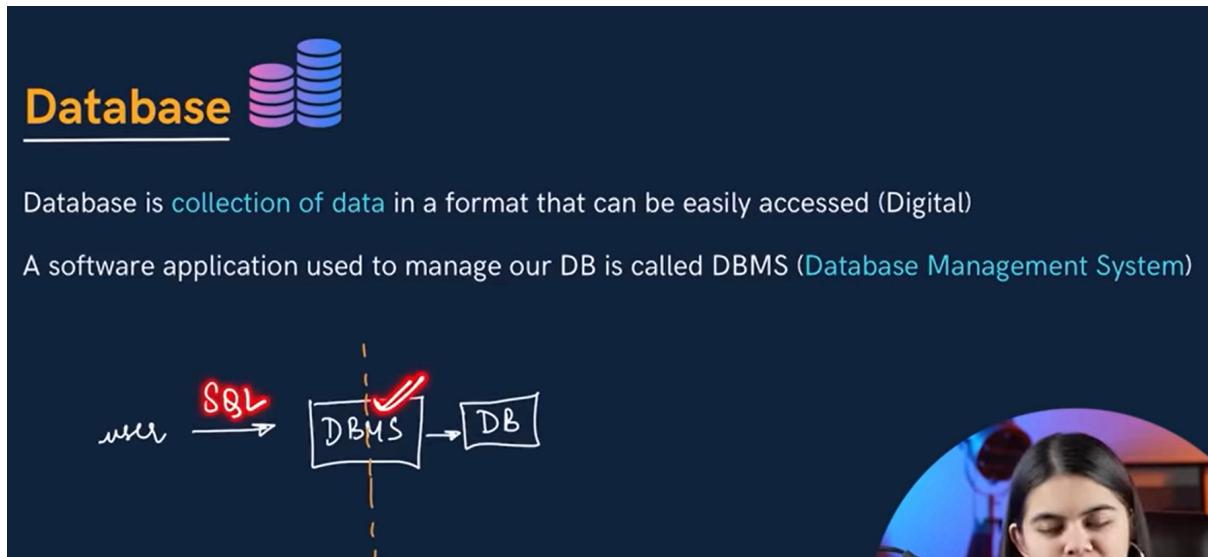


# SQL ( Structured Query Language )

---

To access databases we use SQL .

Firstly let's study what **Databases** are .



## DATABASE :- ( collection of data )

Big companies have a lot of data , E.x. We signed in and we saved our information in google , instagram , facebook . So these big companies use **databases** to save this information.

So because of this we have a collection of data , and collection is stored in a format which is easily accessible e.x. We can search data , to add new data , to delete data , etc .

So such data collection which is easily accessible in specific format is known as **Database**.

## Database Management System ( DBMS ):-

There is **software** which is managing **Data** i.e. to add , delete , update or search data .

The process is basically , Suppose we are a user i.e. we are using database , then we are not able to access the data directly. To access data we use **DBMS** , **DBMS** is a middle layer i.e. A Software application. WE use DBMS and , DBMS do all the changes in backend and give us information .

Hence we use **SQL** for **DBMS**. i.e to interact with Database management System .

---

# Types of Databases:-

There are TWO types of Databases: **Relational** and **Non-relational** .

## Relational Databases:

Here all the data is stored in the form of **Tables** i.e. rows and columns.

## Non-Relational Databases:

When we store data **Without Tables** then such databases are known as **Non-relational** , we can also say them **NoSQL** databases. Because such databases **Don't Know SQL**.

---

To work on **Relational Databases** we need to use **SQL** .Hence such databases also known as **RDBMS( Relational Database Management System )**.

E.x. The popular example of a relational database is **MySQL**.

**MySQL** is such type of **Database Management system** , which works on **relational** and use **SQL** to store data in form of **Tables** .

Apart from this for **relational** there are other **DBMS** like **ORACLE**, **Microsoft SQL Server** , **PostgreSQL**.

Examples of **Non-Relational**,  
Most popular is **mongoDB**.

But now Here will specifically focus on **MySQL DBMS(Database Management System)**.  
And there will create database and will apply queries on it , and from there will try to retrieve , add or delete data.

The diagram is titled "Types of Databases". It compares two main categories: Relational (RDBMS) and Non-relational (NoSQL).  
**Relational (RDBMS):** Data is stored in tables. Logos shown include MySQL, Microsoft SQL Server, ORACLE, and PostgreSQL.  
**Non-relational (NoSQL):** Data is not stored in tables. Logo shown is mongoDB.  
A note at the bottom states: \*\* We use SQL to work with relational DBMS

# What is SQL ?

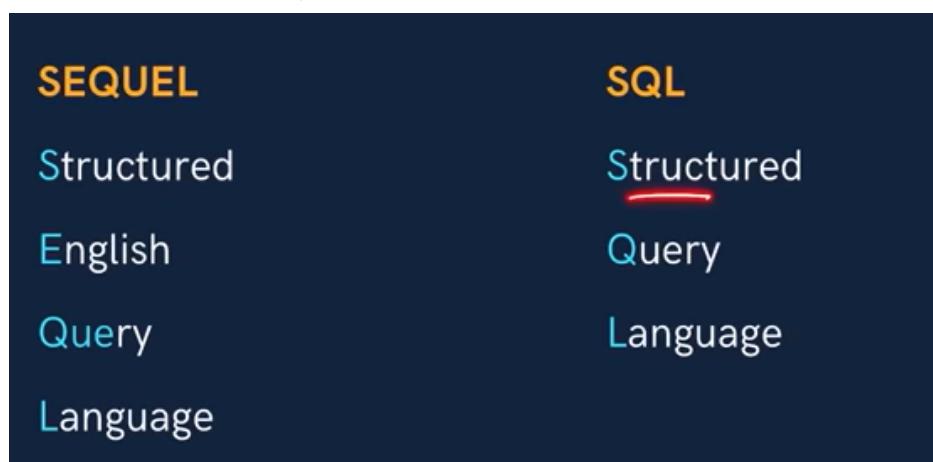
## Structured Query Language

This language is understood by computers and its use to interact with relational databases . On our data we perform four Major operations. I.e **CRUD** operations. ( **create, read , update and delete** ).



Earlier SQL's full name was **Structured English Query Language** and was created by **IBM** company , So earlier it was known as **SEQUEL** .

Later it was named **SQL** .



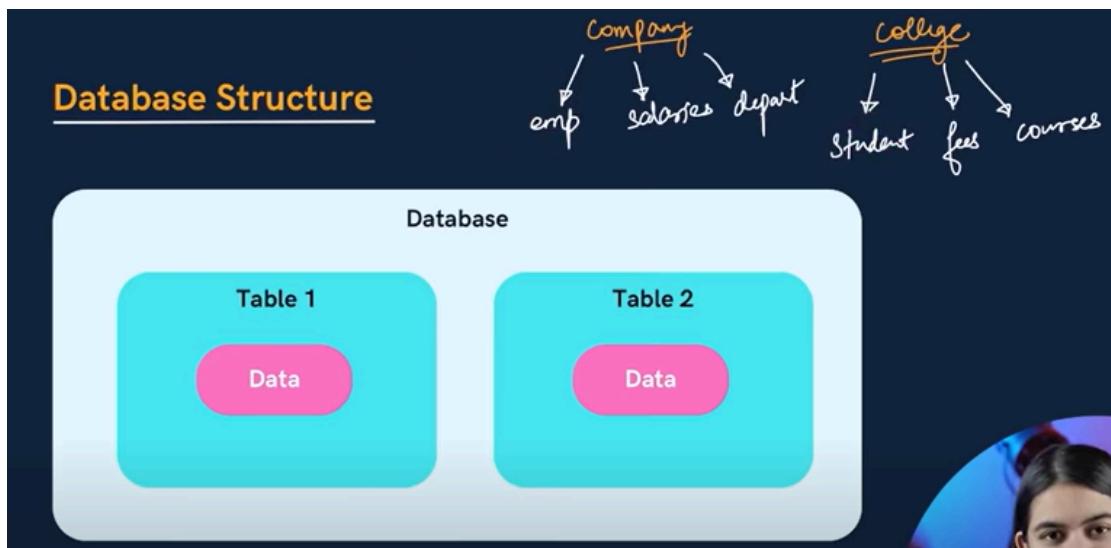
# Database Structure:-

There are multiple tables in Database . Each table will have some particular set of interrelated data .

Interrelated means , Suppose we are creating a Database **Student** , then tables will be Student , fees , courses i.e. related to **Student** . Same for **Company** databases , then tables will be employee , salaries, departments, etc i.e. related to the **company** .

In Databases It's **not** Structure like in the same database we have information of employee and student together.

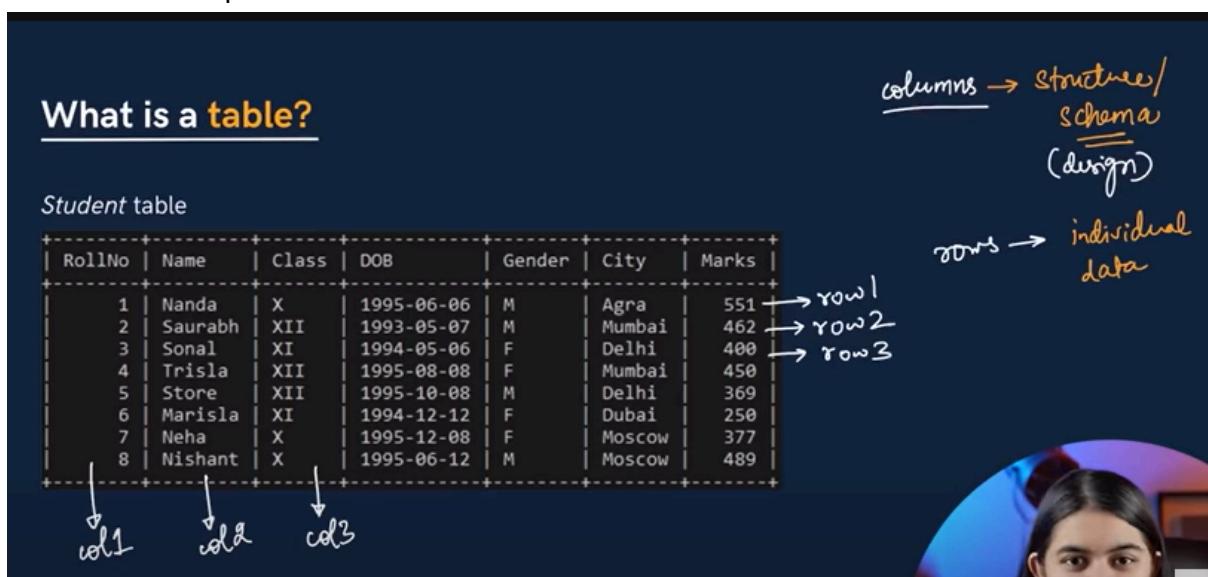
Generally we store interrelated data in the form of tables .



By **columns** we understand the Structure/Schema( Design ) of table i.e. **Columns** will tell us what is stored in the table .

Ex. By column Rollno , Name , Class , DOB , etc we can understand what is stored in **Student Table** .

And **Rows** will represent Individual data of students .

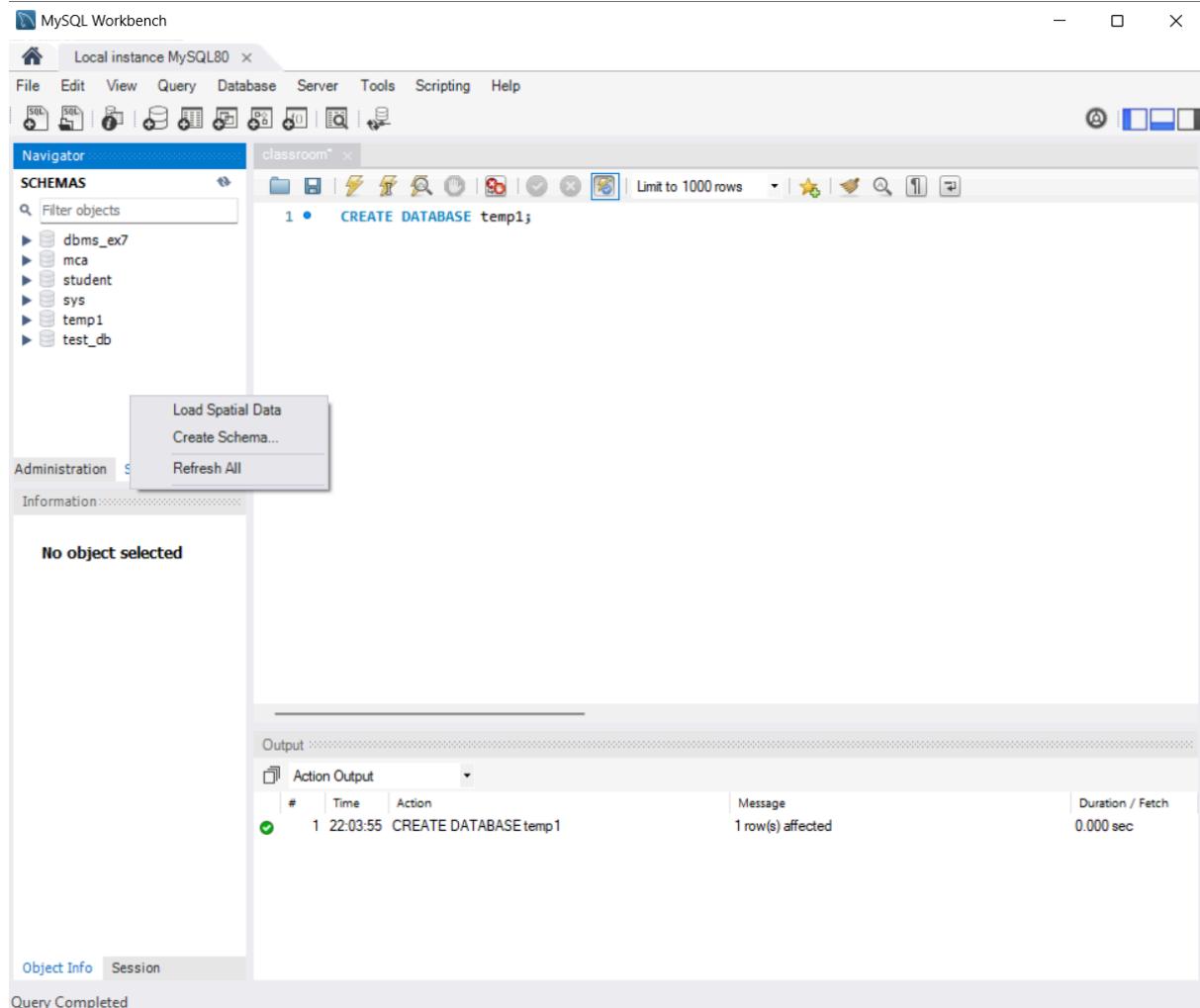


# Create our First Database:-

Our first SQL Query

**CREATE DATABASE db\_name;**

Example,



Here as we can see we can see green tick at bottom , SO query has been executed successfully and our **temp1** database has been created , and by doing **Right CLICK -> Refresh all** in the **SCHEMAS** section we can see the database has been created.

SQL is **not case sensitive** , hence we can write in small case letters also .  
I.e. **create database temp1** is similar to **CREATE DATABASE temp1**

Generally will use Uppercase .

Now to use the database , i.e. for performing operations on some particular database , firstly we have to choose that , to do so we will use ...

## **USE db\_name;**

This will choose the particular database where we have to do data operations.

Example ,

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Local instance MySQL80' is selected. The 'Schemas' tab in the Navigator pane shows databases like dbms\_ex7, mca, student, sys, temp1, and test\_db. The 'temp1' database is currently selected. In the central query editor, there are two statements: 'CREATE DATABASE temp1;' and 'USE temp1;'. The 'Output' pane at the bottom displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	22:03:55	CREATE DATABASE temp1	1 row(s) affected	0.000 sec
2	23:21:08	USE temp1	0 row(s) affected	0.000 sec

## **Creating our First Table:-**

Now after creating the Database we have to create tables in it ,there will define **columns** i.e. **Schema** , for that will use ...

```

CREATE TABLE table_name(
    column_name1 datatype constraint,
    column_name2 datatype constraint,
    .....
    column_nameNth datatype constraint
);

```

Here **column name** is name of the column i.e Schema, and **datatype** is basically the type of data which we want to store for that particular column, and SQL **constraints** are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table.

Example,

Let's create a Database college , where will store Student data . For that lets first create a Student table in it .

**Creating our First Table**

```

USE db_name;

CREATE TABLE table_name (
    column_name1 datatype constraint,
    column_name2 datatype constraint,
    column_name2 datatype constraint
);

```

**student**

<b>id</b>	<b>name</b>	<b>age</b>

```

CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT NOT NULL
);

```



Also we can insert values into the table by using ...

# INSERT INTO Syntax

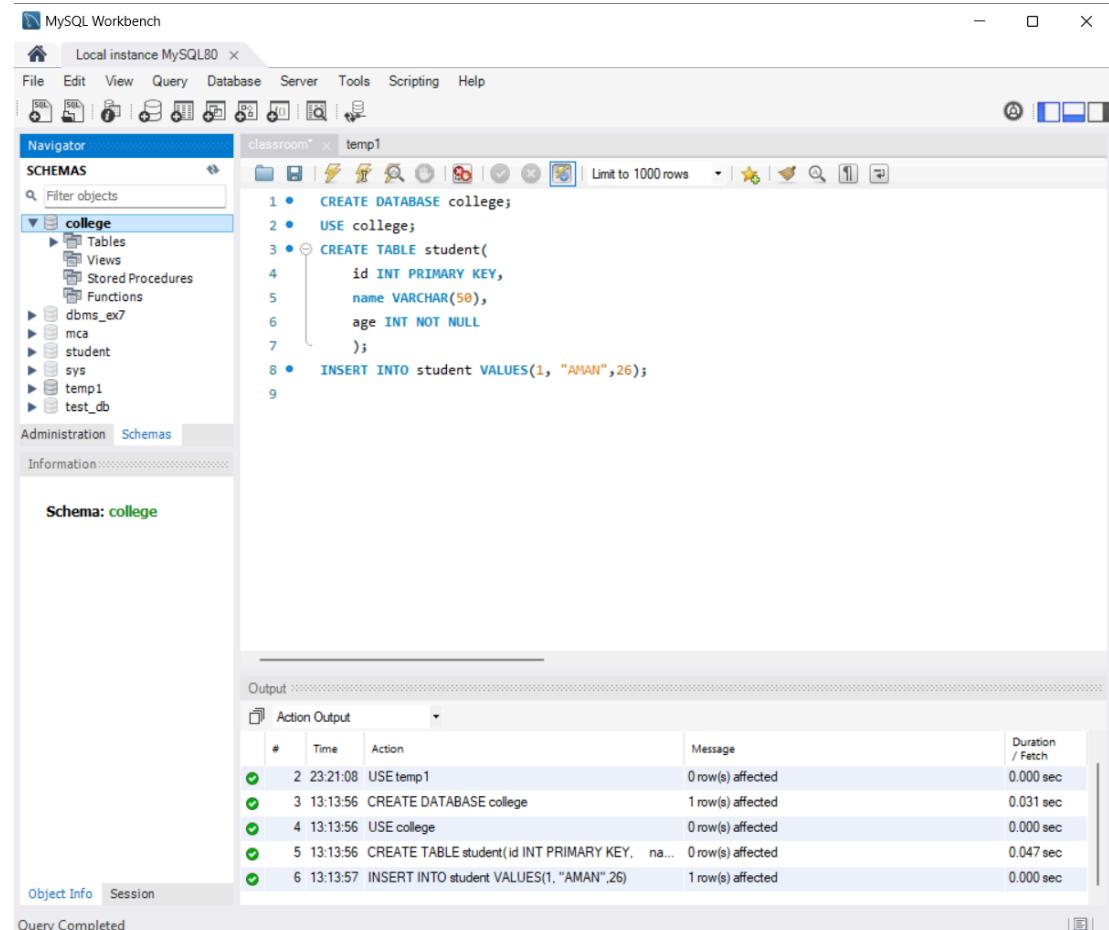
**It is possible to write the `INSERT INTO` statement in two ways:**

**1. Specify both the column names and the values to be inserted:**

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

**2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the `INSERT INTO` syntax would be as follows:**

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```



Here as we can see , the college database has been created by **CREATE DATABASE college**, then we accessed the database by **USE college** , and then we created a table by **CREATE TABLE student** and also we inserted values in it by **INSERT INTO student VALUES();**

We have used constraint **PRIMARY KEY** for **id** , i.e. it should be unique . So if we try to insert values in a table with the same existing **id** , then it will give an error . So **PRIMARY KEY** should be different in every value.

Example ,

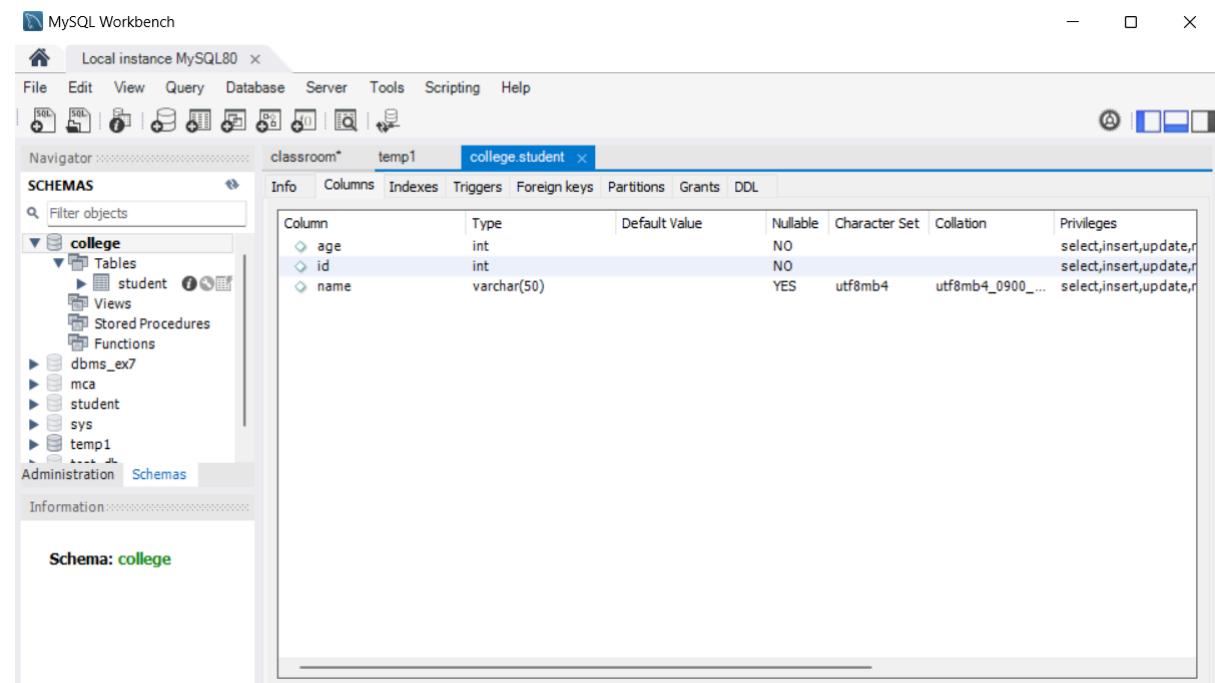
```
8 •    INSERT INTO student VALUES(1, "AMAN",26);
9 •    INSERT INTO student VALUES(1, "Shraddha",24);
```

o/p,

```
6 13:13:57 INSERT INTO student VALUES(1, "AMAN",26) 1 row(s) affected 0.000 sec
X 7 14:20:39 INSERT INTO student VALUES(1, "Shraddha",24) Error Code: 1062. Duplicate entry '1' for key 'student.PR... 0.016 sec
```

---

We can see the **schema** by clicking on (i) button of table **student** of database **student** under **SCHEMAS** section.



The screenshot shows the MySQL Workbench interface. The title bar says "MySQL Workbench" and "Local instance MySQL80". The menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, Help. The toolbar has various icons for database management. The left sidebar is the Navigator, showing the "SCHEMAS" section with "college" selected, which contains "Tables", "student", "Views", "Stored Procedures", and "Functions". Below "Tables" are "dbms\_ex7", "mca", "student", "sys", and "temp1". The main central area shows the "Info" tab of the "college.student" table. The table has three columns: "age" (int), "id" (int), and "name" (varchar(50)). The "Privileges" column shows "select,insert,update,r" for age and id, and "select,insert,update,r" for name. The "Collation" is utf8mb4\_0900\_... for all columns. The "Default Value" is empty for all columns. The "Nullable" column shows "NO" for age and id, and "YES" for name. The "Character Set" is utf8mb4 for all columns.

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges
age	int		NO			select,insert,update,r
id	int		NO			select,insert,update,r
name	varchar(50)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,r

Now as we saw from SQL DBMS software , we can also see it by using SQL command i.e. we can print our whole table , for that will use

**SELECT \* FROM table\_name;**

The screenshot shows the MySQL Workbench interface. In the central query editor window, the following SQL code is displayed:

```
6     age INT NOT NULL
7   );
8 •   INSERT INTO student VALUES(1, "AMAN",26);
9 •   INSERT INTO student VALUES(1, "Shraddha",24);
10
11 •   SELECT * FROM student;
```

Below the code, the Result Grid shows the output of the SELECT query:

	id	name	age
1	1	AMAN	26
*	NULL	NULL	NULL

At the bottom of the interface, the Output panel displays the session log:

#	Time	Action	Message	Duration / Fetch
3	13:13:56	CREATE DATABASE college	1 row(s) affected	0.031 sec
4	13:13:56	USE college	0 row(s) affected	0.000 sec
5	13:13:56	CREATE TABLE student(id INT PRIMARY KEY, ...)	0 row(s) affected	0.047 sec
6	13:13:57	INSERT INTO student VALUES(1, "AMAN",26)	1 row(s) affected	0.000 sec
7	14:20:39	INSERT INTO student VALUES(1, "Shraddha",24)	Error Code: 1062. Duplicate entry '1' for key 'student_id'	0.016 sec
8	14:31:59	SELECT * FROM student LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

After executing we can see our whole table displayed .

Now this was overall view of SQL , now will do detail study ...  
Lets start from DATA TYPES ..

---

# SQL Datatypes:-

Most commonly used datatypes ...

## SQL Datatypes

They define the **type of values** that can be stored in a column

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer( -2,147,483,648 to 2,147,483,647 )	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer( -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 )	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

### 1. CHAR and VARCHAR:-

Difference between **CHAR** and **VARCHAR**,



Now 50 whole bytes will be reserved for **CHAR** even if we just use 4 bytes for string "PUNE".  
I.e Inefficient use of memory .

But , **VARCHAR** has just given a max limit as 50 , and will use only the necessary space which is required . i.e. efficient use of memory .

### 2. BLOB:-

Its use to store large strings. E.x. suppose we want to store a file data .

### 3. INT , TINYINT , BIGINT :-

Here we can store integers , difference is just that how big number is .

### 4. BIT:-

If we choose BIT(1) then we can store only 1 bit of data i.e. either **0** or **1**

If we choose BIT(2) then we can store 2 bits of data i.e. either **00** or **01** or **10** or **11**

And so on ...

In computers all data is stored in BITS , **0** means computer got NO SIGNAL , **1** means current has been gone into computer and got SIGNAL .

### **5. FLOAT, DOUBLE :-**

To store decimal values we use **FLOAT** , and if we want to store decimal values only but .. Big Numbers , then we use **DOUBLE** .

### **6. BOOLEAN :-**

It is used to store TRUE or FALSE . i.e. 1 or 0

To implement **BOOLEAN** we use **TINYINT**.

### **7. DATE, YEAR:-**

We store **DATE** in format YYYY-MM-DD, and **YEAR** in format YYYY

We also have TIME , DATETIME, etc .

Except all these , there are also further data types

**SIGNED** and **UNSIGNED**

So the Datatypes which we use to store Numbers E.x. INT, FLOAT , DOUBLE and TINYINT.  
It can consist of negative as well as positive numbers .

So Numbers which may be **negative** or positive , such numbers we can say come under  
**SIGNED** Datatype i.e. It may have sign ( - or + )

E.x TINYINT UNSIGNED (

But, in some cases we know where we will get only positive value . E.x. Age, salary  
So in such cases we use **UNSIGNED** datatype to increase our range .

**TINYINT UNSIGNED (0 to 255)**

**TINYINT (-128 to 127)**

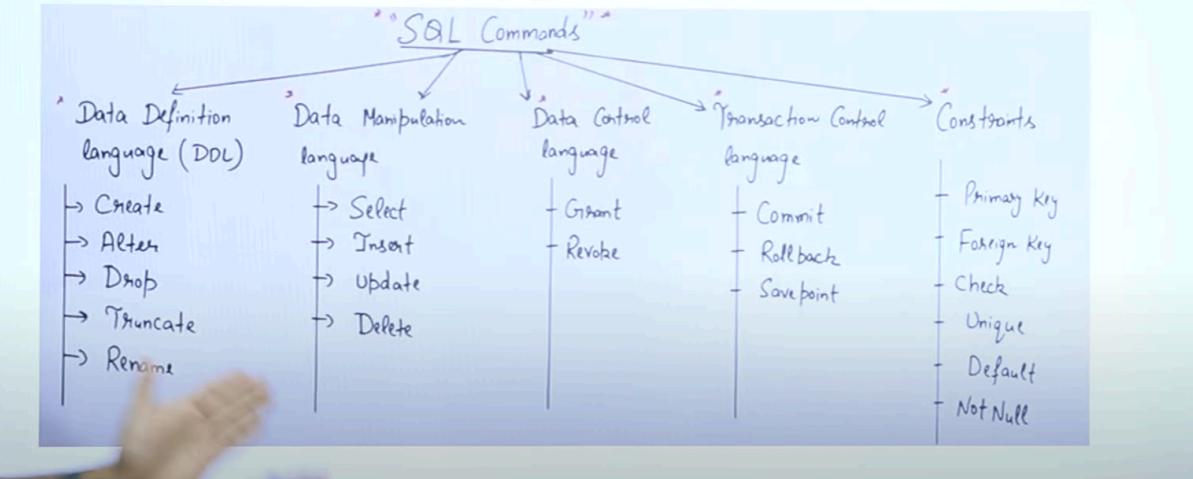
---

## Types of SQL Commands :-

Till now we learned basic SQL commands and queries , also we learned about Datatypes .

Now all the SQL commands we can classify into five different types .

## 5. Explain different types of SQL Commands



1. DDL ( Data Definition Language ): ( to change table schema/definition/structure)

**create, alter, rename, truncate & drop**

DDL i.e. Suppose if we want to create a database , table , or we want to rename it or delete it.

So such kind of operations i.e. creation or deletion , works on it .

Hence How data is going to be defined , how schema is going to be created or how columns are going to be defined , or how we can change the type of column in future .

2. DQL ( Data Query Language ): **select**

To display the table data .

3. DML ( Data Manipulation Language ): ( table data )

**insert , update & delete**

Suppose we want to change rows , new data to insert , update the old data , or want to delete row .

## 4. DCL ( Data control Language ):

### grant & revoke permissions to users

Suppose there are two tables in the database , one is **fees** and another is **marks**. Now in the **college** database there are two tables .

Now we know the **accounts** team should have access to the **fees** table , and **teachers** should have access to the **marks** table .

Now in this case we can decide which user can access which table .

## 5. TCL ( Transaction Control Language ):

### start transaction , commit , rollback

It comes under advanced DBMS .

---

# Database related Queries :-

**CREATE DATABASE db\_name;**

**CREATE DATABASE IF NOT EXISTS db\_name;**

Here if we try to create database which already exist , then for that we can write condition in command like **IF NOT EXISTS** . So database only be created if its not exist .

Example ,

```
13 •  CREATE DATABASE college;
14 •  CREATE DATABASE IF NOT EXISTS college;
```

Output :::::::				
Action Output				
#	Time	Action	Message	Duration / Fetch
✓ 5	13:13:56	CREATE TABLE student(id INT PRIMARY KEY, ...	0 row(s) affected	0.047 sec
✓ 6	13:13:57	INSERT INTO student VALUES(1, "AMAN",26)	1 row(s) affected	0.000 sec
✗ 7	14:20:39	INSERT INTO student VALUES(1, "Shraddha",24)	Error Code: 1062. Duplicate entry '1' for key 'studen...'	0.016 sec
✓ 8	14:31:59	SELECT * FROM student LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
✗ 9	13:55:21	CREATE DATABASE college	Error Code: 1007. Can't create database 'college'; d...	0.015 sec
⚠ 10	13:56:01	CREATE DATABASE IF NOT EXISTS college	1 row(s) affected, 1 warning(s): 1007 Can't create d...	0.015 sec

Here as we can see , the first time we got an error when we tried to create a database , and next time we didn't get an error , but we got just a warning .

So it's a good practice to write **IF NOT EXISTS** while creating a database.  
So In code ERROR is BAD thing , But WARNING is ok.

---

Same thing we can do with **DROP DATABASE** ,

**DROP DATABASE db\_name;**

**DROP DATABASE IF EXISTS db\_name;**

Example,

```
15 •  DROP DATABASE company;  
16 •  DROP DATABASE IF EXISTS company;
```

Output :::::::					
Action Output					
#	Time	Action	Message	Duration / Fetch	
✖	7 14:20:39	INSERT INTO student VALUES(1, "Shraddha",24)	Error Code: 1062. Duplicate entry '1' for key 'studen...'	0.016 sec	
✓	8 14:31:59	SELECT * FROM student LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec	
✖	9 13:55:21	CREATE DATABASE college	Error Code: 1007. Can't create database 'college'; d...	0.015 sec	
⚠	10 13:56:01	CREATE DATABASE IF NOT EXISTS college	1 row(s) affected, 1 warning(s): 1007 Can't create d...	0.015 sec	
✖	11 14:02:21	DROP DATABASE company	Error Code: 1008. Can't drop database 'company'; d...	0.031 sec	
⚠	12 14:03:27	DROP DATABASE IF EXISTS company	0 row(s) affected, 1 warning(s): 1008 Can't drop dat...	0.016 sec	

As we can see, the first time we got a warning , and the second time we got an error .  
So always keep this practice to write **IF EXISTS** while deleting databases .

---

To Display Databases we can write ,

**SHOW DATABASES;**

Example,

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** The query `SHOW DATABASES;` is entered.
- Result Grid:** The results show a list of databases:
  - college
  - dbms\_ex7
  - information\_schema
  - mca
  - mysql
  - performance\_schema
  - student
  - sys
  - temp1
  - test\_db
- Action Output:** A table showing the history of actions:

#	Time	Action	Message	Duration / Fetch
8	14:31:59	SELECT * FROM student LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
9	13:55:21	CREATE DATABASE college	Error Code: 1007. Can't create database 'college'; d...	0.015 sec
10	13:56:01	CREATE DATABASE IF NOT EXISTS college	1 row(s) affected, 1 warning(s): 1007 Can't create d...	0.015 sec
11	14:02:21	DROP DATABASE company	Error Code: 1008. Can't drop database 'company'; d...	0.031 sec
12	14:03:27	DROP DATABASE IF EXISTS company	0 row(s) affected, 1 warning(s): 1008 Can't drop dat...	0.016 sec
13	14:08:01	SHOW DATABASES	10 row(s) returned	0.016 sec / 0.000 sec
- Status:** Read Only

Now we can see , all databases has been displayed .

Similarly we can see the TABLES of current database which we are using .

## SHOW TABLES;

Example,

18 • SHOW TABLES;

Tables_in_college	
▶	student

Result Grid | Filter Rows:  Export: Wrap Cell Content:

Result 3 × Read Only

Output

Action Output		
#	Time	Action
✖ 9	13:55:21	CREATE DATABASE college
⚠ 10	13:56:01	CREATE DATABASE IF NOT EXISTS college
✖ 11	14:02:21	DROP DATABASE company
⚠ 12	14:03:27	DROP DATABASE IF EXISTS company
✓ 13	14:08:01	SHOW DATABASES
✓ 14	14:10:02	SHOW TABLES

## Table related Queries :-

To create a table , we use the CREATE command .

We already studied the syntax of table ,

```
CREATE TABLE table_name (
    column_name1 datatype constraint,
    column_name2 datatype constraint,
);
```

This command helps us to define our **schema/ Design / Columns** .

Example,

```
CREATE TABLE student (
    rollno INT PRIMARY KEY,
    name VARCHAR(50)
);
```

There are two specialities of **PRIMARY KEY** ,

1. It's always **NOT NULL** i.e. we must fill value , it can't be empty.
2. Should have **UNIQUE VALUE**

```

5     name VARCHAR(50),
6     age INT NOT NULL
7   );
8 •  INSERT INTO student VALUES(1, "AMAN",26);
9 •  INSERT INTO student VALUES(1, "Shradha",24);
10
11 •  SELECT * FROM student;
12
13 •  CREATE DATABASE college;
14 •  CREATE DATABASE IF NOT EXISTS college;
15 •  DROP DATABASE company;
16 •  DROP DATABASE IF EXISTS company;
17 •  SHOW DATABASES;
18 •  SHOW TABLES;
19 •  DROP TABLE student;
20 •  CREATE TABLE student(
21   rollno INT PRIMARY KEY,
22   name VARCHAR(50)
23 );

```

Output

#	Time	Action	Message	Duration / Fetch
11	14:02:21	DROP DATABASE company	Error Code: 1008. Can't drop database 'company'; d...	0.031 sec
12	14:03:27	DROP DATABASE IF EXISTS company	0 row(s) affected, 1 warning(s): 1008 Can't drop dat...	0.016 sec
13	14:08:01	SHOW DATABASES	10 row(s) returned	0.016 sec / 0.000 sec
14	14:10:02	SHOW TABLES	1 row(s) returned	0.016 sec / 0.000 sec
15	14:20:58	DROP TABLE student	0 row(s) affected	0.032 sec
16	14:22:33	CREATE TABLE student(rollno INT PRIMARY KE...	0 row(s) affected	0.031 sec

As we can see in the SCHEMA section , Table has been created with the columns .  
Also we can see in ACTION OUTPUT .

To see the data we know we have to use **SELECT** command.

**SELECT \* FROM *table\_name*;**

**SELECT \* FROM student;**

Here \* means **all** .

I.e select all rows and columns from Table.

Example ,

24 • `SELECT * FROM student;`

The screenshot shows the MySQL Workbench interface. At the top, a query window displays the command `SELECT * FROM student;`. Below it, the "Result Grid" pane shows a single row with columns `rollno` and `name`, both containing `NULL`. To the right of the Result Grid is a vertical toolbar with icons for "Result Grid", "Form Editor", and "Field Types". Below the Result Grid is the "student 4" connection status bar with "Apply" and "Revert" buttons. Underneath the connection status is the "Output" tab, which is currently selected. It contains a table titled "Action Output" with the following data:

#	Time	Action	Message	Duration / Fetch
12	14:03:27	DROP DATABASE IF EXISTS company	0 row(s) affected, 1 warning(s): 1008 Can't drop dat...	0.016 sec
13	14:08:01	SHOW DATABASES	10 row(s) returned	0.016 sec / 0.000 sec
14	14:10:02	SHOW TABLES	1 row(s) returned	0.016 sec / 0.000 sec
15	14:20:58	DROP TABLE student	0 row(s) affected	0.032 sec
16	14:22:33	CREATE TABLE student(rollno INT PRIMARY KE...	0 row(s) affected	0.031 sec
17	15:12:48	SELECT * FROM student LIMIT 0, 1000	0 row(s) returned	0.047 sec / 0.000 sec

Table has Columns , but there is NO data in it .

So for inserting we know that ,we have to use the **INSERT** command .

**Insert**

```

INSERT INTO table_name
(colname1, colname2)
VALUES
(col1_v1, col2_v1),
(col1_v2, col2_v2);

```

```

INSERT INTO student
(rollno, name)
VALUES
(101, "karan"),
(102, "arjun");

```

Example ,

```

25 • INSERT INTO student
26   (rollno, name)
27   VALUES
28   (101, "karan"),
29   (102,"arjun");
30

```

Output :

Action Output

#	Time	Action	Message	Duration / Fetch
13	14:08:01	SHOW DATABASES	10 row(s) returned	0.016 sec / 0.000 sec
14	14:10:02	SHOW TABLES	1 row(s) returned	0.016 sec / 0.000 sec
15	14:20:58	DROP TABLE student	0 row(s) affected	0.032 sec
16	14:22:33	CREATE TABLE student(rollno INT PRIMARY KE...	0 row(s) affected	0.031 sec
17	15:12:48	SELECT * FROM student LIMIT 0, 1000	0 row(s) returned	0.047 sec / 0.000 sec
18	15:18:14	INSERT INTO student (rollno, name) VALUES (10...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec

As we can see data has been successfully inserted ,

Lets display the data ...

```

30 • SELECT * FROM student;

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

Result Grid

Form Editor

Field Types

student 5

Apply

Revert

Output :

Action Output

#	Time	Action	Message	Duration / Fetch
14	14:10:02	SHOW TABLES	1 row(s) returned	0.016 sec / 0.000 sec
15	14:20:58	DROP TABLE student	0 row(s) affected	0.032 sec
16	14:22:33	CREATE TABLE student(rollno INT PRIMARY KE...	0 row(s) affected	0.031 sec
17	15:12:48	SELECT * FROM student LIMIT 0, 1000	0 row(s) returned	0.047 sec / 0.000 sec
18	15:18:14	INSERT INTO student (rollno, name) VALUES (10...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
19	15:19:53	SELECT * FROM student LIMIT 0, 1000	2 row(s) returned	0.016 sec / 0.000 sec

Instead of writing

**INSERT INTO student(rollno, name) VALUES(101, "karan"), (102,"arjun");**

We can also write

**INSERT INTO student VALUES(101, "karan"), (102,"arjun");**

We can skip this part:- **(rollno, name)** , if we write values in correct order .

Example ,

```
31 • INSERT INTO student VALUES(103, "shyam");
```

Output

Action Output
# Time Action Message Duration / Fetch
15 14:20:58 DROP TABLE student 0 row(s) affected 0.032 sec
16 14:22:33 CREATE TABLE student(rollno INT PRIMARY KE... 0 row(s) affected 0.031 sec
17 15:12:48 SELECT * FROM student LIMIT 0, 1000 0 row(s) returned 0.047 sec / 0.000 sec
18 15:18:14 INSERT INTO student (rollno, name) VALUES (10... 2 row(s) affected Records: 2 Duplicates: 0 Wamin... 0.000 sec
19 15:19:53 SELECT * FROM student LIMIT 0, 1000 2 row(s) returned 0.016 sec / 0.000 sec
20 15:25:19 INSERT INTO student VALUES(103, "shyam") 1 row(s) affected 0.047 sec

No lets display ,

```
32 • SELECT * FROM student;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

rollno	name
101	karan
102	arjun
103	shiyam
NULL	NULL

student 6 x Apply Revert

Output

Action Output
# Time Action Message Duration / Fetch
17 15:12:48 SELECT * FROM student LIMIT 0, 1000 0 row(s) returned 0.047 sec / 0.000 sec
18 15:18:14 INSERT INTO student (rollno, name) VALUES (10... 2 row(s) affected Records: 2 Duplicates: 0 Wamin... 0.000 sec
19 15:19:53 SELECT * FROM student LIMIT 0, 1000 2 row(s) returned 0.016 sec / 0.000 sec
20 15:25:19 INSERT INTO student VALUES(103, "shyam") 1 row(s) affected 0.047 sec
21 15:26:37 SELECT * FROM studnet LIMIT 0, 1000 Error Code: 1146. Table 'college.studnet' doesn't ex... 0.031 sec
22 15:26:47 SELECT * FROM student LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000 sec

# Practice Question:

**Practice Qs 1**  *25,000.25*

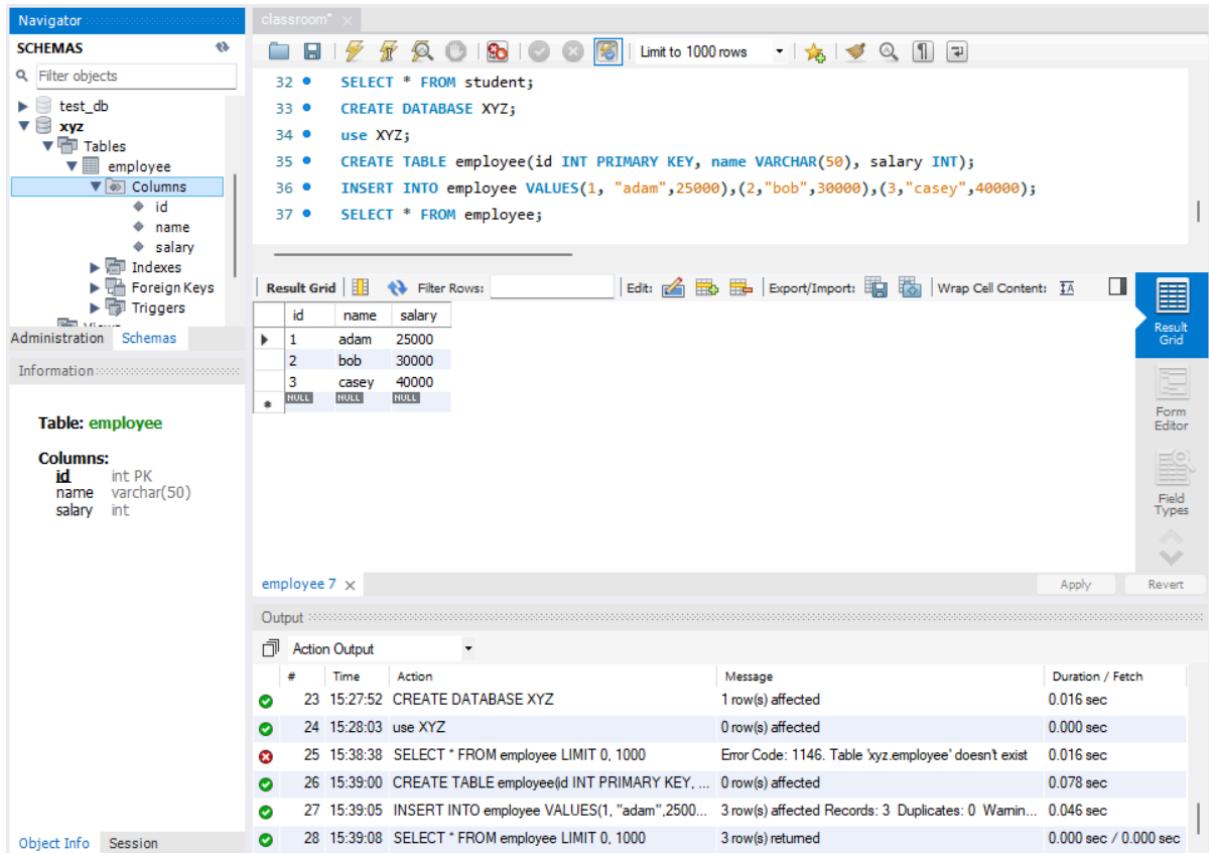
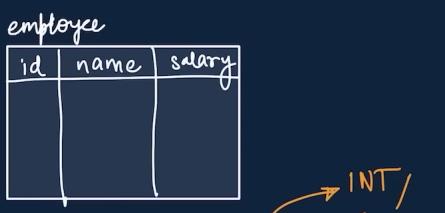
Qs: Create a database for your company named XYZ.

Step1 : create a table inside this DB to store employee info (id, name and salary).

Step2 : Add following information in the DB :

1, "adam", 25000  
2, "bob", 30000  
3, "casey", 40000

Step3 : Select & view all your table data.



Navigator

SCHEMAS

Filter objects

test\_db

xyz

Tables

employee

Columns

id

name

salary

Indexes

Foreign Keys

Triggers

Administration Schemas Information

Table: employee

Columns:

<b>id</b>	int PK
name	varchar(50)
salary	int

Result Grid

#	id	name	salary
1	1	adam	25000
2	2	bob	30000
3	3	casey	40000
*	NULL	NULL	NULL

Result Grid Form Editor Field Types

employee 7

Action Output

#	Time	Action	Message	Duration / Fetch
23	15:27:52	CREATE DATABASE XYZ	1 row(s) affected	0.016 sec
24	15:28:03	use XYZ	0 row(s) affected	0.000 sec
25	15:38:38	SELECT * FROM employee LIMIT 0, 1000	Error Code: 1146. Table 'xyz.employee' doesn't exist	0.016 sec
26	15:39:00	CREATE TABLE employee(id INT PRIMARY KEY, name VARCHAR(50), salary INT)	0 row(s) affected	0.078 sec
27	15:39:05	INSERT INTO employee VALUES(1, "adam", 25000), (2, "bob", 30000), (3, "casey", 40000)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.046 sec
28	15:39:08	SELECT * FROM employee LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

# Keys:-

In Tables there are some special columns , So such kind of special columns are known as **Keys**.

## Primary Key:( Unique and Not Null )

In the English language **Primary** means **main thing** . SO that's the work of PRIMARY KEY . PRIMARY KEY can be a column or set of columns i.e. combination of columns can be a PRIMARY key , ,or a single column can be a primary key .

In a Table there is one column which uniquely identifies that row i.e. The column which we will make the PRIMARY key , that column values must be **unique** for each row .

Every Table will have only **1 PRIMARY KEY** , and it should be **NOT NULL**.

## Foreign Key:

FOREIGN KEY is a column or set of columns in a table that refers to the PRIMARY KEY of another Table.

FOREIGN KEY **can have duplicate & NULL values**.

table1 - Student				table2 - City	
id (PK)	name	cityId	city	id (PK)	city_name
101	karan	1	Pune	1	Pune
102	arjun	2	Mumbai	2	Mumbai
103	ram	1	Pune	3	Delhi
104	shyam	3	Delhi		

As we can see there are duplicate values of foreign keys , and PRIMARY key of **City** table is FOREIGN KEY of the **Student** Table.

So **PRIMARY KEY must be only 1 , but FOREIGN keys can be MANY** .

## Other Constraints:-

SQL constraints are used to specify rules for data in a table . like PRIMARY KEY is a constraint with rule unique and it should be only one in a table .

# NOT NULL :- ( columns cannot have null value )

## **Column\_name INT NOT NULL**

So here we ust store values , this cant be empty .

# UNIQUE:- ( all values in columns are different )

## **Column\_name INT UNIQUE**

Here the values we should enter , that must be unique .

Example ,

The screenshot shows a MySQL Workbench interface. On the left, a code editor displays the following SQL statements:

```
38 • CREATE TABLE temp1(
39     id INT UNIQUE
40 );
41 • INSERT INTO temp1 VALUES(101);
42 ✘ INSERT INTO temp1VALUES(101);
```

On the right, an "Output" window titled "Action Output" shows the execution results:

#	Time	Action	Message	Duration / Fetch
26	15:39:00	CREATE TABLE employee(id INT PRIMARY KEY, ...)	0 row(s) affected	0.078 sec
27	15:39:05	INSERT INTO employee VALUES(1, "adam", 2500...)	3 row(s) affected Records: 3 Duplicates: 0 Wamin...	0.046 sec
28	15:39:08	SELECT * FROM employee LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
29	22:00:42	CREATE TABLE temp1(id INT UNIQUE )	0 row(s) affected	0.015 sec
30	22:01:07	INSERT INTO temp1 VALUES(101)	1 row(s) affected	0.031 sec
31	22:01:27	INSERT INTO temp1VALUES(101)	Error Code: 1064. You have an error in your SQL sy...	0.000 sec

We created table **temp1** and inserted 101 value in **id** which was defined with constraint **unique** , so while inserting same value next time we got error .

Hence The error came because we have defined with unique constraint .

# Primary Key :- ( UNIQUE and NOT NULL )

But the PRIMARY KEY constraint makes both **UNIQUE** and also **NOT NULL**.

```

CREATE TABLE temp1 (
    id INT,
    name VARCHAR(50),
    age INT,
    city VARCHAR(20),
    PRIMARY KEY (id)
);

```

This is another syntax for PRIMARY KEY .

We can also write in combination ,

```

CREATE TABLE temp1 (
    id INT,
    name VARCHAR(50),
    age INT,
    city VARCHAR(20),
    PRIMARY KEY (id, name)
);

```

So the combination of **id** and **name** should be UNIQUE and NOT NULL .

Here the **id** of another can be the same , or the **name** of another row can be the same . But , the combination should be UNIQUE .

Example ,

<b>id</b>	<b>name</b>
101	Shyam
101	Ram
102	Ram
103	

Here as we can see , more than one **id** is the same, more than one **name** is also the same . BUt combination is UNIQUE .

**So we can also make PRIMARY KEY of two columns .**

# FOREIGN KEY:-

**Constraints**

**FOREIGN KEY** prevent actions that would destroy links between tables

```
CREATE TABLE temp (
    cust_id int,
    FOREIGN KEY (cust_id) REFERENCES customer(id)
);
```

**DEFAULT** sets the default value of a column

```
salary INT DEFAULT 25000
```

**Cust\_id** is the normal column in that **temp** Table. Now this is the table **temp** , and one table is been already created named **customer** , and customer has a column **id** which is PRIMARY KEY . and we know that will store the same PRIMARY key in **cust\_id** i.e. column of **temp1**.

For that we have declared **cust\_id** as FOREIGN KEY by ,  
**FOREIGN KEY (cust\_id) references customer(id)**

Where , **customer** is our already existing table , and **id** is the PRIMARY key of **customer** table , and we are making **cust\_id** as FOREIGN KEY .

The **Cust\_id** column is from the new table , and the **id** column is from the old existing table in which our FOREIGN KEY is the PRIMARY KEY . and in between we write **references** i.e. **cust\_id** column is referring to **customer(id)** column.

## DEFAULT :-

It sets default value of an Column .

E.x. **salary INT DEFAULT 25000**

Lets create an table **emp** , and lets add values in it

```

45 • CREATE TABLE emp(id INT , salary INT DEFAULT 25000);
46 • INSERT INTO emp (id) VALUES(101);
47 • SELECT * FROM emp;

```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor with the following SQL statements:

```

CREATE TABLE emp(id INT , salary INT DEFAULT 25000);
INSERT INTO emp (id) VALUES(101);
SELECT * FROM emp;

```

Below the code editor is a result grid showing the data in the 'emp' table:

	id	salary
▶	101	25000

On the right side of the interface, there is a sidebar with a 'Result Grid' tab selected. Below it, an 'Action Output' section displays the following log entries:

#	Time	Action	Message	Duration / Fetch
1	14:39:16	INSERT INTO emp (id) VALUES(101)	Error Code: 1046. No database selected Select the d...	0.000 sec
2	14:39:41	use XYZ	0 row(s) affected	0.000 sec
3	14:39:53	desc XYZ	Error Code: 1146. Table 'xyz.xyz' doesn't exist	0.000 sec
4	14:40:04	desc emp	2 row(s) returned	0.047 sec / 0.000 sec
5	14:48:30	INSERT INTO emp (id) VALUES(101)	1 row(s) affected	0.016 sec
6	14:48:53	SELECT * FROM emp LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Now here we have just added **id** , still **salary** got inserted with default amount 25000. Because we have defined a **DEFAULT** constraint with 25000.

## CHECK :-

By using CHECK constraint we can limit values

**CHECK** it can limit the values allowed in a column

```

CREATE TABLE city (
    id INT PRIMARY KEY,
    city VARCHAR(50),
    age INT,
    CONSTRAINT age_check CHECK (age >= 18 AND city="Delhi")
);

```

```

CREATE TABLE newTab (
    age INT CHECK (age >= 18)
);

```

Now here we have defined **city** table , **age\_check** is the name of CONSTRAINT where and , in CHECK we have given condition .

SO now will not able to add any row in which **age>=18** and **city="Delhi"**

Lets create an sample database ,

The screenshot shows the MySQL Workbench interface. On the left, a tooltip says "Create this sample table". On the right, a tooltip says "Insert this data". The code area contains two snippets of SQL:

```
CREATE DATABASE college;
USE college;

CREATE TABLE student (
    rollno INT PRIMARY KEY,
    name VARCHAR(50),
    marks INT NOT NULL,
    grade VARCHAR(1),
    city VARCHAR(20)
);
```

```
INSERT INTO student
(rollno, name, marks, grade, city)
VALUES
(101, "anil", 78, "C", "Pune"),
(102, "bhumika", 93, "A", "Mumbai"),
(103, "chetan", 85, "B", "Mumbai"),
(104, "dhruv", 96, "A", "Delhi"),
(105, "emanuel", 12, "F", "Delhi"),
(106, "farah", 82, "B", "Delhi");
```

Code :-

```
48 •  DROP DATABASE college;
49 •  CREATE DATABASE college;
50 •  USE college;
51 •  CREATE TABLE student( rollno INT PRIMARY KEY, name VARCHAR(50),marks INT NOT NULL,
52                               grade VARCHAR(1), city VARCHAR(20));
53 •  INSERT INTO student(rollno, name, marks, grade, city) VALUES (101, "anil", 78, "C", "Pune"),
54             (102, "bhumika", 93, "A", "Mumbai"),(103,"chetan",85,"B","Mumbai"),(104, "dhruv", 96,"A","Delhi"),
55             (105, "emanuel",12,"F","Delhi"),(106,"farah",82,"B","Delhi");
56
57
```

Output

Action Output			
#	Time	Action	Message
✖ 8	15:00:02	CREATE DATABASE college	Error Code: 1007. Can't create database 'college': data...
✓ 9	15:26:34	DROP DATABASE college	1 row(s) affected
✓ 10	15:26:52	CREATE DATABASE college	1 row(s) affected
✓ 11	15:27:00	USE college	0 row(s) affected
✓ 12	15:37:44	CREATE TABLE student(rollno INT PRIMARY KEY, ...)	0 row(s) affected
✓ 13	15:37:51	INSERT INTO student(rollno, name, marks, grade, city...)	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 0.016 sec

# Select in Detail

## Select in Detail

used to select any data from the database

### Basic Syntax

`SELECT col1, col2 FROM table_name;`

### To Select ALL

`SELECT * FROM table_name;`

Example,

Suppose we want to see **name** and **marks** of **student** table .

The screenshot shows the MySQL Workbench interface. At the top, there are two numbered statements: 56 (a comment) and 57 (the query). Below them is the 'Result Grid' showing the data from the 'student' table. The 'Action Output' section at the bottom lists the history of database actions, including the creation of the 'college' database and the 'student' table, along with the execution of the current query.

#	Time	Action	Message	Duration / Fetch
9	15:26:34	DROP DATABASE college	1 row(s) affected	0.125 sec
10	15:26:52	CREATE DATABASE college	1 row(s) affected	0.000 sec
11	15:27:00	USE college	0 row(s) affected	0.000 sec
12	15:37:44	CREATE TABLE student(rollno INT PRIMARY KEY, ...)	0 row(s) affected	0.078 sec
13	15:37:51	INSERT INTO student(rollno, name, marks, grade, city...)	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0	0.016 sec
14	15:41:55	SELECT name , marks FROM student LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec

Now lets display all data i.e. `SELECT * FROM students ;`

57 • `SELECT * FROM student;`

58

rollno	name	marks	grade	city
101	anil	78	C	Pune
102	bhumika	93	A	Mumbai
103	chetan	85	B	Mumbai
104	dhruv	96	A	Delhi
105	emanuel	12	F	Delhi

student 4 ×

Output :

Action Output	#	Time	Action	Message	Duration / Fetch
10 15:26:52 CREATE DATABASE college				1 row(s) affected	0.000 sec
11 15:27:00 USE college				0 row(s) affected	0.000 sec
12 15:37:44 CREATE TABLE student(rollno INT PRIMARY KEY, ...)				0 row(s) affected	0.078 sec
13 15:37:51 INSERT INTO student(rollno, name, marks, grade, city...)				6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0	0.016 sec
14 15:41:55 SELECT name , marks FROM student LIMIT 0, 1000				6 row(s) returned	0.000 sec / 0.000 sec
15 15:49:14 SELECT * FROM student LIMIT 0, 1000				6 row(s) returned	0.000 sec / 0.000 sec

Suppose we want to display unique values of the table from any particular row, For that we can use our special keyword i.e. DISTINCT.( this means unique i.e. duplicate values wont display , only unique values will display )

Example,

`SELECT DISTINCT city FROM student;`

58 • `SELECT DISTINCT city FROM student;`

city
Pune
Mumbai
Delhi

student 5 ×

Output :

Action Output	#	Time	Action	Message	Duration / Fetch
17 09:29:24 use XYZ SELECT DISTINCT city FROM students				Error Code: 1064. You have an error in your SQL syntax...	0.000 sec
18 09:29:34 use XYZ				0 row(s) affected	0.000 sec
19 09:29:36 SELECT DISTINCT city FROM students LIMIT 0, 1000				Error Code: 1146. Table 'xyz.students' doesn't exist	0.000 sec
20 09:30:24 SELECT DISTINCT city FROM student LIMIT 0, 1000				Error Code: 1146. Table 'xyz.student' doesn't exist	0.015 sec
21 09:30:44 use college				0 row(s) affected	0.000 sec
22 09:30:47 SELECT DISTINCT city FROM student LIMIT 0, 1000				3 row(s) returned	0.000 sec / 0.000 sec

Now as we can see , only unique values has been displayed , NO duplicate values i.e. city names.

SO whenever we want a data , where we want NO repetition , there we can use DISTINCT keyword .

# Where Clause

With SELECT we use various clauses , Clause means like a legal document thing where clauses means conditions i.e. The Rules we define while displaying data.

# Where Clause

To define some conditions

```
SELECT col1, col2 FROM table_name  
WHERE conditions;
```

**SELECT \* FROM student WHERE marks > 80;**  
**SELECT \* FROM student WHERE city = "Mumbai";**

As we can see here in example ,

1. SELECT \* FROM student WHERE marks > 80;

```
59 •     SELECT *
60      FROM student
61      WHERE marks > 80;
```

So here all the students whose marks are greater than 80 has been displayed .

2. SELECT \* FROM student WHERE city = "Mumbai":

```

62 • SELECT *
63   FROM student
64   WHERE city = "Mumbai";

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid

	rollno	name	marks	grade	city
▶	102	bhumika	93	A	Mumbai
▶	103	chetan	85	B	Mumbai
*	NULL	NULL	NULL	NULL	NULL

student 7 × | Apply

Output

Action Output

#	Time	Action	Message	Duration / Fetch
21	09:30:44	use college	0 row(s) affected	0.000 sec
22	09:30:47	SELECT DISTINCT city FROM student LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
23	10:58:17	SELECT * FROM student WHERE marks > 80 LIMIT...	4 row(s) returned	0.000 sec / 0.000 sec
24	11:00:12	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.016 sec
25	11:00:35	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.000 sec
26	11:00:50	SELECT * FROM student WHERE city = "Mumbai" Li...	2 row(s) returned	0.000 sec / 0.000 sec

Here students with city MUMBAI has been displayed .

We can also write combine condition like , **marks>80 AND city="Mumbai"**

```

65 • SELECT *
66   FROM student
67   WHERE marks >80 AND city = "Mumbai";

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid

	rollno	name	marks	grade	city
▶	102	bhumika	93	A	Mumbai
▶	103	chetan	85	B	Mumbai
*	NULL	NULL	NULL	NULL	NULL

student 8 × | Apply

Output

Action Output

#	Time	Action	Message	Duration / Fetch
22	09:30:47	SELECT DISTINCT city FROM student LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
23	10:58:17	SELECT * FROM student WHERE marks > 80 LIMIT...	4 row(s) returned	0.000 sec / 0.000 sec
24	11:00:12	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.016 sec
25	11:00:35	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.000 sec
26	11:00:50	SELECT * FROM student WHERE city = "Mumbai" Li...	2 row(s) returned	0.000 sec / 0.000 sec
27	11:03:38	SELECT * FROM student WHERE marks >80 AND cit...	2 row(s) returned	0.000 sec / 0.000 sec

SO here we combined our condition using **AND** operator , like this we also have other operators we can use with WHERE clause .

## Using Operators in WHERE

**Arithmetic Operators :** +(addition) , -(subtraction), \*(multiplication), /(division), %(modulus)

**Comparison Operators :** = (equal to), != (not equal to), > , >=, <, <=

**Logical Operators :** AND, OR , NOT, IN, BETWEEN, ALL, LIKE, ANY

**Bitwise Operators :** & (Bitwise AND), | (Bitwise OR)

Example ,

1. Suppose we want to display the students whose **marks** will go above 100 if we add 10 in it .  
i.e. WHERE **marks+10 > 100**

The screenshot shows the MySQL Workbench interface. At the top, a query window displays the SQL command: `68 • SELECT * FROM student WHERE marks+10 > 100;`. Below the query window is the 'Result Grid' pane, which contains a table with columns: rollno, name, marks, grade, and city. The data shows two rows: one for bhumika (rollno 102, marks 93, grade A, city Mumbai) and one for dhruv (rollno 104, marks 96, grade A, city Delhi). The bottom pane, titled 'student 9', shows the 'Action Output' log. It lists several actions, including successful queries like 'SELECT \* FROM student WHERE marks > 80 LIMIT...' and failed queries like 'SELECT \* FROM students WHERE city = "Mumbai" ...'. The log includes details such as time, message, and duration.

rollno	name	marks	grade	city
102	bhumika	93	A	Mumbai
104	dhruv	96	A	Delhi
NULL	NULL	NULL	NULL	NULL

#	Time	Action	Message	Duration / Fetch
23	10:58:17	SELECT * FROM student WHERE marks > 80 LIMIT...	4 row(s) returned	0.000 sec / 0.000 sec
24	11:00:12	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.016 sec
25	11:00:35	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.000 sec
26	11:00:50	SELECT * FROM student WHERE city = "Mumbai" Li...	2 row(s) returned	0.000 sec / 0.000 sec
27	11:03:38	SELECT * FROM student WHERE marks >80 AND cit...	2 row(s) returned	0.000 sec / 0.000 sec
28	12:46:02	SELECT * FROM student WHERE marks+10 >100 Li...	2 row(s) returned	0.016 sec / 0.000 sec

Hence we got list of students whose marks can go above 100 , if we add 10 in it.

2. Suppose we want to check the students whose marks is equal to 93

69 • `SELECT * FROM student WHERE marks = 93;`

rollno	name	marks	grade	city
102	bhumika	93	A	Mumbai
NULL	NULL	NULL	NULL	NULL

student 10 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
24	11:00:12	<code>SELECT * FROM students WHERE city = "Mumbai" ...</code>	Error Code: 1146. Table 'college.students' doesn't exist	0.016 sec
25	11:00:35	<code>SELECT * FROM students WHERE city = "Mumbai" ...</code>	Error Code: 1146. Table 'college.students' doesn't exist	0.000 sec
26	11:00:50	<code>SELECT * FROM student WHERE city = "Mumbai" LI...</code>	2 row(s) returned	0.000 sec / 0.000 sec
27	11:03:38	<code>SELECT * FROM student WHERE marks &gt;80 AND cit...</code>	2 row(s) returned	0.000 sec / 0.000 sec
28	12:46:02	<code>SELECT * FROM student WHERE marks+10 &gt;100 LI...</code>	2 row(s) returned	0.016 sec / 0.000 sec
29	14:09:58	<code>SELECT * FROM student WHERE marks = 93 LIMIT ...</code>	1 row(s) returned	0.000 sec / 0.000 sec

Here we got a students with marks 93 .

Now lets learn **logical operators** in detail ,

**AND** (to check for both conditions to be true)

```
SELECT * FROM student WHERE marks > 80 AND city = "Mumbai";
```

**OR** (to check for one of the conditions to be true)

```
SELECT * FROM student WHERE marks > 90 OR city = "Mumbai";
```

**Between** (selects for a given range)

```
SELECT * FROM student WHERE marks BETWEEN 80 AND 90;
```

**In** (matches any value in the list)

```
SELECT * FROM student WHERE city IN ("Delhi", "Mumbai");
```

**NOT** (to negate the given condition)

```
SELECT * FROM student WHERE city NOT IN ("Delhi", "Mumba
```

Here **IN** will give the list of students who belongs to one of the city gie=vn in list , and **NOT IN** will give the results of the students who does NOT belongs to any of the cities given in the list .

## Limit Clause

Sets an upper limit on number of (tuples)rows to be returned

```
SELECT * FROM student LIMIT 3;
```

```
SELECT col1, col2 FROM table_name  
LIMIT number;
```

Suppose we want just 5 students of data , or 2 students data . then in such cases we can use **LIMIT** to set an upper limit on number of rows to be returned .

Example,

```
SELECT * FROM student LIMIT 3;
```

The screenshot shows the MySQL Workbench interface. In the SQL tab, the query `SELECT * FROM student LIMIT 3;` is entered. In the Results tab, the output is a table named 'Result Grid' containing three rows of student data:

rollno	name	marks	grade	city
101	anil	78	C	Pune
102	bhumika	93	A	Mumbai
103	chetan	85	B	Mumbai

Below the table, the 'Output' section displays the execution log:

#	Time	Action	Message	Duration / Fetch
25	11:00:35	SELECT * FROM students WHERE city = "Mumbai" ...	Error Code: 1146. Table 'college.students' doesn't exist	0.000 sec
26	11:00:50	SELECT * FROM student WHERE city = "Mumbai" Li...	2 row(s) returned	0.000 sec / 0.000 sec
27	11:03:38	SELECT * FROM student WHERE marks >80 AND cit...	2 row(s) returned	0.000 sec / 0.000 sec
28	12:46:02	SELECT * FROM student WHERE marks+10 >100 Li...	2 row(s) returned	0.016 sec / 0.000 sec
29	14:09:58	SELECT * FROM student WHERE marks = 93 LIMIT ...	1 row(s) returned	0.000 sec / 0.000 sec
30	14:23:09	SELECT * FROM student LIMIT 3	3 row(s) returned	0.000 sec / 0.000 sec

Now here is the data of the first 3 students .

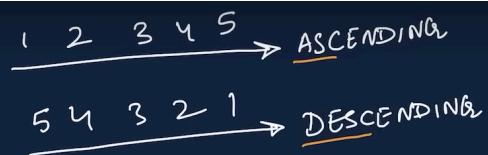
## ORDER BY Clause

In some conditions we want our data in either ascending or descending order , then in such conditions we use ORDER BY clause .( By default ORDER BY is in ascending order i.e. we dont need to specify ascending )

## Order By Clause

To sort in ascending (ASC) or descending order (DESC)

```
SELECT * FROM student  
ORDER BY city ASC;
```



```
SELECT col1, col2 FROM table_name  
ORDER BY col_name(s) ASC;
```

Example ,

Lets display students by their marks order in ascending .

```
SELECT *  
FROM student  
ORDER BY marks ASC
```

The screenshot shows a MySQL query interface. The query entered is:

```
71 • SELECT * FROM student ORDER BY marks ASC;
```

The results grid displays the following data:

rollno	name	marks	grade	city
105	emanuel	12	F	Delhi
101	anil	78	C	Pune
106	farah	82	B	Delhi
103	chetan	85	B	Mumbai
102	bhumika	93	A	Mumbai

The output pane shows the history of queries run on the database:

#	Time	Action	Message	Duration / Fetch
26	11:00:50	SELECT * FROM student WHERE city = "Mumbai" Li...	2 row(s) returned	0.000 sec / 0.000 sec
27	11:03:38	SELECT * FROM student WHERE marks >80 AND cit...	2 row(s) returned	0.000 sec / 0.000 sec
28	12:46:02	SELECT * FROM student WHERE marks+10 >100 Li...	2 row(s) returned	0.016 sec / 0.000 sec
29	14:09:58	SELECT * FROM student WHERE marks = 93 LIMIT ...	1 row(s) returned	0.000 sec / 0.000 sec
30	14:23:09	SELECT * FROM student LIMIT 3	3 row(s) returned	0.000 sec / 0.000 sec
31	14:30:39	SELECT * FROM student ORDER BY marks ASC LIM...	6 row(s) returned	0.000 sec / 0.000 sec

Now suppose we want Top 3 students data in the class .

Now for that will arrange the data in descending order , and in this will put LIMIT of 3 for top 3 students .

i.e.    SELECT \*  
        FROM student  
        ORDER BY marks DESC  
        LIMIT 3

```

72 •   SELECT *
73     FROM student
74     ORDER BY marks DESC
75     LIMIT 3;

```

**Result Grid**

	rollno	name	marks	grade	city
▶	104	dhruv	96	A	Delhi
▶	102	bhumika	93	A	Mumbai
▶	103	chetan	85	B	Mumbai
◀	NONE	NONE	NONE	NONE	NONE

**student 13** × **Apply**

**Output**

#	Time	Action	Message	Duration / Fetch
27	11:03:38	SELECT * FROM student WHERE marks >80 AND cit...	2 row(s) returned	0.000 sec / 0.000 sec
28	12:46:02	SELECT * FROM student WHERE marks+10 >100 LI...	2 row(s) returned	0.016 sec / 0.000 sec
29	14:09:58	SELECT * FROM student WHERE marks = 93 LIMIT ...	1 row(s) returned	0.000 sec / 0.000 sec
30	14:23:09	SELECT * FROM student LIMIT 3	3 row(s) returned	0.000 sec / 0.000 sec
31	14:30:39	SELECT * FROM student ORDER BY marks ASC LIM...	6 row(s) returned	0.000 sec / 0.000 sec
32	14:35:53	SELECT * FROM student ORDER BY marks DESC ...	3 row(s) returned	0.000 sec / 0.000 sec

We got top 3 students in class .

## Aggregate Functions:-

Functions are basically a structure with logic , and it perform the operations and gives us output.

Aggregate means combining things together , So aggregate functions take i/p more than 1 , and return a single value .

Here are some popular aggregate functions

**Aggregate functions perform a calculation on a set of values, and return a single value.**

- COUNT()
  - MAX()
  - MIN()
  - SUM()
  - AVG()
- Get Maximum Marks**
- ```
SELECT max(marks)
FROM student;
```

**Get Average marks**

```
SELECT avg(marks)
FROM student;
```

Example ,

Suppose we want a check **maximum marks** in student list ,

The screenshot shows the MySQL Workbench interface. In the SQL editor, two statements are run:

```
76 • SELECT MAX(marks)
77 FROM student;
```

The results are displayed in a Result Grid:

| MAX(marks) |
|------------|
| 96         |

Below the results, the Action Output window shows the history of database actions:

| #  | Time     | Action                                           | Message           | Duration / Fetch      |
|----|----------|--------------------------------------------------|-------------------|-----------------------|
| 28 | 12:46:02 | SELECT * FROM student WHERE marks+10 > 100 Li... | 2 row(s) returned | 0.016 sec / 0.000 sec |
| 29 | 14:09:58 | SELECT * FROM student WHERE marks = 93 LIMIT ... | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 30 | 14:23:09 | SELECT * FROM student LIMIT 3                    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 31 | 14:30:39 | SELECT * FROM student ORDER BY marks ASC LIM...  | 6 row(s) returned | 0.000 sec / 0.000 sec |
| 32 | 14:35:53 | SELECT * FROM student ORDER BY marks DESC ...    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 33 | 14:43:17 | SELECT MAX(marks) FROM student LIMIT 0, 1000     | 1 row(s) returned | 0.000 sec / 0.000 sec |

Similarly we can check **minimum marks** ,

The screenshot shows the MySQL Workbench interface. In the SQL editor, two statements are run:

```
76 • SELECT MIN(marks)
77 FROM student;
```

The results are displayed in a Result Grid:

| MIN(marks) |
|------------|
| 12         |

Below the results, the Action Output window shows the history of database actions:

| #  | Time     | Action                                           | Message           | Duration / Fetch      |
|----|----------|--------------------------------------------------|-------------------|-----------------------|
| 29 | 14:09:58 | SELECT * FROM student WHERE marks = 93 LIMIT ... | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 30 | 14:23:09 | SELECT * FROM student LIMIT 3                    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 31 | 14:30:39 | SELECT * FROM student ORDER BY marks ASC LIM...  | 6 row(s) returned | 0.000 sec / 0.000 sec |
| 32 | 14:35:53 | SELECT * FROM student ORDER BY marks DESC ...    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 33 | 14:43:17 | SELECT MAX(marks) FROM student LIMIT 0, 1000     | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 34 | 14:44:37 | SELECT MIN(marks) FROM student LIMIT 0, 1000     | 1 row(s) returned | 0.000 sec / 0.000 sec |

## Average of marks

```
76 •   SELECT AVG(marks)
77     FROM student;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL query:

```
76 •   SELECT AVG(marks)
77     FROM student;
```

Below the code editor is a results grid titled "Result Grid". The grid has one row with the following data:

| AVG(marks) |
|------------|
| 74.3333    |

On the right side of the results grid, there is a vertical toolbar with a "Result Grid" icon. Below the results grid, there is a "Result 16" tab and an "Output" section containing an "Action Output" table. The table has columns: #, Time, Action, Message, and Duration / Fetch. It lists the following actions:

| #  | Time     | Action                                          | Message           | Duration / Fetch      |
|----|----------|-------------------------------------------------|-------------------|-----------------------|
| 30 | 14:23:09 | SELECT * FROM student LIMIT 3                   | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 31 | 14:30:39 | SELECT * FROM student ORDER BY marks ASC LIM... | 6 row(s) returned | 0.000 sec / 0.000 sec |
| 32 | 14:35:53 | SELECT * FROM student ORDER BY marks DESC ...   | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 33 | 14:43:17 | SELECT MAX(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 34 | 14:44:37 | SELECT MIN(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 35 | 14:45:32 | SELECT AVG(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |

At the bottom right of the interface, there is a "Read Only" status indicator.

Here we got the class average ...

Now Suppose we want to count number of students ,

```
76 •   SELECT COUNT(rollno)
77     FROM student;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL query:

```
76 •   SELECT COUNT(rollno)
77     FROM student;
```

Below the code editor is a results grid titled "Result Grid". The grid has one row with the following data:

| COUNT(rollno) |
|---------------|
| 6             |

On the right side of the results grid, there is a vertical toolbar with a "Result Grid" icon. Below the results grid, there is a "Result 17" tab and an "Output" section containing an "Action Output" table. The table has columns: #, Time, Action, Message, and Duration / Fetch. It lists the following actions:

| #  | Time     | Action                                          | Message           | Duration / Fetch      |
|----|----------|-------------------------------------------------|-------------------|-----------------------|
| 31 | 14:30:39 | SELECT * FROM student ORDER BY marks ASC LIM... | 6 row(s) returned | 0.000 sec / 0.000 sec |
| 32 | 14:35:53 | SELECT * FROM student ORDER BY marks DESC ...   | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 33 | 14:43:17 | SELECT MAX(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 34 | 14:44:37 | SELECT MIN(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 35 | 14:45:32 | SELECT AVG(marks) FROM student LIMIT 0, 1000    | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 36 | 14:49:12 | SELECT COUNT(rollno) FROM student LIMIT 0, 1000 | 1 row(s) returned | 0.031 sec / 0.000 sec |

At the bottom right of the interface, there is a "Read Only" status indicator.

Hence we can see , we counted rollno to find number of students , and its 6 .

# GROUP BY Clause :-

Groups rows that have the same values into summary rows.

It collects data from multiple records and groups the result by one or more column.

\*Generally we use group by with some *aggregation function*.

Count number of students in each city

```
SELECT city, count(name)
FROM student
GROUP BY city;
```

The screenshot shows a MySQL Workbench session with the following details:

Query Editor (SQL):

```
78 •    SELECT city ,count(name)
79      FROM student
80     GROUP BY city;
```

Result Grid (Output):

| city   | count(name) |
|--------|-------------|
| Pune   | 1           |
| Mumbai | 2           |
| Delhi  | 3           |

Action Output (Logs):

| #  | Time     | Action                                            | Message           | Duration / Fetch      |
|----|----------|---------------------------------------------------|-------------------|-----------------------|
| 32 | 14:35:53 | SELECT * FROM student ORDER BY marks DESC ...     | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 33 | 14:43:17 | SELECT MAX(marks) FROM student LIMIT 0, 1000      | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 34 | 14:44:37 | SELECT MIN(marks) FROM student LIMIT 0, 1000      | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 35 | 14:45:32 | SELECT AVG(marks) FROM student LIMIT 0, 1000      | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 36 | 14:49:12 | SELECT COUNT(rollno) FROM student LIMIT 0, 1000   | 1 row(s) returned | 0.031 sec / 0.000 sec |
| 37 | 14:56:23 | SELECT city ,count(name) FROM student GROUP BY... | 3 row(s) returned | 0.000 sec / 0.000 sec |

Here GROUP BY created group on the basis of cities , and from there we counted Names of all students who are available in the particular city .

NOTE:- No. of columns which we going to display using SELECT , same columns we are going to use in GROUP BY clause . otherwise it will give error .

I.e. We selected **city** and we made a group of **city** .

# Practice Qs



To exit full screen, press Esc

Write the Query to find avg marks in each city in ascending order.

```
81 •  SELECT city, AVG(marks)
82   FROM student
83   GROUP BY city
84   ORDER BY city ASC;
```

| city   | AVG(marks) |
|--------|------------|
| Delhi  | 63.3333    |
| Mumbai | 89.0000    |
| Pune   | 78.0000    |

Result 19 x

Read Only

Output ::

Action Output

| #  | Time     | Action                                            | Message                                                   | Duration / Fetch      |
|----|----------|---------------------------------------------------|-----------------------------------------------------------|-----------------------|
| 34 | 14:44:37 | SELECT MIN(marks) FROM student LIMIT 0, 1000      | 1 row(s) returned                                         | 0.000 sec / 0.000 sec |
| 35 | 14:45:32 | SELECT AVG(marks) FROM student LIMIT 0, 1000      | 1 row(s) returned                                         | 0.000 sec / 0.000 sec |
| 36 | 14:49:12 | SELECT COUNT(rollno) FROM student LIMIT 0, 1000   | 1 row(s) returned                                         | 0.031 sec / 0.000 sec |
| 37 | 14:56:23 | SELECT city ,count(name) FROM student GROUP BY... | 3 row(s) returned                                         | 0.000 sec / 0.000 sec |
| 38 | 15:06:03 | SELECT city, AVG(marks) FROM student GROUP BY ... | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| 39 | 15:07:29 | SELECT city, AVG(marks) FROM student GROUP BY ... | 3 row(s) returned                                         | 0.015 sec / 0.000 sec |

# Practice Qs



For the given table, find the total payment according to each payment method.

| customer_id | customer          | mode        | city        |
|-------------|-------------------|-------------|-------------|
| 101         | Olivia Barrett    | Netbanking  | Portland    |
| 102         | Ethan Sinclair    | Credit Card | Miami       |
| 103         | Maya Hernandez    | Credit Card | Seattle     |
| 104         | Liam Donovan      | Netbanking  | Denver      |
| 105         | Sophia Nguyen     | Credit Card | New Orleans |
| 106         | Caleb Foster      | Debit Card  | Minneapolis |
| 107         | Ava Patel         | Debit Card  | Phoenix     |
| 108         | Lucas Carter      | Netbanking  | Boston      |
| 109         | Isabella Martinez | Netbanking  | Nashville   |
| 110         | Jackson Brooks    | Credit Card | Boston      |

Lets consider the name of the table is **payment** .

```
SELECT mode, count(customer)
FROM payment
GROUP BY mode;
```

Now here it will display the data of customers according to their each payment mode .

---

Example,

Suppose we want to count the student on the basis of their grade  
I.e. how many students got A grade , How many got B , and so on ..  
And lets also order grades it in ascending order..

```
SELECT grade, count(rollno)
FROM student
GROUP BY grade
ORDER BY grade ASC;
```

SO here we have counted student **rollno**, we can also count **name**  
It doesn't matter , we just have to count the students .  
Since **rollno** is unique , so we mostly prefer that for counting .

The screenshot shows a MySQL query editor interface. At the top, there is a code editor with the following SQL query:

```
88 • USE college;
89 • SELECT grade, COUNT(rollno)
90 FROM student
91 GROUP BY grade
92 ORDER BY grade ASC;
```

Below the code editor is a result grid titled "Result Grid". The grid displays the following data:

| grade | COUNT(rollno) |
|-------|---------------|
| A     | 2             |
| B     | 2             |
| C     | 1             |
| F     | 1             |

At the bottom of the interface, there is an "Output" section showing the execution log:

| # | Time     | Action                                             | Message                                                   | Duration / Fetch      |
|---|----------|----------------------------------------------------|-----------------------------------------------------------|-----------------------|
| 1 | 11:47:25 | SELECT grade, COUNT(rollno) FROM student GROUP ... | Error Code: 1046. No database selected Select the defa... | 0.000 sec             |
| 2 | 11:55:21 | USE college                                        | 0 row(s) affected                                         | 0.000 sec             |
| 3 | 11:55:24 | SELECT grade, COUNT(rollno) FROM student GROUP ... | 4 row(s) returned                                         | 0.031 sec / 0.000 sec |

# HAVING Clause

HAVING clause is also used for defining the conditions same like WHERE clause.

Similar to Where i.e. applies some condition on rows.

Used when we want to apply any condition after grouping.

Count number of students in each city where max marks cross 90.

```
SELECT count(name), city
FROM student
GROUP BY city
HAVING max(marks) > 90;
```

Sometimes we have situation where we have to write command in SQL where we can't write WHERE clause , WHERE is used for to apply conditions on **normal rows** , HAVING clause is used to condition on **GROUPS** .

```
93 •   SELECT count(rollno),city
94     FROM student
95     GROUP BY city
96     HAVING max(marks)>90;
```

The screenshot shows the execution of a SQL query in a database environment. The query is:

```
93 •   SELECT count(rollno),city
94     FROM student
95     GROUP BY city
96     HAVING max(marks)>90;
```

The results are displayed in a grid:

| count(rollno) | city   |
|---------------|--------|
| 2             | Mumbai |
| 3             | Delhi  |

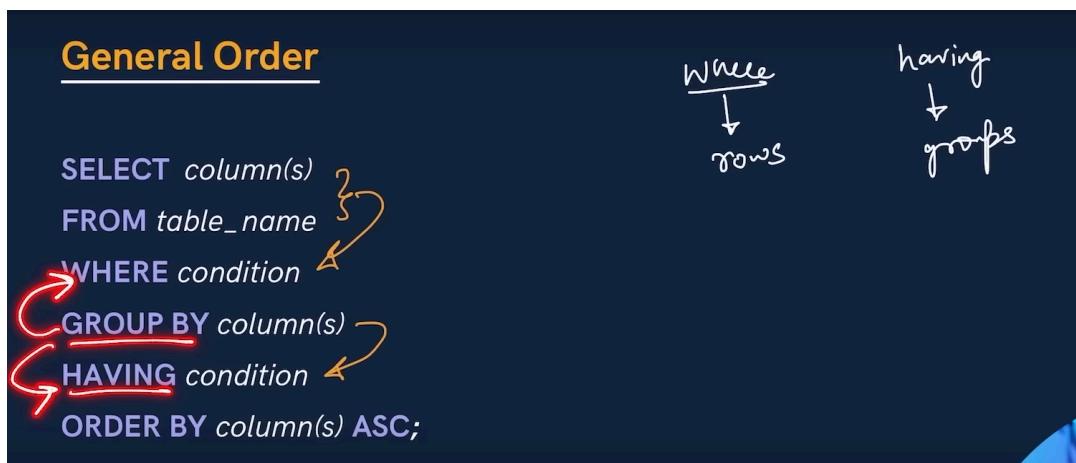
Below the results, there is a log of actions:

| # | Time     | Action                                                | Message                                                   | Duration / Fetch      |
|---|----------|-------------------------------------------------------|-----------------------------------------------------------|-----------------------|
| 1 | 11:47:25 | SELECT grade, COUNT(rollno) FROM student GROUP ...    | Error Code: 1046. No database selected Select the defa... | 0.000 sec             |
| 2 | 11:55:21 | USE college                                           | 0 row(s) affected                                         | 0.000 sec             |
| 3 | 11:55:24 | SELECT grade, COUNT(rollno) FROM student GROUP ...    | 4 row(s) returned                                         | 0.031 sec / 0.000 sec |
| 4 | 12:14:08 | SELECT count(rollno),city FROM student GROUP BY ci... | 2 row(s) returned                                         | 0.000 sec / 0.000 sec |

If we try to do the same using WHERE clause , then it won't be possible.

---

Now let's see the sequence or general order in which we can write the SQL code .



**LIMIT at last.....**

WHERE will apply condition on SELECT and FROM i.e. from which table and what to display , then HAVING will apply condition on GROUP i.e. in the group of data what is the conditions the data should satisfy , and at last will write the ORDER of data in which we want to arrange .

Example ,

Write an SQL query to retrieve the names of cities where students with grade "A" are present. The query should group the results by city and only include cities where the maximum marks obtained by any student in that city exceed 90. The output should be sorted in ascending order by city name.->

```

SELECT city
FROM student
WHERE grade="A"
GROUP BY city
HAVING max(marks)>90
ORDER BY city ASC
    
```

```

97 •  SELECT city
98   FROM student
99   WHERE grade="A"
100  GROUP BY city
101  HAVING max(marks)>90
102  ORDER BY city ASC |
    
```

| Result Grid |        | Filter Rows: | Export: | Wrap Cell Content: | Result Grid |
|-------------|--------|--------------|---------|--------------------|-------------|
|             | city   |              |         |                    |             |
| ▶           | Delhi  |              |         |                    |             |
|             | Mumbai |              |         |                    |             |

| student 3 |            | Output                                                | Read Only                                                 |                       |
|-----------|------------|-------------------------------------------------------|-----------------------------------------------------------|-----------------------|
|           |            | Action Output                                         |                                                           |                       |
| #         | Time       | Action                                                | Message                                                   | Duration / Fetch      |
| ✖         | 1 11:47:25 | SELECT grade, COUNT(rollno) FROM student GROUP ...    | Error Code: 1046. No database selected Select the defa... | 0.000 sec             |
| ✓         | 2 11:55:21 | USE college                                           | 0 row(s) affected                                         | 0.000 sec             |
| ✓         | 3 11:55:24 | SELECT grade, COUNT(rollno) FROM student GROUP ...    | 4 row(s) returned                                         | 0.031 sec / 0.000 sec |
| ✓         | 4 12:14:08 | SELECT count(rollno),city FROM student GROUP BY ci... | 2 row(s) returned                                         | 0.000 sec / 0.000 sec |
| ✓         | 5 14:00:28 | SELECT city FROM student WHERE grade="A" GROU...      | 2 row(s) returned                                         | 0.000 sec / 0.000 sec |

# Table related Queries

## Table related Queries

**Update** (to update existing rows)

```
UPDATE table_name  
SET col1 = val1, col2 = val2  
WHERE condition;
```

```
UPDATE student  
SET grade = "O"  
WHERE grade = "A";
```

Example,

earlier we were giving grades as "A", "B", "C", ... and so on, Suppose college now decided to give grades as , above 90 -> "Outstanding". I.e. we have to change "A" grade to "O" grade.

For that will use UPDATE

```
UPDATE student  
Set grade = "O"  
WHERE grade = "A";
```

```
103 • UPDATE student  
104     SET grade = "O"  
105     WHERE grade = "A";
```

Output :

| Action Output                                                                                                      | # | Time     | Action                                             | Message                                                | Duration / Fetch      |
|--------------------------------------------------------------------------------------------------------------------|---|----------|----------------------------------------------------|--------------------------------------------------------|-----------------------|
| 2 11:55:21 USE college                                                                                             | 2 | 11:55:21 | USE college                                        | 0 row(s) affected                                      | 0.000 sec             |
| 3 11:55:24 SELECT grade, COUNT(rollno) FROM student GROU...                                                        | 3 | 11:55:24 | SELECT grade, COUNT(rollno) FROM student GROU...   | 4 row(s) returned                                      | 0.031 sec / 0.000 sec |
| 4 12:14:08 SELECT count(rollno),city FROM student GROUP BY...                                                      | 4 | 12:14:08 | SELECT count(rollno),city FROM student GROUP BY... | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 5 14:00:28 SELECT city FROM student WHERE grade="A" GRO...                                                         | 5 | 14:00:28 | SELECT city FROM student WHERE grade="A" GRO...    | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 6 14:09:11 SELECT city FROM student WHERE grade="A" GRO...                                                         | 6 | 14:09:11 | SELECT city FROM student WHERE grade="A" GRO...    | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 7 14:10:17 UPDATE student SET grade = "O" WHERE grade = ... Error Code: 1175. You are using safe update mode an... | 7 | 14:10:17 | UPDATE student SET grade = "O" WHERE grade = ...   | Error Code: 1175. You are using safe update mode an... | 0.047 sec             |

Here we got error for “safe mode” i.e. SQL make sure that we wont update the data which is not necessary . because it can cause problem we change database data.

We can simply off the **sql safe mode** using the below command

## **SET SQL\_SAFE\_UPDATES=0**

Now we can update tables , because safe mode is off .

The screenshot shows the MySQL Workbench interface. In the top query editor, line 106 contains the command `SET SQL_SAFE_UPDATES=0;`. Below it, the output window displays the execution history:

| # | Time     | Action                                               | Message                                                | Duration / Fetch      |
|---|----------|------------------------------------------------------|--------------------------------------------------------|-----------------------|
| 3 | 11:55:24 | SELECT grade, COUNT(rollno) FROM student GROUP BY... | 4 row(s) returned                                      | 0.031 sec / 0.000 sec |
| 4 | 12:14:08 | SELECT count(rollno).city FROM student GROUP BY...   | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 5 | 14:00:28 | SELECT city FROM student WHERE grade="A" GRO...      | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 6 | 14:09:11 | SELECT city FROM student WHERE grade="A" GRO...      | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 7 | 14:10:17 | UPDATE student SET grade = "O" WHERE grade = ...     | Error Code: 1175. You are using safe update mode an... | 0.047 sec             |
| 8 | 14:16:24 | SET SQL_SAFE_UPDATES=0                               | 0 row(s) affected                                      | 0.000 sec             |

If we want to ON it in future , then we can use below command .

## **SET SQL\_SAFE\_UPDATES=1**

Now lets run our UPDATE query ...

The screenshot shows the MySQL Workbench interface. In the top query editor, lines 107-109 contain the UPDATE query:

```
107 • UPDATE student
108   SET grade = "O"
109   WHERE grade = "A";
```

Below it, the output window displays the execution history:

| # | Time     | Action                                             | Message                                                  | Duration / Fetch      |
|---|----------|----------------------------------------------------|----------------------------------------------------------|-----------------------|
| 4 | 12:14:08 | SELECT count(rollno).city FROM student GROUP BY... | 2 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 5 | 14:00:28 | SELECT city FROM student WHERE grade="A" GRO...    | 2 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 6 | 14:09:11 | SELECT city FROM student WHERE grade="A" GRO...    | 2 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 7 | 14:10:17 | UPDATE student SET grade = "O" WHERE grade = ...   | Error Code: 1175. You are using safe update mode an...   | 0.047 sec             |
| 8 | 14:16:24 | SET SQL_SAFE_UPDATES=0                             | 0 row(s) affected                                        | 0.000 sec             |
| 9 | 14:25:38 | UPDATE student SET grade = "O" WHERE grade = ...   | 2 row(s) affected Rows matched: 2 Changed: 2 Warnings: 0 | 0.016 sec             |

Now lets check the data using SELECT

```
110 • SELECT * FROM student;
```

**Result Grid**

|   | rollno | name    | marks | grade | city   |
|---|--------|---------|-------|-------|--------|
| ▶ | 101    | anil    | 78    | C     | Pune   |
|   | 102    | bhumika | 93    | O     | Mumbai |
|   | 103    | chetan  | 85    | B     | Mumbai |
|   | 104    | dhruv   | 96    | O     | Delhi  |
|   | 105    | emanuel | 12    | F     | Delhi  |

student 5 ×

**Action Output**

| # | Time        | Action                                           | Message                                                | Duration / Fetch      |
|---|-------------|--------------------------------------------------|--------------------------------------------------------|-----------------------|
| ✓ | 5 14:00:28  | SELECT city FROM student WHERE grade="A" GRO...  | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| ✓ | 6 14:09:11  | SELECT city FROM student WHERE grade="A" GRO...  | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| ✗ | 7 14:10:17  | UPDATE student SET grade = "O" WHERE grade = ... | Error Code: 1175. You are using safe update mode an... | 0.047 sec             |
| ✓ | 8 14:16:24  | SET SQL_SAFE_UPDATES=0                           | 0 row(s) affected                                      | 0.000 sec             |
| ✓ | 9 14:25:38  | UPDATE student SET grade = "O" WHERE grade = ... | 2 row(s) affected Rows matched: 2 Changed: 2 Warn...   | 0.016 sec             |
| ✓ | 10 14:26:29 | SELECT * FROM student LIMIT 0, 1000              | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |

As we can see “A” grade has been updated with “O” grade .

We can also update other data like this .

Example,

Suppose college have given wrong MCQ , for that college decides to increase +1 marks of all students .

## UPDATE student

**SET marks=marks+1 ;**

```
111 • UPDATE student
112     SET marks=marks+1;
113 • select * FROM student;
```

**Result Grid**

|   | rollno | name    | marks | grade | city   |
|---|--------|---------|-------|-------|--------|
| ▶ | 101    | anil    | 79    | C     | Pune   |
|   | 102    | bhumika | 94    | O     | Mumbai |
|   | 103    | chetan  | 86    | B     | Mumbai |
|   | 104    | dhruv   | 97    | O     | Delhi  |
|   | 105    | emanuel | 13    | F     | Delhi  |

student 6 ×

**Action Output**

| # | Time        | Action                                           | Message                                                 | Duration / Fetch      |
|---|-------------|--------------------------------------------------|---------------------------------------------------------|-----------------------|
| ✓ | 8 14:16:24  | SET SQL_SAFE_UPDATES=0                           | 0 row(s) affected                                       | 0.000 sec             |
| ✓ | 9 14:25:38  | UPDATE student SET grade = "O" WHERE grade = ... | 2 row(s) affected Rows matched: 2 Changed: 2 Warn...    | 0.016 sec             |
| ✓ | 10 14:26:29 | SELECT * FROM student LIMIT 0, 1000              | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| ✓ | 11 14:32:13 | UPDATE student SET marks=marks+1                 | 6 row(s) affected Rows matched: 6 Changed: 6 Warn...    | 0.000 sec             |
| ✗ | 12 14:32:23 | select * FROM studnet LIMIT 0, 1000              | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.016 sec             |
| ✓ | 13 14:32:30 | select * FROM student LIMIT 0, 1000              | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |

# Table related Queries

## Delete (to delete existing rows)

**DELETE FROM** *table\_name*  
**WHERE** *condition*;

**DELETE FROM** student  
**WHERE** marks < 33;

Suppose we want to delete the data of students whose marks are less than 33.

DELETE FROM student  
WHERE marks < 33;

In our existing table no one is less than 33 , lets UPDATE the student data with 12 marks of roll no 105.

```
114 • UPDATE student
115 SET marks=12
116 WHERE rollno=105;
```

Output

| #  | Time     | Action                                           | Message                                                        | Duration / Fetch      |
|----|----------|--------------------------------------------------|----------------------------------------------------------------|-----------------------|
| 9  | 14:25:38 | UPDATE student SET grade = "O" WHERE grade = ... | 2 row(s) affected Rows matched: 2 Changed: 2 Warn... 0.016 sec |                       |
| 10 | 14:26:29 | SELECT * FROM student LIMIT 0, 1000              | 6 row(s) returned                                              | 0.000 sec / 0.000 sec |
| 11 | 14:32:13 | UPDATE student SET marks=marks+1                 | 6 row(s) affected Rows matched: 6 Changed: 6 Warn... 0.000 sec |                       |
| 12 | 14:32:23 | select * FROM studnet LIMIT 0, 1000              | Error Code: 1146. Table 'college.studnet' doesn't exist        | 0.016 sec             |
| 13 | 14:32:30 | select * FROM student LIMIT 0, 1000              | 6 row(s) returned                                              | 0.000 sec / 0.000 sec |
| 14 | 14:35:48 | UPDATE student SET marks=12 WHERE rollno=105     | 1 row(s) affected Rows matched: 1 Changed: 1 Warn... 0.000 sec |                       |

Lets check

```
117 • select * FROM student;
```

**Result Grid**

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 105    | emanuel | 12    | F     | Delhi  |

**Action Output**

| #  | Time     | Action                                       | Message                                                 | Duration / Fetch      |
|----|----------|----------------------------------------------|---------------------------------------------------------|-----------------------|
| 10 | 14:26:29 | SELECT * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 11 | 14:32:13 | UPDATE student SET marks=marks+1             | 6 row(s) affected Rows matched: 6 Changed: 6 Warn...    | 0.000 sec             |
| 12 | 14:32:23 | select * FROM studnet LIMIT 0, 1000          | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.016 sec             |
| 13 | 14:32:30 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 14 | 14:35:48 | UPDATE student SET marks=12 WHERE rollno=105 | 1 row(s) affected Rows matched: 1 Changed: 1 Warn...    | 0.000 sec             |
| 15 | 14:37:17 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |

Now lets delete the data with less than 33 marks

```
118 • DELETE FROM student
119 WHERE marks < 33;
```

**Action Output**

| #  | Time     | Action                                       | Message                                                 | Duration / Fetch      |
|----|----------|----------------------------------------------|---------------------------------------------------------|-----------------------|
| 11 | 14:32:13 | UPDATE student SET marks=marks+1             | 6 row(s) affected Rows matched: 6 Changed: 6 Warn...    | 0.000 sec             |
| 12 | 14:32:23 | select * FROM studnet LIMIT 0, 1000          | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.016 sec             |
| 13 | 14:32:30 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 14 | 14:35:48 | UPDATE student SET marks=12 WHERE rollno=105 | 1 row(s) affected Rows matched: 1 Changed: 1 Warn...    | 0.000 sec             |
| 15 | 14:37:17 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 16 | 14:38:24 | DELETE FROM student WHERE marks < 33         | 1 row(s) affected                                       | 0.016 sec             |

Lets see

```
120 • SELECT * FROM student;
```

**Result Grid**

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 106    | farah   | 83    | B     | Delhi  |

**Action Output**

| #  | Time     | Action                                       | Message                                                 | Duration / Fetch      |
|----|----------|----------------------------------------------|---------------------------------------------------------|-----------------------|
| 12 | 14:32:23 | select * FROM studnet LIMIT 0, 1000          | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.016 sec             |
| 13 | 14:32:30 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 14 | 14:35:48 | UPDATE student SET marks=12 WHERE rollno=105 | 1 row(s) affected Rows matched: 1 Changed: 1 Warn...    | 0.000 sec             |
| 15 | 14:37:17 | select * FROM student LIMIT 0, 1000          | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 16 | 14:38:24 | DELETE FROM student WHERE marks < 33         | 1 row(s) affected                                       | 0.016 sec             |
| 17 | 14:38:54 | SELECT * FROM student LIMIT 0, 1000          | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |

Here roll no 105 has been deleted whose marks was less than 33 .

Use it carefully ,

Just **DELETE FROM student** ; can format the data of the **student** table.

---

Lets suppose we have a **department** table , in which we store the data of some departments in which we have columns **id** and **name** .

And another table **teacher** , where we are storing teachers information in which we have columns **id** , **name** and **dept\_id**.

Adam is in the Science department so its department ID is 101 , Bob is in Hindi department so its department ID is 103 , and so on ..

**Revisiting FK**

| dept      | teacher        |
|-----------|----------------|
| <b>id</b> | <b>id</b>      |
| 101       | 101            |
| 102       | 102            |
| 103       | 103            |
| name      | name           |
| Science   | Adam           |
| English   | Bob            |
| Hindi     | Casey          |
|           | Donald         |
|           | <b>dept_id</b> |
|           | 101            |
|           | 103            |
|           | 102            |
|           | 102            |

Department table is linked with teacher table , The foreign key is referencing to department table . i.e **FOREIGN KEY(dept\_id) REFERENCES dept(id)**

```

classroom* x
schemas
Filter objects
college
Tables
dept
student
teacher
Views
Stored Procedure
Functions
dbms_ex7
mca
Administration Schemas
Information
Schema: college

121 • CREATE TABLE dept(
122     id INT PRIMARY KEY,
123     name VARCHAR(50)
124 );
125
126 • CREATE TABLE teacher(
127     id INT PRIMARY KEY,
128     name VARCHAR(50),
129     dept_id INT,
130     FOREIGN KEY (dept_id) REFERENCES dept(id)
131 );
132
133
134
135
136
137
138
139
140

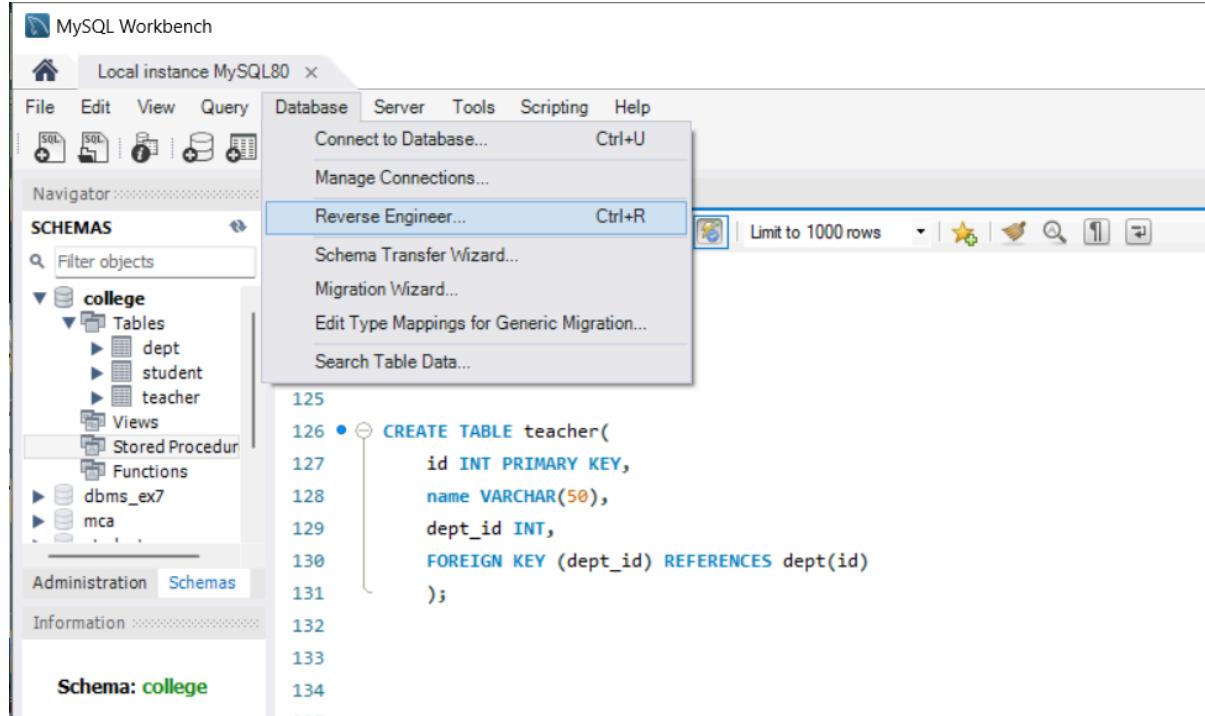
```

**Output**

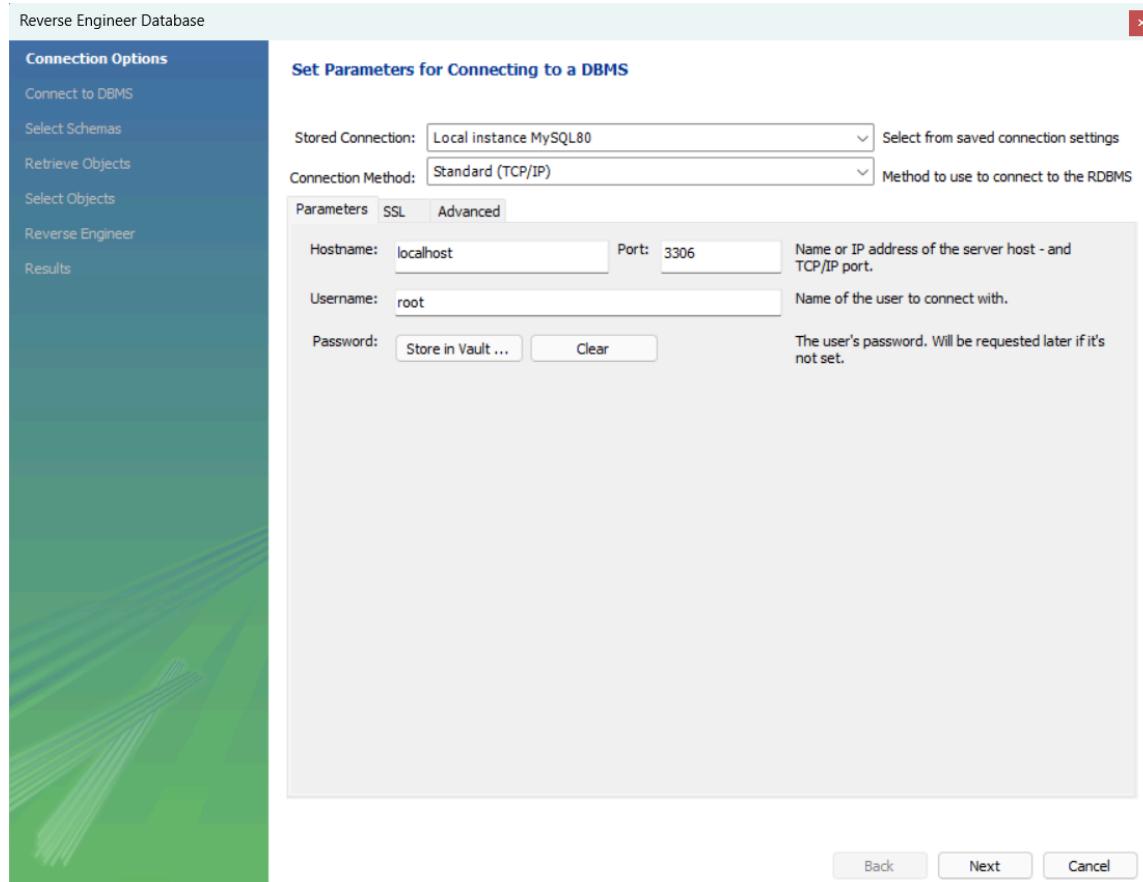
| #  | Time     | Action                                                          | Message                                                        | Duration / Fetch |
|----|----------|-----------------------------------------------------------------|----------------------------------------------------------------|------------------|
| 14 | 14:35:48 | UPDATE student SET marks=12 WHERE rollno=105                    | 1 row(s) affected Rows matched: 1 Changed: 1 Warn... 0.000 sec |                  |
| 15 | 14:37:17 | select * FROM student LIMIT 0, 1000                             | 6 row(s) returned 0.000 sec / 0.000 sec                        |                  |
| 16 | 14:38:24 | DELETE FROM student WHERE marks < 33                            | 1 row(s) affected 0.016 sec                                    |                  |
| 17 | 14:38:54 | SELECT * FROM student LIMIT 0, 1000                             | 5 row(s) returned 0.000 sec / 0.000 sec                        |                  |
| 18 | 15:15:08 | CREATE TABLE dept(id INT PRIMARY KEY, nam... 0 row(s) affected  | 0.031 sec                                                      |                  |
| 19 | 15:15:28 | CREATE TABLE teacher(id INT PRIMARY KEY, n... 0 row(s) affected | 0.031 sec                                                      |                  |

Lets see our ER diagram .

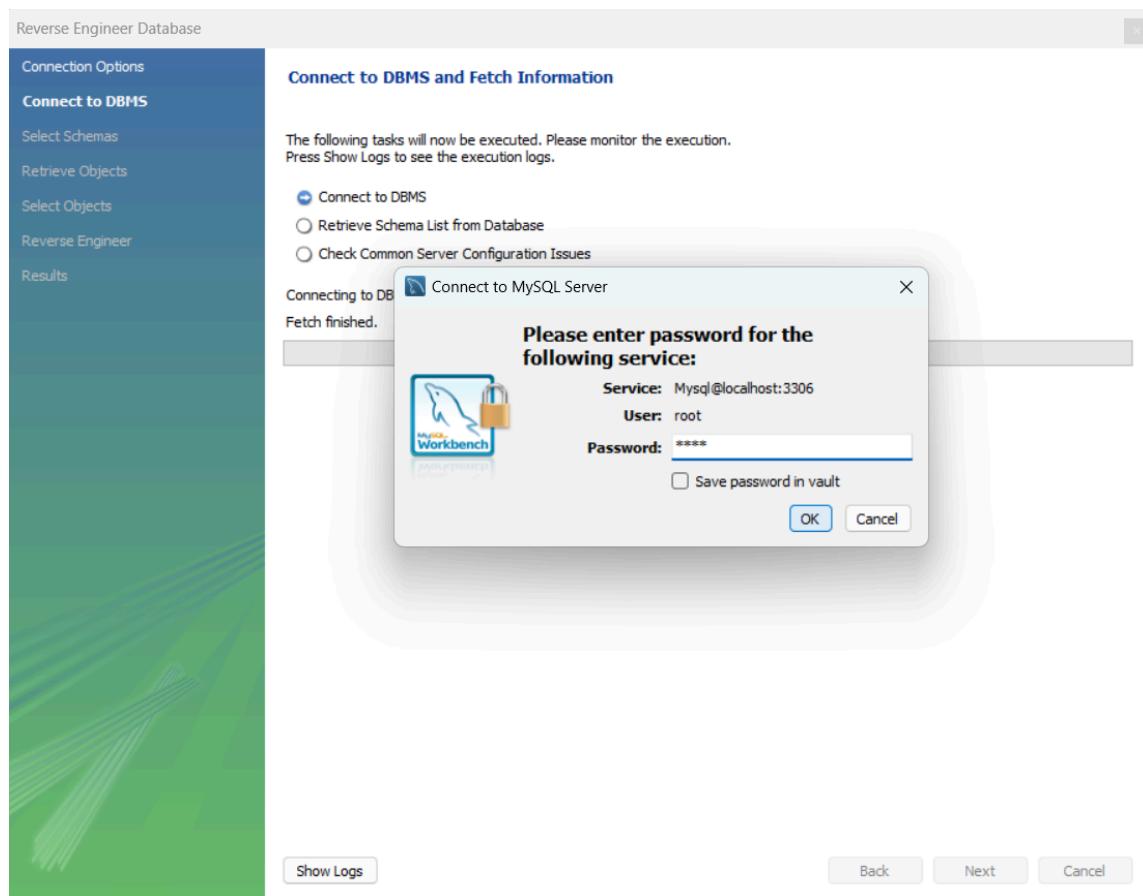
## 1. CLICK on Reverse Engineer under database option



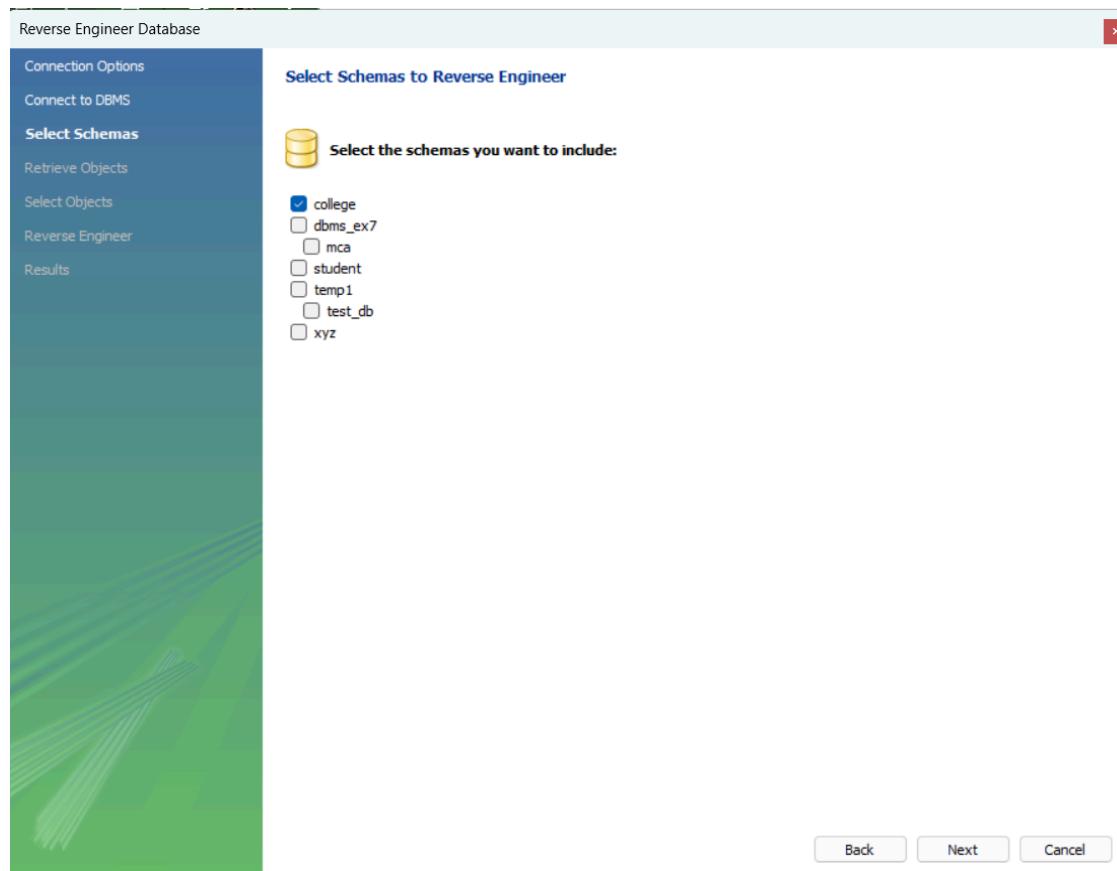
## 2. Then select stored connection and CLICK on next



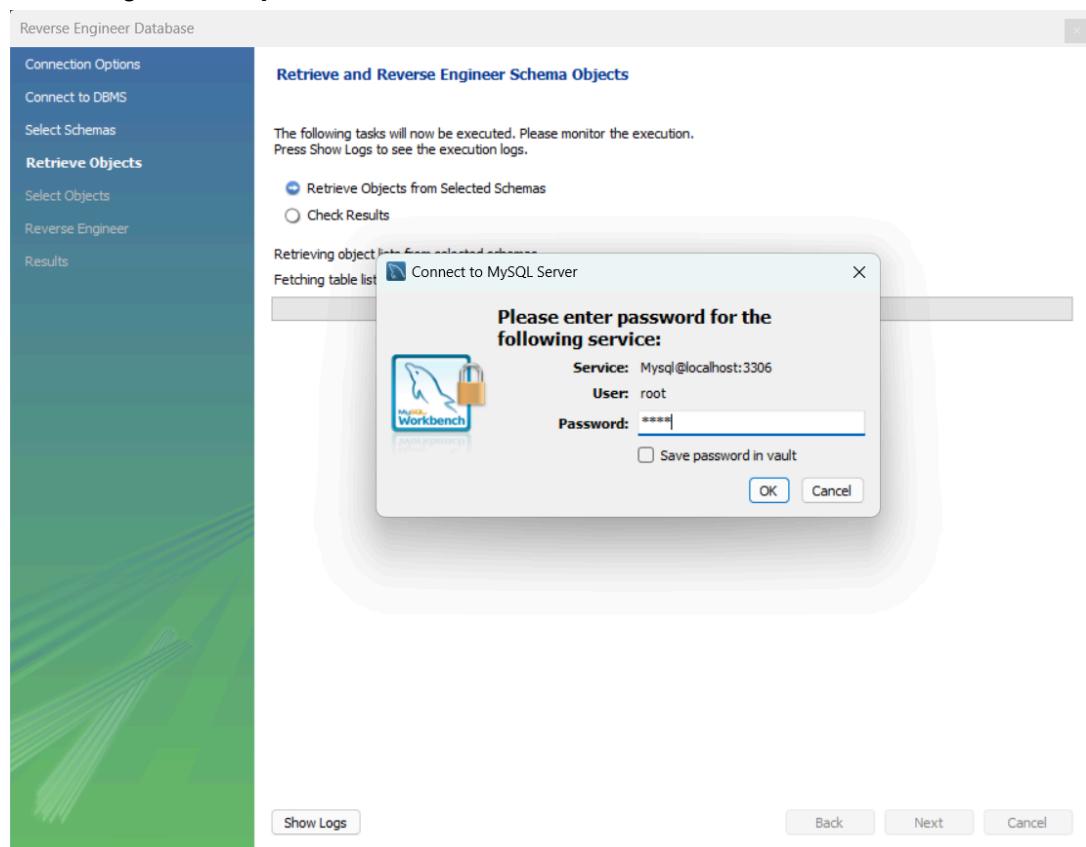
3. Enter password and click on ok-> next.



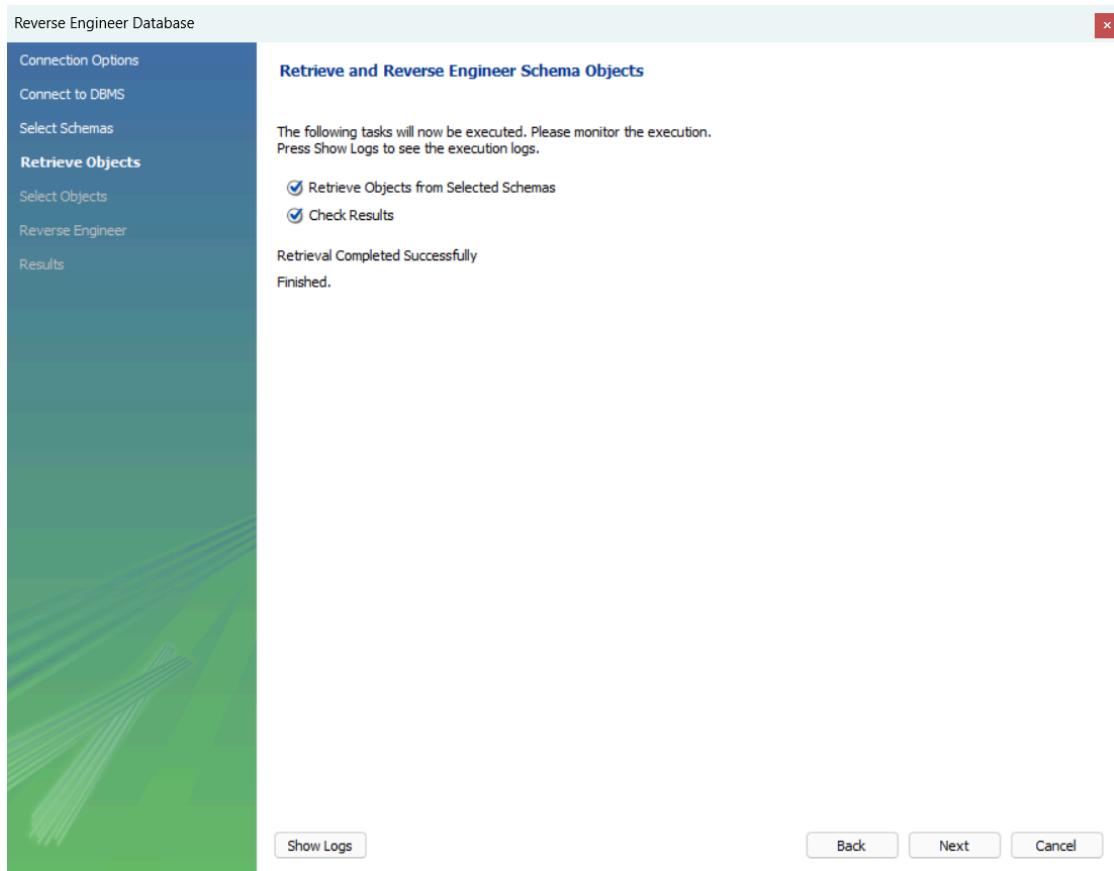
#### 4. Select your schema and CLICK next .



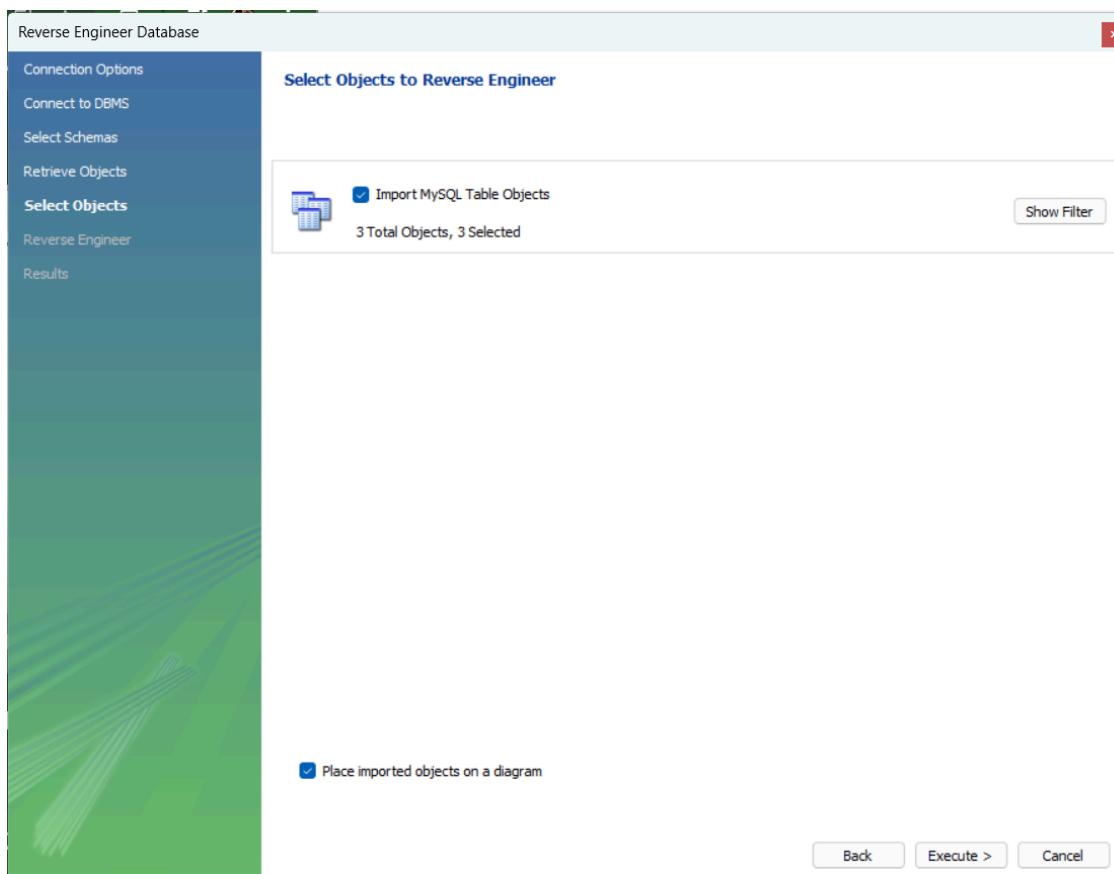
#### 5. Again enter password



## 6. Now its done ...



## 7. Click on execute



## 8. Some more steps...

Reverse Engineer Database

**Reverse Engineering Progress**

The following tasks will now be executed. Please monitor the execution.  
Press Show Logs to see the execution logs.

Reverse Engineer Selected Objects  
 Place Objects on Diagram

Operation Completed Successfully

Show Logs      Back      Next      Cancel



Reverse Engineer Database

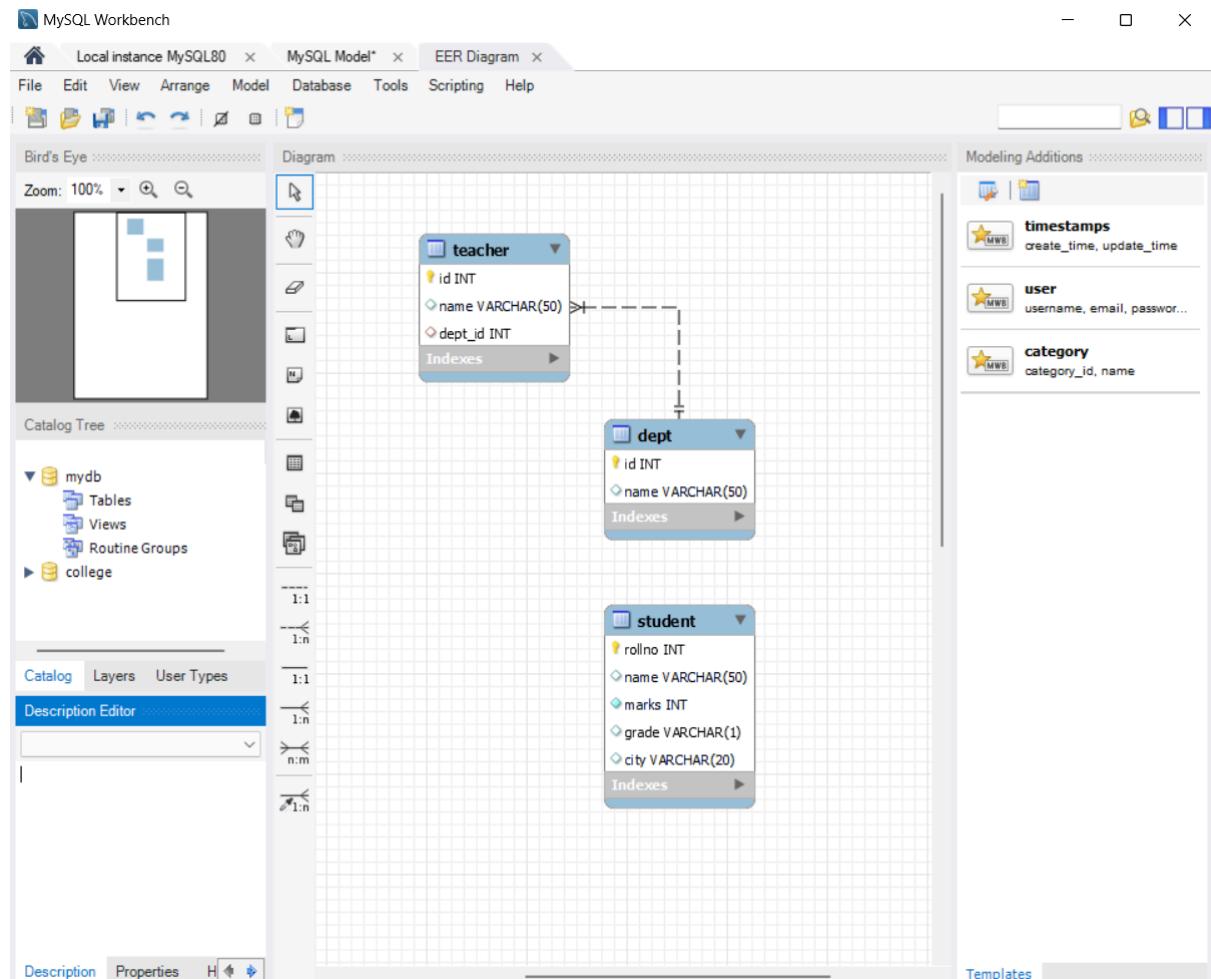
**Reverse Engineering Results**

Summary of Reverse Engineered Objects:  
- 3 tables from schema 'college'

Back      Finish      Cancel



CLICK on finish ..Now we can see our ER diagram



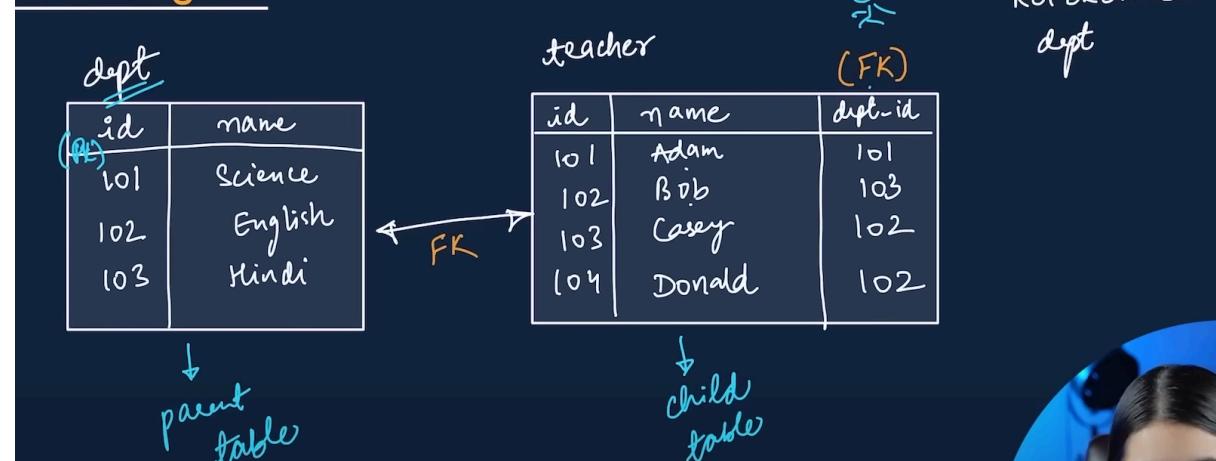
Basically this is our ER diagram ( Entity relationship )...

We can see the golden key like symbol i.e. PRIMARY key .

The arrow is there between **teacher** and **department** , it shows that the tables are connected i.e. though FOREIGN key .

Complete Course in 3 Hours | SQL One Shot using MySQL

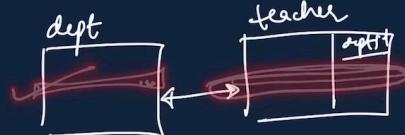
## Revisiting FK



NOTE:- The PRIMARY KEY table is the Parent table . i.e. here **dept** is Parent table , because it has PRIMARY KEY which goes into another table as FOREIGN KEY. And **teacher** table is a Child table .

## Cascading for FK

### ~~Delete~~ On ~~Update~~ Cascade



When we create a foreign key using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a primary key.

### ~~Update~~ On ~~Delete~~ Cascade

When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    courseID INT,
    FOREIGN KEY(courseID) REFERENCES course(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```



Cascading means if change happened in one place , then change must be happen in another place too .

Suppose we have **dept** table , and another is **teacher** table , and we have **dept\_id** in teacher table which is following the FOREIGN KEY relationship.

Suppose in future any department is deleted from the **department** table , then we want that the deletion should happen in the **teacher** table too .

Same goes with Updation.

For that we just have to write **ON DELETE CASCADE** and **ON UPDATE CASCADE** with creation of FOREIGN KEY.

Basically if we do any changes in **dept** table , then the changes should reflect in **teacher** table too.

```

125
126 • CREATE TABLE teacher(
127     id INT PRIMARY KEY,
128     name VARCHAR(50),
129     dept_id INT,
130     FOREIGN KEY (dept_id) REFERENCES dept(id)
131     ON DELETE CASCADE
132     ON UPDATE CASCADE
133 );
134 • drop table teacher;
135
136
137
138

```

| Output        |          |                                               |                                                  |                       |
|---------------|----------|-----------------------------------------------|--------------------------------------------------|-----------------------|
| Action Output |          |                                               |                                                  |                       |
| #             | Time     | Action                                        | Message                                          | Duration / Fetch      |
| ✓ 17          | 14:38:54 | SELECT * FROM student LIMIT 0, 1000           | 5 row(s) returned                                | 0.000 sec / 0.000 sec |
| ✓ 18          | 15:15:08 | CREATE TABLE dept(id INT PRIMARY KEY, nam...  | 0 row(s) affected                                | 0.031 sec             |
| ✓ 19          | 15:15:28 | CREATE TABLE teacher(id INT PRIMARY KEY, n... | 0 row(s) affected                                | 0.031 sec             |
| ✗ 20          | 16:00:42 | CREATE TABLE teacher(id INT PRIMARY KEY, n... | Error Code: 1050. Table 'teacher' already exists | 0.000 sec             |
| ✓ 21          | 16:00:51 | drop table teacher                            | 0 row(s) affected                                | 0.063 sec             |
| ✓ 22          | 16:00:59 | CREATE TABLE teacher(id INT PRIMARY KEY, n... | 0 row(s) affected                                | 0.078 sec             |

Now lets test this by inserting some data .

| Output        |          |                                                     |                                                        |                  |
|---------------|----------|-----------------------------------------------------|--------------------------------------------------------|------------------|
| Action Output |          |                                                     |                                                        |                  |
| #             | Time     | Action                                              | Message                                                | Duration / Fetch |
| ✓ 18          | 15:15:08 | CREATE TABLE dept(id INT PRIMARY KEY, nam...        | 0 row(s) affected                                      | 0.031 sec        |
| ✓ 19          | 15:15:28 | CREATE TABLE teacher(id INT PRIMARY KEY, n...       | 0 row(s) affected                                      | 0.031 sec        |
| ✗ 20          | 16:00:42 | CREATE TABLE teacher(id INT PRIMARY KEY, n...       | Error Code: 1050. Table 'teacher' already exists       | 0.000 sec        |
| ✓ 21          | 16:00:51 | drop table teacher                                  | 0 row(s) affected                                      | 0.063 sec        |
| ✓ 22          | 16:00:59 | CREATE TABLE teacher(id INT PRIMARY KEY, n...       | 0 row(s) affected                                      | 0.078 sec        |
| ✓ 23          | 16:03:41 | INSERT INTO dept VALUES (101,"english"), (102,"I... | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.031 sec        |

Lets display ,

141 • `SELECT * FROM dept;`

|   | <code>id</code> | <code>name</code> |
|---|-----------------|-------------------|
| ▶ | 101             | english           |
| ▶ | 102             | IT                |
| * | HULL            | HULL              |

dept 9 ×

Output :

Action Output

| # | Time        | Action                                              | Message                                                | Duration / Fetch      |
|---|-------------|-----------------------------------------------------|--------------------------------------------------------|-----------------------|
| ✓ | 19 15:15:28 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | 0 row(s) affected                                      | 0.031 sec             |
| ✗ | 20 16:00:42 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | Error Code: 1050. Table 'teacher' already exists       | 0.000 sec             |
| ✓ | 21 16:00:51 | drop table teacher                                  | 0 row(s) affected                                      | 0.063 sec             |
| ✓ | 22 16:00:59 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | 0 row(s) affected                                      | 0.078 sec             |
| ✓ | 23 16:03:41 | INSERT INTO dept VALUES (101,"english"),(102,"I...) | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.031 sec             |
| ✓ | 24 16:04:40 | SELECT * FROM dept LIMIT 0, 1000                    | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |

Similarly lets insert in **teacher** table too

143 • `INSERT INTO teacher`

144      `VALUES`

145      `(101,"Adam",101),`

146      `(102,"Eve",102);`

147

Output :

Action Output

| # | Time        | Action                                              | Message                                                | Duration / Fetch      |
|---|-------------|-----------------------------------------------------|--------------------------------------------------------|-----------------------|
| ✗ | 20 16:00:42 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | Error Code: 1050. Table 'teacher' already exists       | 0.000 sec             |
| ✓ | 21 16:00:51 | drop table teacher                                  | 0 row(s) affected                                      | 0.063 sec             |
| ✓ | 22 16:00:59 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | 0 row(s) affected                                      | 0.078 sec             |
| ✓ | 23 16:03:41 | INSERT INTO dept VALUES (101,"english"),(102,"I...) | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.031 sec             |
| ✓ | 24 16:04:40 | SELECT * FROM dept LIMIT 0, 1000                    | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| ✓ | 25 16:07:11 | INSERT INTO teacher VALUES (101,"Adam",101),(...)   | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.047 sec             |

Lets display ,

148 • `SELECT * FROM teacher;`

|   | <code>id</code> | <code>name</code> | <code>dept_id</code> |
|---|-----------------|-------------------|----------------------|
| ▶ | 101             | Adam              | 101                  |
| ▶ | 102             | Eve               | 102                  |
| * | HULL            | HULL              | HULL                 |

teacher 10 ×

Output :

Action Output

| # | Time        | Action                                              | Message                                                | Duration / Fetch      |
|---|-------------|-----------------------------------------------------|--------------------------------------------------------|-----------------------|
| ✓ | 21 16:00:51 | drop table teacher                                  | 0 row(s) affected                                      | 0.063 sec             |
| ✓ | 22 16:00:59 | CREATE TABLE teacher(id INT PRIMARY KEY, n...)      | 0 row(s) affected                                      | 0.078 sec             |
| ✓ | 23 16:03:41 | INSERT INTO dept VALUES (101,"english"),(102,"I...) | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.031 sec             |
| ✓ | 24 16:04:40 | SELECT * FROM dept LIMIT 0, 1000                    | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |
| ✓ | 25 16:07:11 | INSERT INTO teacher VALUES (101,"Adam",101),(...)   | 2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0 | 0.047 sec             |
| ✓ | 26 16:07:56 | SELECT * FROM teacher LIMIT 0, 1000                 | 2 row(s) returned                                      | 0.000 sec / 0.000 sec |

Now we want our IT department ID to be 103 i.e. we want to change our id 102 to 103.  
Let's update and display the data

```
150 • UPDATE dept
151   SET id = 103
152 WHERE id = 102;
153
154 • SELECT * FROM dept;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statements:

```
150 • UPDATE dept
151   SET id = 103
152 WHERE id = 102;
153
154 • SELECT * FROM dept;
```

Below the code editor is a "Result Grid" table with two rows:

|   | id   | name    |
|---|------|---------|
| ▶ | 101  | english |
| ▶ | 103  | IT      |
| * | HULL | HULL    |

On the right side of the interface, there is a vertical toolbar with a "Result Grid" icon.

Now let's check the data of **teacher** table

```
155 • SELECT * FROM teacher;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statement:

```
155 • SELECT * FROM teacher;
```

Below the code editor is a "Result Grid" table with three rows:

|   | id   | name | dept_id |
|---|------|------|---------|
| ▶ | 101  | Adam | 101     |
| ▶ | 102  | Eve  | 103     |
| * | HULL | HULL | HULL    |

On the right side of the interface, there is a vertical toolbar with a "Result Grid" icon.

Now here as we can see that , **dept\_id** has also changed to 103 .  
We havnt done anything in **teacher** column, still the change happened here too .  
So changes reflected automatically by CASCADING

So wherever we use FOREIGN KEY , there we must think about CASCADING .

## Table related Queries

Alter (to change the schema)

design (columns, datatype, constraints)

### ADD Column

**ALTER TABLE** *table\_name*

**ADD COLUMN** *column\_name datatype constraint;*

### DROP Column

**ALTER TABLE** *table\_name*

**DROP COLUMN** *column\_name;*

### RENAME Table

**ALTER TABLE** *table\_name*

**RENAME TO** *new\_table\_name;*



Using ALTER we can change our schema .

Earlier we have already created a **student** table under the **college** database .

```
49 • CREATE DATABASE college;
50 • USE college;
51 • CREATE TABLE student( rollno INT PRIMARY KEY, name VARCHAR(50),marks INT NOT NULL,
52                               grade VARCHAR(1), city VARCHAR(20));
53 • INSERT INTO student(rollno, name, marks, grade, city)
54   VALUES
55     (101, "anil", 78, "C", "Pune"),
56     (102, "bhumika",93,"A", "Mumbai"),
57     (103,"chetan",85,"B","Mumbai"),
58     (104, "dhruv", 96,"A","Delhi"),
59     (105, "emanuel",12,"F","Delhi"),
60     (106,"farah",82,"B","Delhi");
```

Suppose we want to add an **age** column in it , in which will store INT value .

For that we can write

**ALTER TABLE** *table\_name*

**ADD COLUMN** *column\_name datatype constraint;*

Lets add **age** and display the table ...

```

163 • ALTER TABLE student
164     ADD COLUMN age INT ;
165
166 • SELECT * FROM student;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid | Apply

| rollno | name    | marks | grade | city   | age         |
|--------|---------|-------|-------|--------|-------------|
| 101    | anil    | 79    | C     | Pune   | <b>NULL</b> |
| 102    | bhumika | 94    | O     | Mumbai | <b>NULL</b> |
| 103    | chetan  | 86    | B     | Mumbai | <b>NULL</b> |
| 104    | dhruv   | 97    | O     | Delhi  | <b>NULL</b> |
| 106    | farah   | 83    | B     | Delhi  | <b>NULL</b> |

student 15 x

Output :::::::::::::

Action Output

| #  | Time     | Action                                 | Message                                                 | Duration / Fetch      |
|----|----------|----------------------------------------|---------------------------------------------------------|-----------------------|
| 30 | 17:28:38 | SELECT * FROM dept LIMIT 0, 1000       | 2 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 31 | 17:29:46 | SELECT * FROM teacher LIMIT 0, 1000    | 2 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 32 | 18:03:11 | desc college                           | Error Code: 1146. Table 'college.college' doesn't exist | 0.000 sec             |
| 33 | 18:03:39 | desc student                           | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 34 | 18:04:24 | ALTER TABLE student ADD COLUMN age INT | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0  | 0.016 sec             |
| 35 | 18:04:44 | SELECT * FROM student LIMIT 0, 1000    | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |

As we can see, an **age** column has been added to it .

Similarly we can DROP column using ALTER TABLE .

## ALTER TABLE table\_name

### DROP COLUMN column\_name;

Lets drop **age** and display the table ...

```

168 • ALTER TABLE student
169     DROP COLUMN age;
170
171 • SELECT * FROM student;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid | Apply

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 106    | farah   | 83    | B     | Delhi  |

student 16 x

Output :::::::::::::

Action Output

| #  | Time     | Action                                 | Message                                                 | Duration / Fetch      |
|----|----------|----------------------------------------|---------------------------------------------------------|-----------------------|
| 32 | 18:03:11 | desc college                           | Error Code: 1146. Table 'college.college' doesn't exist | 0.000 sec             |
| 33 | 18:03:39 | desc student                           | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 34 | 18:04:24 | ALTER TABLE student ADD COLUMN age INT | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0  | 0.016 sec             |
| 35 | 18:04:44 | SELECT * FROM student LIMIT 0, 1000    | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 36 | 18:09:58 | ALTER TABLE student DROP COLUMN age    | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0  | 0.063 sec             |
| 37 | 18:10:09 | SELECT * FROM student LIMIT 0, 1000    | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |

As we can see **age** column has been DROP.

We can also rename the table ,

**ALTER TABLE table\_name**

**RENAME TO new\_table\_name;**

### **Table related Queries**

**CHANGE Column (rename)**

**ALTER TABLE table\_name**

**CHANGE COLUMN old\_name new\_name new\_datatype new\_constraint;**

**MODIFY Column (modify datatype/ constraint)**

**ALTER TABLE table\_name**

**MODIFY col\_name new\_datatype new\_constraint;**

Similarly we can also CHANGE the COLUMN NAME by

**ALTER TABLE table\_name**

**CHANGE COLUMN old\_name new\_name new\_datatype  
new\_constraint;**

We can modify the DATATYPE or CONSTRAINT by ,

**ALTER TABLE table\_name**

**MODIFY col\_name new\_datatype new\_constraint;**

---

Lets try some examples ...



**ADD Column**

```
ALTER TABLE student
ADD COLUMN age INT NOT NULL DEFAULT 19;
```

**DROP Column**

```
ALTER TABLE student
DROP COLUMN stu_age;
```

**MODIFY Column**

```
ALTER TABLE student
MODIFY age VARCHAR(2);
```

**RENAME Table**

```
ALTER TABLE student
RENAME TO stu;
```

**CHANGE Column (rename)**

```
ALTER TABLE student
CHANGE age stu_age INT;
```

## 1. ADD Column

```
172 • ALTER TABLE student
173   ADD COLUMN age INT NOT NULL DEFAULT 19;
174 • SELECT * FROM student;
```

| rollno | name    | marks | grade | city   | age |
|--------|---------|-------|-------|--------|-----|
| 101    | anil    | 79    | C     | Pune   | 19  |
| 102    | bhumika | 94    | O     | Mumbai | 19  |
| 103    | chetan  | 86    | B     | Mumbai | 19  |
| 104    | dhruv   | 97    | O     | Delhi  | 19  |
| 106    | farah   | 83    | B     | Delhi  | 19  |

student 17 x

Output :

| #  | Time     | Action                                        | Message                                                | Duration / Fetch      |
|----|----------|-----------------------------------------------|--------------------------------------------------------|-----------------------|
| 34 | 18:04:24 | ALTER TABLE student ADD COLUMN age INT        | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.016 sec             |
| 35 | 18:04:44 | SELECT * FROM student LIMIT 0, 1000           | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 36 | 18:09:58 | ALTER TABLE student DROP COLUMN age           | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.063 sec             |
| 37 | 18:10:09 | SELECT * FROM student LIMIT 0, 1000           | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 38 | 18:20:36 | ALTER TABLE student ADD COLUMN age INT NOT... | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.015 sec             |
| 39 | 18:20:53 | SELECT * FROM student LIMIT 0, 1000           | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |

## 2. MODIFYColumn

```

175 • ALTER TABLE student
176     MODIFY age VARCHAR(2);
177 • desc STUDENT;
178

```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statements:

```

175 • ALTER TABLE student
176     MODIFY age VARCHAR(2);
177 • desc STUDENT;
178

```

Below the code editor is a "Result Grid" showing the structure of the "student" table:

| Field  | Type        | Null | Key | Default | Extra |
|--------|-------------|------|-----|---------|-------|
| rollno | int         | NO   | PRI | NULL    |       |
| name   | varchar(50) | YES  |     | NULL    |       |
| marks  | int         | NO   |     | NULL    |       |
| grade  | varchar(1)  | YES  |     | NULL    |       |
| city   | varchar(20) | YES  |     | NULL    |       |
| age    | varchar(2)  | YES  |     | NULL    |       |

On the right side of the interface, there is a sidebar with icons for "Result Grid" and "Form Editor". Below the sidebar, a message indicates "Read Only".

At the bottom, there is an "Action Output" section showing the history of recent actions:

| #  | Time     | Action                                          | Message                                                 | Duration / Fetch      |
|----|----------|-------------------------------------------------|---------------------------------------------------------|-----------------------|
| 38 | 18:20:36 | ALTER TABLE student ADD COLUMN age INT NOT NULL | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0  | 0.015 sec             |
| 39 | 18:20:53 | SELECT * FROM student LIMIT 0, 1000             | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 40 | 18:22:29 | ALTER TABLE studnet MODIFY age VARCHAR(2)       | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.000 sec             |
| 41 | 18:22:39 | ALTER TABLE student MODIFY age VARCHAR(2)       | 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0  | 0.063 sec             |
| 42 | 18:22:51 | SELECT * FROM STUDENT LIMIT 0, 1000             | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 43 | 18:23:13 | desc STUDENT                                    | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |

We can also write **MODIFY COLUMN** instead of **MODIFY**.

I.e. **ALTER TABLE student MODIFY age VARCHAR(2);**  
**ALTER TABLE student MODIFY COLUMN age VARCHAR(2);**

### 3. CHANGE Column (rename)

```

178 • ALTER TABLE student
179     CHANGE age stu_age INT;
180 • SELECT * FROM student;

```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statements:

```

178 • ALTER TABLE student
179     CHANGE age stu_age INT;
180 • SELECT * FROM student;

```

Below the code editor is a "Result Grid" showing the structure of the "student" table after the change:

| rollno | name    | marks | grade | city   | stu_age |
|--------|---------|-------|-------|--------|---------|
| 101    | anil    | 79    | C     | Pune   | 19      |
| 102    | bhumika | 94    | O     | Mumbai | 19      |
| 103    | chetan  | 86    | B     | Mumbai | 19      |
| 104    | dhruv   | 97    | O     | Delhi  | 19      |
| 106    | farah   | 83    | B     | Delhi  | 19      |
| *      | NULL    | NULL  | NULL  | NULL   | NULL    |

On the right side of the interface, there is a sidebar with icons for "Result Grid" and "Form Editor". Below the sidebar, a message indicates "Apply".

At the bottom, there is an "Action Output" section showing the history of recent actions:

| #  | Time     | Action                                     | Message                                                 | Duration / Fetch      |
|----|----------|--------------------------------------------|---------------------------------------------------------|-----------------------|
| 40 | 18:22:29 | ALTER TABLE studnet MODIFY age VARCHAR(2)  | Error Code: 1146. Table 'college.studnet' doesn't exist | 0.000 sec             |
| 41 | 18:22:39 | ALTER TABLE student MODIFY age VARCHAR(2)  | 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0  | 0.063 sec             |
| 42 | 18:22:51 | SELECT * FROM STUDENT LIMIT 0, 1000        | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 43 | 18:23:13 | desc STUDENT                               | 6 row(s) returned                                       | 0.000 sec / 0.000 sec |
| 44 | 18:25:56 | ALTER TABLE student CHANGE age stu_age INT | 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0  | 0.047 sec             |
| 45 | 18:26:19 | SELECT * FROM student LIMIT 0, 1000        | 5 row(s) returned                                       | 0.000 sec / 0.000 sec |

We can also change the DATATYPE along with NAME here , as we changed from VARCHAR to INT

#### 4. DROP Column

```
181 • ALTER TABLE student
182   DROP COLUMN stu_age;
183 • SELECT * FROM student;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid | Form Editor | Apply

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 106    | farah   | 83    | B     | Delhi  |
| *      | NULL    | NULL  | NULL  | NULL   |

student 21 x

Output :::::

Action Output

| #  | Time     | Action                                     | Message                                                | Duration / Fetch      |
|----|----------|--------------------------------------------|--------------------------------------------------------|-----------------------|
| 42 | 18:22:51 | SELECT * FROM STUDENT LIMIT 0, 1000        | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 43 | 18:23:13 | desc STUDENT                               | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 44 | 18:25:56 | ALTER TABLE student CHANGE age stu_age INT | 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0 | 0.047 sec             |
| 45 | 18:26:19 | SELECT * FROM student LIMIT 0, 1000        | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 46 | 18:27:24 | ALTER TABLE student DROP COLUMN stu_age    | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.063 sec             |
| 47 | 18:27:41 | SELECT * FROM student LIMIT 0, 1000        | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |

#### 5. RENAME Table

Navigator classroom\* | Filter objects | Tables | Views | Stored Procedures | Functions | dbms\_ex7 | mca | student | Administration Schemas | Information

Table: stu

Columns:

- rollno int PK
- name varchar(50)
- marks int
- grade varchar(1)
- city varchar(20)

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid | Form Editor | Apply

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 106    | farah   | 83    | B     | Delhi  |
| *      | NULL    | NULL  | NULL  | NULL   |

stu 22 x

Output :::::

Action Output

| #  | Time     | Action                                     | Message                                                | Duration / Fetch      |
|----|----------|--------------------------------------------|--------------------------------------------------------|-----------------------|
| 44 | 18:25:56 | ALTER TABLE student CHANGE age stu_age INT | 5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0 | 0.047 sec             |
| 45 | 18:26:19 | SELECT * FROM student LIMIT 0, 1000        | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 46 | 18:27:24 | ALTER TABLE student DROP COLUMN stu_age    | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.063 sec             |
| 47 | 18:27:41 | SELECT * FROM student LIMIT 0, 1000        | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 48 | 18:28:34 | ALTER TABLE student RENAME TO stu          | 0 row(s) affected                                      | 0.016 sec             |
| 49 | 18:28:49 | SELECT * FROM stu LIMIT 0, 1000            | 5 row(s) returned                                      | 0.000 sec / 0.000 sec |

Object Info Session

## Table related Queries

Truncate (to delete table's data)

TRUNCATE TABLE *table\_name* ;

DROP

deletes table

TRUNCATE

table data

```
UPDATE student  
SET grade = "O"  
WHERE grade = "A";
```

The main difference between DROP and TRUNCATE is that ,  
DROP deletes the **table** and TRUNCATE deletes the **table data**.

Lets delete whole **student** table data ,

Before deletion

The screenshot shows the Oracle SQL Developer interface. In the top-left, the Navigator pane displays the schema structure under 'classroom' with 'Tables' expanded, showing 'dept', 'student', and 'teacher'. The 'student' table is selected. The top-right pane shows a history of SQL statements from line 177 to 186. The bottom-left pane shows the 'student' table definition with columns: rollno (int PK), name (varchar(50)), marks (int), grade (varchar(8)), and city (varchar(20)). The bottom-right pane is a 'Result Grid' showing the current data for the 'student' table:

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 79    | C     | Pune   |
| 102    | bhumika | 94    | O     | Mumbai |
| 103    | chetan  | 86    | B     | Mumbai |
| 104    | dhruv   | 97    | O     | Delhi  |
| 106    | farah   | 83    | B     | Delhi  |
| *      | NULL    | NULL  | NULL  | NULL   |

The bottom-right pane also contains an 'Action Output' log with several entries, including errors for table modifications and a successful select query.

After deletion ,

**Table: student**

**Columns:**

| rollno | int PK      |
|--------|-------------|
| name   | varchar(50) |
| marks  | int         |
| grade  | varchar(8)  |
| city   | varchar(20) |

**Action Output**

| #  | Time     | Action                                  | Message                                                | Duration / Fetch      |
|----|----------|-----------------------------------------|--------------------------------------------------------|-----------------------|
| 52 | 18:32:33 | ALTER TABLE stu MODIFY grade VARCHAR(8) | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 | 0.000 sec             |
| 53 | 18:41:32 | ALTER TABLE stu RENAME TO student       | 0 row(s) affected                                      | 0.015 sec             |
| 54 | 18:45:12 | SELECT * FROM stu LIMIT 0, 1000         | Error Code: 1146. Table 'college stu' doesn't exist    | 0.000 sec             |
| 55 | 18:45:17 | SELECT * FROM student LIMIT 0, 1000     | 5 row(s) returned                                      | 0.015 sec / 0.000 sec |
| 56 | 18:46:46 | TRUNCATE TABLE student                  | 0 row(s) affected                                      | 0.031 sec             |
| 57 | 18:47:09 | SELECT * FROM student LIMIT 0, 1000     | 0 row(s) returned                                      | 0.016 sec / 0.000 sec |

Whole table data has been deleted ....

Again lets insert the data

**Table: student**

**Columns:**

| rollno | int PK      |
|--------|-------------|
| name   | varchar(50) |
| marks  | int         |
| grade  | varchar(8)  |
| city   | varchar(20) |

**Action Output**

| #  | Time     | Action                                                   | Message                                                | Duration / Fetch      |
|----|----------|----------------------------------------------------------|--------------------------------------------------------|-----------------------|
| 55 | 18:45:17 | SELECT * FROM student LIMIT 0, 1000                      | 5 row(s) returned                                      | 0.015 sec / 0.000 sec |
| 56 | 18:46:46 | TRUNCATE TABLE student                                   | 0 row(s) affected                                      | 0.031 sec             |
| 57 | 18:47:09 | SELECT * FROM student LIMIT 0, 1000                      | 0 row(s) returned                                      | 0.016 sec / 0.000 sec |
| 58 | 18:48:42 | INSERT INTO student(rollno, name, marks, grade, city...) | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.000 sec / 0.000 sec |
| 59 | 18:49:06 | SELECT name, marks FROM student LIMIT 0, 1000            | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 60 | 18:49:28 | SELECT * FROM student LIMIT 0, 1000                      | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |

So whenever we want to delete data , we will not use DROP TABLE instead will use TRUNCATE TABLE .

## Practice Qs



Qs: In the student table :

- Change the name of column "name" to "full\_name".
- Delete all the students who scored marks less than 80.
- Delete the column for grades.

a)->

```
190  CHANGE COLUMN name full_name VARCHAR(20);
191 •  SELECT * FROM student;
```

| rollno | full_name | marks | grade | city   |
|--------|-----------|-------|-------|--------|
| 101    | anil      | 78    | C     | Pune   |
| 102    | bhumika   | 93    | A     | Mumbai |
| 103    | chetan    | 85    | B     | Mumbai |
| 104    | dhruv     | 96    | A     | Delhi  |
| 105    | emanuel   | 12    | F     | Delhi  |
| 106    | farah     | 82    | B     | Delhi  |
| *      | NULL      | NULL  | NULL  | NULL   |

student 27 X

Output ::

| #  | Time     | Action                                                   | Message                                                | Duration / Fetch      |
|----|----------|----------------------------------------------------------|--------------------------------------------------------|-----------------------|
| 57 | 18:47:09 | SELECT * FROM student LIMIT 0, 1000                      | 0 row(s) returned                                      | 0.016 sec / 0.000 sec |
| 58 | 18:48:42 | INSERT INTO student(rollno, name, marks, grade, city...) | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.000 sec             |
| 59 | 18:49:06 | SELECT name , marks FROM student LIMIT 0, 1000           | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 60 | 18:49:28 | SELECT * FROM student LIMIT 0, 1000                      | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 61 | 18:53:54 | ALTER TABLE student CHANGE COLUMN name full...           | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.047 sec             |
| 62 | 18:54:05 | SELECT * FROM student LIMIT 0, 1000                      | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |

b)->

```
192 •  DELETE FROM student  
193      WHERE marks<80;  
194 •  SELECT * FROM student;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statements:

```
192 •  DELETE FROM student  
193      WHERE marks<80;  
194 •  SELECT * FROM student;
```

Below the code editor is a "Result Grid" table with the following data:

|   | rollno | full_name | marks | grade | city   |
|---|--------|-----------|-------|-------|--------|
| ▶ | 102    | bhumika   | 93    | A     | Mumbai |
|   | 103    | chetan    | 85    | B     | Mumbai |
|   | 104    | dhruv     | 96    | A     | Delhi  |
| * | 106    | farah     | 82    | B     | Delhi  |
|   | NULL   | NULL      | NULL  | NULL  | NULL   |

Below the result grid is an "Output" section titled "Action Output" which lists the following history of actions:

| #  | Time     | Action                                         | Message                                                | Duration / Fetch      |
|----|----------|------------------------------------------------|--------------------------------------------------------|-----------------------|
| 59 | 18:49:06 | SELECT name , marks FROM student LIMIT 0, 1000 | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 60 | 18:49:28 | SELECT * FROM student LIMIT 0, 1000            | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 61 | 18:53:54 | ALTER TABLE student CHANGE COLUMN name full... | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.047 sec             |
| 62 | 18:54:05 | SELECT * FROM student LIMIT 0, 1000            | 6 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 63 | 18:59:52 | DELETE FROM student WHERE marks<80             | 2 row(s) affected                                      | 0.000 sec             |
| 64 | 19:00:09 | SELECT * FROM student LIMIT 0, 1000            | 4 row(s) returned                                      | 0.000 sec / 0.000 sec |

c)->

```
195 •  ALTER TABLE student  
196      DROP grade;  
197 •  SELECT * FROM student;
```

The screenshot shows the MySQL Workbench interface. At the top, there is a code editor window with the following SQL statements:

```
195 •  ALTER TABLE student  
196      DROP grade;  
197 •  SELECT * FROM student;
```

Below the code editor is a "Result Grid" table with the following data:

|   | rollno | full_name | marks | city   |
|---|--------|-----------|-------|--------|
| ▶ | 102    | bhumika   | 93    | Mumbai |
|   | 103    | chetan    | 85    | Mumbai |
|   | 104    | dhruv     | 96    | Delhi  |
| * | 106    | farah     | 82    | Delhi  |
|   | NULL   | NULL      | NULL  | NULL   |

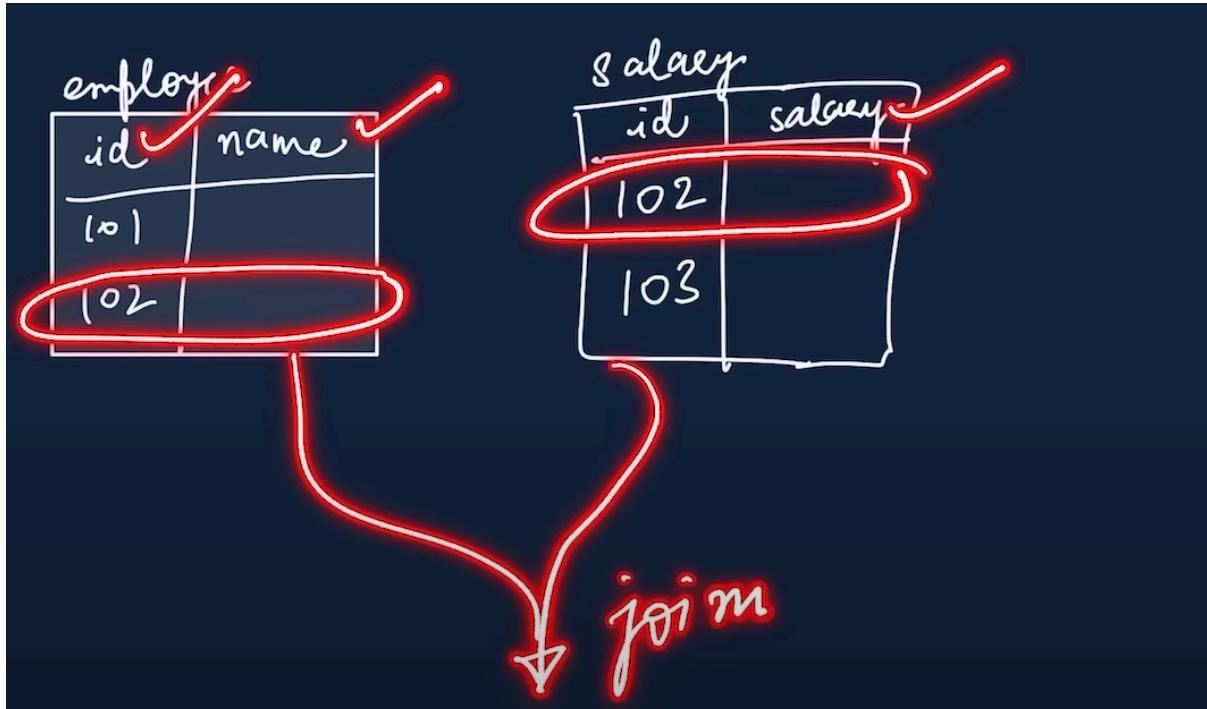
Below the result grid is an "Output" section titled "Action Output" which lists the following history of actions:

| #  | Time     | Action                              | Message                                                   | Duration / Fetch      |
|----|----------|-------------------------------------|-----------------------------------------------------------|-----------------------|
| 62 | 18:54:05 | SELECT * FROM student LIMIT 0, 1000 | 6 row(s) returned                                         | 0.000 sec / 0.000 sec |
| 63 | 18:59:52 | DELETE FROM student WHERE marks<80  | 2 row(s) affected                                         | 0.000 sec             |
| 64 | 19:00:09 | SELECT * FROM student LIMIT 0, 1000 | 4 row(s) returned                                         | 0.000 sec / 0.000 sec |
| 65 | 19:01:44 | ALTER TABLE student DROP grade      | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0    | 0.031 sec             |
| 66 | 19:01:58 | SELECT * FFROM student              | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| 67 | 19:02:04 | SELECT * FROM student LIMIT 0, 1000 | 4 row(s) returned                                         | 0.000 sec / 0.000 sec |

# Joins in SQL

Join is used to combine rows from two or more tables , based on a **related column** between them.

Suppose we have an **employee** table , in which we have **id** and **name** as Column names . One more **salary** table is there where we store salary data of employees, in which we have **id** and **salary** as column names .



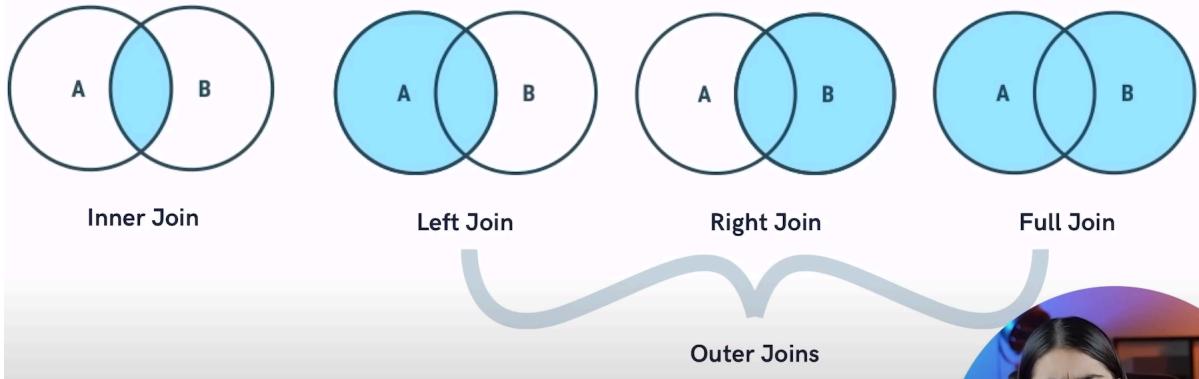
Now we want the data of employee which are in both tables i.e. here as we can see 102 is in both tables , so will get whole data of 102 ( id , name , salary).

So we can JOIN both of the table to get the COMMON INFORMATION.

Since **id** are **related column / common column** in both of the table

JOINS we can also use with the tables where FOREIGN KEY is present , although to perform JOINS its NOT MANDATORY.

## Types of Joins (Venn Diagrams)



### 1. Inner Join :-

Suppose circle A is the whole data of Table A ,and circle B is whole data of Table B.  
Then the **COMMON DATA** between Table A and B.

Example ,

As earlier we saw ,



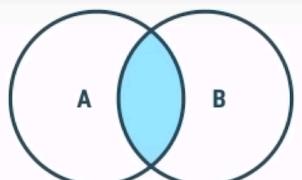
Here matching value will be in both columns .

#### Inner Join

Returns records that have matching values in both tables

#### Syntax

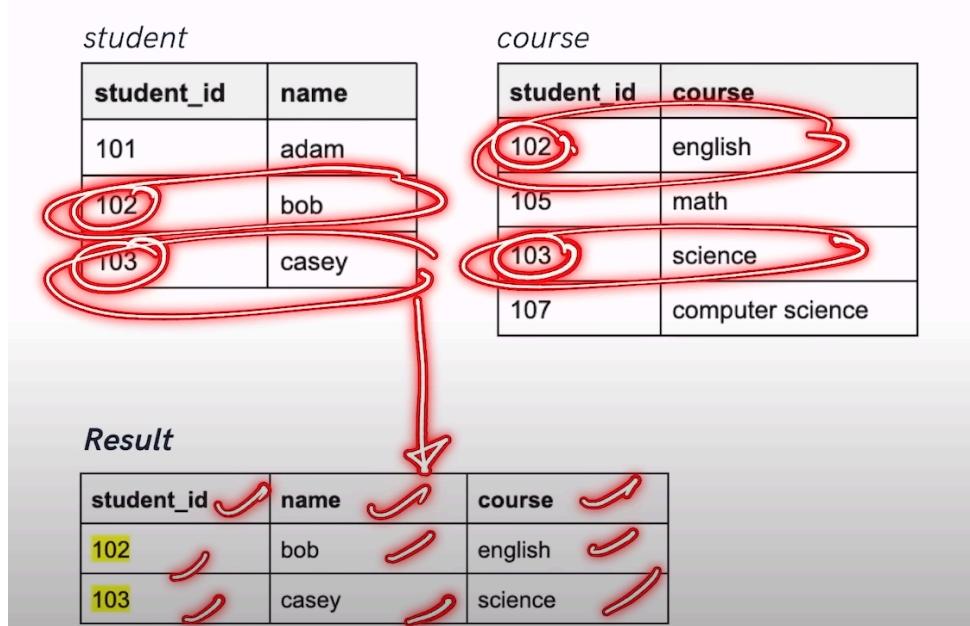
```
SELECT column(s)  
FROM tableA  
INNER JOIN tableB  
ON tableA.col_name = tableB.col_name;
```



Here circle A means table A(left side), and circle B means table B(right side). Will JOIN on the basis of their columns i.e. will take column of table A and column of table

B , where value of both is same . So there should be a column in table A with whom we can compare the value with table B.

Example,



Suppose we have 2 tables **student** and **course** , and currently there is **NO foreign key** . So when will do INNER JOIN of both tables i.e.

```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

By doing this will get the result , where will get the COMMON data based on column **student\_id**. I.e. will get the whole data of student whose column value are same.

So the INNER JOIN result will be the overlapping or same column value .

```

198 •   DROP TABLE student;
199 •   CREATE TABLE student(
200     id INT PRIMARY KEY,
201     name VARCHAR(50)
202 );
203 •   INSERT INTO student (id, name)
204     VALUES
205     (101,"adam"),
206     (102,"bob"),
207     (103,"casey");
208 •   CREATE TABLE course(
209     id INT PRIMARY KEY,
210     course VARCHAR(50)
211 );
212 •   INSERT INTO course(id, course)
213     VALUES
214     (102,"english"),
215     (105,"math"),
216     (103,"science"),
217     (107,"computer science");

```

#### Output :

| Action Output | #  | Time     | Action                                              | Message                                                | Duration / Fetch |
|---------------|----|----------|-----------------------------------------------------|--------------------------------------------------------|------------------|
|               | 69 | 22:25:04 | DROP TABLE student                                  | 0 row(s) affected                                      | 0.047 sec        |
|               | 70 | 22:26:22 | CREATE TABLE student(id INT PRIMARY KEY, n...       | 0 row(s) affected                                      | 0.062 sec        |
|               | 71 | 22:28:04 | desc course                                         | Error Code: 1146. Table 'college.course' doesn't exist | 0.000 sec        |
|               | 72 | 22:30:14 | CREATE TABLE course(id INT PRIMARY KEY, co...       | 0 row(s) affected                                      | 0.031 sec        |
|               | 73 | 22:30:17 | INSERT INTO course(id, course) VALUES (102,"engl... | 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0 | 0.016 sec        |

Here we have created a **student** table and **course** table , also we have inserted the values.

Lets display **student** table ,

```
218 •   SELECT * FROM student;
```

|   | id   | name  |
|---|------|-------|
| ▶ | 101  | adam  |
|   | 102  | bob   |
|   | 103  | casey |
| * | NULL | NULL  |

student 32 x

Apply

Revert

#### Output :

| Action Output | #  | Time     | Action                                              | Message                                                | Duration / Fetch      |
|---------------|----|----------|-----------------------------------------------------|--------------------------------------------------------|-----------------------|
|               | 73 | 22:30:17 | INSERT INTO course(id, course) VALUES (102,"engl... | 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0 | 0.016 sec             |
|               | 74 | 22:36:34 | SELECT * FROM student LIMIT 0, 1000                 | 0 row(s) returned                                      | 0.000 sec / 0.000 sec |
|               | 75 | 22:37:06 | INSERT INTO student (id, name) VALUES (101,"ada...  | 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 | 0.015 sec             |
|               | 76 | 22:37:10 | SELECT * FROM student LIMIT 0, 1000                 | 3 row(s) returned                                      | 0.000 sec / 0.000 sec |

Now lets display course table ,

```
219 • SELECT * FROM course;
```

Result Grid | Filter Rows: Edit: Export/Import: Wrap Cell Content: Result Grid

|   | id   | course           |
|---|------|------------------|
| ▶ | 102  | english          |
|   | 103  | science          |
|   | 105  | math             |
| * | 107  | computer science |
|   | NULL | NULL             |

course 33 x Apply Revert

Output :::::::

Action Output

| #  | Time     | Action                                               | Message                                                | Duration / Fetch      |
|----|----------|------------------------------------------------------|--------------------------------------------------------|-----------------------|
| 74 | 22:36:34 | SELECT * FROM student LIMIT 0, 1000                  | 0 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 75 | 22:37:06 | INSERT INTO student (id, name) VALUES (101,"ada...") | 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 | 0.015 sec             |
| 76 | 22:37:10 | SELECT * FROM student LIMIT 0, 1000                  | 3 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 77 | 22:38:16 | SELECT * FROM course LIMIT 0, 1000                   | 4 row(s) returned                                      | 0.000 sec / 0.000 sec |

Now lets see the INNER JOIN i.e COMMON DATA ,  
i.e.

**SELECT \***

**FROM student**

**INNER JOIN course**

**ON student.id = course.id**

SO will JOIN both tables on the basis of **id** column .

NOTE:- It doesnt matter if column names are DIFFERENT, just the value should be SAME.

```
220 • SELECT *
221   FROM student
222   INNER JOIN course
223   ON student.id = course.id;
```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid

|   | id  | name  | id  | course  |
|---|-----|-------|-----|---------|
| ▶ | 102 | bob   | 102 | english |
|   | 103 | casey | 103 | science |

Result 34 x Read Only

Output :::::::

Action Output

| #  | Time     | Action                                               | Message                                                | Duration / Fetch      |
|----|----------|------------------------------------------------------|--------------------------------------------------------|-----------------------|
| 75 | 22:37:06 | INSERT INTO student (id, name) VALUES (101,"ada...") | 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0 | 0.015 sec             |
| 76 | 22:37:10 | SELECT * FROM student LIMIT 0, 1000                  | 3 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 77 | 22:38:16 | SELECT * FROM course LIMIT 0, 1000                   | 4 row(s) returned                                      | 0.000 sec / 0.000 sec |
| 78 | 23:22:38 | SELECT * FROM student INNER JOIN course ON stu...    | 2 row(s) returned                                      | 0.016 sec / 0.000 sec |

OUTPUT we can see a table in which we got COMBINE COMMON data in one SINGLE TABLE.

We can also use **ALIAS** i.e. Alternate Name by using **AS**

i.e. `SELECT *`

```
FROM student AS s  
INNER JOIN course AS c  
ON student.id = course.id;
```

Here in future if we use **student** or **s** , both will be considered as SAME .

Similarly if we use **course** or **c** , both will be considered as SAME .

So we can write like

```
s.id = c.id ;
```

```
220 •  SELECT *
221   FROM student AS s
222   INNER JOIN course AS c
223     ON s.id = c.id;
```

|   | id  | name  | id  | course  |
|---|-----|-------|-----|---------|
| ▶ | 102 | bob   | 102 | english |
|   | 103 | casey | 103 | science |

Result 35 × Read Only

Output :::::

| Action Output | #                                                 | Time | Action            | Message               | Duration / Fetch |
|---------------|---------------------------------------------------|------|-------------------|-----------------------|------------------|
| 76 22:37:10   | SELECT * FROM student LIMIT 0, 1000               |      | 3 row(s) returned | 0.000 sec / 0.000 sec |                  |
| 77 22:38:16   | SELECT * FROM course LIMIT 0, 1000                |      | 4 row(s) returned | 0.000 sec / 0.000 sec |                  |
| 78 23:22:38   | SELECT * FROM student INNER JOIN course ON stu... |      | 2 row(s) returned | 0.016 sec / 0.000 sec |                  |
| 79 00:09:31   | SELECT * FROM student AS s INNER JOIN course A... |      | 2 row(s) returned | 0.000 sec / 0.000 sec |                  |

We got same result as earlier

## 2. Outer Join :-

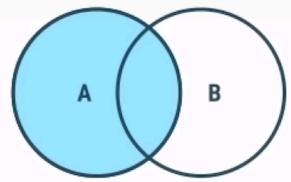
There are three types of outer joins -> **Left Join**, **Right Join** and **Full Join**.

### a) **Left Join** :-

It gives us the data which is either in A Table or the data which is overlapping between A and B Table.

## Left Join

Returns all records from the left table, and the matched records from the right table



### Syntax

```
SELECT column(s)  
FROM tableA  
LEFT JOIN tableB  
ON tableA.col_name = tableB.col_name;
```

Example,

Suppose we consider Table **employee** as A and **salary** as B

| employee A |      | salary B |        |
|------------|------|----------|--------|
| id         | name | id       | salary |
| 101        |      | 102      |        |
| 102        |      | 103      |        |

So will get A data (left) , and overlap or COMMON data .

```
SELECT column(s)  
FROM tableA ..... LEFT TABLE  
LEFT JOIN tableB ..... RIGHT TABLE  
ON tableA.col_name = tableB.col_name
```

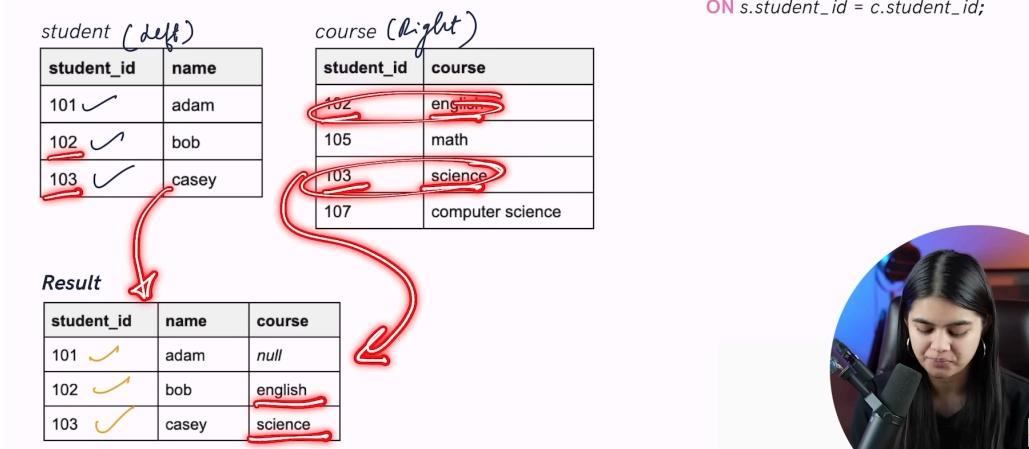
Here **Sequence Matters** i.e we must write **tableA** then **tableB** , so that it will be considered as LEFT JOIN.

So for the table which we want whole data , we must write that table name first .

Example ,

## Left Join

Example



```
SELECT *
FROM student AS s
LEFT JOIN course AS c
ON s.student_id = c.student_id;
```



Here we will get result in which whole LEFT table data is there , and also COMMON data is present .

```
224 •  SELECT *
225   FROM student AS s
226   LEFT JOIN course AS c
```

The screenshot shows a database interface with a SQL query editor and a results grid. The query is:

```
SELECT *
FROM student AS s
LEFT JOIN course AS c
```

The results grid shows the following data:

|   | id  | name  | id   | course  |
|---|-----|-------|------|---------|
| ▶ | 101 | adam  | NULL | NULL    |
|   | 102 | bob   | 102  | english |
|   | 103 | casey | 103  | science |

Below the results, there is a log of actions:

| #  | Time     | Action                                             | Message           | Duration / Fetch      |
|----|----------|----------------------------------------------------|-------------------|-----------------------|
| 77 | 22:38:16 | SELECT * FROM course LIMIT 0, 1000                 | 4 row(s) returned | 0.000 sec / 0.000 sec |
| 78 | 23:22:38 | SELECT * FROM student INNER JOIN course ON stu...  | 2 row(s) returned | 0.016 sec / 0.000 sec |
| 79 | 00:09:31 | SELECT * FROM student AS s INNER JOIN course A...  | 2 row(s) returned | 0.000 sec / 0.000 sec |
| 80 | 00:32:25 | SELECT * FROM student AS s LEFT JOIN course AS ... | 3 row(s) returned | 0.000 sec / 0.000 sec |

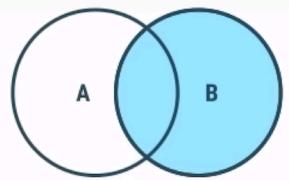
Here as we can see , we got the LEFT table and COMMON data of both .

## b) Right Join:-

Similarly like Left Join , here will get Right side area and overlapping area .

## Right Join

Returns all records from the right table, and the matched records from the left table



### Syntax

```
SELECT column(s)  
FROM tableA  
RIGHT JOIN tableB  
ON tableA.col_name = tableB.col_name;
```

As we can see , it will return RIGHT table and COMMON data of both tables.

NOTE:- Here also sequence matters , and we just have to add RIGHT JOIN in code.  
Example ,

## Right Join

### Example

student

| student_id | name  |
|------------|-------|
| 101        | adam  |
| 102        | bob   |
| 103        | casey |

course

| student_id | course           |
|------------|------------------|
| 102        | english          |
| 105        | math             |
| 103        | science          |
| 107        | computer science |

```
SELECT *  
FROM student as s  
RIGHT JOIN course as c  
ON s.student_id = c.student_id;
```

Result

| student_id | course           | name  |
|------------|------------------|-------|
| 102        | english          | bob   |
| 105        | math             | null  |
| 103        | science          | casey |
| 107        | computer science | null  |

102 and 103 are common , and will get RIGHT side of data .

```

228 •   SELECT *
229     FROM student AS s
230     RIGHT JOIN course AS c
231     ON s.id = c.id;

```

Result Grid | Filter Rows: Export: Wrap Cell Content:

|   | id   | name  | id  | course           |
|---|------|-------|-----|------------------|
| ▶ | 102  | bob   | 102 | english          |
|   | 103  | casey | 103 | science          |
|   | NULL | NULL  | 105 | math             |
|   | NULL | NULL  | 107 | computer science |

Result 37 x Read Only

Output

Action Output

| #  | Time     | Action                                             | Message           | Duration / Fetch      |
|----|----------|----------------------------------------------------|-------------------|-----------------------|
| 78 | 23:22:38 | SELECT * FROM student INNER JOIN course ON stu...  | 2 row(s) returned | 0.016 sec / 0.000 sec |
| 79 | 00:09:31 | SELECT * FROM student AS s INNER JOIN course A...  | 2 row(s) returned | 0.000 sec / 0.000 sec |
| 80 | 00:32:25 | SELECT * FROM student AS s LEFT JOIN course AS ... | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 81 | 00:52:07 | SELECT * FROM student AS s RIGHT JOIN course A...  | 4 row(s) returned | 0.000 sec / 0.000 sec |

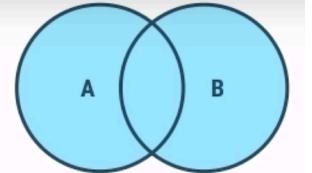
As we can see the output , RIGHT side and CCommon Data .

### c) Full Join:-

Here will get whole data including COMMON data of both tables.

#### Full Join

Returns all records when there is a match in either left or right table



#### Syntax in MySQL

```

SELECT * FROM student as a           LEFT JOIN
LEFT JOIN course as b               UNION
ON a.id = b.id                      RIGHT JOIN
UNION
SELECT * FROM student as a
RIGHT JOIN course as b
ON a.id = b.id;

```



There is statement like FULL JOIN in sql , here we use UNION .

Here we basically do LEFT JOIN and RIGHT JOIN , and we make UNION of both .  
I.e. will get LEFT and overlapping data , RIGHT and overlapping data , and after doing UNION it will JOIN them both .

So basically we do this ,

LEFT JOIN

UNION

RIGHT JOIN

Example ,

## Full Join

### Example

student

| student_id | name  |
|------------|-------|
| 101        | adam  |
| 102        | bob   |
| 103        | casey |

course

| student_id | course           |
|------------|------------------|
| 102        | english          |
| 105        | math             |
| 103        | science          |
| 107        | computer science |

### Result

| student_id | name  | course           |
|------------|-------|------------------|
| 101        | adam  | null             |
| 102        | bob   | english          |
| 103        | casey | science          |
| 105        | null  | math             |
| 107        | null  | computer science |

Lets code ..

```
233 •   SELECT *
234     FROM student AS s
235     LEFT JOIN course AS c
236     ON s.id = c.id
237
238     UNION
239
240     SELECT *
241     FROM student AS s
242     RIGHT JOIN course AS c
243     ON s.id = c.id;
```

|   | id   | name  | id   | course           |
|---|------|-------|------|------------------|
| ▶ | 101  | adam  | NULL | NULL             |
|   | 102  | bob   | 102  | english          |
|   | 103  | casey | 103  | science          |
|   | NULL | NULL  | 105  | math             |
|   | NULL | NULL  | 107  | computer science |

Result 38 x

Read Only

Output ::::::::::::

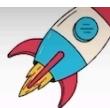
Action Output

| # | Time | Action   | Message                                            | Duration / Fetch                           |
|---|------|----------|----------------------------------------------------|--------------------------------------------|
| ✓ | 79   | 00:09:31 | SELECT * FROM student AS s INNER JOIN course A...  | 2 row(s) returned<br>0.000 sec / 0.000 sec |
| ✓ | 80   | 00:32:25 | SELECT * FROM student AS s LEFT JOIN course AS ... | 3 row(s) returned<br>0.000 sec / 0.000 sec |
| ✓ | 81   | 00:52:07 | SELECT * FROM student AS s RIGHT JOIN course A...  | 4 row(s) returned<br>0.000 sec / 0.000 sec |
| ✓ | 82   | 01:00:02 | SELECT * FROM student AS s LEFT JOIN course AS ... | 5 row(s) returned<br>0.015 sec / 0.000 sec |

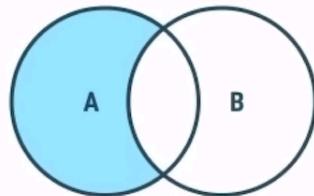
---

Now will do an activity where will learn more JOINS ...

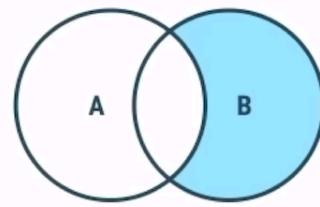
## Think & Ans



Qs: Write SQL commands to display the right exclusive join :



Left Exclusive Join



Right Exclusive Join

```
SELECT *
FROM student as a
LEFT JOIN course as b
ON a.id = b.id
WHERE b.id IS NULL;
```

Suppose we don't want overlapping data i.e COMMON data ,

Earlier in all cases we were getting COMMON data . Now we can ignore Common data by Exclusive Joins.

### 1. Left Exclusive Join:-

Will get just the LEFT table without COMMON data .

### 2. Right Exclusive Join:-

Will get just the RIGHT table without COMMON data.

---

There are NO specific method for Exclusive Joins ,

We write the code same as LEFT or RIGHT JOIN, here we just add condition

**WHERE b.id IS NULL    or            WHERE a.id IS NULL**

**IS NULL** -> checks the value should be NULL .

It is NOT NULL by default in overlapping , and in non-overlapping its NULL .

**Example**

student

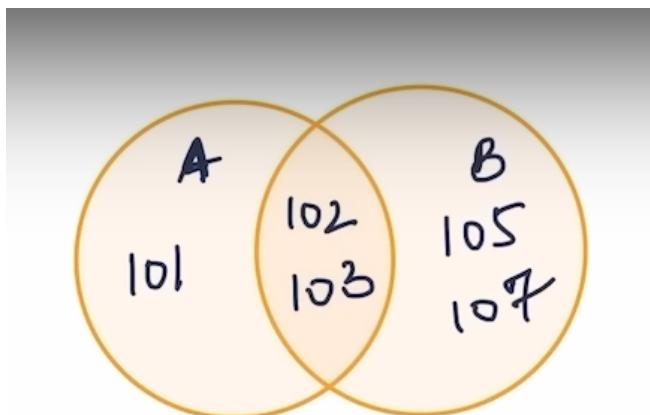
| student_id | name    |
|------------|---------|
| 101 ✓      | adam ✓  |
| 102 ✓      | bob ✓   |
| 103 ✓      | casey ✓ |

course

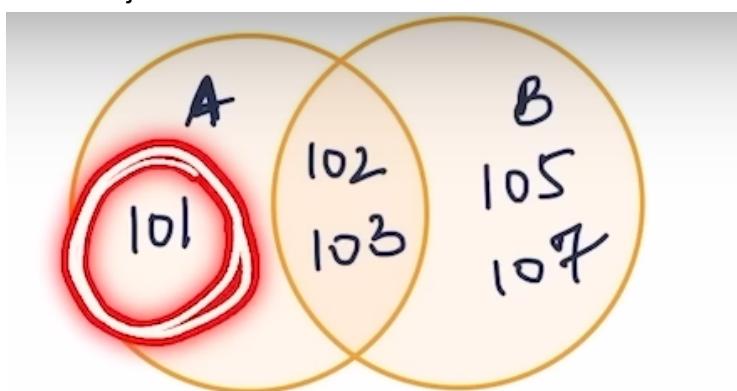
| student_id | course           |
|------------|------------------|
| 102 ✓      | english          |
| 105 ✓      | math             |
| 103 ✓      | science          |
| 107 ✓      | computer science |

**Result**

| student_id | name  | course           |
|------------|-------|------------------|
| 101        | adam  | null             |
| 102        | bob   | english          |
| 103        | casey | science          |
| 105        | null  | math             |
| 107        | null  | computer science |



Example of LEFT Exclusive JOIN,  
Here we just want this data ...



Lets check ,

```

242 •  SELECT *
243   FROM student as a
244   LEFT JOIN course as b
245   ON a.id = b.id
246   WHERE b.id IS NULL;

```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor

|   | id  | name | id   | course |
|---|-----|------|------|--------|
| ▶ | 101 | adam | NULL | NULL   |

Result 39 × Read Only

Output:

Action Output

| # | Time        | Action                                             | Message                                                   | Duration / Fetch      |
|---|-------------|----------------------------------------------------|-----------------------------------------------------------|-----------------------|
| ✓ | 82 01:00:02 | SELECT * FROM student AS s LEFT JOIN course AS ... | 5 row(s) returned                                         | 0.015 sec / 0.000 sec |
| ✗ | 83 01:36:48 | WHERE b.id IS NULL                                 | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| ✗ | 84 01:36:50 | WHERE b.id IS NULL                                 | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| ✓ | 85 01:37:00 | SELECT * FROM student as a LEFT JOIN course as ... | 1 row(s) returned                                         | 0.016 sec / 0.000 sec |

We got 101 .... By LEFT exclusive join ,  
Hence the part which exclusively exist in just LEFT part .  
Here we used LEFT JOIN , ad added condition for RIGHT TABLE data should be  
NULL i.e. **WHERE b.id IS NULL**

Similarly we can check for Right exclusive join ,

```

247 •  SELECT *
248   FROM student as a
249   RIGHT JOIN course as b
250   ON a.id = b.id
251   WHERE a.id IS NULL;

```

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor

|   | id   | name | id  | course           |
|---|------|------|-----|------------------|
| ▶ | NULL | NULL | 105 | math             |
|   | NULL | NULL | 107 | computer science |

Result 40 × Read Only

Output:

Action Output

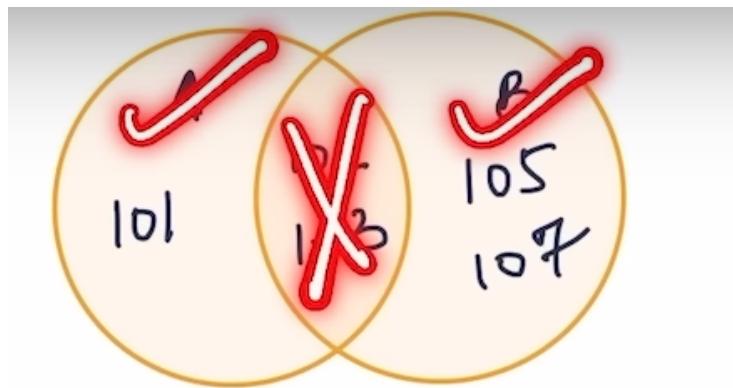
| # | Time        | Action                                             | Message                                                   | Duration / Fetch      |
|---|-------------|----------------------------------------------------|-----------------------------------------------------------|-----------------------|
| ✗ | 83 01:36:48 | WHERE b.id IS NULL                                 | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| ✗ | 84 01:36:50 | WHERE b.id IS NULL                                 | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| ✓ | 85 01:37:00 | SELECT * FROM student as a LEFT JOIN course as ... | 1 row(s) returned                                         | 0.016 sec / 0.000 sec |
| ✓ | 86 01:39:53 | SELECT * FROM student as a RIGHT JOIN course as... | 2 row(s) returned                                         | 0.000 sec / 0.000 sec |

Hence its Right exclusive data .

Here we used RIGHT JOIN , ad added condition for LEFT TABLE data should be NULL i.e. **WHERE a.id IS NULL**

There is also .....

### 3. FULL EXCLUSIVE JOIN:-



Here will get full exclusive part i.e. EXCEPT COMMON DATA  
So will get LEFT TABLE without COMMON data , RIGHT TABLE without COMMON data .

Here also there will be no specific method .

Will just make a UNION of LEFT EXCLUSIVE and RIGHT EXCLUSIVE parts.

|                        |                             |
|------------------------|-----------------------------|
| SELECT *               | ..... LEFT JOIN.....        |
| FROM student as a      | ..... LEFT TABLE            |
| LEFT JOIN course as b  | .....RIGHT TABLE            |
| ON a.id = b.id         | ..... JOIN on basis of id   |
| WHERE b.id IS NULL     | ..... EXCLUSIVE LEFT JOIN   |
| <b>UNION</b>           | <b>===== FULL JOIN=====</b> |
| SELECT *               | ..... RIGHT JOIN.....       |
| FROM student as a      | ..... LEFT TABLE            |
| RIGHT JOIN course as b | ..... RIGHT TABLE           |
| ON a.id = b.id         | ..... JOIN on basis of id   |
| WHERE a.id IS NULL     | ..... EXCLUSIVE RIGHT JOIN  |

Lets try this ...

```

252 •  SELECT *
253   FROM student as a
254   LEFT JOIN course as b
255   ON a.id = b.id
256   WHERE b.id IS NULL
257   UNION
258   SELECT *
259   FROM student as a
260   RIGHT JOIN course as b
261   ON a.id = b.id
262   WHERE a.id IS NULL;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

|   | id   | name | id   | course           |
|---|------|------|------|------------------|
| ▶ | 101  | adam | NULL | NULL             |
|   | NULL | NULL | 105  | math             |
|   | NULL | NULL | 107  | computer science |

Result 41 x Read Only

Action Output

| # | Time | Action                                                       | Message                                                   | Duration / Fetch      |
|---|------|--------------------------------------------------------------|-----------------------------------------------------------|-----------------------|
| ✗ | 84   | 01:36:50 WHERE b.id IS NULL                                  | Error Code: 1064. You have an error in your SQL syntax... | 0.000 sec             |
| ✓ | 85   | 01:37:00 SELECT * FROM student as a LEFT JOIN course as ...  | 1 row(s) returned                                         | 0.016 sec / 0.000 sec |
| ✓ | 86   | 01:39:53 SELECT * FROM student as a RIGHT JOIN course as ... | 2 row(s) returned                                         | 0.000 sec / 0.000 sec |
| ✓ | 87   | 01:51:27 SELECT * FROM student as a LEFT JOIN course as b... | 3 row(s) returned                                         | 0.000 sec / 0.000 sec |

Hence we got Full exclusive data ...

There is one more join known as ,

## SELF JOIN:-

Its basically a INNER JOIN , but this instead of JOINING both table , it JOINS the same table with itself .

**Self Join**

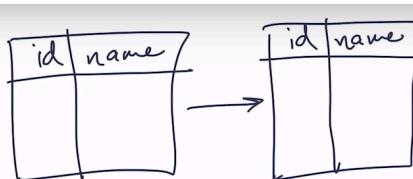
It is a regular join but the table is joined with itself.

*Syntax*

```

SELECT column(s)
FROM table as a
JOIN table as b
ON a.col_name = b.col_name;

```



When we want to join two same tables .....

Here will just write **JOIN** in code ..

Example for usecase ,

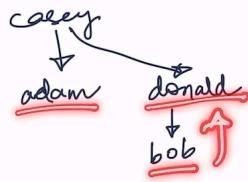
Lets suppose we have table **Employee**

### Self Join

#### Example

Employee

| id    | name   | manager_id   |
|-------|--------|--------------|
| 101   | adam   | 103 (casey)  |
| 102 ✓ | bob    | 104 (donald) |
| 103   | casey  | null         |
| 104   | donald | 103 (casey)  |



```
SELECT a.name as manager_name, b.name  
FROM employee as a  
JOIN employee as b  
ON a.id = b.manager_id;
```

Here as we can see hierarchy , adam has manager\_id 103 -> casey is his manager  
Bob has manager\_id 194 -> donald is his manager  
Casey has NULL -> No manager of casey  
Donald manager\_id is 103 -> casey is his manager

So here basically we are able to find the manager of every employee using SELF JOIN.

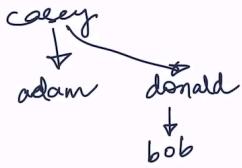
Example,

### Self Join

#### Example

Employee

| id    | name   | manager_id   |
|-------|--------|--------------|
| 101   | adam   | 103 (casey)  |
| 102 ✓ | bob    | 104 (donald) |
| 103   | casey  | null         |
| 104   | donald | 103 (casey)  |



```
SELECT a.name as manager_name, b.name  
FROM employee as a  
JOIN employee as b  
ON a.id = b.manager_id;
```

| a  |      |      | b  |      |      |
|----|------|------|----|------|------|
| id | name | m-id | id | name | m-id |
|    |      |      |    |      |      |
|    |      |      |    |      |      |
|    |      |      |    |      |      |

| a  |      |      | b  |      |      |
|----|------|------|----|------|------|
| id | name | m-id | id | name | m-id |
|    |      |      |    |      |      |
|    |      |      |    |      |      |
|    |      |      |    |      |      |

Lets create employee table

```

27 • CREATE TABLE employee(
28     id INT PRIMARY KEY,
29     name VARCHAR(50),
30     manager_id INT
31 );
32
33 • INSERT INTO employee (id, name, manager_id)
34     VALUES
35     (101, "adam", 103),
36     (102, "bob", 104),
37     (103, "casey", NULL),
38     (104, "donald", 103);
39     I
40     SEL
41
130% 4:40
Action Output
Time | Action | Response
125 16:11:51 SELECT * FROM student as a LEFT JOIN course as b ON a.id = b.id UNION SELECT * FROM student as a RIGHT JOIN... 5 row(s) returned
126 16:15:42 SELECT * FROM student as a LEFT JOIN course as b ON a.id = b.id LIMIT 0, 1000 3 row(s) returned
127 16:16:22 SELECT * FROM student as a LEFT JOIN course as b ON a.id = b.id WHERE c.id IS NULL LIMIT 0, 1000 Error Code: 1054. Unknown column 'c.id' in 'on clause'
128 16:16:31 SELECT * FROM student as a LEFT JOIN course as b ON a.id = b.id WHERE b.id IS NULL LIMIT 0, 1000 1 row(s) returned
129 16:17:53 SELECT * FROM student as a RIGHT JOIN course as b ON a.id = b.id WHERE a.id IS NULL LIMIT 0, 1000 2 row(s) returned
130 16:24:24 CREATE TABLE employee( id INT PRIMARY KEY, name VARCHAR(50), manager_id INT ) 0 row(s) affected
131 16:24:29 INSERT INTO employee (id, name, manager_id) VALUES (101, "adam", 103), (102, "bob", 104), (103, "casey", NULL), (104, "donald", 103) 4 row(s) affected Records: 4

```

Lets display it ,

```

40 • SELECT * FROM employee;
130% 24:40
Result Grid Filter Rows: Search Edit: Export/Import:

```

|  | id   | name   | manager_id |
|--|------|--------|------------|
|  | 101  | adam   | 103        |
|  | 102  | bob    | 104        |
|  | 103  | casey  | NULL       |
|  | 104  | donald | 103        |
|  | NULL | NULL   | NULL       |

Now apply SELF JOIN on it ,

```

42 • SELECT *
43     FROM employee as a
44     JOIN employee as b
45     ON a.id = b.manager_id;

```

Output ,

130% 24:45

**Result Grid** Filter Rows: Search E

|  | <b>id</b> | <b>name</b> | <b>manager_id</b> |  | <b>id</b> | <b>name</b> | <b>manager_id</b> |
|--|-----------|-------------|-------------------|--|-----------|-------------|-------------------|
|  | 103       | casey       | <b>NUL</b>        |  | 101       | adam        | 103               |
|  | 103       | casey       | <b>NUL</b>        |  | 104       | donald      | 103               |
|  | 104       | donald      | 103               |  | 102       | bob         | 104               |

Let's read from **RIGHT to LEFT**, because query executes from right to left

adam 's manager is casey , donald manager is also casey and bob has donald as manager .

SO we don't want everything , we just need names

```
42 • I SELECT a.name, b.name
43   FROM employee as a
44   JOIN employee as b
45   ON a.id = b.manager_id;
```

Output is ,

130% 24:45

**Result Grid** Filter Rows: Search E

| <b>name</b>   | <b>name</b> |
|---------------|-------------|
| casey         | donald      |
| casey         | adam        |
| <b>donald</b> | bob         |

Now both are displaying as **name**. Lets print LEFT table as **manager\_name**

```

42 • I SELECT a.name as manager_name, b.name
43   FROM employee as a
44   JOIN employee as b
45   ON a.id = b.manager_id;

```

Output is ,

Result Grid    Filter Rows:    Search    Export:

|  | manager_name | name       |
|--|--------------|------------|
|  | casey        | adam       |
|  | donald       | <b>bob</b> |
|  | casey        | donald     |

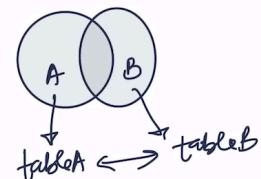
## UNION :-

Union will give us tableA data and also tableB data and will combine both, and Duplicates will get removed and only unique values will get stored .

Just like UNION concept in sets of Mathematics.

### Union

It is used to combine the result-set of two or more SELECT statements.  
Gives UNIQUE records.



To use it :

- every SELECT should have same no. of columns
- columns must have similar data types
- columns in every SELECT should be in same order

### Syntax

```

SELECT column(s) FROM tableA
UNION
SELECT column(s) FROM tableB

```



Example ,

```
43 •  SELECT name FROM employee  
44    UNION  
45    SELECT name FROM employee;
```

130% 27:45

Result Grid



Filter Rows:

Search

Export:



| name          |
|---------------|
| adam          |
| bob           |
| casey         |
| <b>donald</b> |

;

We can use **UNION ALL** to add duplicate data .....

```
43 •  SELECT name FROM employee  
44    UNION ALL  
45    SELECT name FROM employee;
```

Output,

| name          |
|---------------|
| adam          |
| bob           |
| casey         |
| <b>donald</b> |
| adam          |
| bob           |
| casey         |
| donald        |

Here we get repeated data .i.e. Duplicate data

# SQL Sub Queries:-

There are 3 methods to write sub queries ...

- 1) Write subquery within SELECT
- 2) Write subquery within FROM
- 3) Write subquery within WHERE ( most preferred method )

**SQL Sub Queries**

A Subquery or Inner query or a Nested query is a query within another SQL query.

It involves 2 select statements.

*Syntax*

```
SELECT column(s)  
FROM table_name  
WHERE col_name operator  
( subquery );
```

1) Select  
2) From  
3) Where

Select \*



So above is the WHERE method syntax,

**SELECT column(s)**  
**FROM table\_name**  
**WHERE col\_name operator** ..... operator is e.x. BETWEEN , IN , etc  
( subquery );

Example ,

**SQL Sub Queries**

**Example**

Get names of all students who scored more than class average.

Step 1. Find the avg of class  
Step 2. Find the names of students with marks > avg

| Student |         |       |
|---------|---------|-------|
| rollno  | name    | marks |
| 101     | anil    | 78    |
| 102     | bhumika | 93    |
| 103     | chetan  | 85    |
| 104     | dhruv   | 96    |
| 105     | emanuel | 92    |
| 106     | farah   | 82    |

Lets create student table ,

```

265 • CREATE TABLE student( rollno INT PRIMARY KEY, name VARCHAR(50),marks INT NOT NULL,
266                               grade VARCHAR(1), city VARCHAR(20));
267 • INSERT INTO student(rollno, name, marks, grade, city)
268   VALUES
269   (101, "anil", 78, "C", "Pune"),
270   (102, "bhumika", 93, "A", "Mumbai"),
271   (103, "chetan", 85, "B", "Mumbai"),
272   (104, "dhruv", 96, "A", "Delhi"),
273   (105, "emanuel", 12, "F", "Delhi"),
274   (106, "farah", 82, "B", "Delhi");
275 • select * FROM student;

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid

| rollno | name    | marks | grade | city   |
|--------|---------|-------|-------|--------|
| 101    | anil    | 78    | C     | Pune   |
| 102    | bhumika | 93    | A     | Mumbai |
| 103    | chetan  | 85    | B     | Mumbai |
| 104    | dhruv   | 96    | A     | Delhi  |
| 105    | emanuel | 12    | F     | Delhi  |
| 106    | farah   | 82    | B     | Delhi  |
| NULL   | NULL    | NULL  | NULL  | NULL   |

student 43 × Apply Revert

Action Output

| #    | Time     | Action                                                   | Message                                                | Duration / Fetch      |
|------|----------|----------------------------------------------------------|--------------------------------------------------------|-----------------------|
| ✓ 89 | 11:02:27 | DROP TABLE student                                       | 0 row(s) affected                                      | 0.047 sec             |
| ✓ 90 | 11:02:35 | CREATE TABLE student(rollno INT PRIMARY KEY, ...)        | 0 row(s) affected                                      | 0.031 sec             |
| ✓ 91 | 11:02:40 | INSERT INTO student(rollno, name, marks, grade, city...) | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.063 sec             |
| ✓ 92 | 11:02:59 | select * FROM student LIMIT 0, 1000                      | 6 row(s) returned                                      | 0.015 sec / 0.000 sec |

So , TO find the students who scored more than class average

STEP 1 : Find the average of class

**SELECT AVG(marks)**

**FROM student ;**

```

276 • SELECT AVG(marks)
277   FROM student ;

```

Result Grid | Filter Rows: | Export: | Result Grid

| AVG(marks) |
|------------|
| 74.3333    |

STEP2: Find the name of students who scored more than average i.e. marks > avg

**SELECT name**

**FROM student**

**WHERE marks > 74.3333 .....** Earlier we found the average in STEP 1

```

278 •   SELECT name
279     FROM student
280     WHERE marks > 74.3333;

```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

|   | name    |
|---|---------|
| ▶ | anil    |
|   | bhumika |
|   | chetan  |
|   | dhruv   |
|   | farah   |

We can also print marks along with them

```

278 •   SELECT name , marks
279     FROM student
280     WHERE marks > 74.3333;

```

Result Grid | Filter Rows: \_\_\_\_\_ | Export: \_\_\_\_\_ | Wrap Cell Content:

|   | name    | marks |
|---|---------|-------|
| ▶ | anil    | 78    |
|   | bhumika | 93    |
|   | chetan  | 85    |
|   | dhruv   | 96    |
|   | farah   | 82    |

So all are having more than average marks.

Suppose in future we get to know that by mistake extra marks has been added to any student , or some changes happen in marks . So this will affect class average and our average marks will change . and again we have to edit the average marks .

Hence we need a Dynamic process i.e which should not have any fix value , and should automatically find class average and on the basis of class average it should find student details . for these we need SUB QUERIES .

Now here lets use sub query to combine STEP 1 and STEP 2.

### Syntax

```

SELECT column(s)
FROM table_name
WHERE col_name operator
( subquery );

```

```

SELECT name , marks
FROM student
WHERE marks > ( SELECT AVG(marks)
                  FROM student ) ;

```

Here operator is >

Lets execute this

```
281 •  SELECT name , marks  
282      FROM student  
283      WHERE marks > (SELECT AVG(marks)  
284                      FROM student ) ;
```

The screenshot shows the MySQL Workbench interface. At the top, the SQL editor contains the following query:

```
281 •  SELECT name , marks  
282      FROM student  
283      WHERE marks > (SELECT AVG(marks)  
284                      FROM student ) ;
```

The results are displayed in a "Result Grid" table:

| name    | marks |
|---------|-------|
| anil    | 78    |
| bhumika | 93    |
| chetan  | 85    |
| dhruv   | 96    |
| farah   | 82    |

Below the results, the "Action Output" section shows the history of actions:

| #  | Time     | Action                                         | Message           | Duration / Fetch      |
|----|----------|------------------------------------------------|-------------------|-----------------------|
| 93 | 11:06:18 | SELECT AVG(marks) FROM student LIMIT 0, 1000   | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 94 | 11:10:25 | SELECT name FROM student WHERE marks > 74.3... | 5 row(s) returned | 0.000 sec / 0.000 sec |
| 95 | 11:12:41 | SELECT name ,marks FROM student WHERE marks... | 5 row(s) returned | 0.000 sec / 0.000 sec |
| 96 | 12:34:03 | SELECT name ,marks FROM student WHERE marks... | 5 row(s) returned | 0.000 sec / 0.000 sec |

As we can see , we got the same details as earlier ...

Now if in future marks got updated of any student , then we dont have to change query .  
This will be the general query and its dynamic i.e. its not static ( not fix ), So this will update automatically in future .

---

Lets see one more example ,

The screenshot shows a web-based SQL tutorial page titled "SQL Sub Queries". It includes an "Example" section and a "Step 1. Find the even roll numbers" section. To the right, there is a table of student data:

| rollno | name    | marks |
|--------|---------|-------|
| 101    | anil    | 78    |
| 102    | bhumika | 93    |
| 103    | chetan  | 85    |
| 104    | dhruv   | 96    |
| 105    | emanuel | 92    |
| 106    | farah   | 82    |

STEP1:-

```
285 •  SELECT rollno  
286      FROM student  
287      WHERE rollno % 2 = 0;
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
285 •  SELECT rollno  
286      FROM student  
287      WHERE rollno % 2 = 0;
```

The results are displayed in a "Result Grid" table:

| rollno |
|--------|
| 102    |
| 104    |
| 106    |
| NULL   |

## STEP 2:-

```
288 •  SELECT name  
289   FROM student  
290   WHERE rollno IN (102,104,106);
```

| name    |
|---------|
| bhumika |
| dhruv   |
| farah   |

IN will match rollno with all the values and will give result .

Now lets combine ...

```
288 •  SELECT name  
289   FROM student  
290   WHERE rollno IN (SELECT rollno  
291       FROM student  
292       WHERE rollno % 2 = 0);
```

| name    |
|---------|
| bhumika |
| dhruv   |
| farah   |

student 50 x Read Only

Action Output

| #  | Time     | Action                                              | Message           | Duration / Fetch      |
|----|----------|-----------------------------------------------------|-------------------|-----------------------|
| 96 | 12:34:03 | SELECT name , marks FROM student WHERE marks...     | 5 row(s) returned | 0.000 sec / 0.000 sec |
| 97 | 12:52:32 | SELECT rollno FROM student WHERE rollno % 2 = 0 ... | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 98 | 12:54:24 | SELECT name FROM student WHERE rollno IN (102...    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 99 | 12:55:53 | SELECT name FROM student WHERE rollno IN (SEL...    | 3 row(s) returned | 0.000 sec / 0.000 sec |

Now we can also print rollno with it , so that will see even roll nos

```

288 •  SELECT name , rollno
289   FROM student
290   WHERE rollno IN (SELECT rollno
291     FROM student
292       WHERE rollno % 2 = 0);

```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Result Grid | Form Editor | Apply | Revert

|         | name | rollno |
|---------|------|--------|
| bhumika | 102  |        |
| dhruv   | 104  |        |
| farah   | 106  |        |
| *       | NULL | NULL   |

student 51 ×

Output :::::

Action Output

| #   | Time     | Action                                              | Message           | Duration / Fetch      |
|-----|----------|-----------------------------------------------------|-------------------|-----------------------|
| 97  | 12:52:32 | SELECT rollno FROM student WHERE rollno % 2 = 0 ... | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 98  | 12:54:24 | SELECT name FROM student WHERE rollno IN (102...)   | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 99  | 12:55:53 | SELECT name FROM student WHERE rollno IN (SEL...)   | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 100 | 13:11:25 | SELECT name , rollno FROM student WHERE rollno I... | 3 row(s) returned | 0.000 sec / 0.000 sec |

Lets see one more example ,

## SQL Sub Queries

*Example with FROM*

Find the max marks from the students of Delhi

Step 1. Find the students of Delhi

Step 2. Find their max marks using the sublist in step 1

| rollno | name    | marks | city   |
|--------|---------|-------|--------|
| 101    | anil    | 78    | Pune   |
| 102    | bhumika | 93    | Mumbai |
| 103    | chetan  | 85    | Mumbai |
| 104    | dhruv   | 96    | Delhi  |
| 105    | emanuel | 92    | Delhi  |
| 106    | farah   | 82    | Delhi  |

## STEP 1:-

```
293 •  SELECT *
294   FROM student
295   WHERE city = "delhi";
296
```

| rollno | name    | marks | grade | city  |
|--------|---------|-------|-------|-------|
| 104    | dhruv   | 96    | A     | Delhi |
| 105    | emanuel | 12    | F     | Delhi |
| 106    | farah   | 82    | B     | Delhi |
| *      | HULL    | HULL  | HULL  | HULL  |

student 52 x

Output

| #   | Time     | Action                                              | Message           | Duration / Fetch      |
|-----|----------|-----------------------------------------------------|-------------------|-----------------------|
| 98  | 12:54:24 | SELECT name FROM student WHERE rollno IN (102...    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 99  | 12:55:53 | SELECT name FROM student WHERE rollno IN (SEL...    | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 100 | 13:11:25 | SELECT name , rollno FROM student WHERE rollno I... | 3 row(s) returned | 0.000 sec / 0.000 sec |
| 101 | 13:18:30 | SELECT * FROM student WHERE city = "delhi" LIMI...  | 3 row(s) returned | 0.000 sec / 0.000 sec |

## Step 2:-

Here we just want the maximum marks of **delhi** students . So instead of using **student** table in **FROM** , will use the table which we got in **STEP 1**.

```
296 •  SELECT MAX(marks)
297   ⏺ FROM ( SELECT *
298     ⏺   FROM student
299     ⏺   WHERE city = "delhi" );
300
```

| #   | Time     | Action                                              | Message                                                                   | Duration / Fetch      |
|-----|----------|-----------------------------------------------------|---------------------------------------------------------------------------|-----------------------|
| 100 | 13:11:25 | SELECT name , rollno FROM student WHERE rollno I... | 3 row(s) returned                                                         | 0.000 sec / 0.000 sec |
| 101 | 13:18:30 | SELECT * FROM student WHERE city = "delhi" LIMI...  | 3 row(s) returned                                                         | 0.000 sec / 0.000 sec |
| 102 | 13:22:26 | SELECT MAX(marks) FROM ( SELECT * FROM stud...      | Error Code: 1248 Error Code: 1248. Every derived table must have its o... | 0.000 sec             |
| 103 | 13:22:28 | SELECT MAX(marks) FROM ( SELECT * FROM stud...      | Error Code: 1248 Error Code: 1248. Every derived table must have its o... | 0.000 sec             |

We got an error for **ALIAS** , let's use an alias for the table .

```

296 •   SELECT MAX(marks)
297   FROM ( SELECT *
298     FROM student
299     WHERE city = "delhi") AS temp;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid | Form Editor | Read Only

Output :

| #   | Time     | Action                                             | Message                                                  | Duration / Fetch      |
|-----|----------|----------------------------------------------------|----------------------------------------------------------|-----------------------|
| 101 | 13:18:30 | SELECT * FROM student WHERE city = "delhi" LIMI... | 3 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 102 | 13:22:26 | SELECT MAX(marks) FROM ( SELECT * FROM stud...     | Error Code: 1248. Every derived table must have its o... | 0.000 sec             |
| 103 | 13:22:28 | SELECT MAX(marks) FROM ( SELECT * FROM stud...     | Error Code: 1248. Every derived table must have its o... | 0.000 sec             |
| 104 | 13:25:10 | SELECT MAX(marks) FROM ( SELECT * FROM stud...     | 1 row(s) returned                                        | 0.000 sec / 0.000 sec |

As we can see in below table ,

| rollno | name    | marks | city   |
|--------|---------|-------|--------|
| 101    | anil    | 78    | Pune   |
| 102    | bhumika | 93    | Mumbai |
| 103    | chetan  | 85    | Mumbai |
| 104    | dhruv   | 96    | Delhi  |
| 105    | emanuel | 92    | Delhi  |
| 106    | farah   | 82    | Delhi  |

This is the output .

We can also have alternate simple logic than this ,

```

SELECT MAX(marks)
FROM student
WHERE city = "Delhi"

```

Will get the same output ....

```
300 •  SELECT MAX(marks)
301   FROM student
302   WHERE city = "Delhi";
```

Result Grid | Filter Rows: Export: Wrap Cell Content: ▾

|   | MAX(marks) |
|---|------------|
| ▶ | 96         |

Result 54 × Read Only

Output :::::

Action Output

| # | Time | Action   | Message                                          | Duration / Fetch                                                   |
|---|------|----------|--------------------------------------------------|--------------------------------------------------------------------|
| ✖ | 102  | 13:22:26 | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | Error Code: 1248. Every derived table must have its o... 0.000 sec |
| ✖ | 103  | 13:22:28 | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | Error Code: 1248. Every derived table must have its o... 0.000 sec |
| ✓ | 104  | 13:25:10 | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | 1 row(s) returned 0.000 sec / 0.000 sec                            |
| ✓ | 105  | 13:29:11 | SELECT MAX(marks) FROM student WHERE city = "... | 1 row(s) returned 0.000 sec / 0.000 sec                            |

# SQL Sub Queries

## *Example with SELECT*

WE rarely use SELECT for Sub query ,

# MySQL Views

## MySQL Views

A view is a virtual table based on the result-set of an SQL statement.

```
CREATE VIEW view1 AS  
SELECT rollno, name FROM student;  
  
SELECT * FROM view1;
```

\*A view always shows up-to-date data. The database engine recreates the view, every time a user queries it.

In normal tables , they are **real tables** with **real data** and the operations which we do on tables , that actual reflects in **real database**.

In **VIEWS** , they are **virtual tables** i.e. temporary tables in which can store any data .

Suppose there is a Company , in which we have **customer** table and there is **sales** team , so sales team should know customer information . But that information should have an limit . E.x. Sales steam should not know that which CREDIT CARD customer has used for payment.

So will create a small version of table i.e. VIEWS which will have small data . And on that VIEWS we can run different queries .

Syntax ,

**CREATE VIEW view\_name AS**

**{QUERY}** ..... The data that should store in VIEW

Example ,

Now teacher shouldnt know that student is coming from which city . Lets create an view where will store details of student except city .

```

303 • CREATE VIEW view1 AS
304     SELECT rollno, name, marks FROM student ;

```

Output :

| Action Output | #        | Time | Action                                           | Message                                                  | Duration / Fetch      |
|---------------|----------|------|--------------------------------------------------|----------------------------------------------------------|-----------------------|
| 103           | 13:22:28 |      | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | Error Code: 1248. Every derived table must have its o... | 0.000 sec             |
| 104           | 13:25:10 |      | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | 1 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 105           | 13:29:11 |      | SELECT MAX(marks) FROM student WHERE city = "... | 1 row(s) returned                                        | 0.000 sec / 0.000 sec |
| 106           | 14:08:01 |      | CREATE VIEW view1 AS SELECT rollno, name, mar... | 0 row(s) affected                                        | 0.032 sec             |

View has been created ..

Now we can perform operations on view , Lets first display ..

305 • `SELECT * FROM view1;`

| rollno | name    | marks |
|--------|---------|-------|
| 101    | anil    | 78    |
| 102    | bhumika | 93    |
| 103    | chetan  | 85    |
| 104    | dhruv   | 96    |
| 105    | emanuel | 12    |
| 106    | farah   | 82    |

view1 55 x Read Only

Output :

| Action Output | #        | Time | Action                                           | Message           | Duration / Fetch      |
|---------------|----------|------|--------------------------------------------------|-------------------|-----------------------|
| 104           | 13:25:10 |      | SELECT MAX(marks) FROM ( SELECT * FROM stud...   | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 105           | 13:29:11 |      | SELECT MAX(marks) FROM student WHERE city = "... | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 106           | 14:08:01 |      | CREATE VIEW view1 AS SELECT rollno, name, mar... | 0 row(s) affected | 0.032 sec             |
| 107           | 14:25:31 |      | SELECT * FROM view1 LIMIT 0, 1000                | 6 row(s) returned | 0.031 sec / 0.000 sec |

It will look and work like real table , but its actually a VIRTUAL TABLE .

Lets perform more operations ,

Example , Display students with more than 90 marks

306 • `SELECT *`

307     `FROM view1`

308     `WHERE marks > 90;`

| rollno | name    | marks |
|--------|---------|-------|
| 102    | bhumika | 93    |
| 104    | dhruv   | 96    |

view1 56 x Read Only

Output :

| Action Output | #        | Time | Action                                            | Message           | Duration / Fetch      |
|---------------|----------|------|---------------------------------------------------|-------------------|-----------------------|
| 105           | 13:29:11 |      | SELECT MAX(marks) FROM student WHERE city = "...  | 1 row(s) returned | 0.000 sec / 0.000 sec |
| 106           | 14:08:01 |      | CREATE VIEW view1 AS SELECT rollno, name, mar...  | 0 row(s) affected | 0.032 sec             |
| 107           | 14:25:31 |      | SELECT * FROM view1 LIMIT 0, 1000                 | 6 row(s) returned | 0.031 sec / 0.000 sec |
| 108           | 14:27:28 |      | SELECT * FROM view1 WHERE marks > 90 LIMIT 0, ... | 2 row(s) returned | 0.000 sec / 0.000 sec |

We can DROP VIEW just like Normal tables ,  
We can simply write **DROP VIEW view1;**