MySQL Documentation

What is SQL?

- SQL stands for **Structured Query Language**.
- SQL lets you access and manipulate databases.
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

What Can SQL do?

- SQL can execute queries against a database.
- SQL can retrieve data from a database.
- SQL can insert records in a database.
- SQL can update records in a database.
- SQL can delete records from a database.
- SQL can create new databases.
- SQL can create new tables in a database.
- SQL can create stored procedures in a database.
- SQL can create views in a database.

- SQL can set permissions on tables, procedures, and views.

RDBMS

- RDBMS stands for **Relational Database Management System**.
- RDBMS is the basis for SQL and all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Introduction of MySQL:

MySQL is an open-source relational database management system (RDBMS) widely used for storing and retrieving data. It provides a powerful and flexible database management platform and is supported on various operating systems.

Advantages of MySQL:

- Fast and High-Performance database.
- Easy to use, maintain, and administer.
- Easily available and maintains the integrity of the database.
- Provides scalability, usability, and reliability.
- Low-cost hardware.
- MySQL can read simple and complex queries and write operations.
- InnoDB is a default and widely used storage engine.
    - Provides strong indexing support.
- Provides SSL support for secured connections.
- Provides powerful data encryption and accuracy.
- Provides Cross-platform compatibility.

- Provides minimized code repetition.

Database:

A database is a collection of organized data stored in MySQL. It serves as a container for tables, views, procedures, and other database objects. MySQL allows you to create multiple databases within a single MySQL server instance.

Tables:

Tables are the fundamental structure for storing data in MySQL. They consist of rows and columns, forming a grid-like structure. Each table represents a specific entity or concept and holds related data.

Columns:

Columns, also known as fields, represent the individual data elements within a table. They define the type of data stored in a particular field, such as integers, strings, dates, etc.

Rows:

Rows, also called records, contain the actual data within a table. Each row represents a single entry or instance of the data entity defined by the table. Each column in a row corresponds to a specific field.

Primary Key:

A primary key is a unique identifier for each row in a table. It ensures that each row can be uniquely identified and serves as a reference point for other tables that establish relationships with the primary key.

Foreign Key:

A foreign key is a field or set of fields in a table that refers to the primary key of another table. It establishes a

relationship between two tables, enforcing referential integrity and enabling data consistency across the tables.

SQL command:

- Open **MySQL shell => CLI (Command Line Interface)**

Sr. No	Command	Description
1	\sql	Switching SQL mode
2	\connect root@localhost:3306	Connection **Server** with your **PC**
3	show databases;	Show all databases
4	create database dbName	Create New Database
5	use dbName	Select to database
6	show tables;	Show all tables
7	create table tabelName(column datatype, column datatype);	Create New tabel
8	desc tabelName; describe tabelName;	describe the table with all the detail
9	insert into tableName values(val1, val2, val3, ….);	Insert a record into a table
10	select * from tableName;	Select All Records shown
11	\quit	Quit of SQL mode

SQL Data type

A Data Type specifies a particular type of data, like integer, floating points, Boolean, etc. It also identifies the possible values for that type, the operations that can be performed on that type, and the way the values of that type are stored. In MySQL, each database table has many columns and contains specific data types for each column.

▼ **Numeric Data type:**

- bit

- tinyint
- smallint
- **int**
- bigint
- decimal
- numeric
- **float**
- **double**
- bool
- boolean

▼ **Character / String Data type:**

- char
- **varchar**
- tiny text
- text
- long text

▼ **Date / Time Data type:**

- **Date**
- time
- DateTime
- timestamp
- year

▼ **Miscellaneous:**

- JSON
- XML

Create / Insert Query:

- Specify both the column names and the values to be inserted

```
insert into  table_name (col1, col2,...) values (val1, val2, 
```

- Specify both the column names and the values to be inserted

```
insert into  table_name (col1, col2,...) values (val1, val2, 
```

- Adding values for all the columns of the table

```
insert into  table_name values (val1, val2, …);

insert into  table_name values (val1, val2, …), (val1, val2, 
```

Retrieve / Select Query:

- Select all data from the table

```
select * from table_name;
```

- Select a specific column of a table

```
select column1, column2, …., columnN from table_name;
```

- One time show full table and add show table column extra

```
select *, column  from  table_name;
```

- Alias column name

```
select column “alias_name” from table_name;

select column as “alias_name” from table_name;
```

Where Clause:

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.

```
select column1, column2, ….  from  table_name where condition
```

(***condition*** = , < , > , != , <> , <= , >= , BETWEEN , LIKE , IN)

Update Query:

The **UPDATE** statement is used to modify the existing records in a table.

```
update   table_name  set column1 = value1, column2 = value2 ,
```

Notes:- Be careful when updating records in a table Notice the WHERE clause in the **UPDATE** statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Delete Query:

The **Delete** statement is used to delete existing records in a table.

```
delete from table_name ;  // all records deleted

delete from table_name where condition; // specific row delet
```

Notes:- Be careful when deleting records in a table Notice the WHERE clause in the **DELETE** statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Drop Statement:

The **Drop** statement is used to drop an existing table in a database.

```
drop table table_name;   // all record and data structure del

drop database db_name;   // all records and tables delete
```

Notes:- Be careful when dropping a table deleting a table will result in loss of complete information stored in the table.

Order By Keyword:-

- The order by keyword is used to sort the result-set in ascending or descending order.
- The order by keyword escorts the records in ascending order by default. To sort the record in descending order, use the **desc** keyword.

```
-- Ascending order
select * from table_name Order by column_name;

-- Descending Order
Select * from  table_name  Order by column_name desc;

-- Multiple Column order by
Select * from  table_name Order by column_name desc, column_n
```

In operator & Not In operator :

- The in operator allows you to specify multiple values in a where clause.
- The in-operator is a shorthand for multiple **or** conditions.

```
-- in operator
Select * from  table_name Where column_name IN (value1,value2

-- not in operator
Select * from  table_name Where column_name NOT IN (value1,va
```

Between operator:-

- The between operator selects values Within a given range. The values can be numbers, text, or dates.

```
-- between operator
Select * from  table_name Where column_name Between value1 an

-- not between operator
Select * from  table_name Where column_name Not between Value
```

Like operator:-

- The like operator is used in a where clause to search for a specified pattern in a column.
- The percent sign (%) represents zero, one or multiple characters.
- The underscore sign (_) represents one single character.
- **Note**:- Ms Access uses an asterisk(*) instead of the **percent sign(%)** and a question mark(?) instead of the **underscore(_)**.

```
Select * from  table_name  where column like pattern;
```

You can combine any number of conditions using **AND** or **OR** operator.

Pattern	Description
a%	Start with ‘a’
%a	End with ‘a’
_a%	Second character ‘a’
%a_	Second last character ‘a’
%a%	Word in character ‘a’ any position
_a__	Length specific character
a%o	Start with ‘a’ and End with ‘o’

Group by:-

- The group by statement groups raws that have the same value into summary raws, like “find the number of customers in each century”
- The group by statement is often used with aggregate functions (**count()**, **max()**, **min()**, **sum()**, **avg()**) to group the result set by one or more columns.

```
Select column from  table_name Where condition group by colum
```

Distinct statement:-

- The select distinct statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values and sometimes you can count to list the different (distinct) values.

```
Select Distinct column_name, ... from table_name;
```

Limit:-

```
Select * from  table_name  limit number;
```

Constraints

- **Unique Constraint**

  A unique constraint ensures that values in a column or a combination of columns are unique, preventing duplicates. Unlike the primary key constraint, a unique constraint allows null values, but if a column is defined as a unique constraint, only one null value is permitted.

- **Not Null Constraint**

  A not null constraint enforces that a column cannot contain null values. It ensures that a value must be provided for the column during an insert or update operation. This constraint is used to enforce data completeness and avoid unexpected data issues.

- **Check Constraint**

  A check constraint defines a condition or expression that must be true for the data in a column. It allows you to define custom rules to restrict the values stored in the column. For example, you can specify a check constraint to ensure that a numeric column only accepts positive values.

- **Default Constraint**

  The default value will be added to all new records if no other value is specified.

- **Create Index Constraint**

  Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Primary Key:

A primary key is used to ensure data in the specific column is unique. It is a column that cannot have null values. It is either an existing table column or a column that is specifically generated by the databases according to a defined sequence.

- A primary key is used to ensure data in the specific column is unique.
- It uniquely identifies a record in the relational database tables.
- Only one primary key is allowed in a table.
- It is a combination of unique and not null constraints.
- It does not allow not null values.
- Its values cannot be deleted from the parent.
- Its constraint can be implicitly defined on the temporary tables.
- A table can allow a composite primary key.

Create a **primary key** in MySQL, SQL Server, Oracle, Ms Access database tables.

```
Create table table_name (
Column_name  datatype  primary key,
Column_name  datatype,
_ _ _ , _ _ _ );
```

Create a **primary key** in MySQL database table.

```
Create table  table_name (
Column_name  datatype,
Column_name  datatype,
_ _ _ _ , _ _ _ ,
```

```
primary key(Column_name)
);
```

Primary key on Alter table

To create a primary key constraint on the column when the table is already created.

```
Alter table table_name add primary key (column);

-- Using constraint add primary key
Alter table  table_name
Add constraint pk_table name
primary key ( Col_name, col_name);
```

Drop a primary key constraint

To drop a primary key constraint

```
Alter table  table_name Drop primary key;

-- using constraint drop primary key
Alter table  table_name Drop constraint pk_table name;
```

Foreign key

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.

- It refers to the field in a table which is the primary key of another table.
- Where more than one foreign key is allowed in a table.

- It can contain duplicate values and a table in a relational database.
- It can also contain null values.
- Its value can be deleted from the child table.

Create a **foreign key** in MySQL, SQL Server, Oracle, Ms Access database tables.

```
Create table table_name (
Column_name  datatype  primary key,
Column_name  datatype,
_ _ _ , _ _ _,
Foreign key  (colum_name) references ptablename (colum_name)
);       // ptablename = parent table
```

SQL Foreign key on alter table

To create a foreign key constraint on the column when the table is already created.

```
Alter table  table_name
add foreign key (colmn_name) references ptablename (pcol_name

-- Using constraint add foreign key
Alter table  table_name
add constraint fk_tablename
foreign key (col_name) references ptablename (pcol_name);
```

Drop a foreign key constraint.

To drop a foreign key constraint

```
Alter table  table_name Drop foreign key (column_name);
```

Alter table statement

- The alter table statement is used to add, delete, and modify columns in an existing table.
- The alter table statement is also used to add and drop various constraints on an existing table.

```
-- Add a New Column in the table
Alter  table_name
Add column column_name datatype;

-- modifying column in the table
Alter table  table_name
Modify column column_name datatype;

-- Delete Existing Column in the table
Alter table  table_name  Drop column column_name;
```

Join:-

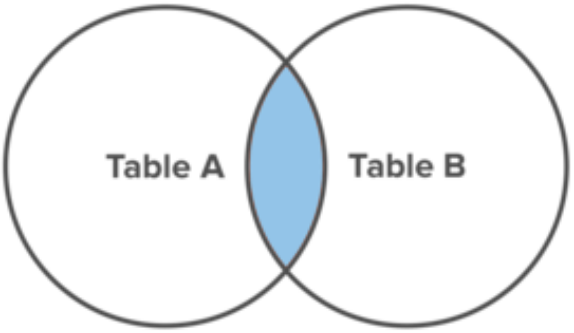
Mysql joins are used with select; it is used to retrieve data from multiple Tables; it is performed whenever you need to fetch records from two or more Tables.

There are three types of MySQL joins.

1. **Inner join** for Returns records that have matching values in both tables
2. **Left join** Returns all records from the left table, and the matched records from the right table
3. **Right join** Returns all records from the right table and the matched records from the left table
4. **Full join** Returns all records when there is a match in either left or right table

Inner join

The `INNER JOIN` is the most common type of join. It returns only those rows that have a match in both joined tables. The following Venn diagram illustrates how inner join works.

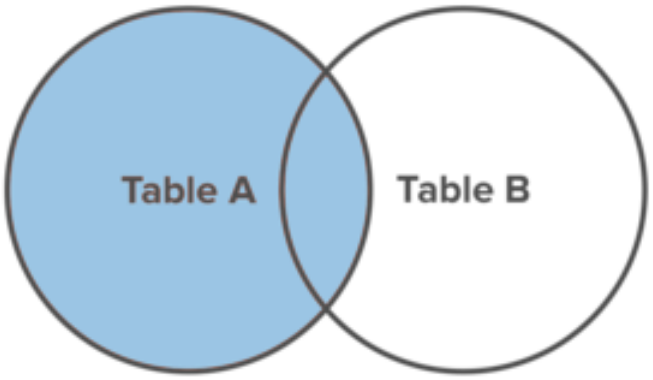


SQL INNER JOIN

```
Select column From table1
Inner join table2
On table1.column = table2.column;
```

Left(outer) join

A `LEFT JOIN` statement returns all rows from the left table along with the rows from the right table for which the join condition is met. Left join is a type of outer join that's why it is also referred to as *left outer join*. The following Venn diagram illustrates how left-join works.

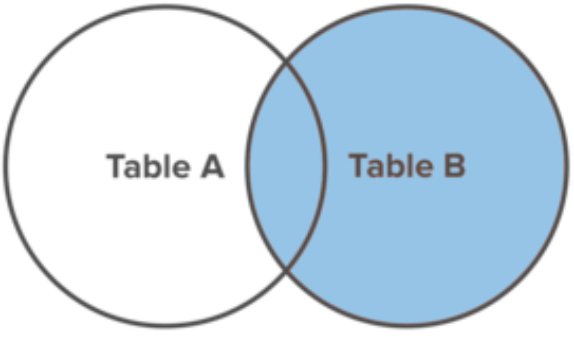


SQL LEFT JOIN

```
Select column From table1
Left join table2 On table1.column = table2.column;
```

Right(outer) join

The `RIGHT JOIN` is the exact opposite of the `LEFT JOIN`. It returns all rows from the right table along with the rows from the left table for which the join condition is met. Right join is a type of outer join that's why it is also referred to as *right outer join*. The following Venn diagram illustrates how right-join works.

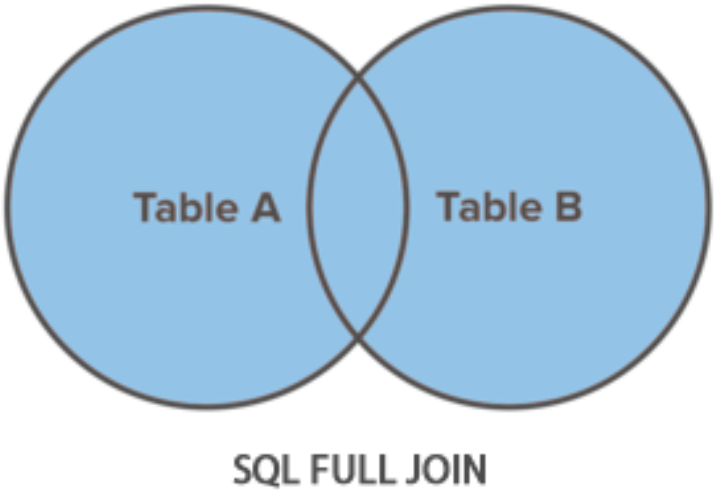


SQL RIGHT JOIN

```
Select column From table1
Right join table2 On table1.column = table2.column;
```

Full join

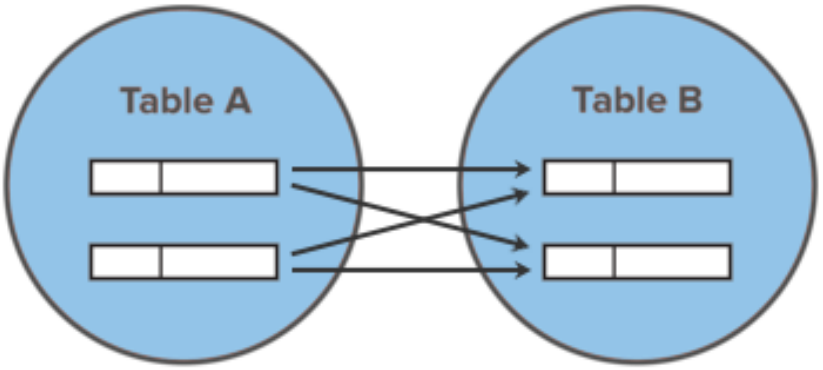
A `FULL JOIN` returns all the rows from the joined tables, whether they are matched or not i.e. you can say a full join combines the functions of a `LEFT JOIN` and a `RIGHT JOIN`. A full join is a type of outer join that's why it is also referred to as *full outer join*. The following Venn diagram illustrates how full-join works.




```
Select column From table1
FUll join table2 On table1.column = table2.column;
```

Cross join

If you don't specify a join condition when joining two tables, the database system combines each row from the first table with each row from the second table. This type of join is called a cross join or a Cartesian product. The following Venn diagram illustrates how cross-join works.



SQL CROSS JOIN

```
Select column From table1 cross join table2;
```

SubQuery:

A subquery is embedded inside another query and acts as input or output for that query. Subqueries are also called **inner queries** and they can be used in various complex operations in SQL.

MYSQL Subquery can be used with an outer query which is used as input to the outer query. It can be used with **SELECT, FROM,** and **WHERE clauses.** MYSQL subquery is executed first before the execution of the outer query.

```
Select * From tableName where
                    column = (select * from table_name where
```