name** | **remarks** |
| 1 | Comedy | Movies with humour |
| 2 | Romantic | Love stories |
| 3 | Epic | Story acient movies |
| 4 | Horror | NULL |
| 5 | Science Fiction | NULL |
| 6 | Thriller | NULL |
| 7 | Action | NULL |
| 8 | Romantic Comedy | NULL |
| 9 | Cartoons | NULL |
| 10 | Cartoons | NULL |

**Summary**

* The INSERT command is used to add new data into a table
* The date and string values should be enclosed in single quotes.
* The numeric values do not need to be enclosed in quotes.
* The INSERT command can also be used to insert data from one table into another.

**MySQL UPDATE & DELETE Query with Example**

**What is the DELETE Keyword?**

The **MySQL DELETE** command is used to delete rows that are no longer required from the database tables. It deletes the whole row from the table and returns count of deleted rows. Delete command comes in handy to delete temporary or obsolete data from your database.The DELETE command can delete more than one row from a table in a single query. This proves to be advantages when removing large numbers of rows from a database table.

Once a row has been deleted, it cannot be recovered. It is therefore strongly recommended to make database backups before deleting any data from the database. This can allow you to restore the database and view the data later on should it be required.

**Delete command syntax**

The basic syntax of the delete command is as shown below.

DELETE FROM `table\_name` [WHERE condition];

HERE

* DELETE FROM `table\_name` tells MySQL server to remove rows from the table ..
* [WHERE condition] is optional and is used to put a filter that restricts the number of rows affected by the DELETE query.

If the WHERE clause is not used in the DELETE query, then all the rows in a given table will be deleted. Before we go into more details discussion the DELETE command, let's insert some sample data into the movies table to work with.

INSERT INTO `movies` (`title`, `director`, `year\_released`, `category\_id`) VALUES ('The Great Dictator', 'Chalie Chaplie', 1920, 7);

INSERT INTO `movies` (`title`, `director`, `category\_id`) VALUES ('sample movie', 'Anonymous', 8);

INSERT INTO movies (`title`, `director`, `year\_released`, `category\_id`) VALUES ('movie 3', 'John Brown', 1920, 8);

Executing the above script adds three (3) movies into the movies table. Before we go any further into our lesson, let's get all the movies in our table. The script shown below does that.

SELECT \* FROM `movies`;

Executing the above script gives us the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **itle** | **director** | **year\_released** | **category\_id** |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 3 | X-Men | NULL | 2008 | NULL |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |
| 6 | Angels and Demons | NULL | 2007 | 6 |
| 7 | Davinci Code | NULL | 2007 | 6 |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 16 | 67% Guilty | NULL | 2012 | NULL |
| 18 | The Great Dictator | Chalie Chaplie | 1920 | 7 |
| 19 | sample movie | Anonymous | NULL | 8 |
| 20 | movie 3 | John Brown | 1920 | 8 |

 Let's suppose that the Myflix video library no longer wishes to be renting out "The Great Dictator" to its members and they want it removed from the database. Its movie id is 18, we can use the script shown below to delete its row from the movies table.

DELETE FROM `movies` WHERE `movie\_id` = 18;

Executing the above script in MySQL WorkBench against the Myflix deletes the movie with id 18 from the database table.

Let's see the current status of movies table.

SELECT \* FROM `movies`;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 3 | X-Men | NULL | 2008 | NULL |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |
| 6 | Angels and Demons | NULL | 2007 | 6 |
| 7 | Davinci Code | NULL | 2007 | 6 |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 16 | 67% Guilty | NULL | 2012 | NULL |
| 19 | sample movie | Anonymous | NULL | 8 |
| 20 | movie 3 | John Brown | 1920 | 8 |

NOTE:

* *the movie with id 18 has not been return in the query result set.*
* *you cannot delete a single column for a table. You can delete an entire row.*

Let's say we  have a list of movies we want to delete . We can use the WHERE clause along with IN.

DELETE FROM `movies` WHERE `movie\_id` IN (20,21);

Executing the above script deletes movies with IDs 20 and 21 from our movies table.

**WHAT IS THE UPDATE COMMAND?**

The Update command is used to modify rows in a table. The update command can be used to update a single field or multiple fields at the same time. It can also be used to update a table with values from another table .

**Update command syntax**

The basic syntax of the SQL Update command is as shown below.

UPDATE `table\_name` SET `column\_name` = `new\_value' [WHERE condition];

HERE

* UPDATE `table\_name` is the command that tells MySQL to update the data in a table .
* SET `column\_name` = `new\_value' are the names and values of the fields to be affected by the update query. Note, when setting the update values, strings data types must be in single quotes. Numeric values do not need to be in quotation marks. Date data type must be in single quotes and in the format 'YYYY-MM-DD'.
* [WHERE condition]  is optional and can be used to put a filter that restricts the number of rows affected by the UPDATE query.

Let's now look at a practical example that updates data in the members table. Let's suppose that our member's membership numbers 1 and 2 have the following updates to be made to their data records.

|  |  |
| --- | --- |
| Membership number | Updates required |
| 1 | Changed contact number from 999 to 0759 253 532 |
| 2 | Change the name to Janet Smith Jones and physical address should be updated to Melrose 123 |

We will start with making updates for membership number  1 before we make any updates to our data, let's retrieve the record for membership number 1. The script shown below helps us to do that.

SELECT \* FROM `members` WHERE `membership\_number` = 1;

Executing the above script gives us the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 999 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |

Let's now update the contact number using the script shown below.

UPDATE `members` SET `contact\_number` = '0759 253 542' WHERE `membership\_number` = 1;

Executing the above script updates the contact number from 999 to 0759 253 532 for membership number 1. Let's now look at the record for membership number 1 after executing the update script.

SELECT \* FROM `members` WHERE `membership\_number` = 1;

Executing the above script gives us the following results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |

Let's now look at the updates required for membership number 2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 2 | Smith Jones | Female | 23-06-1980 | Park Street | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |

The following script helps us to do that.

UPDATE `members` SET `full\_names` = 'Janet Smith Jones', `physical\_address` = 'Melrose 123' WHERE `membership\_number` = 2;

Executing the above script in updates the full names for membership number 2 to Janet Smith Jones and the physical address to Melrose 123.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |

**Summary**

* The delete command is used to remove data that is no longer required from a table.
* The "WHERE clause" is used to limit the number of rows affected by the DELETE query.
* Once data has been deleted, it cannot be recovered, it is therefore strongly recommend make backups before deleting data.
* The update command is used to modify existing data.
* The "WHERE clause" is used to limit the number of rows affected by the UPDATE query.

**ORDER BY in MySQL: DESC & ASC**

**Sorting Results**

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

**Order by clause**

The order by clause is used to sort the query result sets in either ascending or descending order. It is used in conjunction with the SELECT query. It has the following basic syntax.

SELECT statement... [WHERE condition | GROUP BY `field\_name(s)` HAVING condition] ORDER BY `field\_name(s)` [ASC | DESC];

**HERE**

* **"SELECT statement..."** is the regular select query
* **" | "** represents alternatives
* **"[WHERE condition | GROUP BY `field\_name(s)` HAVING condition"** is the optional condition used to filter the query result sets.
* **"ORDER BY"** performs the query result set sorting
* **"[ASC | DESC]"** is the keyword used to sort result sets in either ascending or descending order. Note ***ASC*** is used as the default.

**What are DESC and ASC Keywords?**

|  |  |
| --- | --- |
| **[ORDER BY in MySQL: DESC & ASC](https://www.guru99.com/images/asc.png)ASC is the short form for ascending** | **[ORDER BY in MySQL: DESC & ASC](https://www.guru99.com/images/desc.png)DESC is the short form for descending** |
| It is used to sort the query results in a top to bottom style. | It is used to sort the query results in a bottom to top style |
| When working on date data types, the earliest date is shown on top of the list. | . When working on date types, the latest date is shown on top of the list. |
| When working with numeric data types, the lowest values are shown on top of the list. | When working with numeric data types, the highest values are shown at top of the query result set. |
| When working with string data types, the query result set is sorted from those starting with the letter A going up to the letter Z. | When working with string data types, the query result set is sorted from those starting with the letter Z going down to the letter A. |

Both the DESC and ASC keywords are used together in conjunction with the SELECT statement and the ORDER BY clause.

**DESC and ASC syntax**

The DESC sort keyword has the following basic syntax.

SELECT {fieldName(s) | \*} FROM tableName(s) [WHERE condition] ORDER BY fieldname(s) ASC /DESC [LIMIT N]

**HERE**

* **SELECT {fieldName(s) | \*} FROM tableName(s)** is the statement containing the fields and table(s) from which to get the result set from.
* **[WHERE condition]** is optional but can be used to filter the data according to the given condition.
* **ORDER BY** fieldname(s) is mandatory and is the field on which the sorting is to be performed. The DESC keyword specifies that the sorting is to be in descending order.
* **[LIMIT]** is optional but can be used to limit the number of results returned from the query result set.

**Examples:**

Let's now look at a practical example -

SELECT \* FROM members;

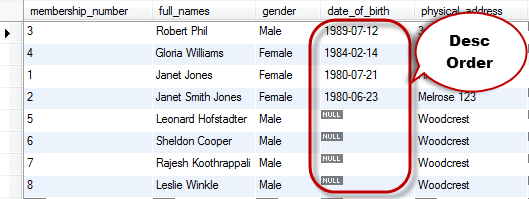
Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 |  |

Let's suppose the marketing department wants the members details arranged in decreasing order of Date of Birth.  This will help them send birthday greetings in a timely fashion. We can get the said list by executing a query like below -

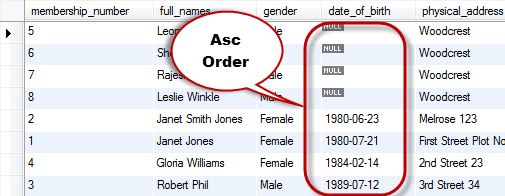
SELECT \* FROM members ORDER BY date\_of\_birth DESC;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

[](https://www.guru99.com/images/DescQuery.png)

The same query in ascending order

SELECT \* FROM members ORDER BY date\_of\_birth ASC

[](https://www.guru99.com/images/AscQuery.png)

Note: NULL values means no values (not zero or empty string) . Observe the way they have been sorted.

**More examples**

Let's consider the following script that lists all the member records.

SELECT \* FROM `members`;

Executing the above script gives the following results shown below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 | NULL |

Suppose we want to get a list that sorts the query result set using the gender field, we would use the script shown below.

SELECT \* FROM `members` ORDER BY `gender`;

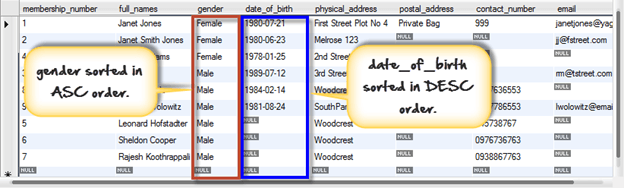
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contct\_number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 | NULL |

"Female" members have been displayed first followed by "Male" members, this is because when order by clause is used without specifying the ASC or DESC keyword, by default, MySQL has sorted the query result set in an ascending order.

Let's now look at an example that does the **sorting using two columns**; the first one is **sorted** in **ascending order** by default while the second column is **sorted** in **descending order.**

SELECT \* FROM `members` ORDER BY `gender`,`date\_of\_birth` DESC;

Executing the above script in MySQL workbench against the myflixdb gives the following results.

[](https://www.guru99.com/images/order_by_two_columns%281%29.png)

The gender column was sorted in ascending order by default while the date of birth column was sorted in descending order explicitly

**Why we may use DESC and ASC?**

Suppose we want to print a payments history for a video library member to help answer queries from the front desk, wouldn't it be more logical to have the payments printed in a descending chronological order starting with the recent payment to the earlier payment?

The DESC key word comes in handy in such situations. We can write a query that sorts the list in descending order using the payment date.

Suppose the marketing department wants to get a list of movies by category that members can use to decide which movies are available in the library when renting movies, wouldn't it be more logical to look sort the movie category names and title in ascending so that members can quickly lookup the information from the list?

The ASC keyword comes in handy in such situations; we can get the movies list sorted by category name and movie title in an ascending order.

**Summary**

* Sorting query results is re-arranging the rows returned from a query result set either in ascending or descending order.
* The DESC keyword is used to sort the query result set in a descending order.
* The ASC keyword is used to sort the query result set in an ascending order.
* Both DESC and ASC work in conjunction with the ORDER BY keyword. They can also be used in combination with other keywords such as WHERE clause and LIMIT
* The default for ORDER BY when nothing has been explicitly specified is ASC.

**MySQL GROUP BY and HAVING Clause with Examples**

**What is the Group by Clause?**

The GROUP BY clause is a SQL command that is used to **group rows that have the same values**.

 The GROUP BY clause is used in the SELECT statement .Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

That's what it does, **summarizing data** from the database.

The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

**GROUP BY Syntax**

Now that we know what the GROUP By clause is, let's look at the syntax for a basic group by query.

SELECT statements... GROUP BY column\_name1[,column\_name2,...] [HAVING condition];

**HERE**

* "SELECT statements..." is the standard SQL SELECT command query.
* "**GROUP BY** *column\_name1*" is the clause that performs the grouping based on column\_name1.
* "[,column\_name2,...]" is optional; represents other column names when the grouping is done on more than one column.
* "[HAVING condition]" is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the  WHERE clause.

**Grouping using a Single Column**

In order to help understand the effect of Group By clause, let's execute a simple query that returns all the gender entries from the members table.

SELECT `gender` FROM `members` ;

|  |
| --- |
| **gender** |
| Female |
| Female |
| Male |
| Female |
| Male |
| Male |
| Male |
| Male |
| Male |

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

SELECT `gender` FROM `members` GROUP BY `gender`;

Executing the above script in MySQL workbench against the Myflixdb gives us the following results.

|  |
| --- |
| **gender** |
| Female |
| Male |

Note only two results have been returned. This is because we only have two gender types Male and Female. The GROUP BY clause grouped all the "Male" members together and returned only a single row for it. It did the same with the "Female" members.

**Grouping using multiple columns**

Suppose that we want to get a list of movie category\_id  and corresponding years in which they were released.

Let's observe the output of this simple query

SELECT `category\_id`,`year\_released` FROM `movies` ;

|  |  |
| --- | --- |
| **category\_id** | **year\_released** |
| 1 | 2011 |
| 2 | 2008 |
| NULL | 2008 |
| NULL | 2010 |
| 8 | 2007 |
| 6 | 2007 |
| 6 | 2007 |
| 8 | 2005 |
| NULL | 2012 |
| 7 | 1920 |
| 8 | NULL |
| 8 | 1920 |

The above result has many duplicates.

Let's execute the same query using group by -

SELECT `category\_id`,`year\_released` FROM `movies` GROUP BY `category\_id`,`year\_released`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

|  |  |
| --- | --- |
| **category\_id** | **year\_released** |
| NULL | 2008 |
| NULL | 2010 |
| NULL | 2012 |
| 1 | 2011 |
| 2 | 2008 |
| 6 | 2007 |
| 7 | 1920 |
| 8 | 1920 |
| 8 | 2005 |
| 8 | 2007 |

The GROUP BY clause operates on both the category id and year released to identify **unique** rows in our above example.

**If the category id is the same but the year released is different, then a row is treated as a unique one .If the category id and the year released is the same for more than one row, then it's considered a duplicate and only one row is shown.**

**Grouping and aggregate functions**

Suppose we want total number of males and females in our database. We can use the following script shown below to do that.

SELECT `gender`,COUNT(`membership\_number`) FROM `members` GROUP BY `gender`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |
| --- | --- |
| **gender** | **COUNT('membership\_number')** |
| Female | 3 |
| Male | 5 |

The results shown below are grouped by every unique gender value posted and the number of grouped rows is counted using the COUNT aggregate function.

**Restricting query results using the HAVING clause**

It's not always that we will want to perform groupings on all the data in a given table. There will be times when we will want to restrict our results to a certain given criteria.  In such cases , we can use the HAVING clause

Suppose we want to know all the release years for movie category id 8. We would use the following script to achieve our results.

SELECT \* FROM `movies` GROUP BY `category\_id`,`year\_released` HAVING `category\_id` = 8;

Executing the above script in MySQL workbench against the Myflixdb gives us the following results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |

Note only movies with category id 8 have been affected by our GROUP BY clause.

**Summary**

* The GROUP BY Clause is used to group rows with same values .
* The GROUP BY Clause is used together with the SQL SELECT statement.
* The SELECT statement used in the GROUP BY clause can only be used contain column names, aggregate functions, constants and expressions.
* The HAVING clause is used to restrict the results returned by the GROUP BY clause.

**MySQL Wildcards Tutorial: Like, NOT Like, Escape, ( % ), ( \_ )**

**What are wildcards?**

Wildcards are characters that help search data matching complex criteria. Wildcards are used in conjunction with the LIKE comparison operator or the NOT LIKE comparison operator.

**Why use WildCards ?**

If you are familiar with using the SQL,  you may think that you can search for any complex data using SELECT and WHERE clause . Then why use Wildcards ?

Before we answer that question, let's look at an example. Suppose that the marketing department of Myflix video library carried out marketing promotions in the city of Texas and would like to get some feedback on the number of members

 that registered from Texas, you can use the following SELECT statement together with the WHERE clause to get the desired information.

SELECT \* FROM members WHERE postal\_address = 'Austin , TX' OR postal\_address = Dallas , TX OR postal\_address = Iola,TX OR postal\_adress = Houston ,TX';

As you can see from the above query, the "WHERE clause" becomes complex. Using wildcards however, simplifies the query as we can use something simple like the script shown below.

SELECT \* FROM members WHERE postal\_address like '% TX';

In short, wildcards allow us to develop power search engines into our data driven applications.

**Types of wildcards**

**% the percentage**

% the percentage character is used to specify a pattern of **zero (0) or more characters**. It has the following basic syntax.

SELECT statements... WHERE fieldname LIKE 'xxx%';

**HERE**

* "SELECT statement..." is the standard SQL SELECT command.
* "WHERE" is the key word used to apply the filter.
* "LIKE" is the comparison operator that is used in conjunction with wildcards
* 'xxx' is any specified starting pattern such as a single character or more and "%" matches any number of characters starting from zero (0).

To fully appreciate the above statement, let's look at a practical example

Suppose we want to get all the movies that have the word "code" as part of the title, we would use the percentage wildcard to perform a pattern match on both sides of the word "code". Below is the SQL statement that can be used to achieve the desired results.

SELECT \* FROM movies WHERE title LIKE '%code%';

Executing the above script in MySQL workbench against the myflixdb gives us the results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 7 | Davinci Code | NULL | NULL | 6 |

Notice that even if the search key word "code" appears on the beginning or end of the title, it is still returned in our result set. This is because our code includes any number of characters at the beginning then matches the pattern "code" followed by any number of characters at the end.

Let's now modify our above script to include the percentage wildcard at the beginning of the search criteria only.

SELECT \* FROM movies WHERE title LIKE '%code';

Executing the above script in MySQL workbench against the myflixdb gives us the results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 7 | Davinci Code | NULL | NULL | 6 |

Notice that only one record has been returned from the database. This is because our code matches any number of characters at the beginning of the movie title and gets only records that end with the pattern "code".

Let's now shift the percentage wildcard to the end of the specified pattern to be matched. The modified script is shown below.

SELECT \* FROM movies WHERE title LIKE 'code%';

Executing the above script in MySQL workbench against the myflixdb gives us the results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |

Notice only one record has been returned from the database. This is because our code matches all titles that start with the pattern "code" followed by any number of characters.

**\_ underscore wildcard**

The underscore wildcard is used to **match exactly one character**. Let's suppose that we want to search for all the movies that were released in the years 200x where x is exactly one character that could be any value. We would use the underscore wild card to achieve that. The script below select all the movies that were released in the year "200x"

SELECT \* FROM movies WHERE year\_released LIKE '200\_';

Executing the above script in MySQL workbench against the myflixdb gives us the results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 9 | Honey mooners | Jhon Shultz | 2005 | 8 |

Notice that only movies that have 200 follows by any character in the field year released have been returned in our result set. This is because the underscore wildcard matched the pattern 200 followed by any single character

**NOT Like**

The NOT logical operator can be used together with the wildcards to return rows that do not match the specified pattern.

Suppose we want to get movies that were not released in the year 200x. We would use the NOT logical operator together with the underscore wildcard to get our results. Below is the script that does that.

SELECT \* FROM movies WHERE year\_released NOT LIKE '200\_';

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 8 | Underworld-Awakeninh | Michahel Eal | 2012 | 6 |

Notice only movies that do not start with 200 in the year released have been returned in our result set. This is because we used the NOT logical operator in our wildcard pattern search.

**Escape keyword.**

The ESCAPE keyword is used to **escape pattern matching characters** such as the (%) percentage and underscore (\_) if they form part of the data.

 Let's suppose that we want to check for the string "67%" we can use;

LIKE '67#%%' ESCAPE '#';

If we want to search for the movie "67% Guilty", we can use the script shown below to do that.

SELECT \* FROM movies WHERE title LIKE '67#%%' ESCAPE '#';

Note the double "**%**%" in the LIKE clause, the first one in red "**%**" is treated as part of the string to be searched for. The other one is used to match any number of characters that follow.

The same query will also work if we use something like

SELECT \* FROM movies WHERE title LIKE '67=%%' ESCAPE '=';

**Summary**

* Like & Wildcards powerful tools that help search data matching complex patterns.
* There are a number of wildcards that include the percentage, underscore and charlist(not supported by MySQL ) among others
* The percentage wildcard is used to match any number of characters starting from zero (0) and more.
* The underscore wildcard is used to match exactly one character.

**MYSQL Regular Expressions (REGEXP) with Syntax & Examples**

**What are regular expressions?**

Regular Expressions help search data matching complex criteria.  We looked at wildcards in the previous tutorial. If you have worked with wildcards before, you may be asking why learn regular expressions when you can get similar results using the wildcards. Because, compared to wildcards, regular expressions allow us to search data matching even more complex criterion.

**Basic syntax**

The basic syntax for a regular expression is as follows

SELECT statements... WHERE fieldname REGEXP 'pattern';

**HERE -**

* **"SELECT statements..."** is the standard SELECT statement
* **"WHERE fieldname"** is the name of the column on which the regular expression is to be performed on.
* **"REGEXP 'pattern'"** REGEXP is the regular expression operator and 'pattern' represents the pattern to be matched by REGEXP. **RLIKE** is the **synonym for REGEXP** and achieves the same results as REGEXP. To avoid confusing it with the LIKE operator, it **better to use REGEXP** instead.

Let's now look at a practical example-

SELECT \* FROM `movies` WHERE `title` REGEXP 'code';

The above query searches for all the movie titles that have the word code in them. It does not matter whether the "code" is at the beginning, middle or end of the title. As long as it is contained in the title then it will be considered.

Let's suppose that we want to search for movies that start with a, b, c or d , followed by any number of other characters, how would we go about to achieve that. We can use a regular expression together with the metacharacters to achieve our desired results.

SELECT \* FROM `movies` WHERE `title` REGEXP '^[abcd]';

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 4 | Code Name Black | Edgar Jimz | 2010 | NULL |
| 5 | Daddy's Little Girls | NULL | 2007 | 8 |
| 6 | Angels and Demons | NULL | 2007 | 6 |
| 7 | Davinci Code | NULL | 2007 | 6 |

Let's now take a close look at our regular expression responsible for the above result.

'^[abcd]' the caret (^) means that the pattern match should be applied at the beginning and the charlist [abcd] means that only movie titles that start with a, b, c or d are returned in our result set.

Let's modify our above script and use the NOT charlist and see what results we will get after executing our query.

SELECT \* FROM `movies` WHERE `title` REGEXP '^[^abcd]';

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **title** | **director** | **year\_released** | **category\_id** |
| 1 | Pirates of the Caribean 4 | Rob Marshall | 2011 | 1 |
| 2 | Forgetting Sarah Marshal | Nicholas Stoller | 2008 | 2 |
| 3 | X-Men |  | 2008 |  |
| 9 | Honey mooners | John Schultz | 2005 | 8 |
| 16 | 67% Guilty |  | 2012 |  |
| 17 | The Great Dictator | Chalie Chaplie | 1920 | 7 |
| 18 | sample movie | Anonymous |  | 8 |
| 19 | movie 3 | John Brown | 1920 | 8 |

Let's now take a close look at our regular expression responsible for the above results.

'^[^abcd]' the caret (^) means that the pattern match should be applied at the beginning and the charlist [^abcd] means that the movie titles starting with any of the enclosed characters is excluded from the result set.

**Regular expression metacharacters**

What we looked at in the above example is the simplest form of a regular expression. Let's now look at more advanced regular expression pattern matches. Suppose we want to search for movie titles that start with the pattern "code" only using a regular expression, how would we go about it? The answer is metacharacters. They allow us to fine tune our pattern search results using regular expressions.

|  |  |  |
| --- | --- | --- |
| **Char** | **Description** | **Example** |
| **\*** | The **asterisk (\*)**metacharacter is used to match zero (0) or more instances of the strings preceding it | *SELECT \* FROM movies WHERE title REGEXP 'da\*';*will give all movies containing characters "da" .For Example, Da Vinci Code , Daddy's Little Girls. |
| **+** | The **plus (+)** metacharacter is used to match one or more instances of strings preceding it. | *SELECT \* FROM `movies` WHERE `title` REGEXP 'mon+';*will give all movies containing characters "mon" .For Example, Angels and Demons. |
| **?** | The question**(?)** metacharacter is used to match zero (0) or one instances of the strings preceding it. | *SELECT \* FROM `categories` WHERE `category\_name` REGEXP 'com?';*will give all the categories containing string com .For Example, comedy , romantic comedy . |
| **.** | The **dot (.)** metacharacter is used to match any single character in exception of a new line. | *SELECT \* FROM movies WHERE `year\_released` REGEXP '200.';*will give all the movies released in the years starting with characters "200" followed by any single character .For Example, 2005,2007,2008 etc. |
| **[abc]** | The **charlist [abc]** is used to match any of the enclosed characters. | *SELECT \* FROM `movies` WHERE `title` REGEXP '[vwxyz]';*will give all the movies containing any single character in "vwxyz" .For Example, X-Men, Da Vinci Code, etc. |
| **[^abc]** | The **charlist [^abc]** is used to match any characters excluding the ones enclosed. | *SELECT \* FROM `movies` WHERE `title` REGEXP '^[^vwxyz]';*will give all the movies containing characters other than the ones in "vwxyz". |
| **[A-Z]** | The **[A-Z]** is used to match any upper case letter. | *SELECT \* FROM `members` WHERE `postal\_address` REGEXP '[A-Z]';*will give all the members that have postal address containing any character from A to Z. .For Example, Janet Jones with membership number 1. |
| **[a-z]** | The **[a-z]** is used to match any lower case letter | *SELECT \* FROM `members` WHERE `postal\_address` REGEXP '[a-z]';*will give all the members that have postal addresses containing any character from  a to z. .For Example, Janet Jones with membership number 1. |
| **[0-9]** | The **[0-9]** is used to match any digit from 0 through to 9. | *SELECT \* FROM `members` WHERE `contact\_number` REGEXP '[0-9]'*will give all the members have submitted contact numbers containing characters "[0-9]" .For Example, Robert Phil. |
| **^** | The **caret (^)** is used to start the match at beginning. | *SELECT \* FROM `movies` WHERE `title` REGEXP '^[cd]';*gives all the movies with the title starting with any of the characters in "cd" .For Example, Code Name Black, Daddy's Little Girls and Da Vinci Code. |
| **|** | The **vertical bar (|)** is used to isolate alternatives. | *SELECT \* FROM `movies` WHERE `title` REGEXP '^[cd]|^[u]';*gives all the movies with the title starting with any of the characters in "cd"  or "u" .For Example, Code Name Black, Daddy's Little Girl, Da Vinci Code and Underworld - Awakening. |
| **[[:<:]]** | The**[[:<:]]** matches the beginning of words. | *SELECT \* FROM `movies` WHERE `title` REGEXP '[[:<:]]for';* gives all the movies with titles starting with the characters. For Example:  Forgetting Sarah Marshal. |
| **[[:>:]]** | The **[[:>:]]**matches the end of words. | *SELECT \* FROM `movies` WHERE `title` REGEXP 'ack[[:>:]]';*  gives all the movies with titles ending with the characters "ack" .For Example, Code Name Black. |
| **[:class:]** | The **[:class:]** matches a character class i.e. *[:alpha:]* to match letters, *[:space:]* to match white space, *[:punct:]* is match punctuations and [:upper:] for upper class letters. | *SELECT \* FROM `movies` WHERE `title` REGEXP '[:alpha:]';*gives all the movies with titles contain letters only .For Example, Forgetting Sarah Marshal, X-Men etc. Movie like Pirates of the Caribbean 4 will be omitted by this query. |

The backslash (\) is used to as an escape character. If we want to use it as part of the pattern in a regular expression, we should use double backslashes (\\)

**Summary**

* Regular expressions provide a powerful and flexible pattern match that can help us implement power search utilities for our database systems.
* REGEXP is the operator used when performing regular expression pattern matches. RLIKE is the synonym
* Regular expressions support a number of metacharacters which allow for more flexibility and control when performing pattern matches.
* The backslash is used as an escape character in regular expressions. It's only considered in the pattern match if double backslashes have used.
* Regular expressions are not case sensitive.

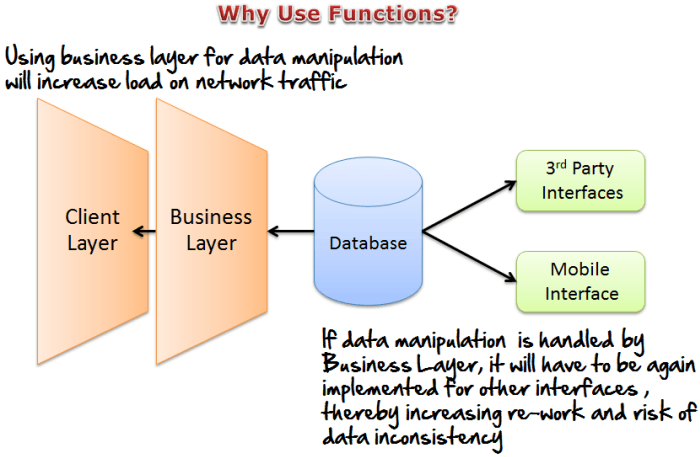
# MySQL Functions: String, Numeric, User-Defined, Stored

### What are functions?

**MySQL can do much more than just store and retrieve data**. We can also **perform manipulations on the data** before retrieving or saving it. That's where MySQL Functions come in. Functions are simply pieces of code that perform some operations and then return a result. Some functions accept parameters while other functions do not accept parameters.

 Let' briefly look at an example of MySQL function. By default, MySQL saves date data types in the format "YYYY-MM-DD". Suppose we have built an application and our users want the date to be returned in the format "DD-MM-YYYY", we can use MySQL built in function DATE\_FORMAT to achieve this. DATE\_FORMAT is one of the most used functions in MySQL. We will look at it in more details as we unfold the lesson.

### Why use functions?



Based on the example given in the introduction, people with experience in computer programming may be thinking *"Why bother MySQL Functions? The same effect can be achieved with scripting/programming language?"*  It's true we can achieve that by writing some procedures/function in the application program.

Getting back to our DATE example in the introduction, for our users to get the data in the desired format, business layer will have to do necessary processing.

This becomes a problem when the application has to integrate with other systems. When we use MySQL functions such as the DATE\_FORMAT, then we can have that functionality embedded into the database and any application that needs the data gets it in the required format. This **reduces re-work in the business logic and reduce data inconsistencies.**

Another reason why we should consider using **MySQL functions is the fact that it can help reducing network traffic in client/server applications**. Business Layer will only need to make call to the stored functions without the need manipulate data .On average, the use of functions can help greatly improve overall system performance.

**Types of functions**

**Built-in functions**

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
* [**Aggregate functions**](https://www.guru99.com/aggregate-functions.html) - operate on all of the above data types and produce summarized result sets.
* **Other functions** - MySQL also supports other types of built in functions but we will limit our lesson to the above named functions only.

Let's now look at each of the functions mentioned above in detail. We will be explaining the most used functions using our "Myflixdb".

**String functions**

We already looked at what string functions do. We will look at a practical example that uses them. In our movies table, the movie titles are stored using combinations of lower and upper case letters. Suppose we want to get a query list that returns the movie titles in upper case letters. We can use the "UCASE" function to do that. It takes a string as a parameter and converts all the letters to upper case. The script shown below demonstrates the use of the "UCASE" function.

SELECT `movie\_id`,`title`, UCASE(`title`) FROM `movies`;

**HERE**

* UCASE(`title`) is the built in function that takes the title as a parameter and returns it in upper case letters with the alias name `upper\_case\_title`.

Executing the above script in MySQL workbench against the Myflixdb gives us the following results shown below.

|  |  |  |
| --- | --- | --- |
| **movie\_id** | **title** | **UCASE('title')** |
| 16 | 67% Guilty | 67% GUILTY |
| 6 | Angels and Demons | ANGELS AND DEMONS |
| 4 | Code Name Black | CODE NAME BLACK |
| 5 | Daddy's Little Girls | DADDY'S LITTLE GIRLS |
| 7 | Davinci Code | DAVINCI CODE |
| 2 | Forgetting Sarah Marshal | FORGETTING SARAH MARSHAL |
| 9 | Honey mooners | HONEY MOONERS |
| 19 | movie 3 | MOVIE 3 |
| 1 | Pirates of the Caribean 4 | PIRATES OF THE CARIBEAN 4 |
| 18 | sample movie | SAMPLE MOVIE |
| 17 | The Great Dictator | THE GREAT DICTATOR |
| 3 | X-Men | X-MEN |

MySQL supports a number of string functions. For a complete list of all the built in string functions, refere to this link <http://dev.mysql.com/doc/refman/5.0/en/string-functions.html> on MySQL website.

**Numeric functions**

As earlier mentioned, these functions operate on numeric data types. We can perform mathematic computations on numeric data in the SQL statements.

**Arithematic operators**

MySQL supports the following arithmatic operators that can be used to perform computations in the SQL statements.

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

Let's now look at examples of each of the above operator

**Integer Division (DIV)**

SELECT 23 DIV 6 ;

Executing the above script gives us the following results.

3

**Division operator (/)**

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

SELECT 23 / 6 ;

Executing the above script gives us the following results.

3.8333

**Subtraction operator (-)**

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

SELECT 23 - 6 ;

Executing the above script gives us 17

**Addition operator (+)**

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

SELECT 23 + 6 ;

Executing the above script gives us 29

**Multiplication operator (\*)**

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

SELECT 23 \* 6 AS `multiplication\_result`;

Executing the above script gives us the following results.

|  |
| --- |
| **multiplication\_result** |
| 138 |

**Modulo operator (-)**

The modulo operator divides N by M and gives us the reminder. Let's now look at the modulo operator example. We will use the same values as in the previous examples.

SELECT 23 % 6 ;

OR

SELECT 23 MOD 6 ;

Executing the above script gives us  5

Let's now look at some of the common numeric functions in MySQL.

**Floor** - this function removes decimals places from a number and rounds it to the nearest lowest number. The script shown below demonstrates its usage.

SELECT FLOOR(23 / 6) AS `floor\_result`;

Executing the above script gives us the following results.

|  |
| --- |
| **Floor\_result** |
| 3 |

**Round** - this function rounds a number with decimal places to the nearest whole number. The script shown below demonstrates its usage.

SELECT ROUND(23 / 6) AS `round\_result`;

Executing the above script gives us the following results.

|  |
| --- |
| **Round\_result** |
| 4 |

**Rand** - this function is used to generate a random number, its value changes every time that the function is called. The script shown below demonstrates its usage.

SELECT RAND() AS `random\_result`;

**Stored functions**

Stored functions are just like built in functions except that you have to define the stored function yourself. Once a stored function has been created, it can be used in SQL statements just like any other function. The basic syntax for creating a stored function is as shown below

CREATE FUNCTION sf\_name ([parameter(s)])

RETURNS data type

DETERMINISTIC

STATEMENTS

**HERE**

* **"CREATE FUNCTION sf\_name ([parameter(s)]) "** is mandatory and tells MySQL server to create a function named `sf\_name' with optional parameters defined in the parenthesis.
* **"RETURNS data type"** is mandatory and specifies the data type that the function should return.
* **"DETERMINISTIC"** means the function will return the same values if the same arguments are supplied to it.
* **"STATEMENTS"** is the procedural code that the function executes.

Let's now look at a practical example that implements a built in function. Suppose we want to know which rented movies are past the return date. We can create a stored function that accepts the return date as the parameter and then compares it with the current date in MySQL server. If the current date is less than the return movie date, then we return "No" else we return "Yes". The script shown below helps us to achieve that.

DELIMITER |

CREATE FUNCTION sf\_past\_movie\_return\_date (return\_date DATE)

RETURNS VARCHAR(3)

DETERMINISTIC

BEGIN

DECLARE sf\_value VARCHAR(3);

IF curdate() > return\_date

THEN SET sf\_value = 'Yes';

ELSEIF curdate() <= return\_date

THEN SET sf\_value = 'No';

END IF;

RETURN sf\_value;

END|

Executing the above script created the stored function `sf\_past\_movie\_return\_date`.

Let's now test our stored function.

SELECT `movie\_id`,`membership\_number`,`return\_date`,CURDATE() ,sf\_past\_movie\_return\_date(`return\_date`) FROM `movierentals`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **movie\_id** | **membership\_number** | **return\_date** | **CURDATE()** | **sf\_past\_movie\_return\_date('return\_date')** |
| 1 | 1 | NULL | 04-08-2012 | NULL |
| 2 | 1 | 25-06-2012 | 04-08-2012 | yes |
| 2 | 3 | 25-06-2012 | 04-08-2012 | yes |
| 2 | 2 | 25-06-2012 | 04-08-2012 | yes |
| 3 | 3 | NULL | 04-08-2012 | NULL |

**User-defined functions**

MySQL also supports user defined functions that extend MySQL. User defined functions are functions that you can create using a programming language such as C, C++ etc. and then add them to MySQL server. Once added, they can be used just like any other function.

**Summary**

* Functions allow us to enhance the capabilities of MySQL.
* Functions always return a value and can optionally accept parameters.
* Built in functions are functions that are shipped with MySQL. They can be categorized according to the data types that they operate on i.e. strings, date and numeric built in functions.
* Stored functions are created by the user within MySQL server and can be used in SQL statements.
* User defined functions are created outside MySQL and can be incorporated into MySQL server.

**MySQL Aggregate Functions Tutorial : SUM, AVG, MAX, MIN , COUNT, DISTINCT**

Aggregate Functions are all about

* Performing  calculations on multiple rows
* Of a single column of a table
* And returning a single value.

The ISO standard defines five (5) aggregate functions namely;

1) COUNT  
2) SUM  
3) AVG  
4) MIN  
5) MAX

**Why use aggregate functions.**

From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

>Aggregate functions allow us to easily produce summarized data from our database.

For instance, from our myflix database , management may require following reports

* Least rented movies.
* Most rented movies.
* Average number that each movie is rented out in a month.

We easily produce above reports using aggregate functions.

Let's look into aggregate functions in detail.

**COUNT Function**

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. **All aggregate functions by default exclude nulls values before working on the data.**

COUNT (\*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (\*) also considers Nulls and duplicates.

The table shown below shows data in movierentals table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **reference\_ number** | **transaction\_ date** | **return\_date** | **membership\_ number** | **movie\_id** | **movie\_ returned** |
| 11 | 20-06-2012 | NULL | 1 | 1 | 0 |
| 12 | 22-06-2012 | 25-06-2012 | 1 | 2 | 0 |
| 13 | 22-06-2012 | 25-06-2012 | 3 | 2 | 0 |
| 14 | 21-06-2012 | 24-06-2012 | 2 | 2 | 0 |
| 15 | 23-06-2012 | NULL | 3 | 3 | 0 |

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

SELECT COUNT(`movie\_id`) FROM `movierentals` WHERE `movie\_id` = 2;

Executing the above query in MySQL workbench against myflixdb gives us the following results.

|  |
| --- |
| **COUNT('movie\_id')** |
| 3 |

**DISTINCT Keyword**

[](https://www.guru99.com/images/DistinctApple.png)

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

To appreciate the concept of Distinct, lets execute a simple query

SELECT `movie\_id` FROM `movierentals`;

|  |
| --- |
| **movie\_id** |
| 1 |
| 2 |
| 2 |
| 2 |
| 3 |

Now let's execute the same query with the distinct keyword -

SELECT DISTINCT `movie\_id` FROM `movierentals`;

As shown below , distinct omits duplicate records from the results.

|  |
| --- |
| **movie\_id** |
| 1 |
| 2 |
| 3 |

**MIN function**

The MIN function **returns the smallest value in the specified table field**.

As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

The following query helps us achieve that

SELECT MIN(`year\_released`) FROM `movies`;

Executing the above query in MySQL workbench against myflixdb gives us the following results.

|  |
| --- |
| **MIN('year\_released')** |
| 2005 |

**MAX function**

Just as the name suggests, the MAX function is the opposite of the MIN function. It **returns the largest value from the specified table field**.

Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.

SELECT MAX(`year\_released`) FROM `movies`;

Executing the above query in MySQL workbench using myflixdb gives us the following results.

|  |
| --- |
| **MAX('year\_released')** |
| 2012 |

**SUM function**

Suppose we want  a report that gives total  amount of payments made so far. We can use the MySQL **SUM** function which **returns the sum of all the values in the specified column**. **SUM works on numeric fields only**. **Null values are excluded from the result returned.**

The following table shows the data in payments table-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **payment\_ id** | **membership\_ number** | **payment\_ date** | **description** | **amount\_ paid** | **external\_ reference \_number** |
| 1 | 1 | 23-07-2012 | Movie rental payment | 2500 | 11 |
| 2 | 1 | 25-07-2012 | Movie rental payment | 2000 | 12 |
| 3 | 3 | 30-07-2012 | Movie rental payment | 6000 | NULL |

The query shown below gets the all payments made and sums them up to return a single result.

SELECT SUM(`amount\_paid`) FROM `payments`;

Executing the above query in MySQL workbench against the myflixdb gives the following results.

|  |
| --- |
| **SUM('amount\_paid')** |
| 10500 |

**AVG function**

MySQL  AVG function **returns the average of the values in a specified column**. Just like the SUM function, it **works only on numeric data types**.

 Suppose we want to find the average amount paid. We can use the following query -

SELECT AVG(`amount\_paid`) FROM `payments`;

Executing the above query in MySQL workbench, gives us the following results.

|  |
| --- |
| **AVG('amount\_paid')** |
| 3500 |

**Summary**

* MySQL supports all the five (5) ISO standard aggregate functions COUNT, SUM, AVG, MIN and MAX.
* SUM and AVG functions only work on numeric data.
* If you want to exclude duplicate values from the aggregate function results, use the DISTINCT keyword. The ALL keyword includes even duplicates. If nothing is specified the ALL is assumed as the default.
* Aggregate functions can be used in conjunction with other SQL clauses such as GROUP BY

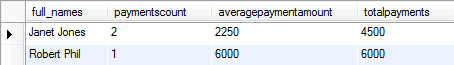
**Brain Teaser**

You think aggregate functions are easy. Try this!

The following example groups members by name, counts the total number of payments, the average payment amount and the grand total of the payment amounts.

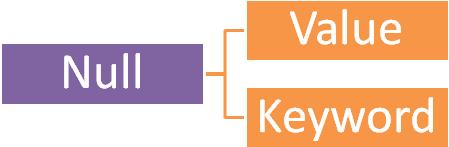
SELECT m.`full\_names`,COUNT(p.`payment\_id`) AS `paymentscount`,AVG(p.`amount\_paid`) AS `averagepaymentamount`,SUM(p.`amount\_paid`) AS `totalpayments` FROM members m, payments p WHERE m.`membership\_number` = p.`membership\_number` GROUP BY m.`full\_names`;

Executing the above example in MySQL workbench gives us the following results.

[](https://www.guru99.com/images/practical_example.png)

# MySQL IS NULL & IS NOT NULL Tutorial with EXAMPLES

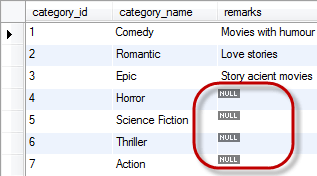
  In SQL Null is both a value as well as a keyword. Let's look into NULL value first -



**Null as a Value**

In simple terms, NULL is simply a place holder for data that does not exist. When performing insert operations on tables, they will be times when some  field values will not be available.

In order to meet the requirements of true relational database management systems, MySQL uses NULL as the place holder for the values that have not been submitted. The screenshot below shows how NULL values look in database.

[](https://www.guru99.com/images/Query.png)

Let's now look at some of the basics for NULL before we go further into the discussion.

* **NULL is not a data type** - this means it is not recognized as an "int", "date" or any other defined data type.
* **Arithmetic operations** involving **NULL** always **return NULL** for example, 69 + NULL = NULL.
* All **aggregate functions** **affect only rows that do not have NULL values**.

Let's now demonstrate how the count function treats null values.  Let's see the current contents of the members table-

SELECT \* FROM `members`;

Executing the above script gives us the following results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_ number** | **full\_ names** | **gender** | **date\_of\_ birth** | **physical\_ address** | **postal\_ address** | **contact\_ number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 2 | Janet Smith Jones | Female | 23-06-1980 | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 4 | Gloria Williams | Female | 14-02-1984 | 2nd Street 23 | NULL | NULL | NULL |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 | lwolowitz[at]email.me |

Let's count all members who have updated their contact\_number

SELECT COUNT(contact\_number) FROM `members`;

Executing the above query gives us the following results.

|  |
| --- |
| **COUNT(contact\_number)** |
| 7 |

Note: Values that are NULL have not been included

**What is NOT?**

The NOT logical operator is used to test for Boolean conditions and returns true if the condition is false. The NOT operator returns false if the condition been tested is true

|  |  |
| --- | --- |
| Condition | NOT Operator Result |
| True | False |
| False | True |

**Why use NOT null?**

There will be cases when we will have to perform computations on a query result set and return the values. Performing any arithmetic operations on columns that have the NULL value returns null results. In order to avoid such situations from happening, we can employ the use of the NOT NULL clause to limit the results on which our data operates.

**NOT NULL Values**

Let's suppose that we want to create a table with certain fields that should always be supplied with values when inserting new rows in a table. We can use the NOT NULL clause on a given field when creating the table.

The example shown below creates a new table that contains employee's data. The employee number should always be supplied

CREATE TABLE `employees`(

employee\_number int NOT NULL,

full\_names varchar(255) ,

gender varchar(6)

);

Let's now try to insert a new record without specifying the employee name and see what happens.

INSERT INTO `employees` (full\_names,gender) VALUES ('Steve Jobs', 'Male');

Executing the above script in MySQL workbench gives the following error -

[MySQL IS NULL & IS NOT NULL Tutorial with Examples](https://www.guru99.com/images/ErrorCode.png)

**NULL Keywords**

NULL can also be used as a keyword when performing Boolean operations on values that include NULL. The "IS/NOT" keyword is used in conjunction with the NULL word for such purposes. The basic syntax when null is used as a keyword is as follows

`comlumn\_name' IS NULL

`comlumn\_name' NOT NULL

**HERE**

* ***"IS NULL"*** is the keyword that performs the Boolean comparison. It returns true if the supplied value is NULL and false if the supplied value is not NULL.
* ***"NOT NULL"***is the keyword that performs the Boolean comparison. It returns true if the supplied value is not NULL and false if the supplied value is null.

Let's now look at a practical example that uses the NOT NULL keyword to eliminate all the column values that have null values.

Continuing with the example above , suppose we need details of members whose contact number is not null . We can execute a query like

SELECT \* FROM `members` WHERE contact\_number IS NOT NULL;

Executing the above query gives only records where contact number is not null.

Suppose we want member records where contact number is null. We can use following query

SELECT \* FROM `members` WHERE contact\_number IS NULL;

Executing the above query gives member details whose contact number is NULL

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_ number** | **full\_names** | **gender** | **date\_of\_birth** | **physical\_address** | **postal\_address** | **contact\_ number** | **email** |
| 1 | Janet Jones | Female | 21-07-1980 | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) |
| 3 | Robert Phil | Male | 12-07-1989 | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) |
| 5 | Leonard Hofstadter | Male | NULL | Woodcrest | NULL | 845738767 | NULL |
| 6 | Sheldon Cooper | Male | NULL | Woodcrest | NULL | 976736763 | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | Woodcrest | NULL | 938867763 | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | Woodcrest | NULL | 987636553 | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | SouthPark | P.O. Box 4563 | 987786553 | lwolowitz[at]email.me |

**Comparing null values**

**Three-value logic** - performing Boolean operations on conditions that involve NULL can either return **"Unknown", "True" or "False".**

For example, **using the "IS NULL" keyword** when doing comparison operations **involving NULL** can either return **true** or **false**. Using other comparison operators returns **"Unknown"(NULL).**

**Suppose you compare number five with 5**

SELECT 5 =5;

The query result is 1 which means TRUE

|  |
| --- |
| **5 =5** |
| 1 |

Let's do the same operation with NULL

SELECT NULL = NULL;

|  |
| --- |
| **NULL = NULL** |
| NULL |

Let's look at another example

SELECT 5 > 5;

|  |
| --- |
| **5 > 5** |
| 0 |

The query result is 0 which means FALSE

Let's look at same example using NULL

SELECT NULL > NULL;

|  |
| --- |
| **NULL > NULL** |
| NULL |

Lets use the IS NULL keyword

SELECT 5 IS NULL;

|  |
| --- |
| **5 IS NULL** |
| 0 |

The query result is 0 which is FALSE

SELECT NULL IS NULL;

|  |
| --- |
| **NULL IS NULL** |
| 1 |

The query result is 1 which is TRUE

**Summary**

* NULL is a value place holder for optional table fields.
* MySQL treats the NULL value differently from other data types. The NULL values when used in a condition evaluates to the false Boolean value.
* The NOT logical operate is used to test for Boolean values and evaluates to true if the Boolean value is false and false if the Boolean value is true.
* The NOT NULL clause is used to eliminate NULL values from a result set
* Performing arithmetic operations on NULL values always returns NULL results.
* The comparison operators such as [, =, etc.] cannot be used to compare NULL values.

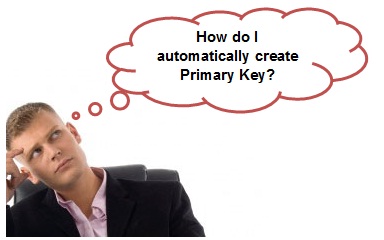
**MySQL AUTO\_INCREMENT with Examples**

**What is auto increment?**

Auto Increment is a function that operates on numeric data types. It automatically generates sequential numeric values every time that a record is inserted into a table for a field defined as auto increment.

**When use auto increment?**

In the lesson on database normalization, we looked at how data can be stored with minimal redundancy, by storing data into many small tables ,related to each other using primary and foreign keys.

[](https://www.guru99.com/images/AutoIncrement.jpg)

A primary key must be unique as it uniquely identifies a row in a database. But, how can we ensure that the primary key is always unique? One of the possible solutions would be, to use a formula to generate the primary key, which checks for existence of the key, in the table, before adding data. This may work well but as you can see the approach is complex and not foolproof. In order to avoid such complexity and to ensure that the primary key is always unique, we can use MySQL's Auto increment feature to generate primary keys. Auto increment is used with the INT data type. The INT data type supports both signed and unsigned values.  Unsigned data types can only contain positive numbers. As a best practice, it is recommended to define the unsigned constraint on the auto increment primary key.

**Auto increment syntax**

Let's now look at the script used to create the movie categories table.

CREATE TABLE `categories` (

`category\_id` int(11) AUTO\_INCREMENT,

`category\_name` varchar(150) DEFAULT NULL,

`remarks` varchar(500) DEFAULT NULL,

PRIMARY KEY (`category\_id`)

);

*Notice* the "AUTO\_INCREMENT" on the category\_id field. This causes the category Id to be automatically generated every time a new row is inserted into the table. It is not supplied when inserting data into the table, MySQL generates it.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record

Let's examine the current contents of the categories table.

SELECT \* FROM `categories`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |
| --- | --- | --- |
| **category\_id** | **category\_name** | **remarks** |
| 1 | Comedy | Movies with humour |
| 2 | Romantic | Love stories |
| 3 | Epic | Story acient movies |
| 4 | Horror | NULL |
| 5 | Science Fiction | NULL |
| 6 | Thriller | NULL |
| 7 | Action | NULL |
| 8 | Romantic Comedy | NULL |

Let's now insert a new category into the categories table .

INSERT INTO `categories` (`category\_name`) VALUES ('Cartoons');

Executing the above script against the myflixdb in MySQL workbench gives us the following results shown below.

|  |  |  |
| --- | --- | --- |
| **category\_id** | **category\_name** | **remarks** |
| 1 | Comedy | Movies with humour |
| 2 | Romantic | Love stories |
| 3 | Epic | Story acient movies |
| 4 | Horror | NULL |
| 5 | Science Fiction | NULL |
| 6 | Thriller | NULL |
| 7 | Action | NULL |
| 8 | Romantic Comedy | NULL |
| 9 | Cartoons | NULL |

Note we didn't supply the category id. MySQL automatically generated it for us because the category id is defined as auto increment.

If you want to get the last insert id that was generated by MySQL, you can use the LAST\_INSERT\_ID function to do that. The script shown below gets the last id that was generated.

SELECT LAST\_INSERT\_ID();

Executing the above script gives the last Auto increment number generated by the INSERT query. The results are shown below.

[MySQL AUTO_INCREMENT with Examples](https://www.guru99.com/images/last_insert_id.png)

**Summary**

* Auto increment attribute when specified on a column with a numeric data types, generates numbers sequentially whenever a new row is added into the database.
* The Auto increment is commonly used to generate primary keys.
* The defined data type on the Auto increment should be large enough to accommodate many records. Defining TINYINT as the data type for an auto increment field limits the number of records that can be added to the table to 255 only since any values beyond that would not be accepted by the TINYINT data type.
* It is considered a good practice to specify the unsigned constraint on auto increment primary keys to avoid having negative numbers.
* When a row is deleted from a table, its auto incremented id is not re-used. MySQL continues generating new numbers sequentially.
* By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record
* To let AUTO\_INCREMENT sequence start with another value , use AUTO\_INCREMENT = 10

**MYSQL - ALTER, DROP, RENAME, MODIFY**

**WHAT IS THE ALTER COMMAND?**

As the saying goes **Change is the only constant**

With time business requirements change as well. As business requirements change, Database designs need changing as well.

MySQL provides the **ALTER** function that helps us **incorporate the changes to the already existing database design**.

The alter command is used to modify an existing database, table, view or other database objects that might need to change during the life cycle of a database.

Let's suppose that we have completed our database design and it has been implemented. Our database users are using it and then they realize some of the vital information was left out in the design phase. They don't want to lose the existing  data but just want to incorporate the new information. The alter command comes in handy in such situations. We can use the alter command to change the data type of a field from say string to numeric, change the field name to a new name or even add a new column in a table.

**Alter- syntax**

The basic syntax used to add a column to an already existing table is shown below

ALTER TABLE `table\_name` ADD COLUMN `column\_name` `data\_type`;

**HERE**

* **"ALTER TABLE `table\_name`"** is the command that tells MySQL server to modify the table named `table\_name`.
* **"ADD COLUMN `column\_name` `data\_type`"** is the command that tells MySQL server to add a new column named `column\_name` with data type `data\_type'.

Let's suppose that Myflix has introduced online billing and payments. Towards that end, we have been asked to add a field for the credit card number in our members table. We can use the ALTER command to do that. Let's first look at the structure of the members table before we make any amendments. The script shown below helps us to do that.

SHOW COLUMNS FROM `members`;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| full\_names | varchar(350) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

 We can use the script shown below to add a new field to the members table.

ALTER TABLE `members` ADD COLUMN `credit\_card\_number` VARCHAR(25);

Executing the above script in MySQL against the Myflixdb adds a new column named credit card number to the members table with VARCHAR as the data type. Executing the show columns script gives us the following results.

SHOW COLUMNS FROM `members`;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| full\_names | varchar(350) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |
| credit\_card\_number | varchar(25) | YES |  |  |  |

As you can see from the results returned, credit card number has been added to the members table. The data contained in the members' data is not affected by the addition of the new column.

**WHAT IS THE DROP COMMAND?**

The DROP command is used to

1. Delete a database from MySQL server
2. Delete an object (like Table , Column)from a database.

Let's now look at practical examples that make use of the DROP command.

In our previous example on the Alter Command, we added a column named credit card number to the members table.

Suppose the online billing functionality will take some time and we want to DROP the credit card column

We can use the following script

ALTER TABLE `members` DROP COLUMN `credit\_card\_number`;

Executing the above script drops the column credit\_card\_number from the members table

 Let's now look at the columns in the members table to confirm if our column has been dropped.

SHOW COLUMNS FROM `members`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| full\_names | varchar(350) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

*Notice that the credit card number has been dropped from the fields list.*

**DROP TABLE**

The syntax to DROP a table from Database is as follow -

DROP TABLE `sample\_table`;

Let'look at an example

DROP TABLE `categories\_archive`;

Executing the above script deletes the table named ` categories\_archive ` from our database.

**WHAT IS THE RENAME COMMAND?**

The rename command is used to **change the name of an existing database object(like Table,Column) to a new name**.

**Renaming a table does not make it to lose any data is contained within it.**

Syntax:-

The rename command has the following basic syntax.

RENAME TABLE `current\_table\_name` TO `new\_table\_name`;

Let's suppose that we want to rename the movierentals table to movie\_rentals, we can use the script shown below to achieve that.

RENAME TABLE `movierentals` TO `movie\_rentals`;

Executing the above script renames the table `movierentals` to `movie\_rentals`.

We will now rename the movie\_rentals table back to its original name.

RENAME TABLE `movie\_rentals` TO `movierentals`;

**CHANGE KEYWORD**

Change Keywords allows you to

1. Change Name of Column
2. Change Column Data Type
3. Change Column Constraints

 Let's look at an example. The full names field in the members table is of varchar data type and has a width of 150.

SHOW COLUMNS FROM `members`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| full\_names | varchar(150) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

Suppose we want to

1. Change the field name from "full\_names" to "fullname
2. Change it to char data type with a width of 250
3. Add a NOT NULL constraint

 We can accomplish this using the change command as follows -

ALTER TABLE `members` CHANGE COLUMN `full\_names` `fullname` char(250) NOT NULL;

Executing the above script in MySQL workbench against myflixdb and then executing the show columns script given above gives the following results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| fullnames | char(250) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

**MODIFY KEYWORD**

**The MODIFY Keyword allows you to**

1. Modify Column Data Type
2. Modify Column Constraints

In the CHANGE example above, we had to change the field name as well other details. **Omitting the field name from the CHANGE statement will generate an error.**Suppose we are only interested in changing the data type and constraints on the field without affecting the field name, we can use the MODIFY keyword to accomplish that.

The script below changes the width of "fullname" field from 250 to 50.

ALTER TABLE `members`MODIFY `fullname` char(50) NOT NULL;

Executing the above script in MySQL workbench against myflixdb and then executing the show columns script given above gives the following results shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| fullnames | char(50) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

**AFTER KEYWORD**

Suppose that we want to add a new column at a specific position in the  table.

We can use the alter command together with the AFTER keyword.

The script below adds "date\_of\_registration" just after the date of birth in the members table.

ALTER TABLE `members` ADD `date\_of\_registration` date NULL AFTER `date\_of\_birth`;

Executing the above script in MySQL workbench against myflixdb and then executing the show columns script given above gives the following results shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| membership\_number | int(11) | NO | PRI | NULL | auto\_increment |
| fullnames | char(50) | NO |  | NULL |  |
| gender | varchar(6) | YES |  | NULL |  |
| date\_of\_birth | date | YES |  | NULL |  |
| date\_of\_registration | date | YES |  | NULL |  |
| physical\_address | varchar(255) | YES |  | NULL |  |
| postal\_address | varchar(255) | YES |  | NULL |  |
| contact\_number | varchar(75) | YES |  | NULL |  |
| email | varchar(255) | YES |  | NULL |  |

Note: The Hilighted row is **added after date\_of\_birth** cloumn

**Summary**

* The alter command is used when we want to modify a database or any object contained in the database.
* The drop command is used to delete databases from MySQL server or objects within a database.
* The rename command is used to change the name of a table to a new table name.
* The Change keyword allows you to change a column name , data type and constraints
* The Modify Keyword allows you to modify a column data type and constraints
* The After keyword is used to specify position of a column in a table

**MySQL LIMIT & OFFSET with Examples**

**What is the LIMIT keyword?**

The limit keyword is used to limit the number of rows returned in a  query result.

It can be used in conjunction with the SELECT, UPDATE OR DELETE commands LIMIT keyword syntax

The syntax for the LIMIT keyword is as follows

SELECT {fieldname(s) | \*} FROM tableName(s) [WHERE condition] LIMIT N;

**HERE**

* **"SELECT {fieldname(s) | \*} FROM tableName(s)"** is the SELECT statement containing the fields that we would like to return in our query.
* **"[WHERE condition]"** is optional but when supplied, can be used to specify a filter on the result set.
* **"LIMIT  N"** is the keyword and **N** is any number starting from 0, putting 0 as the limit does not return any records in the query. Putting a number say 5 will return five records. If the records in the specified table are less than N, then all the records from the queried table are returned in the result set.

**Let's look at an example -**

SELECT \* FROM members LIMIT 2;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_ number** | **full\_ names** | **gender** | **date\_of \_birth** | **date\_of \_registration** | **physical\_ address** | **postal\_ address** | **contact\_ number** | **email** | **credit\_ card\_ number** |
| 1 | Janet Jones | Female | 21-07-1980 | NULL | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) | NULL |
| 2 | Janet Smith Jones | Female | 23-06-1980 | NULL | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) | NULL |

As you can see from the above screenshot, only two members have been returned.

**Getting a list of ten (10) members only from the database**

Let's suppose that we want to get a list of the first 10 registered members from the Myflix database. We would use the following script to achieve that.

SELECT \* FROM members LIMIT 10;

Executing the above script gives us the results shown below

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_ number** | **full\_ names** | **gender** | **date\_of \_birth** | **date\_of \_registration** | **physical\_ address** | **postal\_ address** | **contact\_ number** | **email** | **credit\_ card\_ number** |
| 1 | Janet Jones | Female | 21-07-1980 | NULL | First Street Plot No 4 | Private Bag | 0759 253 542 | [janetjones@yagoo.cm](mailto:janetjones@yagoo.cm) | NULL |
| 2 | Janet Smith Jones | Female | 23-06-1980 | NULL | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) | NULL |
| 3 | Robert Phil | Male | 12-07-1989 | NULL | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) | NULL |
| 4 | Gloria Williams | Female | 14-02-1984 | NULL | 2nd Street 23 | NULL | NULL | NULL | NULL |
| 5 | Leonard Hofstadter | Male | NULL | NULL | Woodcrest | NULL | 845738767 | NULL | NULL |
| 6 | Sheldon Cooper | Male | NULL | NULL | Woodcrest | NULL | 976736763 | NULL | NULL |
| 7 | Rajesh Koothrappali | Male | NULL | NULL | Woodcrest | NULL | 938867763 | NULL | NULL |
| 8 | Leslie Winkle | Male | 14-02-1984 | NULL | Woodcrest | NULL | 987636553 | NULL | NULL |
| 9 | Howard Wolowitz | Male | 24-08-1981 | NULL | SouthPark | P.O. Box 4563 | 987786553 | lwolowitz[at]email.me | NULL |

Note only 9 members have been returned in our query since N in the LIMIT clause is greater than the number of total records in our table.

Re-writing the above script as follows

SELECT \* FROM members LIMIT 9;

Only returns 9 rows in our query result set.

**Using the OFF SET in the LIMIT query**

The **OFF SET** value is also most often used together with the LIMIT keyword. The OFF SET value allows us to specify which row to start from retrieving data

Let's suppose that we want to get a limited number of members starting from the middle of the rows, we can use the LIMIT keyword together with the offset value to achieve that. The script shown below gets data starting the second row and limits the results to 2.

SELECT \* FROM `members` LIMIT 1, 2;

Executing the above script in MySQL workbench against the myflixdb gives the following results.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **membership\_ number** | **full\_ names** | **gender** | **date\_of \_birth** | **date\_of \_registration** | **physical\_ address** | **postal\_ address** | **contact\_ number** | **email** | **credit\_ card\_ number** |
| 2 | Janet Smith Jones | Female | 23-06-1980 | NULL | Melrose 123 | NULL | NULL | [jj@fstreet.com](mailto:jj@fstreet.com) | NULL |
| 3 | Robert Phil | Male | 12-07-1989 | NULL | 3rd Street 34 | NULL | 12345 | [rm@tstreet.com](mailto:rm@tstreet.com) | NULL |

Note that here **OFFSET = 1** Hence row#2 is returned & **Limit = 2**, Hence only 2 records are returned

**When should we use the LIMIT keyword?**

Let's suppose that we are developing the application that runs on top of myflixdb. Our system designer have asked us to limit the number of records displayed on a page to say 20 records per page to counter slow load times. How do we go about implementing the system that meets such user requirements? The LIMIT keyword comes in handy in such situations. We would be able to limit the results returned from a query to 20 records only per page.

**Summary**

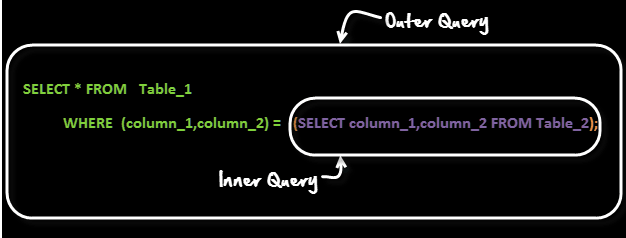
* The LIMIT keyword of is used to limit the number of rows returned from a result set.
* The LIMIT number can be any number from zero (0) going upwards. When zero (0) is specified as the limit, no rows are returned from the result set.
* The OFF SET value allows us to specify which row to start from retrieving data
* It can be used in conjunction with the SELECT, UPDATE OR DELETE commands LIMIT keyword syntax

# MySQL SubQuery Tutorial with Examples

### What are sub queries?

A sub query is a select query that is contained inside another query. The inner select query is usually used to determine the results of the outer select query.

Let's look into the sub query syntax -



A common customer complaint at the MyFlix Video Library is the low number of movie titles. The management wants to buy movies for a category which has least number of titles.

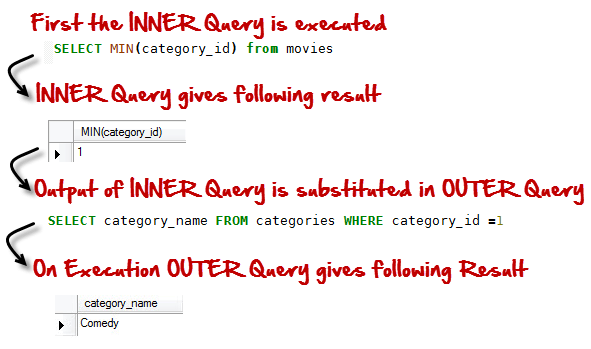
You can use a query like

SELECT category\_name FROM categories WHERE category\_id =( SELECT MIN(category\_id) from movies);

It gives a result

[MySQL SubQuery Tutorial with Examples](https://www.guru99.com/images/row_sub_query.png)

Let's see how this query works

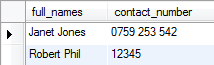
[](https://www.guru99.com/images/SubQueryExecution.png)

The above is a form of **Row Sub-Query**. In such sub-queries the , inner query can give only ONE result. The permissible operators when work with row subqueries are [=, >, =, <=, ,!=,  ]

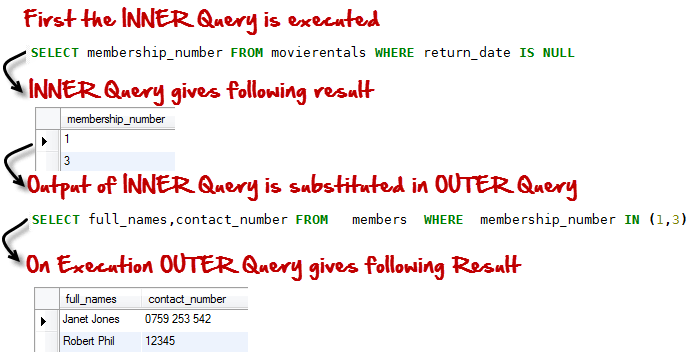
Let's look at another example ,

Suppose you want Names and Phone numbers of members of people who have rented a movie and are yet to return them.  Once you get Names and Phone Number you call them up to give a reminder. You can use a query like

SELECT full\_names,contact\_number FROM members WHERE membership\_number IN (SELECT membership\_number FROM movierentals WHERE return\_date IS NULL );



Let's see how this query works

[](https://www.guru99.com/images/TableSubQuery.png)

In this case, the inner query returns more than one results. The above is type of T**able sub-quer**y.

Till now we have seen two queries , lets now see an example of **triple query**!!!

Suppose the management wants to reward the highest paying member.

We can run a query like

Select full\_names From members WHERE membership\_number = (SELECT membership\_number FROM payments WHERE amount\_paid = (SELECT MAX(amount\_paid) FROM payments));

The above query gives the following result -

[https://www.guru99.com/images/SubQueryResult1.jpg](https://www.guru99.com/images/SubQueryResult1.jpg)

### Sub-Queries Vs Joins!

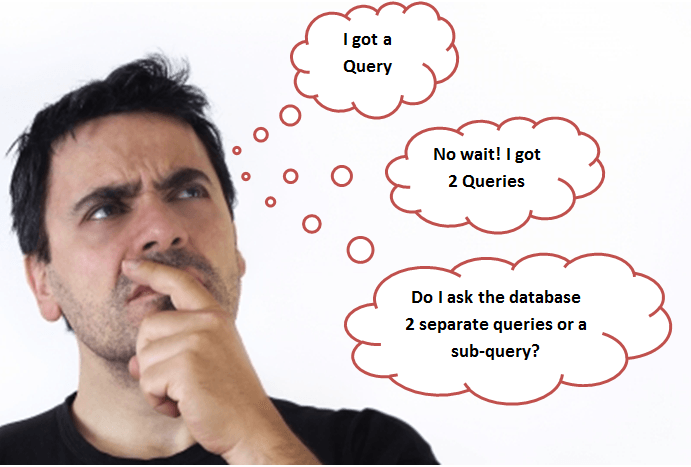
When compare with Joins , sub-queries are simple to use and easy to read. They are not as complicated as Joins

Hence there are frequently used by SQL beginners.

But sub-queries have performance issues.  Using a join instead of a sub-query can at times give you upto 500 times performance boost.

Given a choice, it is recommended to use a JOIN over a sub query.

**Sub-Queries  should only be used as a fallback solution when you cannot use a JOIN operation to achieve the above**



**Summary**

* Subqueries are embedded queries inside another query. The embedded query is known as the inner query and the container query is known as the outer query.
* Sub queries are easy to use, offer great flexibility and can be easily broken down into single logical components making up the query which is very useful when[Testing](https://www.guru99.com/software-testing.html)and debugging the queries.
* MySQL supports three types of subqueries, scalar, row and table subqueries.
* Scalar sub queries only return a single row and single column.
* Row sub queries only return a single row but can have more than one column.
* Table subqueries can return multiple rows as well as columns.
* Subqueries can also be used in INSERT, UPDATE and DELETE queries.
* For performance issues, when it comes to getting data from multiple tables, it is strongly recommended to use JOINs instead of subqueries. Sub queries should only be used with good reason.

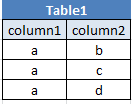
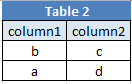
**MySQL UNION - Complete Tutorial**

**What is a union?**

Unions combine the results from multiple SELECT queries into a consolidated result set.

The only requirements for this to work is that the number of columns should be the same from all the SELECT queries which needs to be combined .

Suppose we have two tables as follows

[[](https://www.guru99.com/images/Table1%281%29.png)](https://www.guru99.com/images/Table1%281%29.png)[](https://www.guru99.com/images/Table2%281%29.png)

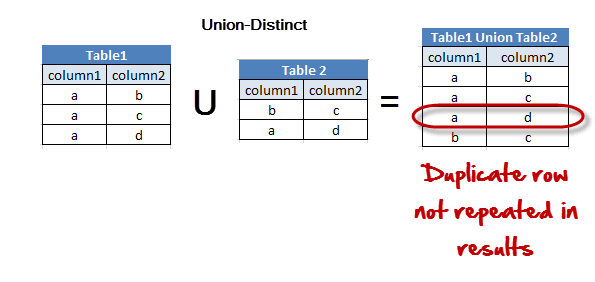
Let's now create a UNION query to combines both tables using DISTINCT

SELECT `column1`,` column1 FROM `table1`

UNION DISTINCT

SELECT ` column1`,` column1` FROM `table2`;

Here duplicate rows are removed and only unique rows are returned.

[](https://www.guru99.com/images/Table1UnionTable2Distinct.png)

Note: MySQL uses the DISTINCT clause as default when executing UNION queries if nothing is specified.

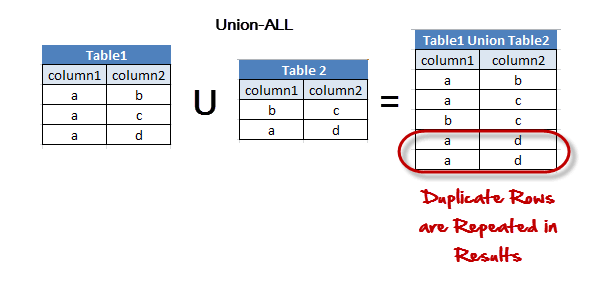
Let's now create a UNION query to combines both tables using ALL

SELECT `column1`,` column1` FROM `table1`

UNION ALL

SELECT ` column1`,` column1` FROM `table2`;

Here duplicate rows are included and since we use ALL.

[](https://www.guru99.com/images/Table1UnionTable2ALL.png)

**Why use unions**

Suppose there is a flaw in your database design and you are using two different tables meant for the same purpose. You want to consolidate these two tables into one while omitting any duplicate records from creeping into the new table. You can use UNION in such cases.

**Summary**

* The UNION command is used to combine more than one SELECT query results into a single query contain rows from all the select queries.
* The number of columns and data types in the SELECT statements must be the same in order for the UNION command to work.
* The DISTINCT clause is used to eliminate duplicate values from the UNION query result set. MySQL uses the DISTINCT clause as the default when executing UNION queries if nothing is specified.
* The ALL clause is used to return all even the duplicate rows in the UNION query.

**Practical examples using MySQL workbench**

In our myFlixDB lets combine

membership\_number and full\_names from Members table

with

movie\_id and title from movies table

We can use the following query

SELECT `membership\_number`,`full\_names` FROM `members`

UNION

SELECT `movie\_id`,`title` FROM `movies`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

|  |  |
| --- | --- |
| **membership\_number** | **full\_names** |
| 1 | Janet Jones |
| 2 | Janet Smith Jones |
| 3 | Robert Phil |
| 4 | Gloria Williams |
| 5 | Leonard Hofstadter |
| 6 | Sheldon Cooper |
| 7 | Rajesh Koothrappali |
| 8 | Leslie Winkle |
| 9 | Howard Wolowitz |
| 16 | 67% Guilty |
| 6 | Angels and Demons |
| 4 | Code Name Black |
| 5 | Daddy's Little Girls |
| 7 | Davinci Code |
| 2 | Forgetting Sarah Marshal |
| 9 | Honey mooners |
| 19 | movie 3 |
| 1 | Pirates of the Caribean 4 |
| 18 | sample movie |
| 17 | The Great Dictator |
| 3 | X-Men |

**MySQL JOINS Tutorial: INNER, OUTER, LEFT, RIGHT, CROSS**

**What are JOINS?**

Joins help retrieving data from two or more database tables.

The tables are mutually related using primary and foreign keys.

Note: JOIN is the most misunderstood topic amongst SQL leaners. For sake of simplicity and ease of understanding , we will be using a new Database to practice sample.  As shown below

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **first\_name** | **last\_name** | **movie\_id** |
| 1 | Adam | Smith | 1 |
| 2 | Ravi | Kumar | 2 |
| 3 | Susan | Davidson | 5 |
| 4 | Jenny | Adrianna | 8 |
| 6 | Lee | Pong | 10 |
| **id** | **title** | | **category** |
| 1 | ASSASSIN'S CREED: EMBERS | | Animations |
| 2 | Real Steel(2012) | | Animations |
| 3 | Alvin and the Chipmunks | | Animations |
| 4 | The Adventures of Tin Tin | | Animations |
| 5 | Safe (2012) | | Action |
| 6 | Safe House(2012) | | Action |
| 7 | GIA | | 18+ |
| 8 | Deadline 2009 | | 18+ |
| 9 | The Dirty Picture | | 18+ |
| 10 | Marley and me | | Romance |

**Types of joins**

**Cross JOIN**

Cross JOIN is a simplest form of JOINs which matches each row from one database table to all rows of another.

In other words it gives us combinations of each row of first table with all records in second table.

Suppose we want to get all member records against all the movie records, we can use the script shown below to get our desired results.

[](https://www.guru99.com/images/CrossJoin.png)

SELECT \* FROM `movies` CROSS JOIN `members`

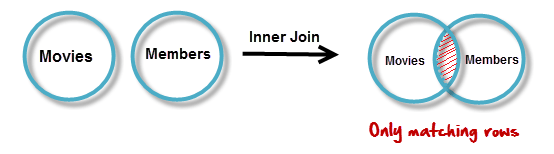
Executing the above script in MySQL workbench gives us the following results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **id** | **title** |  | **id** | **first\_name** | **last\_name** | **movie\_id** |
| 1 | ASSASSIN'S CREED: EMBERS | Animations | 1 | Adam | Smith | 1 |
| 1 | ASSASSIN'S CREED: EMBERS | Animations | 2 | Ravi | Kumar | 2 |
| 1 | ASSASSIN'S CREED: EMBERS | Animations | 3 | Susan | Davidson | 5 |
| 1 | ASSASSIN'S CREED: EMBERS | Animations | 4 | Jenny | Adrianna | 8 |
| 1 | ASSASSIN'S CREED: EMBERS | Animations | 6 | Lee | Pong | 10 |
| 2 | Real Steel(2012) | Animations | 1 | Adam | Smith | 1 |
| 2 | Real Steel(2012) | Animations | 2 | Ravi | Kumar | 2 |
| 2 | Real Steel(2012) | Animations | 3 | Susan | Davidson | 5 |
| 2 | Real Steel(2012) | Animations | 4 | Jenny | Adrianna | 8 |
| 2 | Real Steel(2012) | Animations | 6 | Lee | Pong | 10 |
| 3 | Alvin and the Chipmunks | Animations | 1 | Adam | Smith | 1 |
| 3 | Alvin and the Chipmunks | Animations | 2 | Ravi | Kumar | 2 |
| 3 | Alvin and the Chipmunks | Animations | 3 | Susan | Davidson | 5 |
| 3 | Alvin and the Chipmunks | Animations | 4 | Jenny | Adrianna | 8 |
| 3 | Alvin and the Chipmunks | Animations | 6 | Lee | Pong | 10 |
| 4 | The Adventures of Tin Tin | Animations | 1 | Adam | Smith | 1 |
| 4 | The Adventures of Tin Tin | Animations | 2 | Ravi | Kumar | 2 |
| 4 | The Adventures of Tin Tin | Animations | 3 | Susan | Davidson | 5 |
| 4 | The Adventures of Tin Tin | Animations | 4 | Jenny | Adrianna | 8 |
| 4 | The Adventures of Tin Tin | Animations | 6 | Lee | Pong | 10 |
| 5 | Safe (2012) | Action | 1 | Adam | Smith | 1 |
| 5 | Safe (2012) | Action | 2 | Ravi | Kumar | 2 |
| 5 | Safe (2012) | Action | 3 | Susan | Davidson | 5 |
| 5 | Safe (2012) | Action | 4 | Jenny | Adrianna | 8 |
| 5 | Safe (2012) | Action | 6 | Lee | Pong | 10 |
| 6 | Safe House(2012) | Action | 1 | Adam | Smith | 1 |
| 6 | Safe House(2012) | Action | 2 | Ravi | Kumar | 2 |
| 6 | Safe House(2012) | Action | 3 | Susan | Davidson | 5 |
| 6 | Safe House(2012) | Action | 4 | Jenny | Adrianna | 8 |
| 6 | Safe House(2012) | Action | 6 | Lee | Pong | 10 |
| 7 | GIA | 18+ | 1 | Adam | Smith | 1 |
| 7 | GIA | 18+ | 2 | Ravi | Kumar | 2 |
| 7 | GIA | 18+ | 3 | Susan | Davidson | 5 |
| 7 | GIA | 18+ | 4 | Jenny | Adrianna | 8 |
| 7 | GIA | 18+ | 6 | Lee | Pong | 10 |
| 8 | Deadline(2009) | 18+ | 1 | Adam | Smith | 1 |
| 8 | Deadline(2009) | 18+ | 2 | Ravi | Kumar | 2 |
| 8 | Deadline(2009) | 18+ | 3 | Susan | Davidson | 5 |
| 8 | Deadline(2009) | 18+ | 4 | Jenny | Adrianna | 8 |
| 8 | Deadline(2009) | 18+ | 6 | Lee | Pong | 10 |
| 9 | The Dirty Picture | 18+ | 1 | Adam | Smith | 1 |
| 9 | The Dirty Picture | 18+ | 2 | Ravi | Kumar | 2 |
| 9 | The Dirty Picture | 18+ | 3 | Susan | Davidson | 5 |
| 9 | The Dirty Picture | 18+ | 4 | Jenny | Adrianna | 8 |
| 9 | The Dirty Picture | 18+ | 6 | Lee | Pong | 10 |
| 10 | Marley and me | Romance | 1 | Adam | Smith | 1 |
| 10 | Marley and me | Romance | 2 | Ravi | Kumar | 2 |
| 10 | Marley and me | Romance | 3 | Susan | Davidson | 5 |
| 10 | Marley and me | Romance | 4 | Jenny | Adrianna | 8 |
| 10 | Marley and me | Romance | 6 | Lee | Pong | 10 |

**INNER JOIN**

The inner JOIN is used to return rows from both tables that satisfy the given condition.

Suppose , you want to get list of members who have rented movies together with titles of movies rented by them. You can simply use an INNER JOIN for that, which returns rows from both tables that satisfy with given conditions.

[](https://www.guru99.com/images/InnerJoin.png)

SELECT members.`first\_name` , members.`last\_name` , movies.`title`

FROM members ,movies

WHERE movies.`id` = members.`movie\_id`

Executing the above script give

|  |  |  |
| --- | --- | --- |
| **first\_name** | **last\_name** | **title** |
| Adam | Smith | ASSASSIN'S CREED: EMBERS |
| Ravi | Kumar | Real Steel(2012) |
| Susan | Davidson | Safe (2012) |
| Jenny | Adrianna | Deadline(2009) |
| Lee | Pong | Marley and me |

Note the above results script can also be written as follows to achieve the same results.

SELECT A.`first\_name` , A.`last\_name` , B.`title`

FROM `members`AS A

INNER JOIN `movies` AS B

ON B.`id` = A.`movie\_id`

**Outer JOINs**

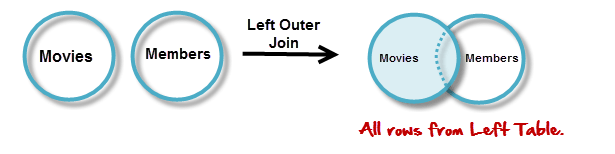
MySQL Outer JOINs return all records matching from both tables .

It can detect records having no match in joined table. It returns **NULL** values for records of joined table if no match is found.

Sounds Confusing ? Let's look into an example -

**LEFT JOIN**

Assume now you want to get titles of all movies together with names of members who have rented them. It is clear that some movies have not being rented by any one. We can simply use **LEFT JOIN**for the purpose.

[](https://www.guru99.com/images/LeftOuterJoin.png)

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. **Where no matches have been found in the table on the right, NULL is returned.**

SELECT A.`title` , B.`first\_name` , B.`last\_name`

FROM `movies` AS A

LEFT JOIN `members` AS B

ON B.`movie\_id` = A.`id`

Executing the above script in MySQL workbench gives.You can see that in the returned result which is listed below that for movies which are not rented, member name fields are having NULL values. That means no matching member found members table for that particular movie.

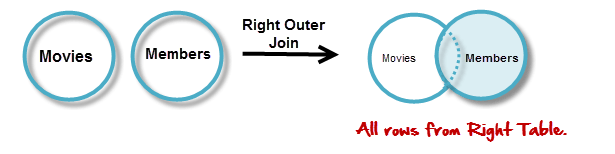
|  |  |  |
| --- | --- | --- |
| **title** | **first\_name** | **last\_name** |
| ASSASSIN'S CREED: EMBERS | Adam | Smith |
| Real Steel(2012) | Ravi | Kumar |
| Safe (2012) | Susan | Davidson |
| Deadline(2009) | Jenny | Adrianna |
| Marley and me | Lee | Pong |
| Alvin and the Chipmunks | NULL | NULL |
| The Adventures of Tin Tin | NULL | NULL |
| Safe House(2012) | NULL | NULL |
| GIA | NULL | NULL |
| The Dirty Picture | NULL | NULL |

Note: Null is returned for non-matching rows on right

**RIGHT JOIN**

RIGHT JOIN is obviously the opposite of LEFT JOIN. The RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

In our example,  let's assume that you need to get names of members and movies rented by them. Now we have a new member who has not rented any movie yet

[](https://www.guru99.com/images/RightOuterJoin.png)

SELECT A.`first\_name` , A.`last\_name`, B.`title`

FROM `members` AS A

RIGHT JOIN `movies` AS B

ON B.`id` = A.`movie\_id`

Executing the above script in MySQL workbench gives the following results.

|  |  |  |
| --- | --- | --- |
| **first\_name** | **last\_name** | **title** |
| Adam | Smith | ASSASSIN'S CREED: EMBERS |
| Ravi | Kumar | Real Steel(2012) |
| Susan | Davidson | Safe (2012) |
| Jenny | Adrianna | Deadline(2009) |
| Lee | Pong | Marley and me |
| NULL | NULL | Alvin and the Chipmunks |
| NULL | NULL | The Adventures of Tin Tin |
| NULL | NULL | Safe House(2012) |
| NULL | NULL | GIA |
| NULL | NULL | The Dirty Picture |

Note: Null is returned for non-matching rows on left

**"ON" and "USING" clauses**

In above JOIN query examples, we have used ON clause to match the records between table.

USING clause can also be used for the same purpose. The difference with **USING**is it **needs to have identical names for matched columns in both tables.**

In "movies" table so far we used its primary key with the name "id". We referred to same in  "members" table with the name "movie\_id".

Let's rename "movies" tables "id" field to have the name "movie\_id". We do this in order to have identical matched field names.

ALTER TABLE `movies` CHANGE `id` `movie\_id` INT( 11 ) NOT NULL AUTO\_INCREMENT;

Next let's use USING with above LEFT JOIN example.

SELECT A.`title` , B.`first\_name` , B.`last\_name`

FROM `movies` AS A

LEFT JOIN `members` AS B

USING ( `movie\_id` )

Apart from using **ON**and **USING with JOINs**you can use many other MySQL clauses like **GROUP BY, WHERE**and even functions like **SUM**, **AVG**, etc.

**Why should we use joins?**

Now you may think, why we use JOINs when we can do the same task running queries. Especially if you have some experience in database programming you know we can run queries one by one, use output of each in successive queries. Of course, that is possible. But using JOINs, you can get the work done by using only a one query with any search parameters. On the other hand **MySQL can achieve better performance** with JOINs as it can use Indexing. Simply use of single JOIN query instead running multiple queries do reduce server overhead. Using multiple queries instead that leads more data transfers between MySQL and applications (software). Further it requires more data manipulations in application end also.

**It is clear that we can achieve better MySQL and application performances by use of JOINs.**

**Summary**

* JOINS allow us to combine data from more than one table into a single result set.
* JOINS have better performance compared to sub queries
* INNER JOINS only return rows that meet the given criteria.
* OUTER JOINS can also return rows where no matches have been found. The unmatched rows are returned with the NULL keyword.
* The major JOIN types include Inner, Left Outer, Right Outer, Cross JOINS etc.
* The frequently used clause in JOIN operations is "ON". "USING" clause requires that matching columns be of the same name.
* JOINS can also be used in other clauses such as GROUP BY, WHERE, SUB QUERIES, AGGREGATE FUNCTIONS etc.

**Views in MySQL Tutorial: Create, Join & Drop with Examples**

**What are views?**

Simply put, VIEWS are virtual tables .By virtual, we mean, the tables do not store any data of their own but display data stored in other tables.

In other words, VIEWS are nothing but SELECT Queries.

**Views syntax**

Let's now look at the basic syntax used to create a view in MySQL.

CREATE VIEW `view\_name` AS SELECT statement;

**WHERE**

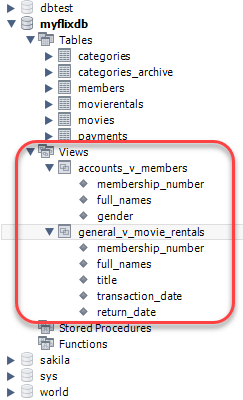
* **"CREATE VIEW `view\_name`"** tells MySQL server to create a view object in the database named `view\_name`
* **"AS SELECT statement"** is the SQL statements to be packed in the views. It can be a SELECT statement can contain data from one table or multiple tables.

Let's now create our first view using the "myflixdb" we will create a simple view that restricts the columns seen in the members table.

Suppose authorization requirements  state that the accounts department can only see member's number , name and gender from the member's table. To achieve this you can create a VIEW -

CREATE VIEW `accounts\_v\_members` AS SELECT `membership\_number`,`full\_names`,`gender` FROM `members`;

Executing the above script in MySQL workbench against the myflixdb and expanding the views node in the database explorer gives us the following results.

[](https://www.guru99.com/images/database_explorer.png)

Note the accounts\_v\_members object is now visible in the database views objects. Let's now execute a SELECT statement that selects all the fields from the view.

SELECT \* FROM `accounts\_v\_members`;

Executing the above script in MySQL workbench against myflixdb gives us the following results shown below.

|  |  |  |
| --- | --- | --- |
| **membership\_number** | **full\_names** | **gender** |
| 1 | Janet Jones | Female |
| 2 | Janet Smith Jones | Female |
| 3 | Robert Phil | Male |
| 4 | Gloria Williams | Female |
| 5 | Leonard Hofstadter | Male |
| 6 | Sheldon Cooper | Male |
| 7 | Rajesh Koothrappali | Male |
| 8 | Leslie Winkle | Male |
| 9 | Howard Wolowitz | Male |

Only the authorized columns for accounts department have been returned. Other details found in the members table have been hidden .

If we want to see the SQL statements that make up a particular view, we can use the script shown below to do that.

SHOW CREATE VIEW `accounts\_v\_members`;

Executing the above script gives you the view name and the SQL SELECT statements used to create the view.

**Joins and views**

Let's now look at a fairly complex example which involves multiple tables and uses joins.

We will package the JOIN created that gets information from three (3) tables namely members, movies and movie rentals. Below is the script that helps us to achieve that.

CREATE VIEW `general\_v\_movie\_rentals` AS SELECT mb.`membership\_number`,mb.`full\_names`,mo.`title`,mr.`transaction\_date`,mr.`return\_date` FROM `movierentals` AS mr INNER JOIN `members` AS mb ON mr.`membership\_number` = mb.`membership\_number` INNER JOIN `movies` AS mo ON mr.`movie\_id` = mo.`movie\_id`;

Executing the above scripts creates the view named general\_v\_movie\_rentals in our myflixdb

Let's now select all the fields from a table named general\_v\_movie\_rentals.

SELECT \* FROM `general\_v\_movie\_rentals`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **membership\_number** | **full\_names** | **title** | **transaction\_date** | **return\_date** |
| 1 | Janet Jones | Pirates of the Caribean 4 | 20-06-2012 | 28-06-2012 |
| 1 | Janet Jones | Forgetting Sarah Marshal | 22-06-2012 | 25-06-2012 |
| 3 | Robert Phil | Forgetting Sarah Marshal | 22-06-2012 | 25-06-2012 |
| 2 | Janet Smith Jones | Forgetting Sarah Marshal | 21-06-2012 | 24-06-2012 |
| 3 | Robert Phil | X-Men | 23-06-2012 | 28-06-2012 |

Note we didn't have to write the complex JOIN query to get information about members, movies and movie rental details. We simply used the view in a regular SELECT statement as any other ordinary table. The view can be called from anywhere in the application system running on top of the myflixdb.

**Dropping views**

The DROP command can be used to delete a view from the database that is no longer required. The basic syntax to drop a view is as follows.

DROP VIEW ` general\_v\_movie\_rentals `;

Why use views?

You may want to use views primarily for following 3 reasons

* Ultimately , you will use your SQL knowledge , to create applications , which will use a  database for data requirements. It's recommended that you use VIEWS of the original table structure in your application instead of using the tables themselves. This ensures that when you refactor your DB, your legacy code will see the orignal schema via the view without breaking the application.
* VIEWS increase re-usability. You will not have to create complex queries involving joins repeatedly. All the complexity is converted into a single line of query use VIEWS. Such condensed code will be easier to integrate in your application. This will eliminates chances of typos and your code will be more readable.
* VIEWS help in data security. You can use views to show only authorized information to users and hide sensitive data like credit card numbers.

**Summary**

* Views are virtual tables; they do not contain the data that is returned. The data is stored in the tables referenced in the SELECT statement.
* Views improve security of the database by showing only intended data to authorized users. They hide sensitive data.
* Views make life easy as you do not have write complex queries time and again.
* It's possible to use INSERT, UPDATE  and DELETE on a VIEW. These operations will change the underlying tables of the VIEW.  The only consideration is that VIEW should contain all NOT NULL columns of the tables it references. Ideally, you should not use VIEWS for updating.

# MySQL Index Tutorial - Create, Add & Drop

### What are Index?

Nobody likes slow systems.

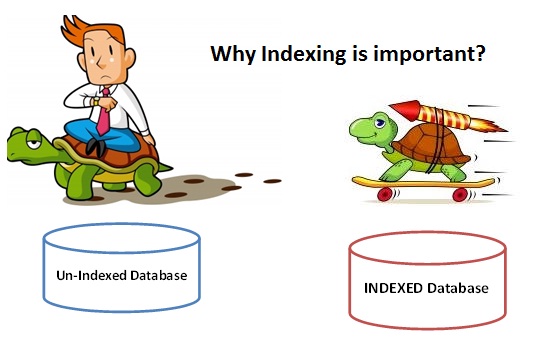
High system performance is of prime importance in almost all database systems .

Most businesses invest heavily in hardware so that data retrievals and manipulations can be faster.

But there is limit to hardware investments a business can make.

Optimizing your database is a cheaper and better solution.

Towards this end we can use INDEXES.



* The slowness in the response time is usually due to the records being stored randomly in database tables.
* Search queries have to loop through the entire randomly stored records one after the other to locate the desired data.
* This results in poor performance databases when it comes to retrieving data from large tables
* Indexes come in handy in such situations. Indexes sort data in an organized sequential way.Think of an index as an alphabetically sorted list. It is easier to lookup names that have been sorted in alphabetical order than ones that are not sorted.
* INDEX's are created on the column(s) that will be used to filter the data.
* Using indexes on tables that are frequently updated can result in poor performance. This is because MySQL creates a new index block every time that data is added or updated in the table. Generally, indexes should be used on tables whose data does not change frequently but is used a lot in select search queries.

**Create index basic syntax**

Indexes can be defined in 2 ways

1.       At the time of table creation

2.       After table has been created

Example:

For our myflixdb we expect lots of searches to the database on full name.

We will add the "full\_names" column to Index in a new table "members\_indexed".

The script shown below helps us to achieve that.

CREATE TABLE `members\_indexed` (

`membership\_number` int(11) NOT NULL AUTO\_INCREMENT,

`full\_names` varchar(150) DEFAULT NULL,

`gender` varchar(6) DEFAULT NULL,

`date\_of\_birth` date DEFAULT NULL,

`physical\_address` varchar(255) DEFAULT NULL,

`postal\_address` varchar(255) DEFAULT NULL,

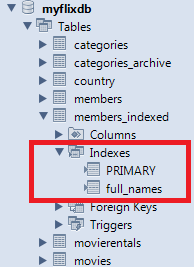
`contact\_number` varchar(75) DEFAULT NULL,

`email` varchar(255) DEFAULT NULL,

PRIMARY KEY (`membership\_number`),INDEX(full\_names)

) ENGINE=InnoDB;

Execute the above SQL script in MySQL workbench against the "myflixdb".

[](https://www.guru99.com/images/tables_explorer.png)

Refreshing the myflixdb shows the newly created table named members\_indexed.

*"Note"* members\_indexed table has "full\_names" in the indexes node.

As the members base expand and the number of records increases , search queries on the members\_indexed table that use the WHERE and ORDER BY clauses will be much faster compared to the ones performed the members table without the index defined.

**Add index basic syntax**

The above example created the index when defining the database table. Suppose we already have a table defined and search queries on it are very slow. They take too long to return the results. After investigating the problem, we discover that we can greatly improve the system performance by creating INDEX on the most commonly used column in the WHERE clause.

 We can use following query to add index

CREATE INDEX id\_index ON table\_name(column\_name);

Let's suppose that search queries on the movies table are very slow and we want to use an index on the "movie title" to speed up the queries, we can use the following script to achieve that.

CREATE INDEX `title\_index` ON `movies`(`title`);

Executing the above query creates an index on the title field in the movies table.

This means all the search queries on the movies table using the "title" will be faster.

Search queries on other fields in the movies table will however still are slower compared to the ones based on the indexed field.

Note you can create indexes on multiple columns if necessary depending on the fields that you intend to use for your database search engine.

If you want to view the indexes defined on a particular table, you can use the following script to do that.

SHOW INDEXES FROM table\_name;

Let's now take a look at all the indexes defined on the movies table in the myflixdb.

SHOW INDEXES FROM `movies`;

Executing the above script in MySQL workbench against the myflixdb gives us the following results shown below.

Note the primary and foreign keys on the table have already been indexed by MySQL. Each index has its own unique name and the column on which it is defined is shown as well.

**Drop index basic syntax**

The drop command is used to remove already defined indexes on a table.

There may be times when you have already defined an index on a table that is frequently updated. You may want to remove the indexes on such a table to improve the UPDATE and INSERT queries performance. The basic syntax used to drop an index on a table is as follows.

DROP INDEX `index\_id` ON `table\_name`;

Let's now look at a practical example.

DROP INDEX ` full\_names` ON `members\_indexed`;

Executing the above command drops the index with id ` full\_names ` from the members\_indexed table.

**Summary**

* Indexes are very powerful when it comes to greatly improving the performance of MySQL search queries.
* Indexes can be defined when creating a table or added later on after the table has already been created.
* You can define indexes on more than one column on a table.
* The SHOW INDEX FROM table\_name is used to display the defined indexes on a table.
* The DROP command is used to remove a defined index on a given table.

**Your First Application using MySQL and PHP**

**Now that Myflixdb, what's next?**

***Congratulations***for your success completion of the SQL tutorial series.

We looked at how we can design a database from scratch, the relational database management system model among others, MySQL as a database management system and how it implements the SQL language, what tools are available for us to use in the database design phase, SQL development and MySQL server administration such as MySQL workbench and finally produced a working database for a fictional video library system "Myflix".

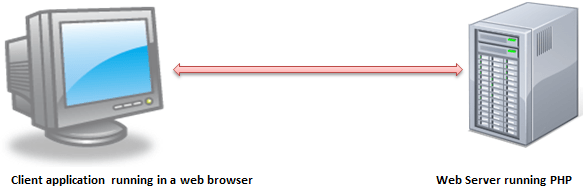
Excited as you might be with your newly acquired knowledge, have you given any thoughts' as to how your database uses will use your database?

How about developing an application that hides all the SQL statements inside and provide a visual graphical user interface for the users to use when interacting with the database?

We can use[PHP](https://www.guru99.com/php-tutorials.html)for this...

**What is PHP?**

PHP is a **general purpose server side scripting language** that we can use to develop dynamic web sites and applications. PHP only needs to be **installed on the web server** that will host the web application and **client applications** can **access**the **server resources** via **web browsers**. The diagram shown below illustrates the basic architecture of a PHP web application.

[](https://www.guru99.com/images/client_server.png)

**Why PHP?**

You have obviously head of a number of programming languages out there; you may be wondering why we would want to use PHP over other languages to develop our video library application? PHP just like MySQL is **open source and free**, you don't need to pay for you to get PHP; all you have to do is download it. Packages such as **XAMPP** come bundled already with a web server, MySQL and PHP among others. This is also unbelievably free. This makes PHP a cost effective choice compared to other scripting languages such as CFML or ASP.

Other benefit that you get with PHP is that it's a **server side scripting language**; this means you only need to install it on the server and client computers requesting for the resources from the server do not need to have PHP installed; only a web browser would be enough.

PHP also has **in built support for working hand in hand with MySQL**; this doesn't mean you can't use PHP with other database management systems.

PHP is **cross platform** which means you can deploy your application on a number of different operating systems such as windows, Linux, Mac OS etc.

**PHP and MySQL**

PHP and MySQL are like two different sides of the same coin. Just like MySQL has built in functions for data manipulations, so does PHP has built in functions for connecting to MySQL server and manipulating the data in the database. Let's now look at some of PHP functions that allow us to manipulate MySQL databases

*MySQL connect*

$dh = mysql\_connect(servername,username,password);

**HERE**

* "mysql\_connect" is the PHP built in function for connecting to MySQL database
* "servername" is the name of the server running MySQL server.
* "username" is the name of the user that we will use to authenticate ourselves when connecting to the server.
* "password" is the password that we will use to authenticate ourselves when connecting to the server.

*PHP Data Object [PDO]*

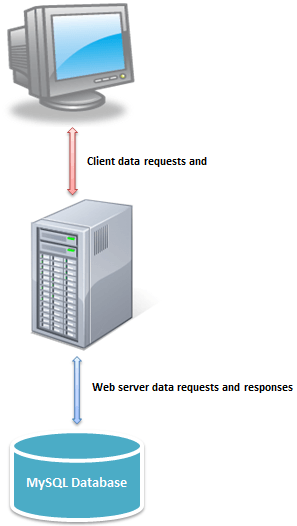
Alternatively we can also use the PHP Data Object PDO to connect to MySQL server. It has the following basic syntax.

$dbconn = new PDO(mysql:host=servername;dbname=databasename,username,password);

**HERE**

* "dbconn" is the database connection object that we can use to manipulate our database.
* "new PDO (...)" creates an instance of the PDO class.
* "mysql:host=servername;" is the name of the server running MySQL server.
* "dbname=databasename" is the name of the database to be manipulated from PHP.
* "username,password" are login credentials we will use to authenticate ourselves.

As you can see from the above database connection examples, not only is PHP simple to use but it's also flexible. The diagram shown below illustrates how the client application, PHP powered web server and MySQL interact.

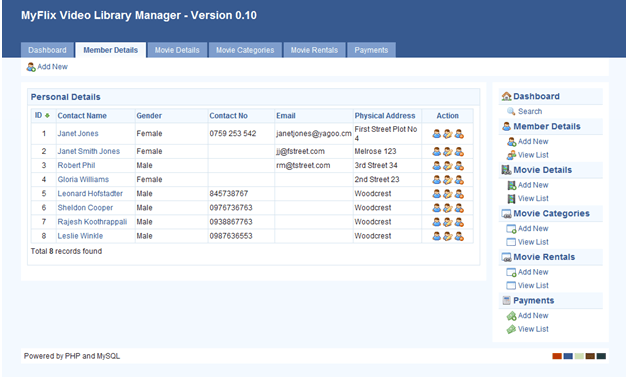
[](https://www.guru99.com/images/client_server_database.png)

**Introducing Myflix Video Library Application**

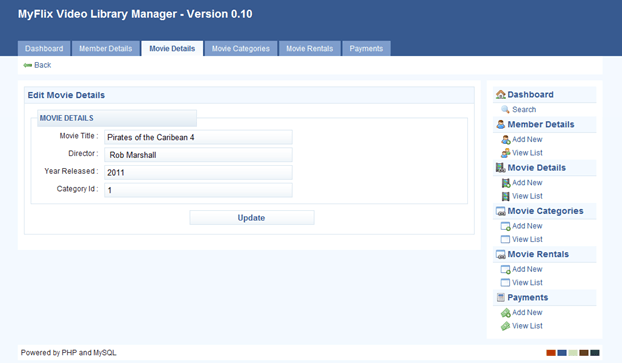
Assuming we have acquired knowledge of the PHP language combined with what we just learnt in the MySQL tutorial series, we can write the web based application in PHP that runs on top of the MySQL database to provide our Myflix database users with a Visual Graphical User Interface for interactions.

**Myflix Video Library Screenshots**

The screenshots below show how the video library application running on top of Myflixdb would look like.

[](https://www.guru99.com/images/myflix_video_library_1.png)

Members listing window shown above

[](https://www.guru99.com/images/myflix_video_library_2.png)

Movie details editing window shown above

**Summary**

* MySQL is a powerful open source database engine that can meet the requirements of both stand-alone as well as client server applications.
* PHP is a general purpose scripting language that can be used to develop client applications that run on top of MySQL databases.
* Equipping yourself with both MySQL and PHP knowledge is double blessing.

**SQL vs MySQL: What's the Difference?**

**What is SQL?**

SQL is a language which is used to operate your database. SQL is the basic language used for all the databases. There are minor syntax changes amongst different databases, but the basic SQL syntax remains largely the same. SQL is a short abbreviation of **S**tructured **Q**uery **L**anguage. According to ANSI (American National Standards Institute), SQL is the standard language to operate a relational database management system.

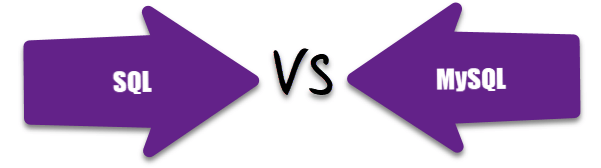
SQL is used in the accessing, updating, and manipulation of data in a database. Its design allows for the management of data in an RDBMS, such as MYSQL. SQL language also used for controlling data access and for creation and modification of Database schemas.

**What is MYSQL?**

Developed in the mid-90s., MySQL was one of the first open-source database available in the market. Today there are many alternatives variants of MySQL,. However, the differences between the variants are not significant as they use the same syntax, and basic functionality also remains same.

MySQL is an RDBMS that allows keeping the data that exists in a database organized. MySQL is pronounced as "My S-Q-L," but it is also called "My Sequel." It is named after co-founder Michael Widenius' daughter. MySQL provides a multi-user access to databases. This RDBMS system is used with the combination of PHP and Apache Web Server, on top of a Linux distribution. MySQL uses the SQL language to query the database.

Now let see the Difference between SQL and MySQL

[](https://www.guru99.com/images/1/051818_1204_SQLvsMySQLW1.png)

**Difference between SQL and MySQL**

| **Parameter** | **SQL** | **MYSQL** |
| --- | --- | --- |
| Definition | SQL is a Structured Query Language. It is useful to manage relational databases. | MySQL is an RDBMS tostore, retrieve, modify and administrate a database usingSQL. |
| Complexity | You need to learn the SQL language to use it effectively. | It is readily available through download and installation. |
| Type | SQL is a query language. | MySQL is database software. It used "SQL" language to query the database. |
| Support for connector | SQL does not provide connectors. | MySQL offers an integrated tool called 'MySQL workbench' to design and develop databases. |
| Purpose | To query and operate database system. | Allows data handling, storing, modifying, deleting in a tabular format. |
| Usage | SQL code and commands are used in various DBMS and RDMS systems including MYSQL. | MYSQL is used as an RDBMS database. |
| Updates | The language is fixed, and command remains the same. | Get the frequent updates |

**KEY DIFFERENCE:**

* **SQL** is a language which is used to operate your database whereas **MySQL** was one of the first open-source database available in the market
* SQL is used in the accessing, updating, and manipulation of data in a database while MySQL is an RDBMS that allows keeping the data that exists in a database organized
* SQL is a Structured Query Language and MySQL is a RDBMS to store, retrieve, modify and administrate a database.
* SQL is a query language while MYSQL is a database software

**20 Best SQL Management Tools in 2020**

There are many SQL management tools available in the market, and therefore it is challenging to select the best tool for your SQL project management. The following is a curated list of best SQL tools with key features, and download links.

**1)**[**Interbase**](https://bit.ly/35jw20Q)

[](https://bit.ly/35jw20Q)

[Interbase](https://bit.ly/35jw20Q) is a ultra-fast, scalable, embeddable SQL database with commercial grade data security, disaster recovery and change synchronization.

**Features:**

* Follows SQL industry standards, supports unicode and is ideal for any character set globally.
* Offers multiple join unicode, real-time event alerts, and the Change Views’ SQL-based change tracking.
* Runs on Windows, Linux, Mac OS X, Android and iOS.
* Supports Java, C, C++, .NET, Delphi, PHP and Ruby.
* Small lightweight footprint. Leverages today's multi-core CPUs and multi-CPU systems, along with a unique versioning approach and advanced disk I/O caching.
* Admin-Free, rapid crash recovery and disaster recovery.
* Cross-Platform encryption, separate security login and role based user security.

[](https://bit.ly/35jw20Q)

**2)**[**Studio for SQL Server**](https://bit.ly/2ND0RFs)

[](https://bit.ly/2ND0RFs)

[dbForge Studio for SQL Server](https://bit.ly/2ND0RFs) is a powerful IDE for SQL Server management, administration, development, data reporting, analysis, and a lot more. SQL developers and DBAs performing complex database tasks can use the GUI tool to speed up almost any database experience, such as designing databases, writing SQL code, comparing databases, synchronizing schemas and data, generating meaningful test data, and much more.  
  
**Features:**

* reliable database change management
* smart coding assistance
* better data protection and problems resolving
* professional SQL Server administration for secure database management
* better SQL Server reporting

[](https://bit.ly/2EE5xrr)

**3)**[**dbWatch**](https://bit.ly/2y4pTWP)

[](https://bit.ly/2y4pTWP)

[dbWatch](https://bit.ly/2y4pTWP) is a complete database monitoring and management solution for SQL Server, Oracle, PostgreSQL, Sybase, MySQL and Azure. Designed for proactive management and automation of routine maintenance in large scale on-premise, hybrid or cloud database environments.  
  
**Features:**

* Performance monitoring & Health Reports
* Automatic memory reduction for SQLServer
* DB Management with SMS and email extensions
* IDE for developing alerts & reporting with 3rd party extensions
* Enterprise performance views package with Maintenance automation
* SQL Server and Oracle license management & control
* Multi-site support and Database Cluster support
* Database Autodiscover with CLI scripting engine
* Bulk install database servers and bulk task and alerts configuration

[](https://bit.ly/2UccOVA)

**4)**[**SQL Sentry**](https://bit.ly/2Zafucj)

[https://www.guru99.com/images/1/sql_sentry_logo.png](https://bit.ly/2Zafucj)

[SQL One](https://bit.ly/2Zafucj) is the top-rated SQL Server database performance monitoring solution, with proven scalability to handle growing enterprise workloads. Used by hundreds of companies with high demands for data performance—including Humana, Subway, Subaru and Verizon

* Helps data professionals collect and present detailed, actionable performance metrics, customize alerts for proactive monitoring, and manage high-impact SQL Server events.
* Provides powerful SQL query analysis and tuning capabilities with the popular SentryOne Plan Explorer functionality.
* SQL Sentry is consistently rated highest among competitors on independent review site TrustRadius and was named “Best DBA Solution” in the 2019 Database Trends and Applications Readers’ Choice Awards.

[](https://bit.ly/2Zl3gIN)

**5)**[**Visual Expert**](https://bit.ly/33IEVzv)

[https://www.guru99.com/images/1/visual_expert_logo_v1.png](https://bit.ly/33IEVzv)

[Visual Expert](https://bit.ly/33IEVzv) is a powerful static code analysis toolbox for Oracle and SQL Server database code. With 200+ features, developers can start working immediately on any PL/SQL or T-SQL code.

Explore your code with treeviews & diagrams synchronized with source code view. Visualize dependencies and call trees. Analyze and improve the response time of your procedures. Anticipate the effect of a change. Build code documentation. Compare 2 versions of your code in just few clicks!  
  
**Features:**

* CRUD matrix
* Diagrams
* Code performance
* Code review
* Automatic HTML documentation generation
* Code comparison
* And much more!

[](https://bit.ly/33IEVzv)

**6)**[**Jet Profiler for MySQL**](https://bit.ly/2So8e5X)

[](https://bit.ly/2So8e5X)

[Jet Profiler for MySQL](https://bit.ly/2So8e5X) is a real-time query performance and diagnostics tool for the MySQL database server. The tool is developed since 2009 and is highly stable  
  
**Features:**

* Query, table and user performance
* Graphical visualisation
* Low overhead
* User friendly

[](https://bit.ly/2M1ZeSP)

**7) Adminer**

[](https://www.adminer.org/en/editor/)

Adminer is a SQL management tool for managing databases, tables, relations, indexes, users. It has support for all the popular database management systems like MySQL, PostgreSQL, SQLite, MS SQL, Oracle, and MongoDB.

**Features:**

* Connect with database server with username and password
* Option for selecting existing database or create a new one
* Allows modification of the name, type, collation, comment and default values of columns
* Add and drop tables and columns. Supports all data types, blobs through file transfer
* Display users and rights and change them. Display variables with links to documentation
* Manage events and table partitions
* Extensive customization options

**Download link:**<https://www.adminer.org/en/editor/>

**8) DBComparer:**

[](http://dbcomparer.com/Download/Default.aspx)

It is a SQL management tool for analysing the differences in Microsoft SQL Server database structures. It allows comparing database objects like tables, columns, indexes, foreign keys, schemas, etc.

**Features:**

* Comparing and synchronizing databases on different servers as well as on a single server
* Using the Console Application
* Support for working with many projects simultaneously
* Plenty of options for comparison and synchronization
* Built-in SQL Script editor with syntax highlight
* New state-of-the-art graphical user interface
* Latest SQL Server version support

**Download link:**<http://dbcomparer.com/Download/Default.aspx>

**9) EMS SQL Manager:**

EMS SQL Manager allows users to create and edit SQL Server database objects and create, modify, execute and save SQL queries.

**Features:**

* Full support of PostgreSQL
* Visual and text tools for query building
* Rapid database management and navigation
* Advanced tool for data manipulation
* Effective security management
* Impressive data export and import capabilities
* Powerful visual database designer
* Easy-to-use wizards performing PostgreSQL maintenance tasks
* Access to PostgreSQL Server through HTTP protocol
* Database change management

**Download link:** <https://www.sqlmanager.net/en/products/mssql/manager/download>

**10) SQuirrel SQL:**

It is a JAVA-based database administration tool. This JDBC compliant SQL management tool allows users to view the database structure and issue SQL commands. It also supports databases such as Firebird, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, Sybase, etc.

**Features:**

* Quickly view and edit data in any JDBC-compliant database
* Allows to view the database's metadata
* Work with multiple databases on local and remote machines
* Use a single interface to work with various database engines

**Download link:** <http://squirrel-sql.sourceforge.net/>

**11) SQLite Database Browser:**

SQLite Database Browser is an open source SQL tool that allows users to create, design and edits SQLite database files. It lets users show a log of all the SQL commands that have been issued by them and by the application itself.

**Features:**

* Create and delete indexes
* Browse, edit, add and delete records
* Import and export records as text
* Import and export tables from CSV files
* It Import and export databases from SQL dump files
* Examine a log of all SQL commands issued by the application

**Download link:** <http://sqlitebrowser.org/>

**12) DBeaver:**

[](http://dbeaver.jkiss.org/download/)

DBeaver is an open source database tool for developers and database administrators. It offers supports for JDBC compliant databases such as MySQL, Oracle, IBM DB2, SQL Server, Firebird, SQLite, and Sybase.

**Feature:**

* It is free and open source tool
* Allows to browse and change database metadata, tables, keys and indexes and column.
* It's SQL editor includes auto-formatting, auto-completion, and hyperlinks
* It allows searching database objects like tables, columns, constraints, and procedures

**Download link:** <http://dbeaver.jkiss.org/download/>

**13) DBVisualizer Free**

[https://www.guru99.com/images/2-2017/072817_0646_Top20SQLMan4.png](https://www.dbvis.com/download/)

DbVisualizer Free is a SQL management tool. It allows users to manage a wide range of databases including Oracle, Sybase, SQL Server, MySQL, Informix, H3, and SQLite.

**Features:**

* Auto Completion support in the SQL editor
* Permits execution of multi-statement SQL scripts
* Export Schema and Database
* Allows to Import data from CSV files
* Come with inbuilt Query Builder
* Foreign Key Data Navigator

**Download link:** <https://www.dbvis.com/download/>

**14) HeidiSQL:**

[](https://www.heidisql.com/download.php)

HeidiSQL is another reliable SQL management tool. It is designed using the popular MySQL server, Microsoft SQL databases, and PostgreSQL. It allows users to browse and edit data, create and edit tables, views, triggers and scheduled events.

**Features:**

* It's free and open source tool
* Allows to connect to multiple servers in one window
* Create and modify tables, views, stored routines and scheduled events.
* Export from one server database into another server
* Helps to manage user privileges
* Browse and edit table data using a comfortable grid
* Monitor and kill client-processes
* Repair and Optimize tables in a batch manner

**Download link:** <https://www.heidisql.com/download.php>

**15) FlySpeed SQL Query:**

FlySpeed SQL Query is data handling tool for all database users and developers. It allows users to build queries on different database servers without any deep knowledge of SQL syntax.

**Features:**

* Get connected to various database servers natively or using ODBC
* Browse and modified data from tables and queries in the grid or using the customizable form view
* Instantly find, sort and filter data
* Analyse all the complex SQL queries by representing them visually
* Manage SQL queries via the professional editor with code completion and syntax highlighting
* Access your SQL query execution history and data browsing settings and saved between work sessions
* Export data to Excel, CSV, HTML and XML files

**Download link:** <http://www.activedbsoft.com/download-querytool.html>

**16) SQL Diagnostic Manager:**

SQL diagnostic manager is a performance monitoring tool developed by Idera. It offers diagnostics solution that helps users to evaluate health and performance within their SQL Server.

**Features:**

* It offers easy installation and use
* Monitor performance for physical and virtual environments
* It analysis the entire SQL Server environment. It also gives notifications when performance or availability issues are discovered.
* Monitor queries and query plans to see the reasons of blocks and deadlocks.
* Stores data in the centralized SQL diagnostic manager repository
* View summary of top issues and alerts with the web console add-on
* Provides a comprehensive view of current performance
* Monitor application transactions with SQL Workload Analysis

**Download link:** <https://www.idera.com/productssolutions/sqlserver/sqldiagnosticmanager>

**17) Free SQL Performance Monitoring Tool:**

Free SQL Performance Monitoring tool developed by ManageEngine. It allows users to keep an eye on SQL Server performance! This tool also helps users to monitor MS SQL Servers for performance and availability. It can use with all the MS SQL versions like MS SQL 2012 and 2014.

**Features:**

* Real time monitoring of MS SQL server performance
* Get instant alerts when resources like CPU, Disk, Memory, and SQL cache exceeds the threshold value
* View per second details of Deadlocks, Lock waits and Latch wait time
* View instant details about the active database connections

**Download link:** <https://www.manageengine.com/download.html>

**18) ApexSQL Monitor:**

[https://www.guru99.com/images/2-2017/072817_0646_Top20SQLMan6.png](https://www.apexsql.com/Download.aspx?download=Monitor)

ApexSQL Monitor is a web-based SQL management application. It provides support for monitoring multiple SQL Server instances.

**Features:**

* View system performance metrics
* Identify deadlocks and performance issues
* Use comprehensive dashboard for viewing metrics
* Analyze query waits and review query execution plans
* Calculate baselines and thresholds

**Download Link:** <https://www.apexsql.com/Download.aspx?download=Monitor>

**19) AppDynamics:**

[https://www.guru99.com/images/2-2017/072817_0646_Top20SQLMan7.png](https://www.appdynamics.com/free-trial/)

The AppDynamics Database management product for Microsoft SQL Server is a comprehensive SQL tool. It is used to monitor versions 2000, 2005, 2008, 2012 and 2014 of SQL Server.

**Features:**

* Deep visibility into SQL Server performance
* It allows viewing SQL Server instance objects
* Complete record of Historical SQL Server instance performance
* Comprehensive reporting of time series comparisons and top SQL reports

**Download link:** <https://www.appdynamics.com/free-trial/>

**20) Toad:**

[](https://www.toadworld.com/p/downloads)

Toad is another SQL Server DBMS tool. It maximizes productivity using extensive automation, intuitive workflows, and built-in expertise. This SQL management tool resolve issues, manage change and promote the highest levels of code quality.

**Features:**

* Access key data quickly for in-place analysis and single-click export to an Excel instance
* It can easily identify differences by comparing and syncing servers, data, and schemas.
* Roll back transactions directly from the transaction log without need to restore from a backup
* Get powerful query tuning capabilities
* Execute scripts and T-SQL snippets for numerous instances and servers
* Enable performance tuning with automated SQL scan and optimization
* Get free performance health checks benchmarked from the community
* Perform application performance tuning with automated query rewrites and optimization
* Automate repetitive processes like data and schema comparisons

**Download link:** <https://www.toadworld.com/p/downloads>

**21) ZenPacks**

[https://www.guru99.com/images/2-2017/072817_0646_Top20SQLMan10.png](https://www.zenoss.com/product/what-we-monitor/applications)

Zenoss offers ZenPacks for manage Microsoft SQL Server, My SQL, Oracle Database, and PostgreSQL. Each ZenPack has specific features of each platform, with the available APIs to determine what should be monitored.

**Features:**

* Open source and free of cost
* Monitor performance metrics for database instances
* Tables and disk space
* Collect events and forward back to the Zenoss console
* Map service impact relationships between components
* Report on database activity with the help of Zenoss Console

**Download link:** <https://www.zenoss.com/product/what-we-monitor/applications>

**22) SQL Server Storage Manager:**

[](https://www.lepide.com/sql-storage-manager/download.html)

Lepide's SQL Server storage manager is an open source utility to analyse the performance of SQL Servers. It provides complete visibility of storage space and performance.

**Features:**

* Eliminate the need to writing complex stored procedures
* It offers simple graphical displays showing all elements of SQL servers
* Provides a centralized platform to create consolidated reports of all SQL resources
* Give real time alerts based on easy to configure criteria
* Effectively track, manage and plan disk space for various SQL servers
* Identify issues with table partitioning, defragmentation, and indexes
* Decrease network traffic time by partitioning indexes, databases, and heaps
* It provides extensive SQL reports
* Offers better visibility of databases, files and tables and more

**Download link:** <https://www.lepide.com/sql-storage-manager/download.html>

**23) Microsoft SQL Server Management Studio Express:**

SQL Server Management Studio Express is an open source tool for accessing, managing and developing all components of SQL. This tool supports most administrative tasks for SQL Server.

**Features:**

* A single, integrated environment for SQL Server Database Engine management and authoring
* It allows users to send them to a Code Editor, or script them for later execution.
* Non-modal and resizable dialogs help users to access to multiple tools while a dialog is open
* A common scheduling dialog that allows users to perform the action of the management dialogs at a later time
* Exporting and importing SQL Server Management Studio
* Save or print XML Deadlock and Showplan files automatically created by SQL Server
* A tutorial on SQL Server Management Studio to help users to take advantage of the many new features to become more productive right away
* A new activity monitor with filtering and automatic refresh
* Integrated Database Mail interfaces
* An integrated Web browser for browsing of online or MSDN help
* Integration of Help from online communities
* A new error and informational message box which provide more information. It also allows users to send e-mail and messages to the support team

**Download link:**<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

**24) Database Performance Analyzer for SQL Server:**

[https://www.guru99.com/images/2-2017/072817_0646_Top20SQLMan12.png](http://www.solarwinds.com/downloads)

SolarWinds Database Performance Analyzer is performance monitoring and analysis tool for database admin. It also finds the reason of bottlenecks and reduces the overall cost of database operations.

**Features:**

* Monitor capacity constraints, queries, and server health performance
* Expert database and SQL query tuning advisors
* Quickly find the cause of complex problems
* Identify database problems in real-time
* Cross-vendor database support from a single interface
* Low overhead on monitored databases
* Blocking and Deadlock analysis
* Analyze every factor that impacts SQL Server execution
* Monitors database in hybrid environments, on-premises, virtualized, and in the cloud
* Helps proactively tune queries so applications can respond faster

**Download link:** <http://www.solarwinds.com/downloads>

**25) Sequel Pro:**

[](https://www.sequelpro.com/)

Sequel Pro is a fast, easy-to-use database management tool for working with MySQL. This SQL management tool helpful for interacting with your database. It is also easy to add new databases, add new tables, add new rows, and any other type of databases using this software.

**Features:**

* Open source SQL management software
* Easy and fast installation
* Support for cloud
* No Java required to run this program
* Multiple result sets
* SQLPro provides support all the major database types including MySQL, MariaDB, Microsoft SQL Server and Oracle 8i
* It is fast, easy-to-use Mac database management application for working with MySQL databases
* It gives you direct access to MySQL Databases on local and remote servers

**Download link:**<https://www.sequelpro.com/>

**SQL Tutorial for Beginners PDF**

Databases can be found in almost all software applications. SQL is the standard language to query a database. This eBook will teach you database design. Also, it teaches you basic to advanced SQL.

**Key Highlights of SQL Tutorial PDF are**

* 220+ pages
* eBook Designed for beginners
* Beautifully annotated screenshots
* You will get lifetime access

[https://www.guru99.com/images/1/ebook_look_inside.png](https://www.guru99.com/pdf/sql_preview.pdf)

[](https://www.guru99.com/pdf/sql_preview.pdf)

**Inside this PDF**

1. What is Database? What is SQL?
2. MySQL Workbench Tutorial & MySQL Introduction
3. Database Design Tutorial: Learn Data Modeling
4. What is Normalization? 1NF, 2NF, 3NF & BCNF with Examples
5. What is ER Modeling? Learn with Example
6. MySQL Create Database, Tables, Data Types
7. MySQL SELECT Statement with Examples
8. MySQL WHERE Clause with Examples - AND, OR, IN, NOT IN
9. MySQL query INSERT INTO Table with Examples
10. MySQL UPDATE & DELETE Query with Example
11. ORDER BY in MySQL: DESC & ASC
12. MySQL GROUP BY and HAVING Clause with Examples
13. MySQL Wildcards Tutorial: Like, NOT Like, Escape, ( % ), ( \_ )
14. MYSQL Regular Expressions (REGEXP) with Syntax & Examples
15. MySQL Functions: String, Numeric, User-Defined, Stored
16. MySQL Aggregate Functions Tutorial : SUM, AVG, MAX, MIN , COUNT, DISTINCT
17. MySQL IS NULL & IS NOT NULL Tutorial with EXAMPLES
18. MySQL AUTO\_INCREMENT with Examples
19. MYSQL - ALTER, DROP, RENAME, MODIFY
20. MySQL LIMIT & OFFSET with Examples
21. MySQL SubQuery Tutorial with Examples
22. MySQL UNION - Complete Tutorial
23. MySQL JOINS Tutorial: INNER, OUTER, LEFT, RIGHT, CROSS
24. Views in MySQL Tutorial: Create, Join & Drop with Examples
25. MySQL Index Tutorial - Create, Add & Drop
26. Your First Application using MySQL and PHP
27. Oracle MySQL 5.6 Certification with Study Guide
28. SQL vs MySQL: What's the Difference?