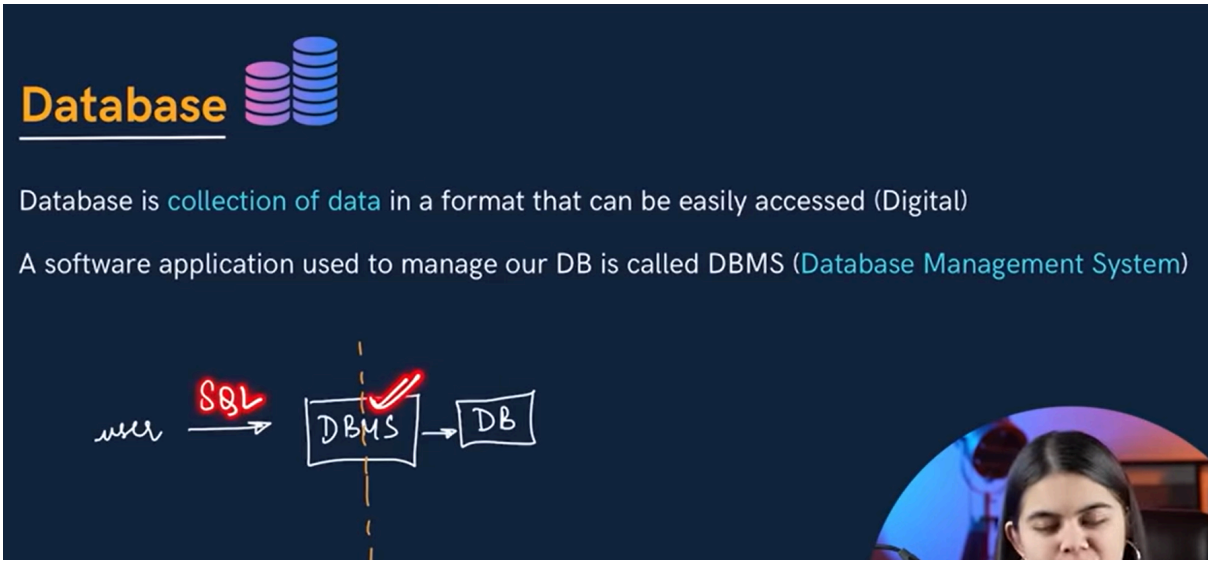
SQL (Structured Query Language)

To access databases we use SQL .
Firstly let's study what **Databases** are **.**



DATABASE :- (collection of data)

Big companies have alot 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:

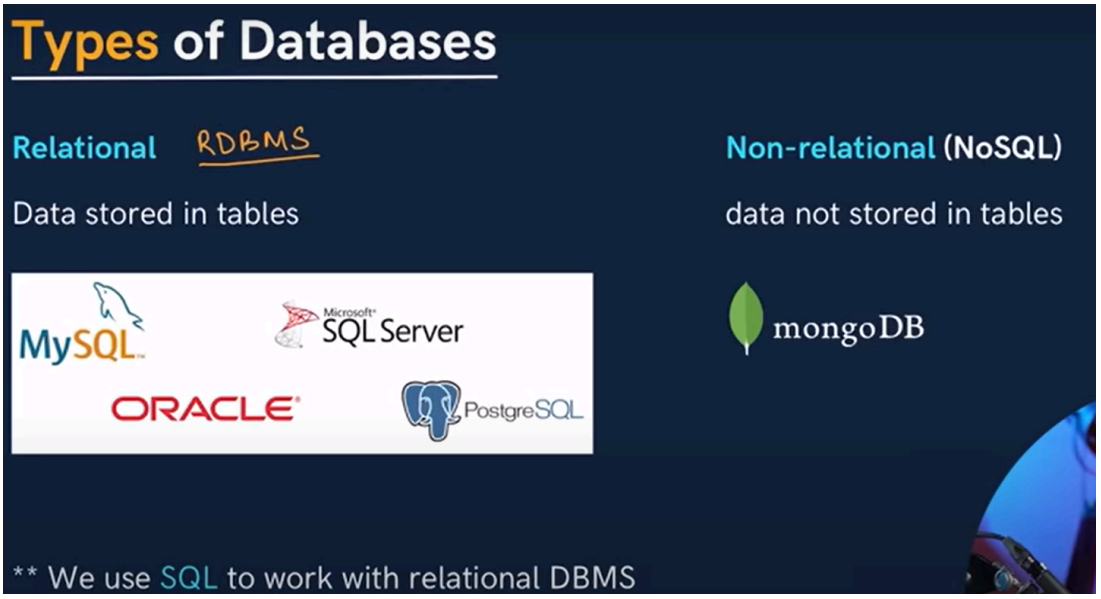
When we store data **Without Tables** then such databases are known as **Non-relational** , we can also say them **NoSQL** databases. Because such databases **Don't Know SQL.**

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.

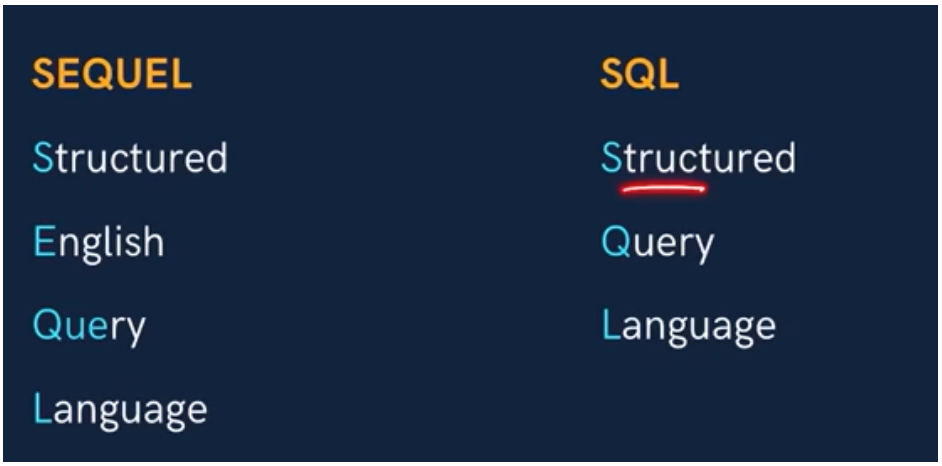


What is SQL ?

Structured Query Language

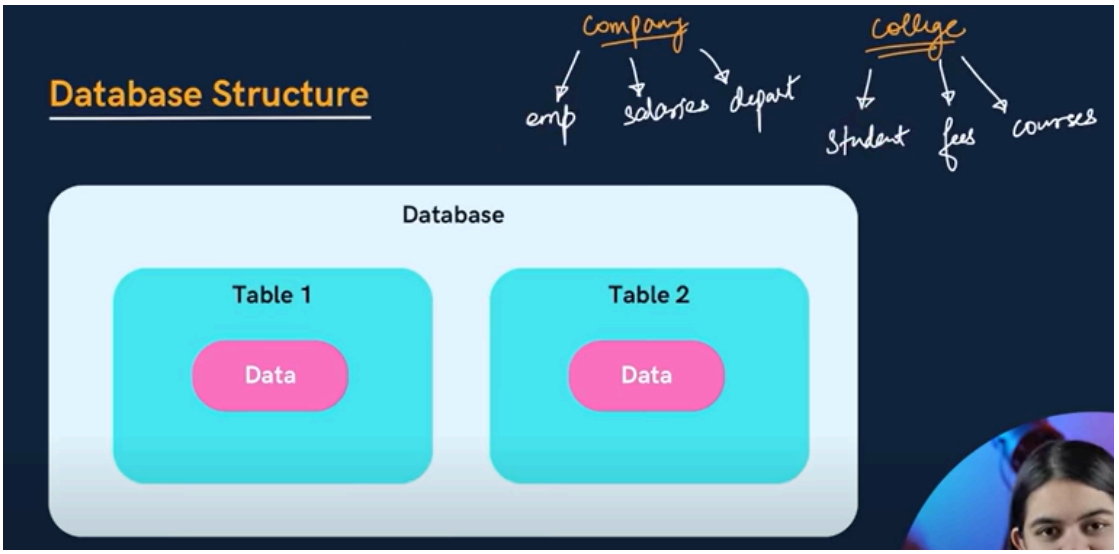
This language is understood by computers and its use to interact with relational databases .
On our data we perform four Major operations. I.e **CRUD** operations. **(create, read , update and delete).**

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** .

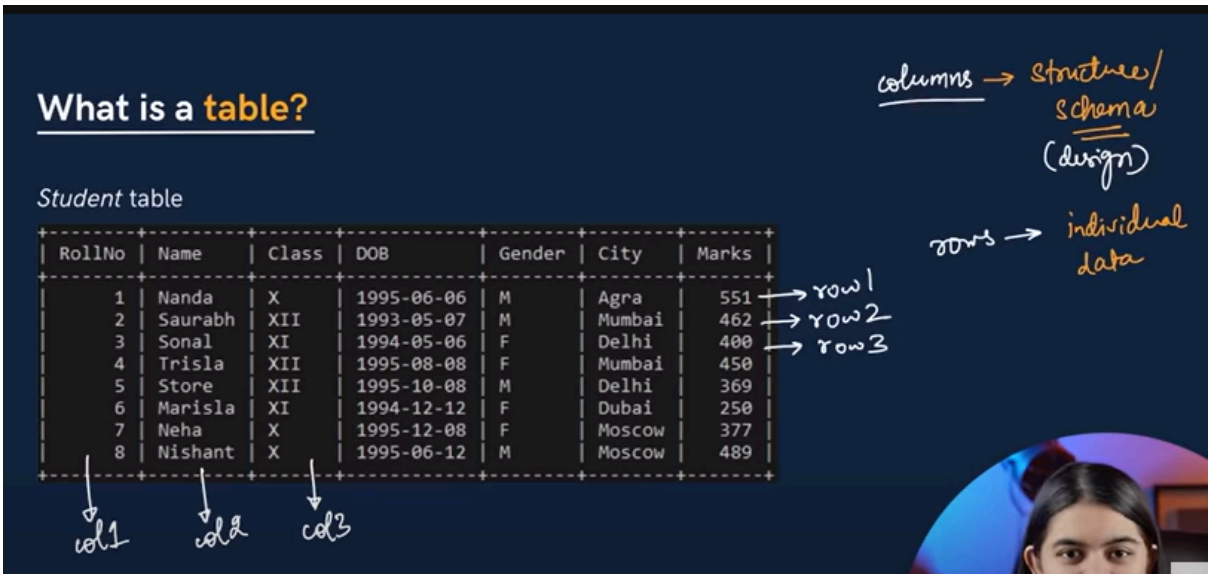


Database Structure:-

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 .

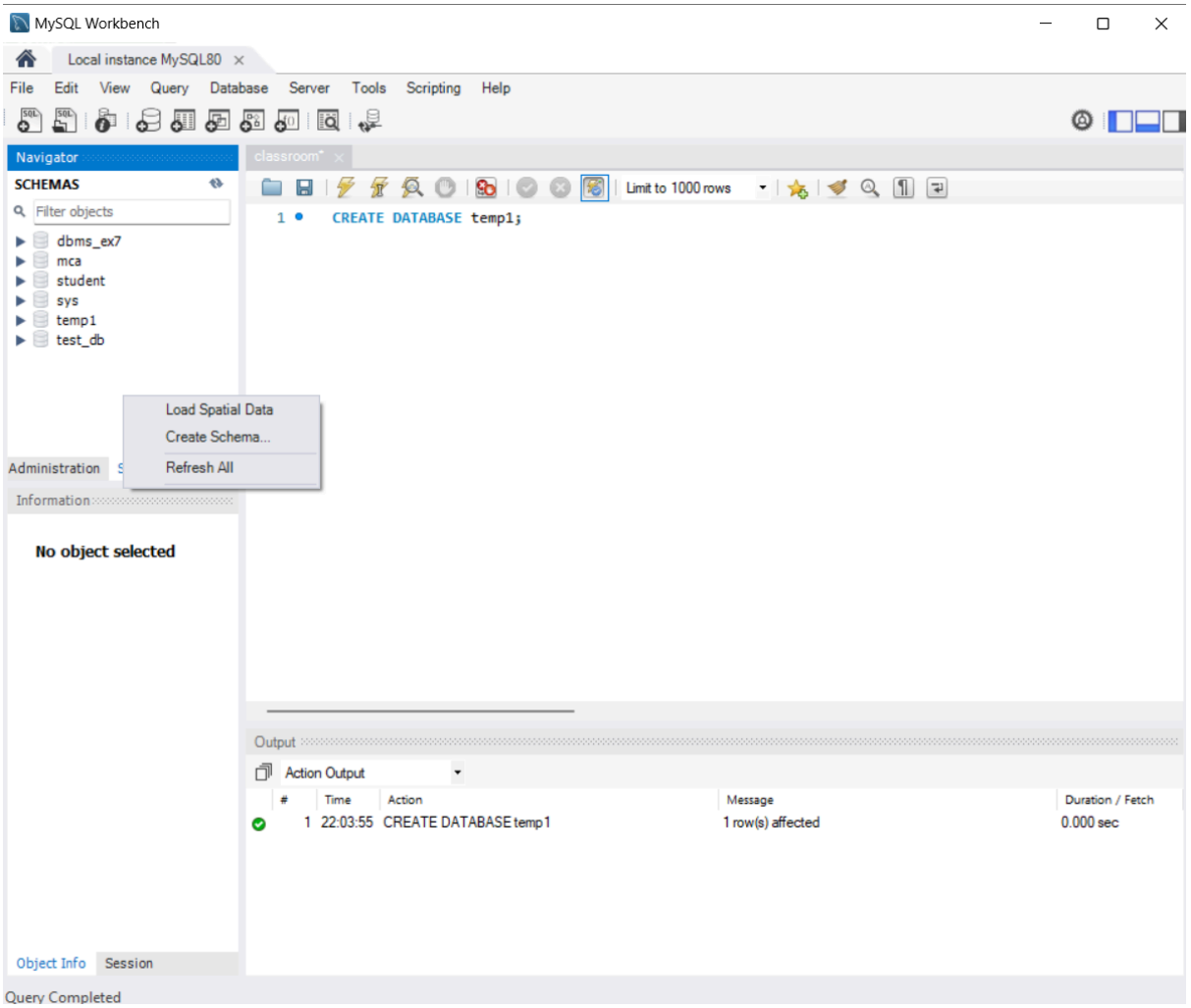


Create our First Database:-

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.

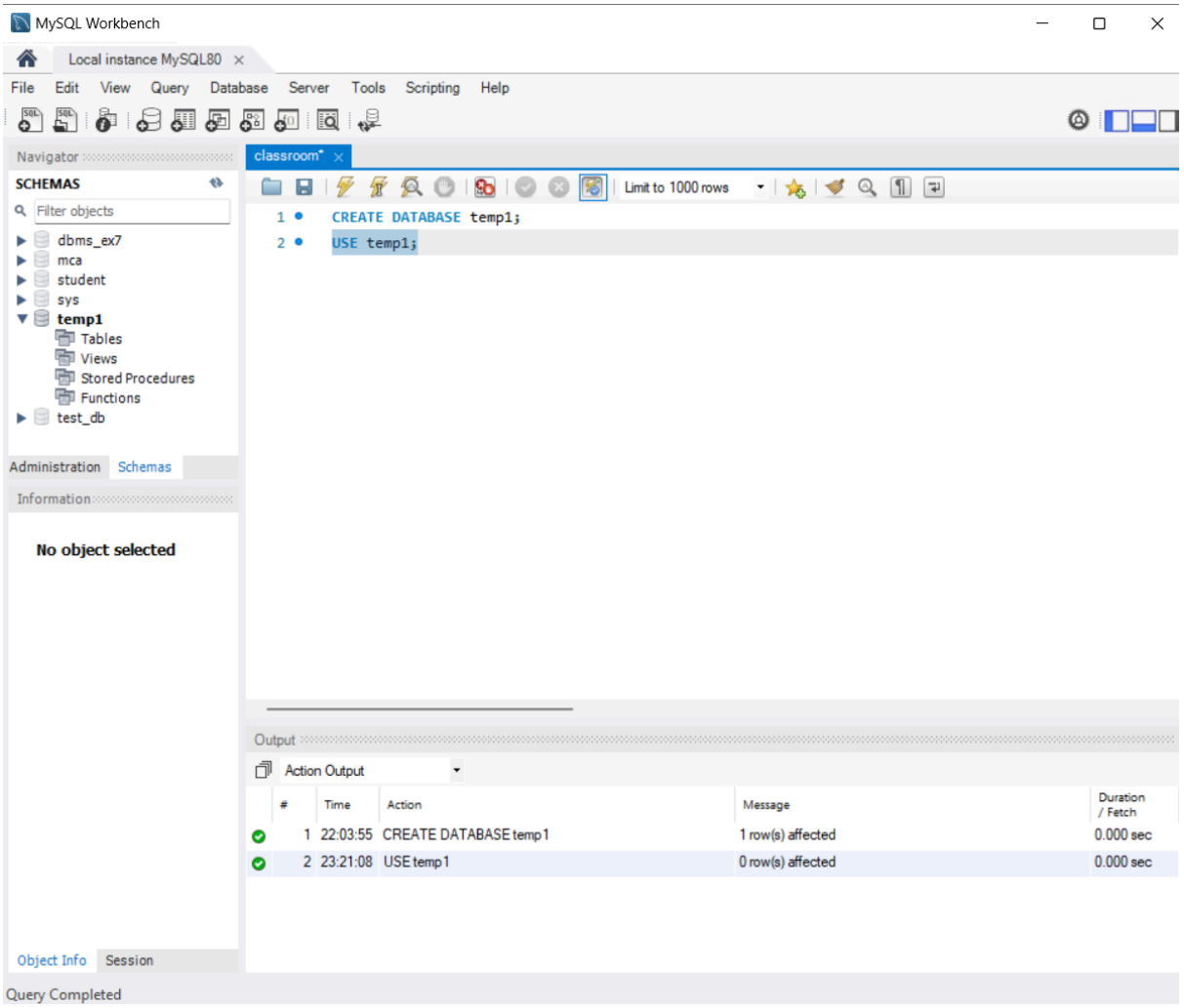
SQL is **not case sensitive** , hence we can write in small case letters also .
I.e. **create database temp1** is similar to **CREATE DATABASE temp1**

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 ,



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 .

Also we can insert values into the table by using …

INSERT INTO Syntax

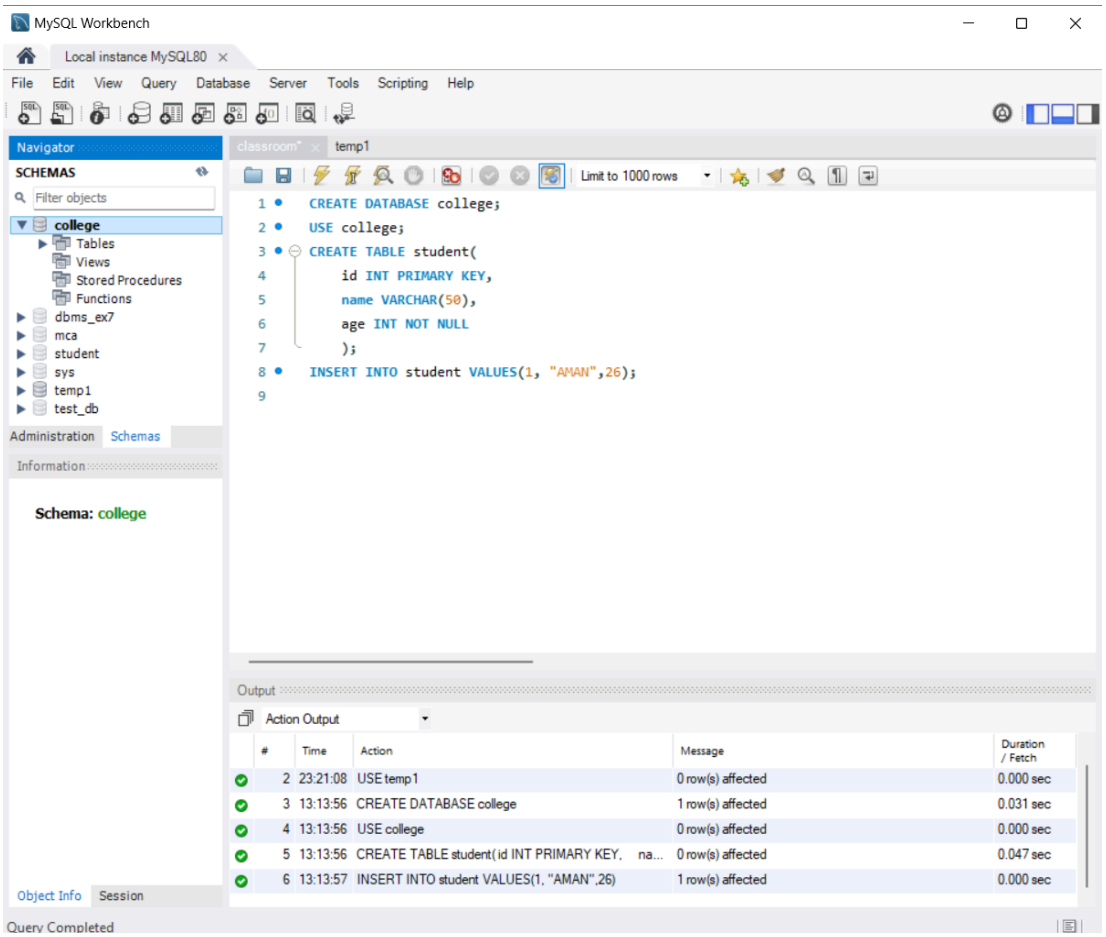
It is possible to write the `INSERT INTO` statement in two ways:

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 ,

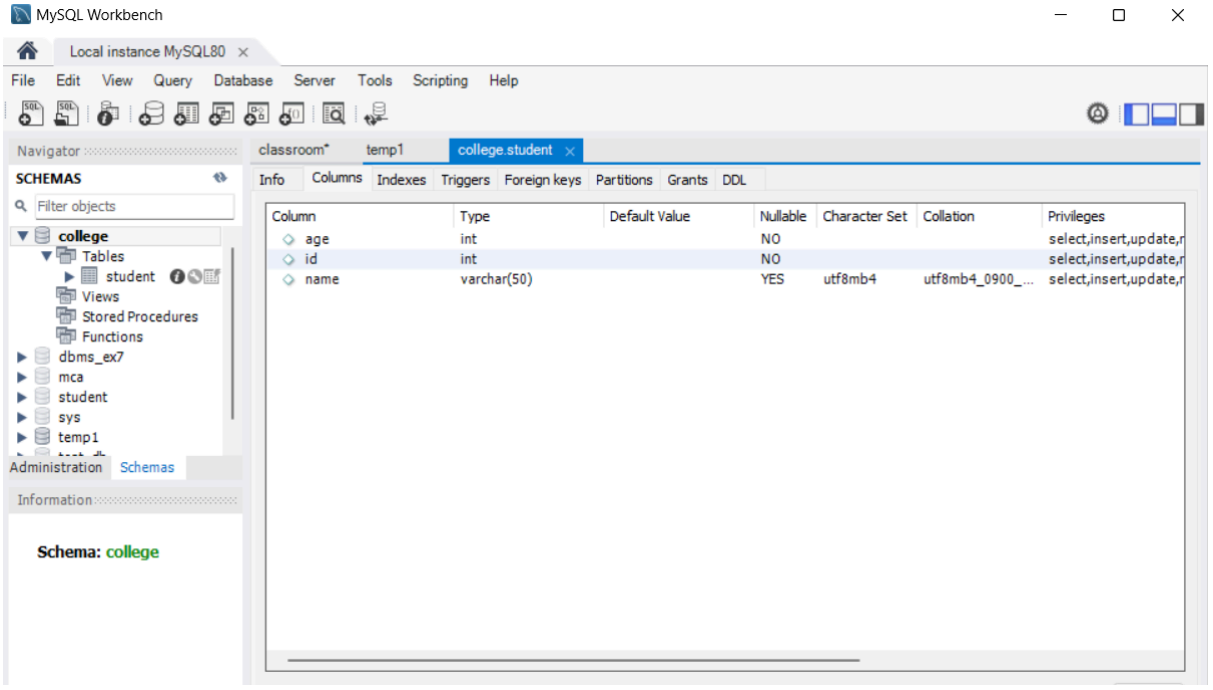
```
8 •  INSERT INTO student VALUES(1, "AMAN",26);
9 •  INSERT INTO student VALUES(1, "Shraddha",24);
```

o/p,

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

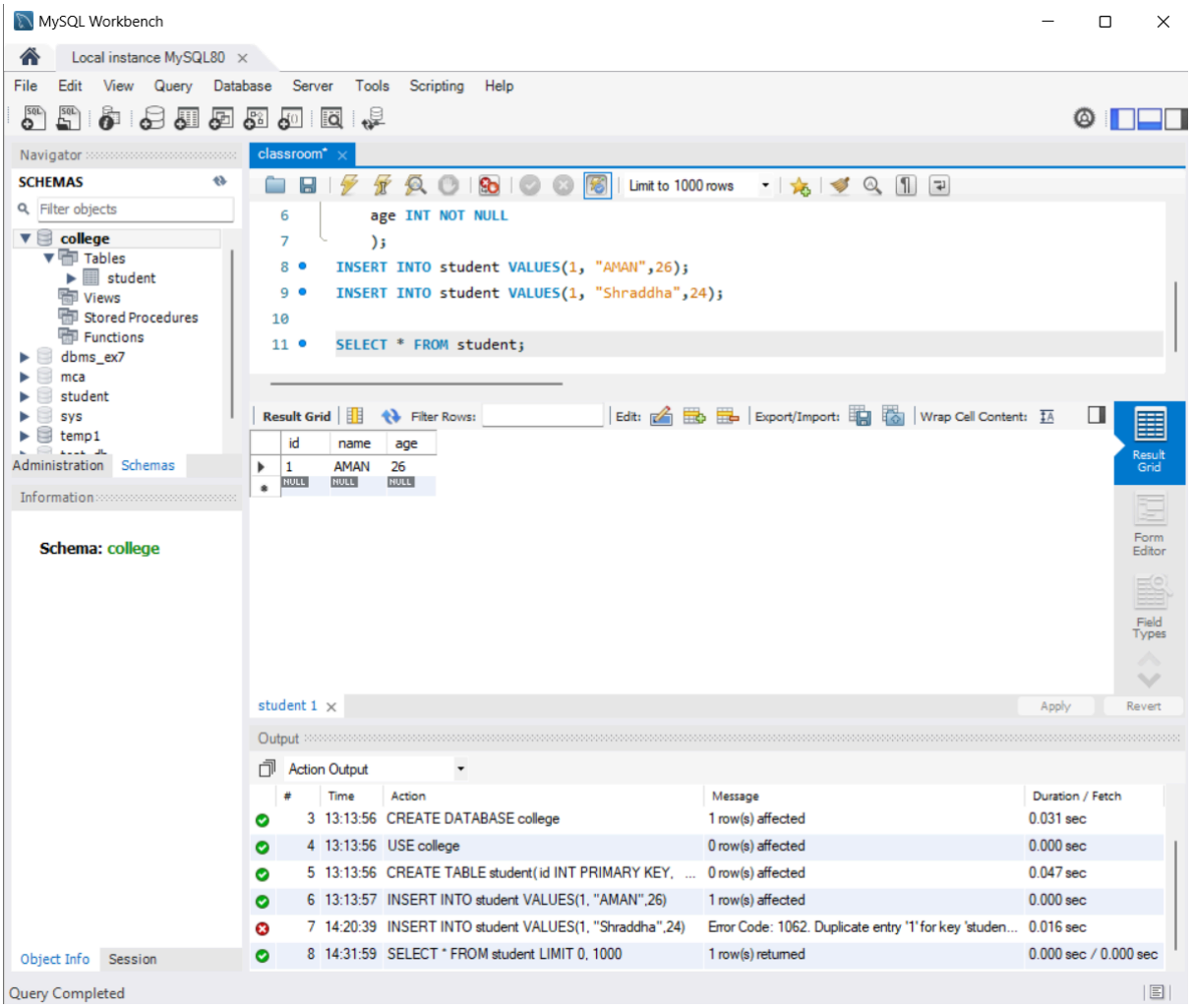
---

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



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;

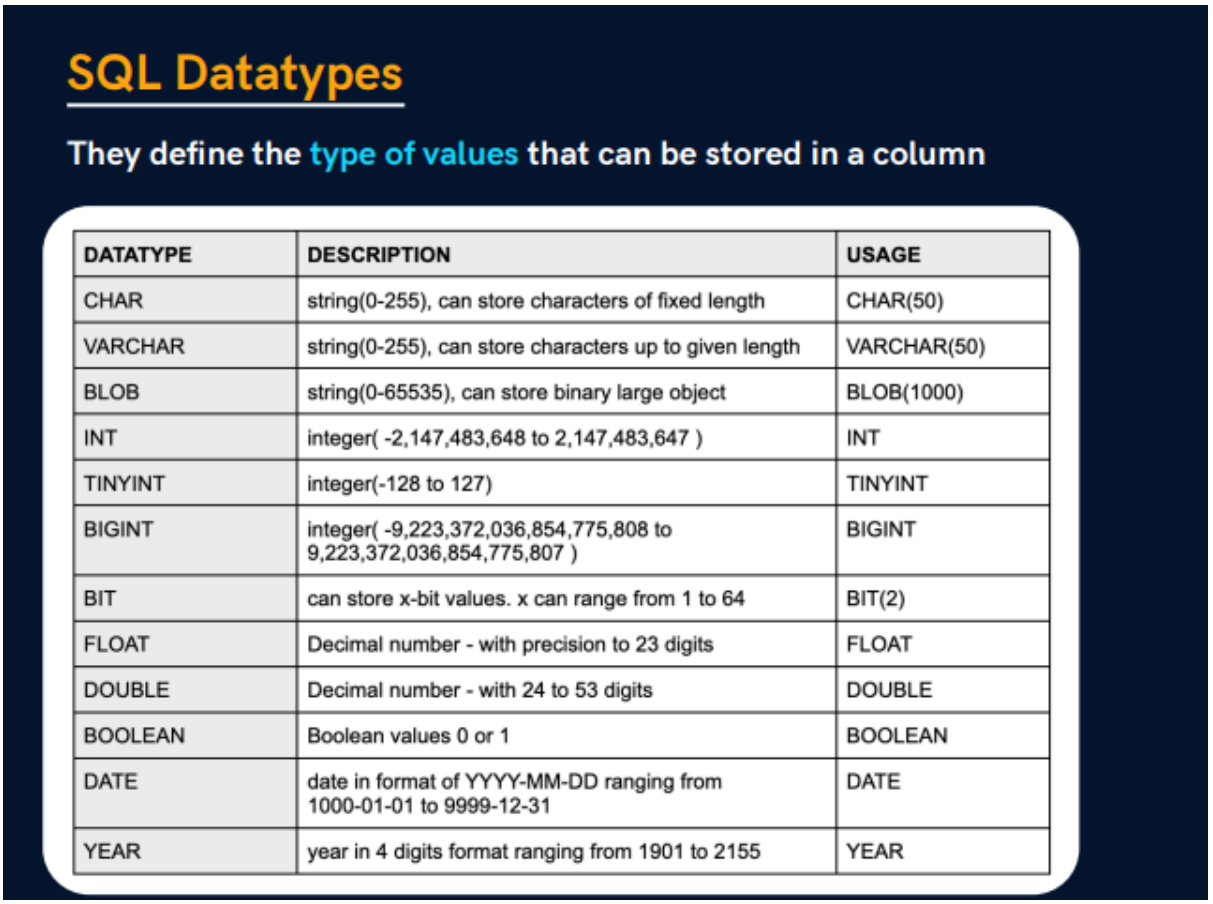


After executing we can see our whole table displayed .

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

SQL Datatypes:-

Most commonly used datatypes …



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 store 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**

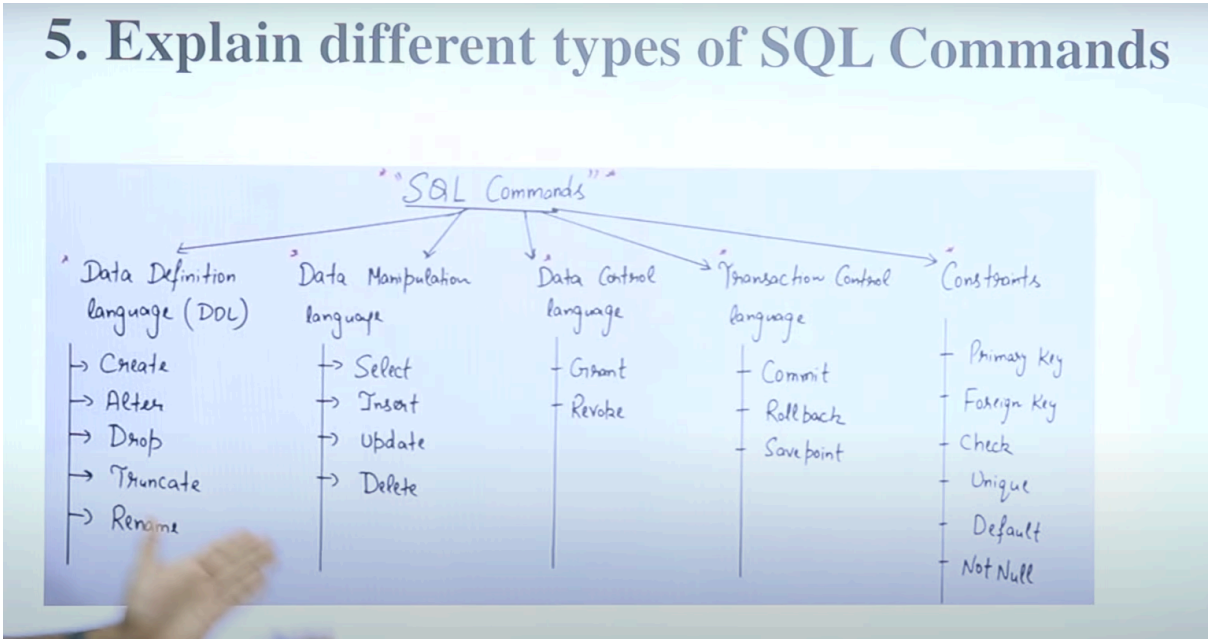
So the Datatypes which we use to store Numbers E.x. INT, FLOAT , DOUBLE and TINYINT. It can consist of negative as well as positive numbers .
So Numbers which may be **negative** or positive , such numbers we can say come under **SIGNED** Datatype i.e. It may have sign (**-** or **+**)
E.x TINYINT UNSIGNED (

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 .

Types of SQL Commands :-

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

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



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

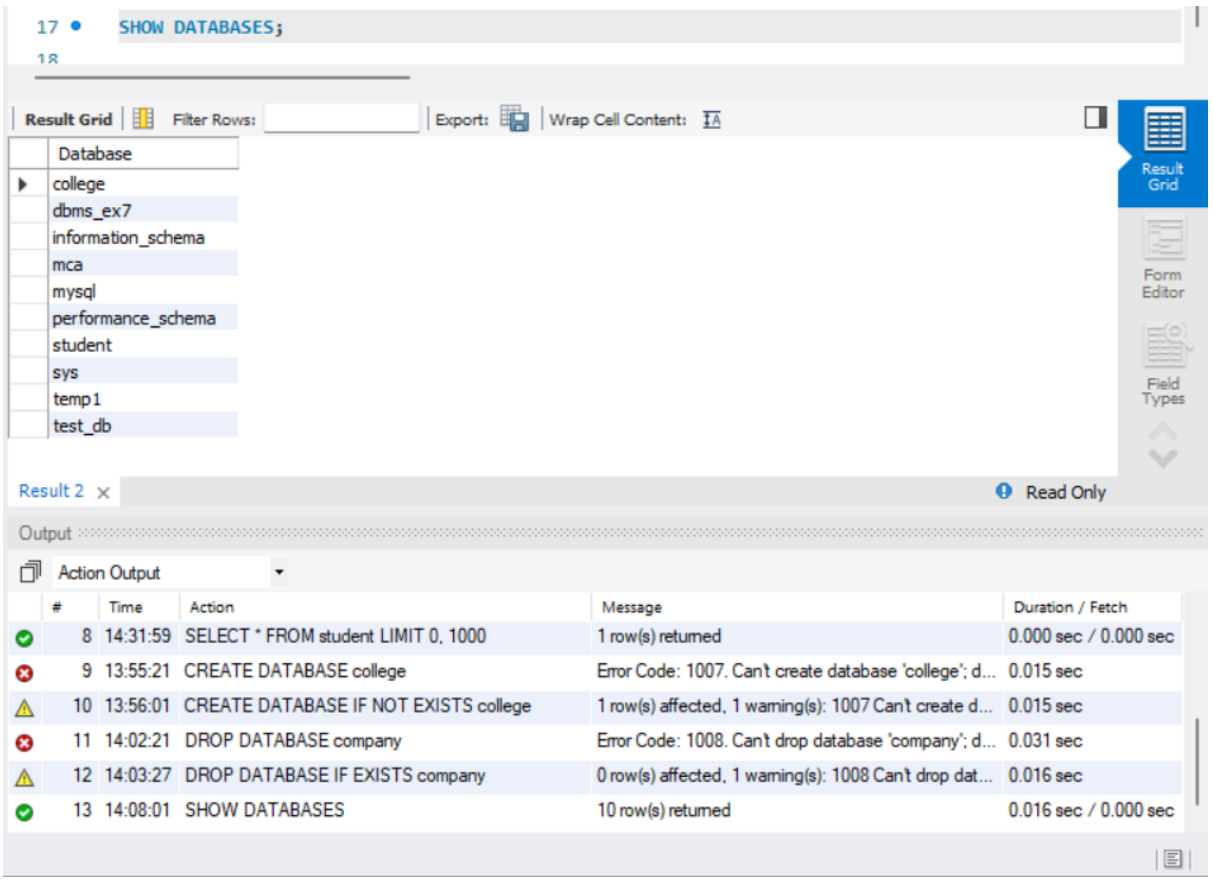
#	Time	Action	Message	Duration / Fetch
7	14:20:39	INSERT INTO student VALUES(1, "Shraddha",24)	Error Code: 1062. Duplicate entry '1' for key 'studen...	0.016 sec
8	14:31:59	SELECT * FROM student LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
9	13:55:21	CREATE DATABASE college	Error Code: 1007. 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,



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,

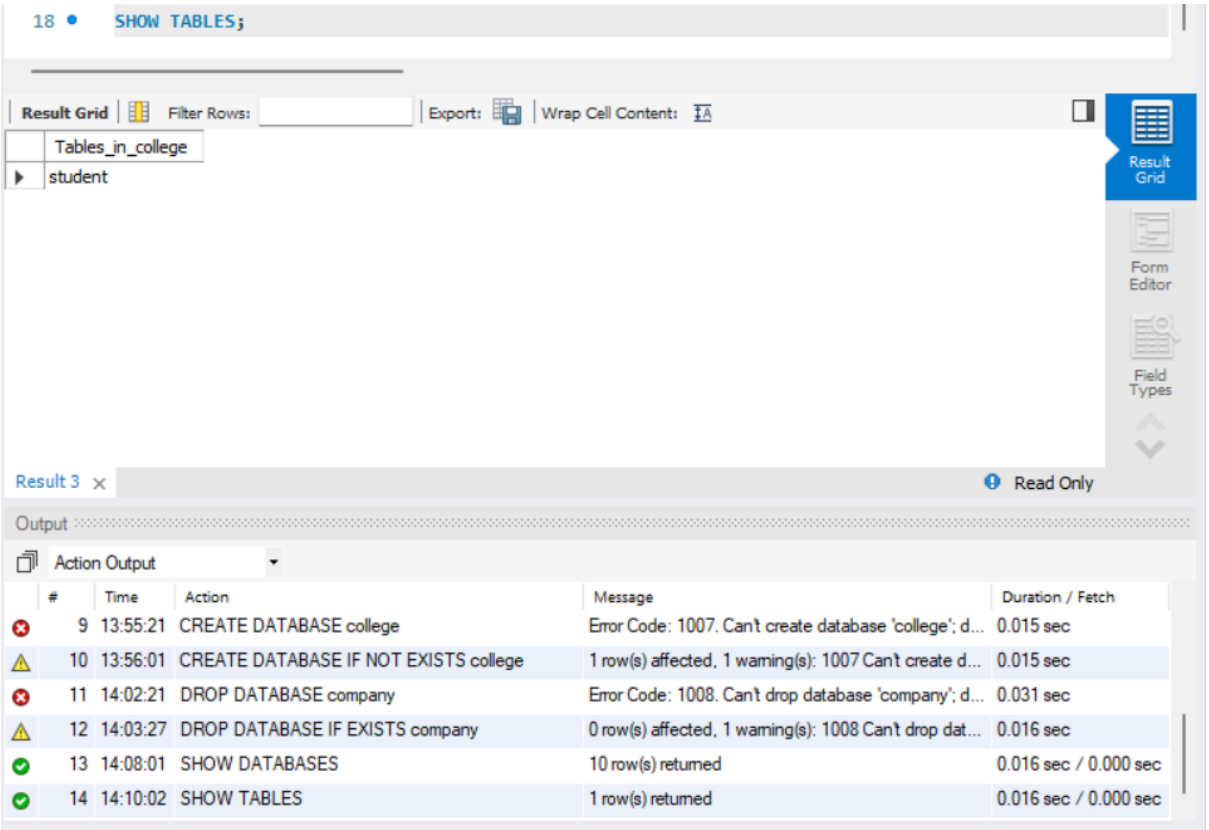


Table related Queries :-

To create a table , we use the CREATE command **.**
We already studied the syntax of table ,

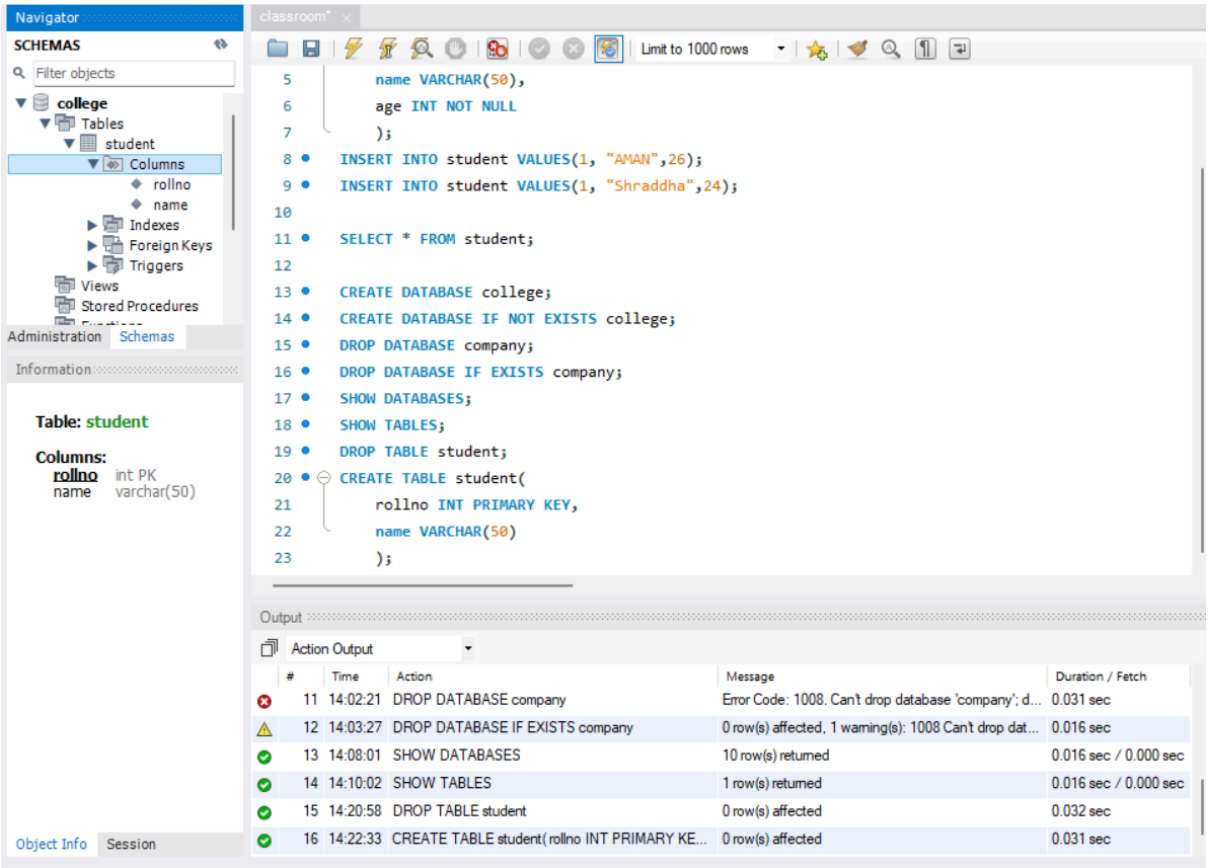
```
CREATE TABLE table_name (
   column_name1 datatype constraint,
   column_name2 datatype constraint,
);
```

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**



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.

Here * means **all .**
I.e select all rows and columns from Table.
EXample ,

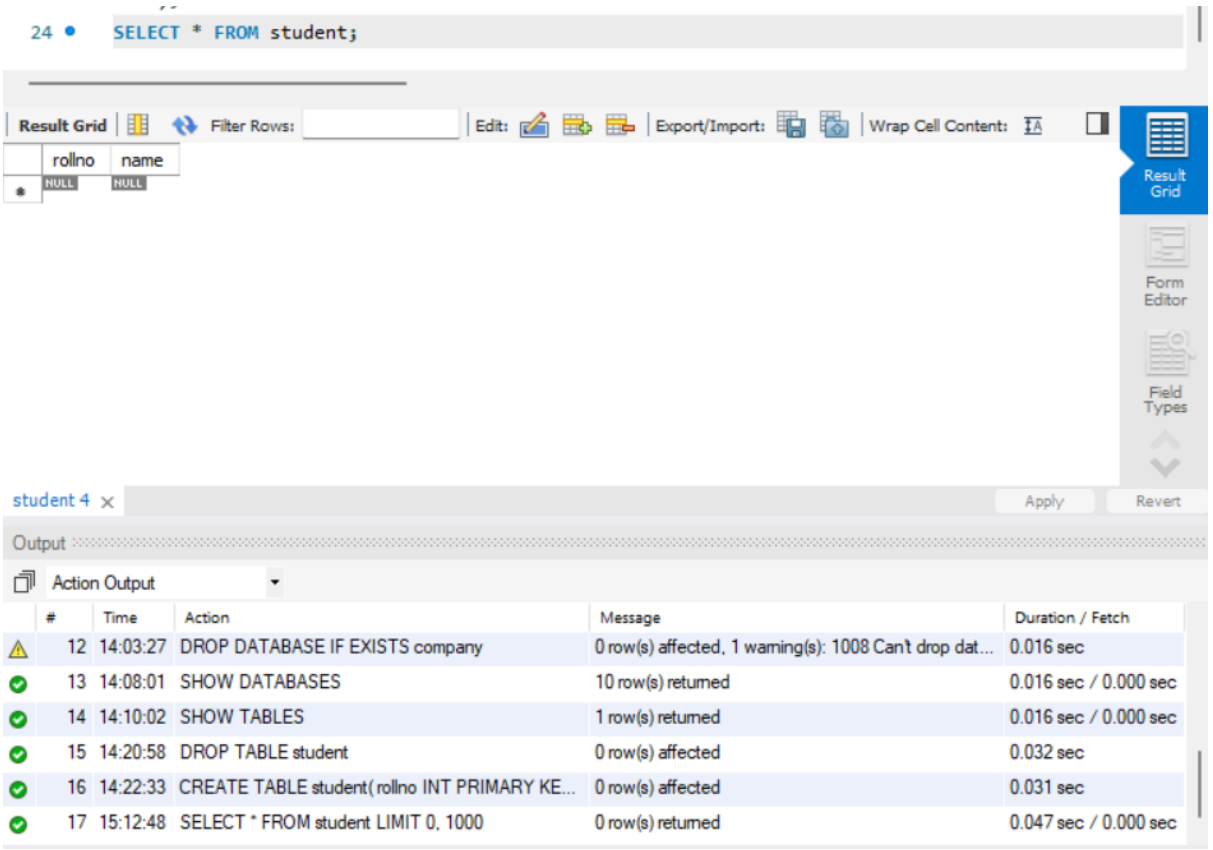
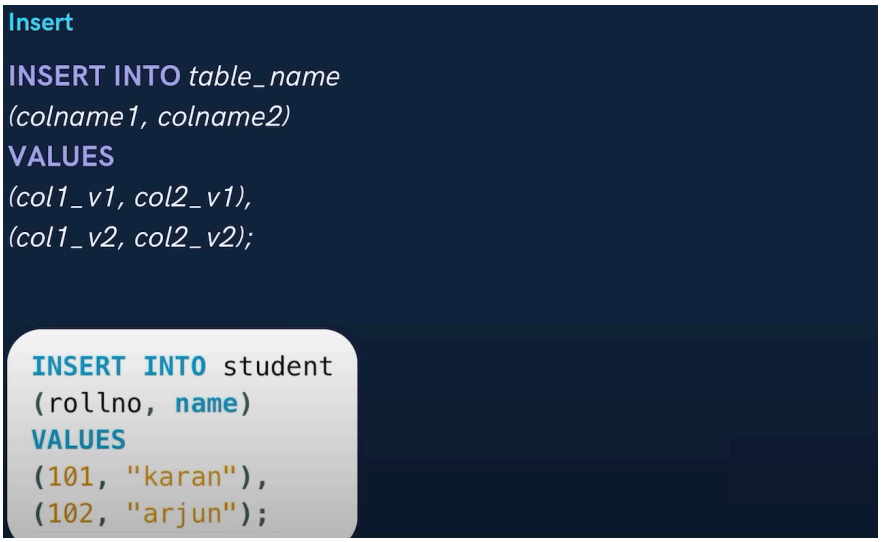


Table has Columns , but there is NO data in it .
So for inserting we know that ,we have to use the **INSERT** command .



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 Warnin...	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:

rollno	name
101	karan
102	arjun
NULL	NULL

Result Grid | Form Editor | Field Types

student 5 | Apply | Revert

Output

Action Output

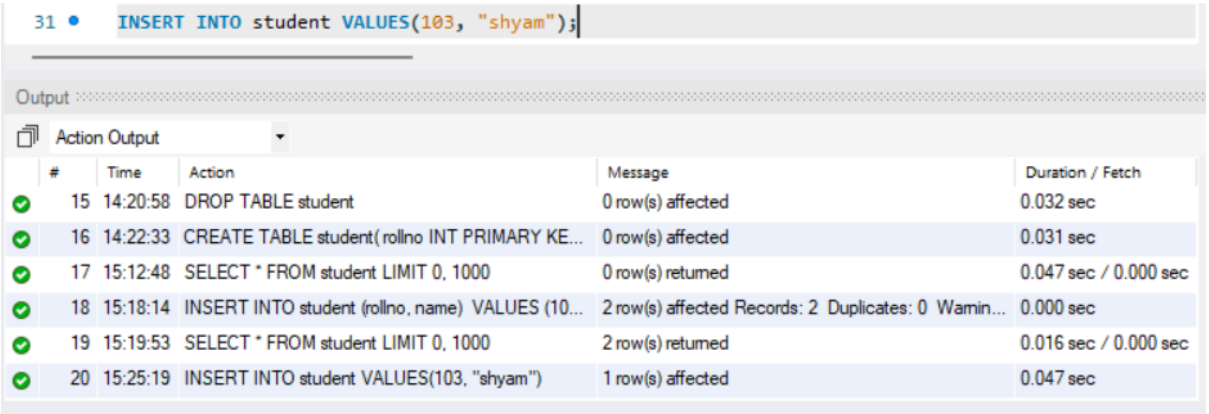
#	Time	Action	Message	Duration / Fetch
14	14:10:02	SHOW TABLES	1 row(s) returned	0.016 sec / 0.000 sec
15	14:20:58	DROP TABLE student	0 row(s) affected	0.032 sec
16	14:22:33	CREATE TABLE student(rollno INT PRIMARY KE...	0 row(s) affected	0.031 sec
17	15:12:48	SELECT * FROM student LIMIT 0, 1000	0 row(s)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

---

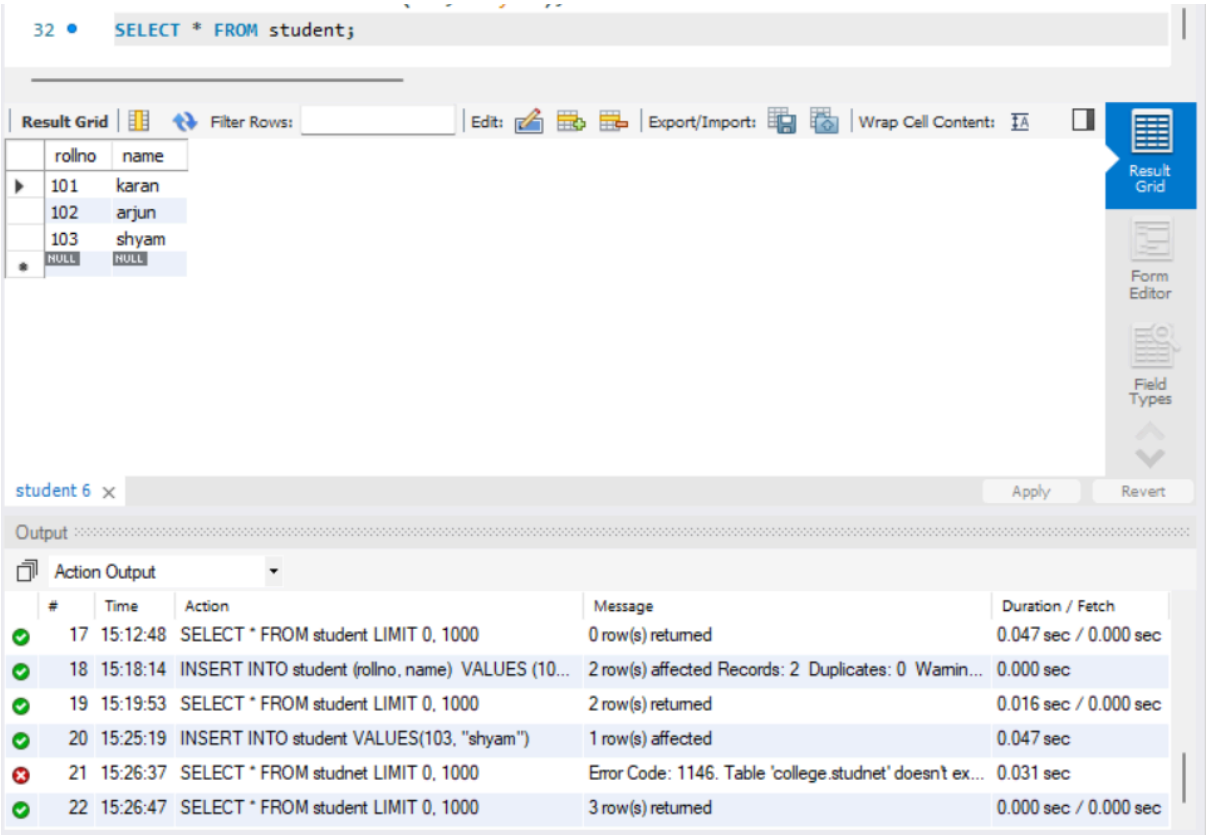
Instead of writing
INSERT INTO student(rollno, name) VALUES(101, “karan”), (102,”arjun”);
We can also write
INSERT INTO student VALUES(101, “karan”), (102,”arjun”);

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

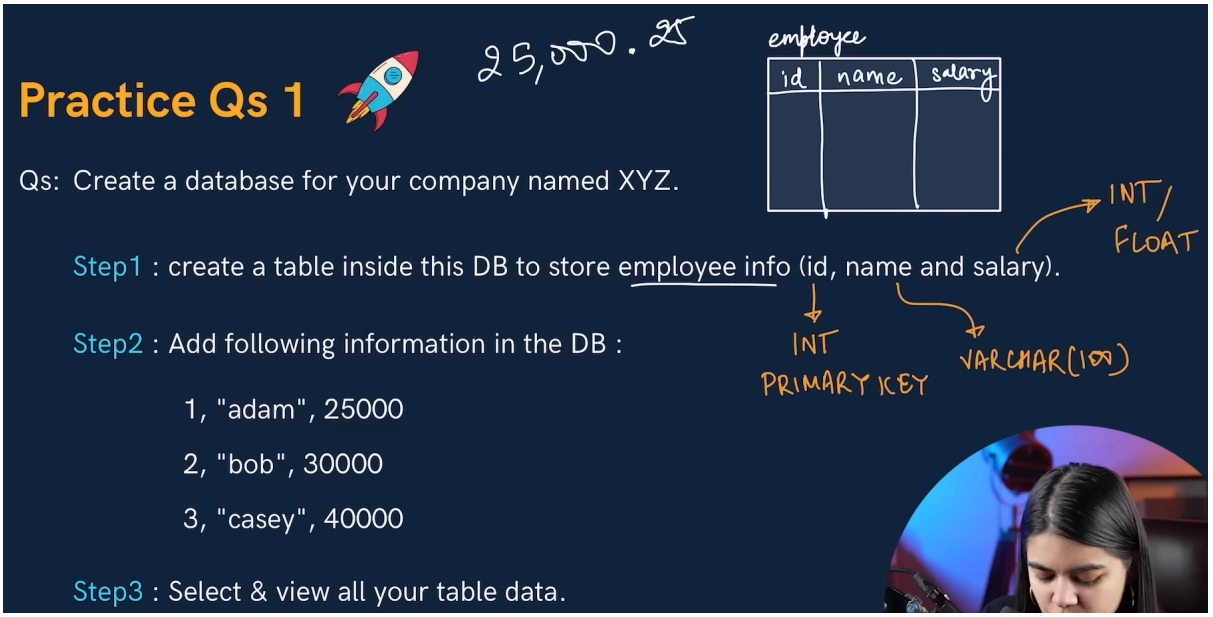
Example ,

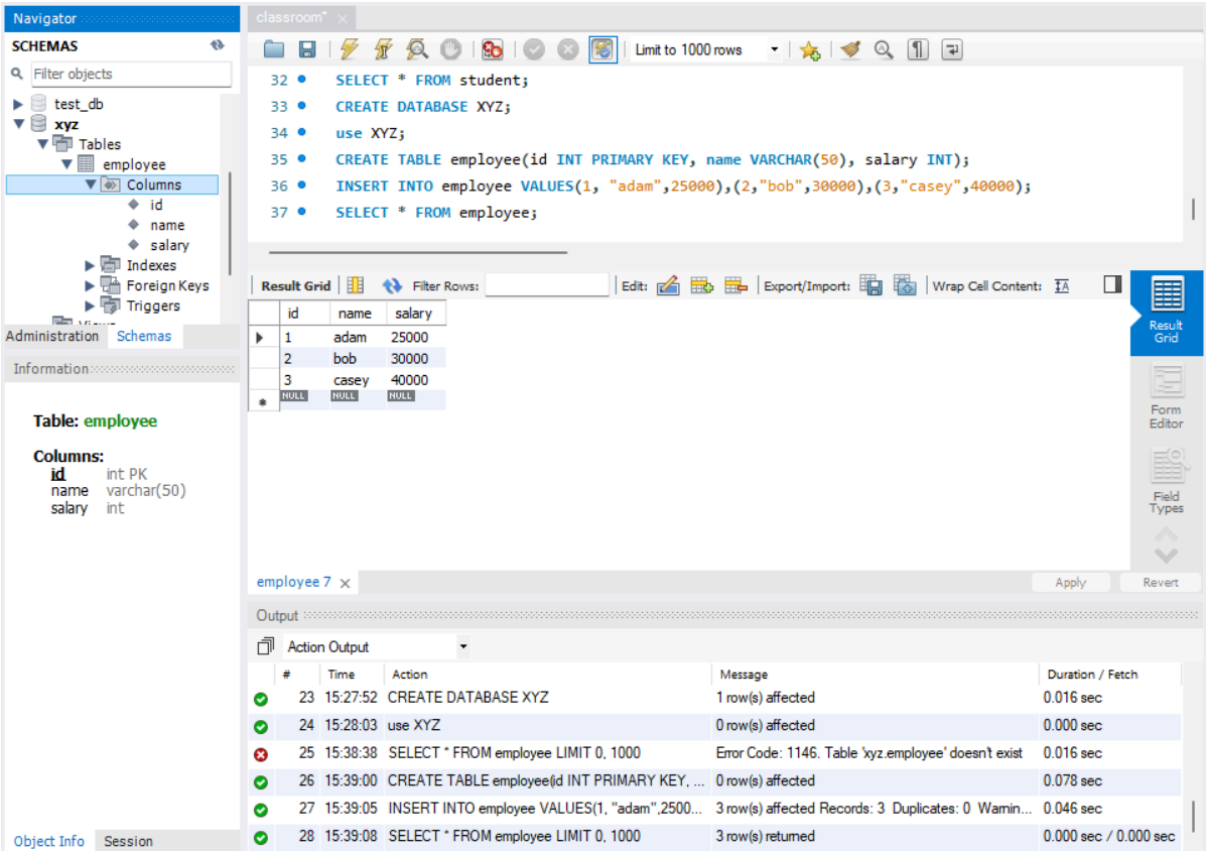


No lets display ,



Practice Question:





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.

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 ,

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

Output

Action Output

#	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 Warnin...	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)

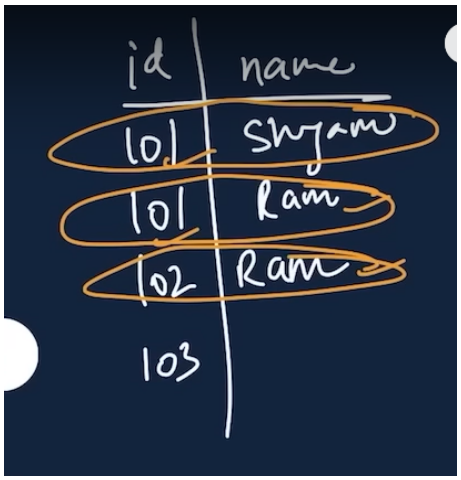
But the PRIMARY KEY constraint makes both **UNIQUE** and also **NOT NULL.**

```sql
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 ,

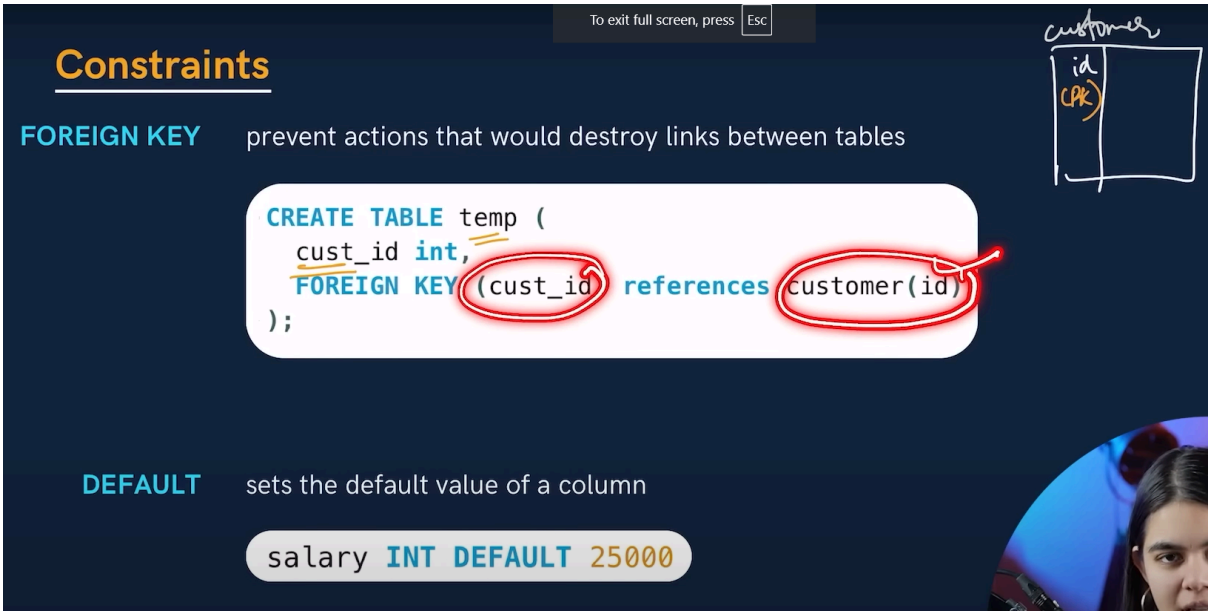
```sql
CREATE TABLE temp1 (
  id INT,
  name VARCHAR(50),
  age INT,
  city VARCHAR(20),
  PRIMARY KEY (id, name)
);
```

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 ,



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:-



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;
```

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

id	salary
101	25000

emp 2 × Read Only

Output

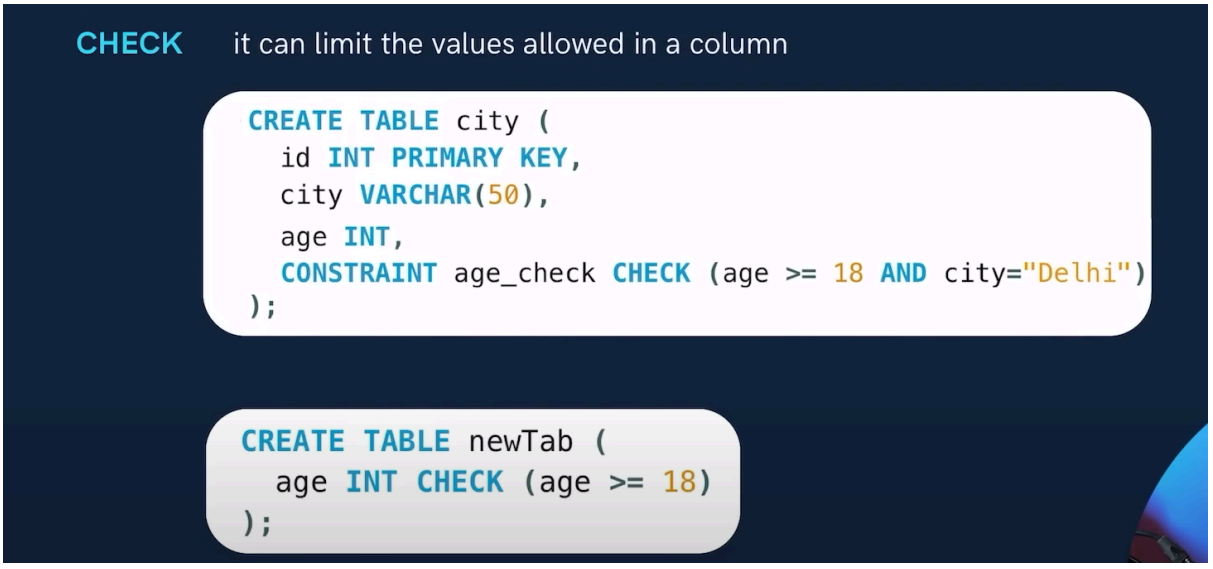
Action Output

#	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



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 ,

Code :-

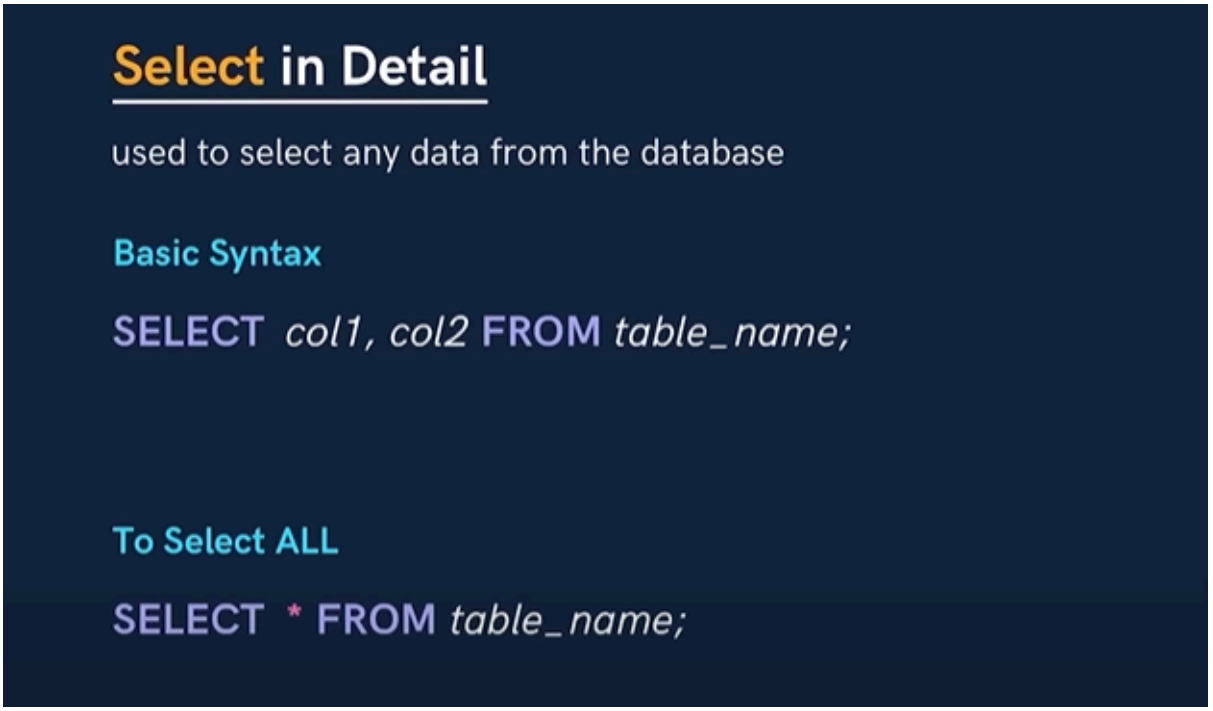
```
48 •  DROP DATABASE college;
49 •  CREATE DATABASE college;
50 •  USE college;
51 •  CREATE TABLE student( rollno INT PRIMARY KEY, name VARCHAR(50),marks INT NOT NULL,
52                          grade VARCHAR(1), city VARCHAR(20));
53 •  INSERT INTO student(rollno, name, marks, grade, city) VALUES (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

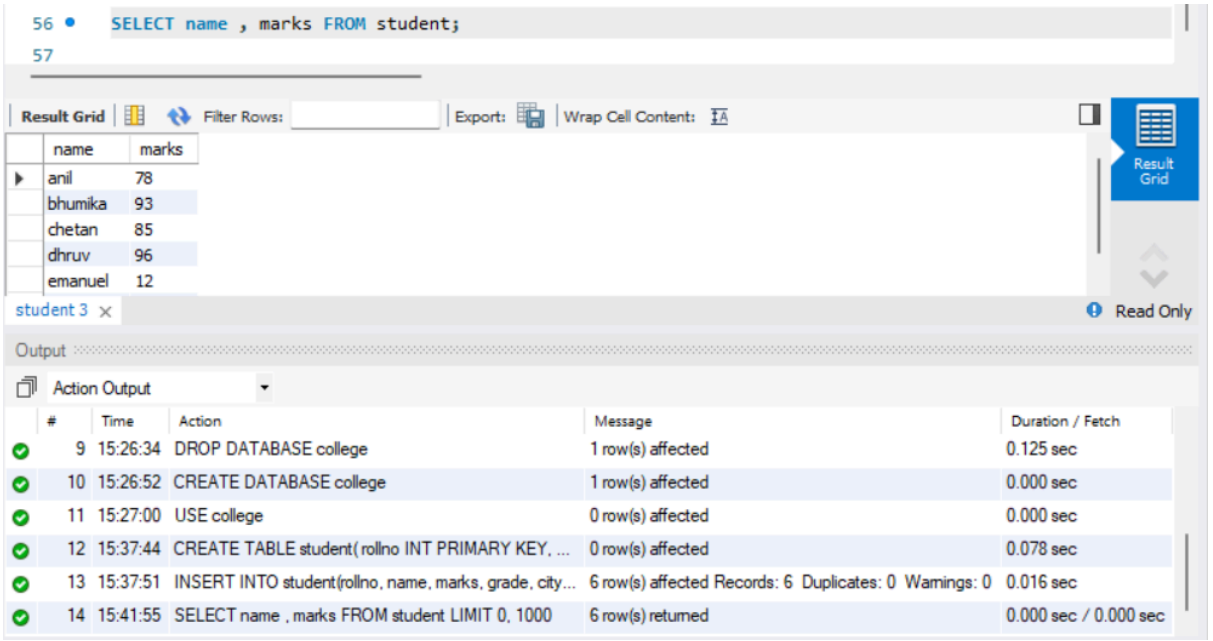
Action Output

#	Time	Action	Message	Duration / Fetch
8	15:00:02	CREATE DATABASE college	Error Code: 1007. Can't create database 'college'; data...	0.000 sec
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

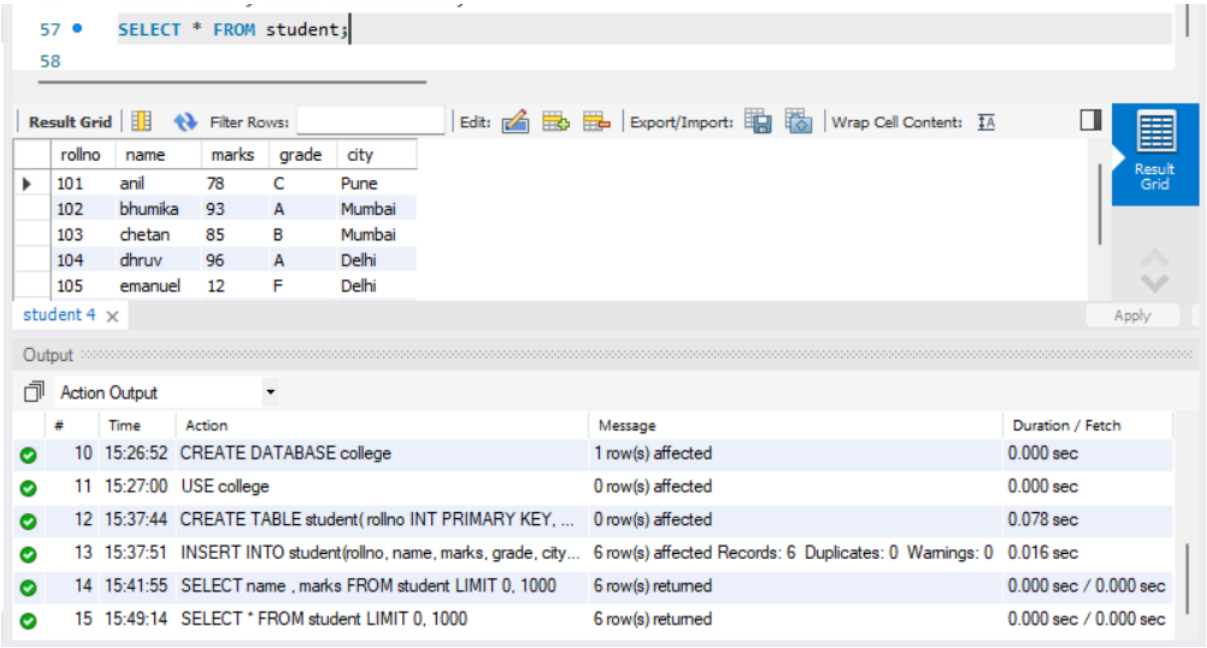
Select in Detail



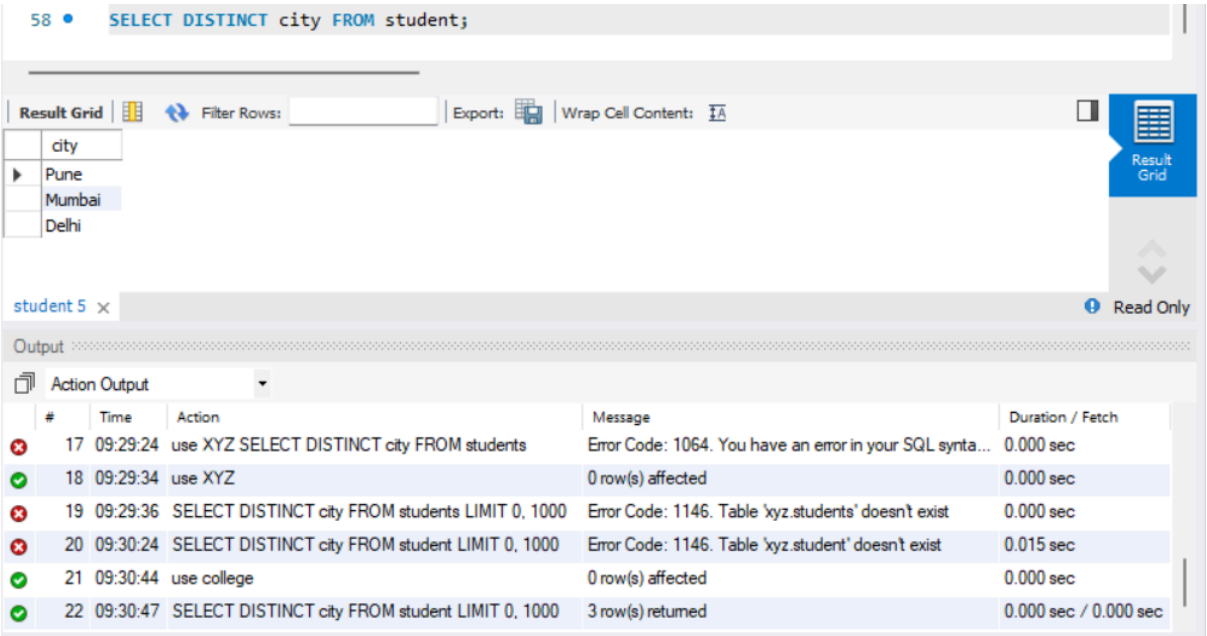
Example,
Suppose we want to see **name** and **marks** of **student** table .



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



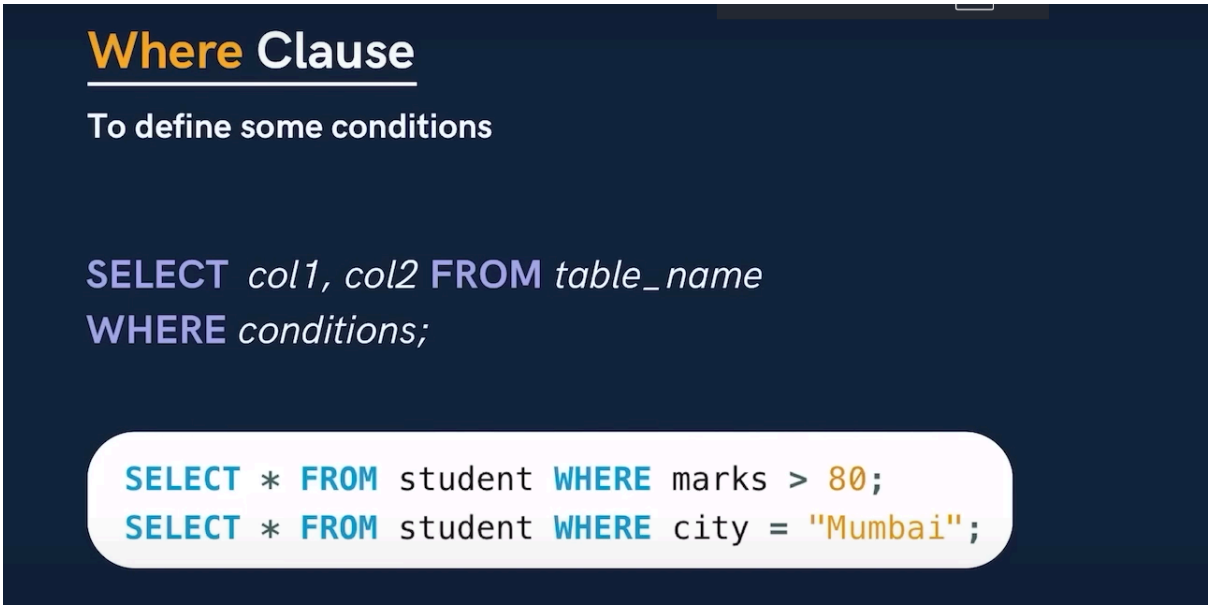
Suppose we want to display unique values of the table from any particular row, For that we can use our special keyword i.e. DISTINCT.(this means unique i.e. duplicate values wont display , only unique values will display)
Example,
SELECT DISTINCT city FROM student;



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 .

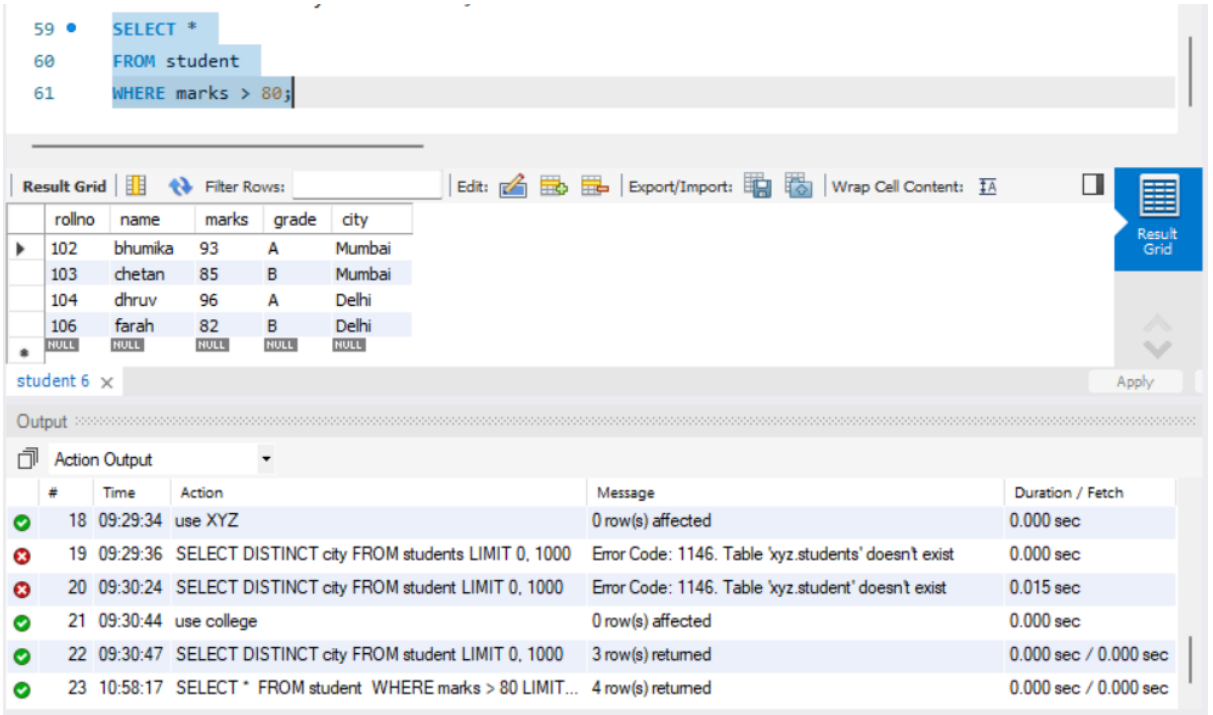
Where Clause

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.



As we can see here in example ,

1. SELECT * FROM student **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 contion 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

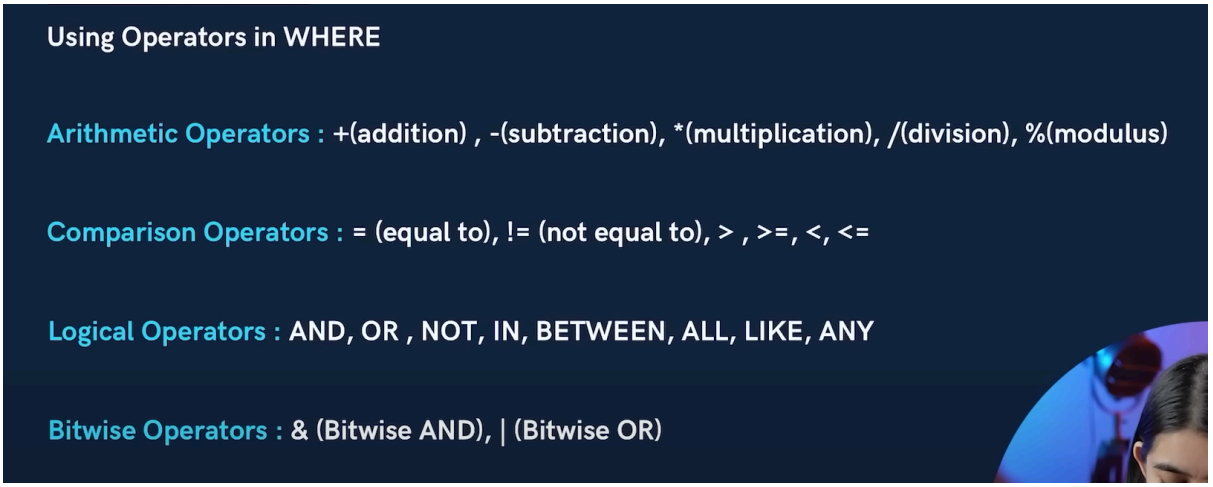
student 8 | Apply

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 .



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**

```
68 •  SELECT * FROM student WHERE marks+10 >100;
```

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

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

student 9 | Apply

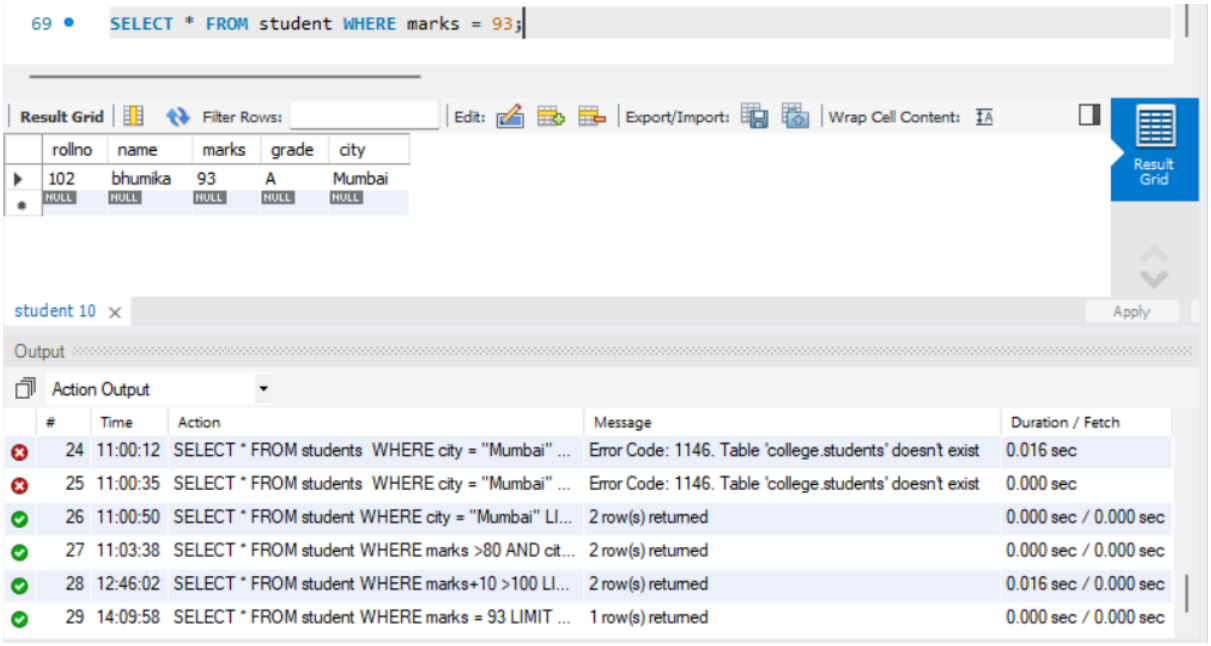
Output

Action Output

#	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

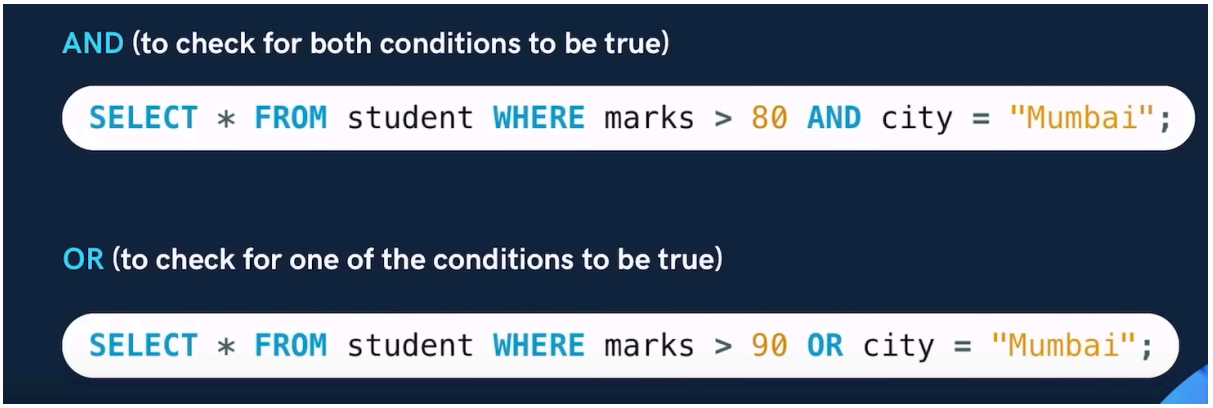
Hence we got list of students whose marks can go above 100 , if we add 10 in it.

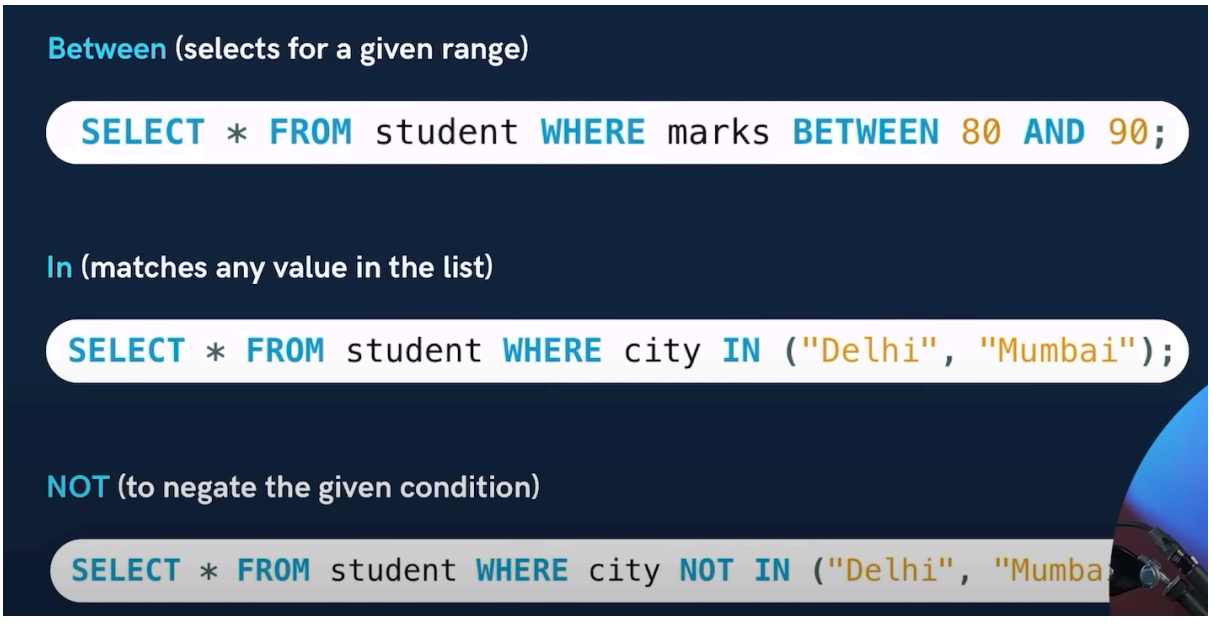
2. Suppose we want to check the students whose marks is equal to 93



Here we got a students with marks 93 .

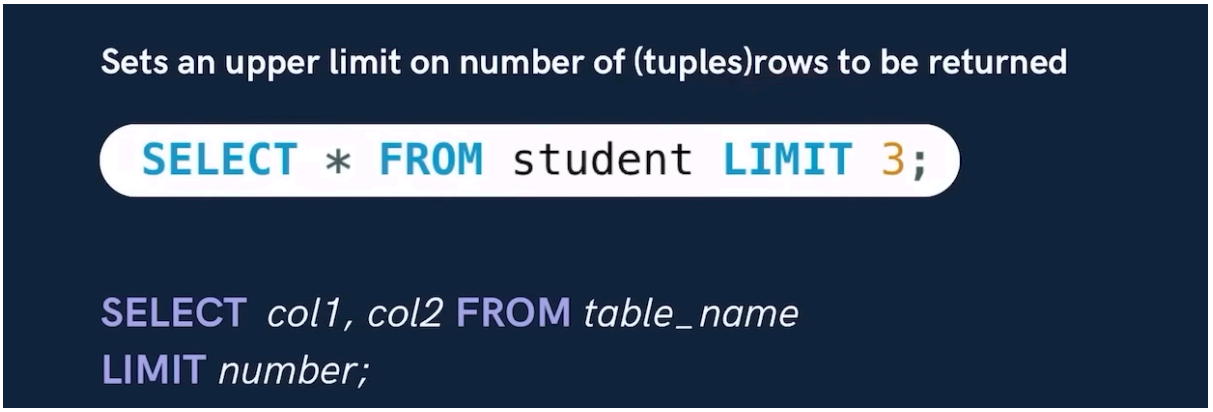
Now lets learn **logical operators** in detail ,



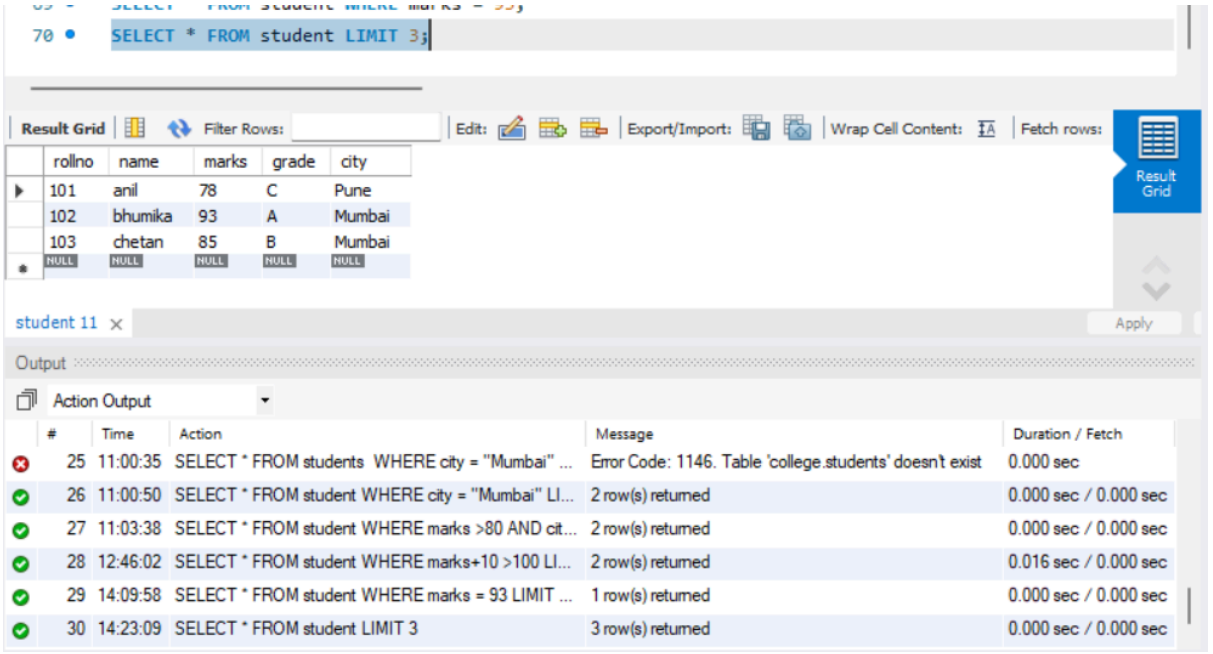


Here **IN** will give the list of students who belongs to one of the city gie=vn in list , and **NOT IN** will give the results of the students who does NOT belongs to any of the cities given in the list .

Limit Clause



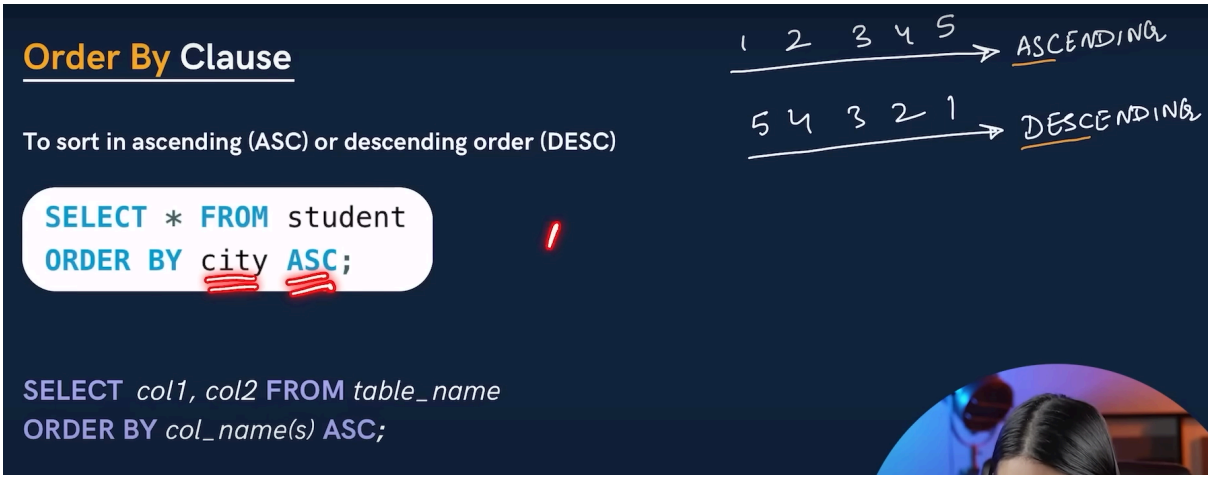
Suppose we want just 5 students of data , or 2 students data . then in such cases we can use **LIMIT** to set an upper limit on number of rows to be returned .
Example,
SELECT * FROM student **LIMIT 3**;



Now here is the data of the first 3 students .

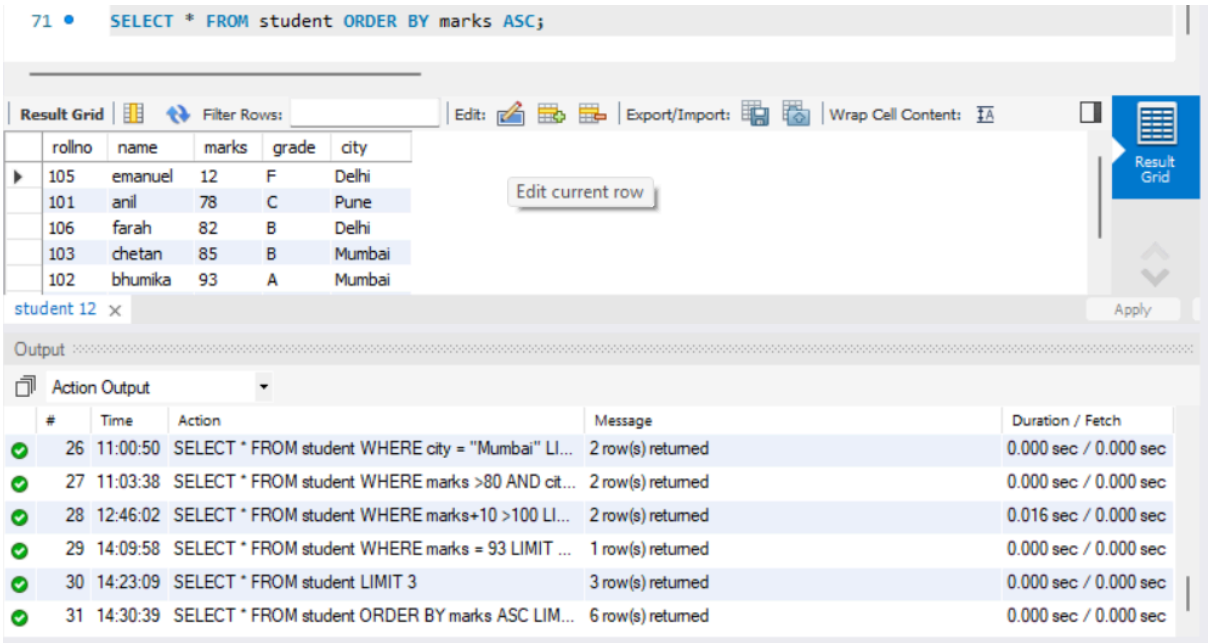
ORDER BY Clause

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)



Example ,
Lets display students by their marks order in ascending .

SELECT *
FROM student
ORDER BY marks ASC



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

rollno	name	marks	grade	city
104	dhruv	96	A	Delhi
102	bhumika	93	A	Mumbai
103	chetan	85	B	Mumbai
NULL	NULL	NULL	NULL	NULL

student 13 | Apply

Output

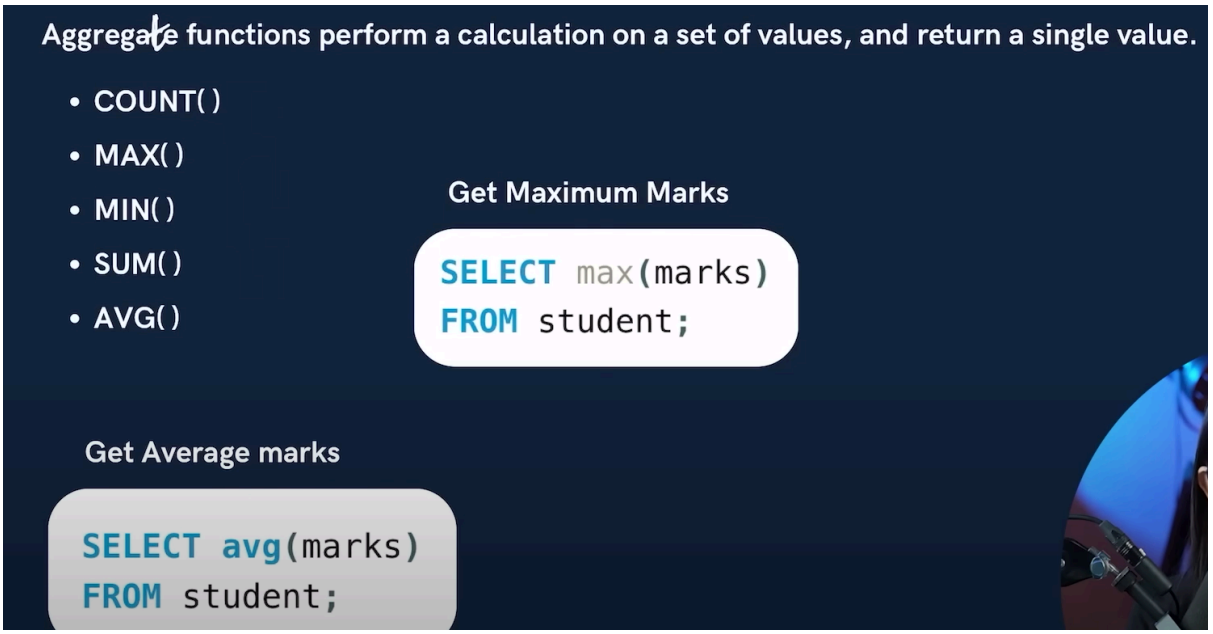
Action Output

#	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

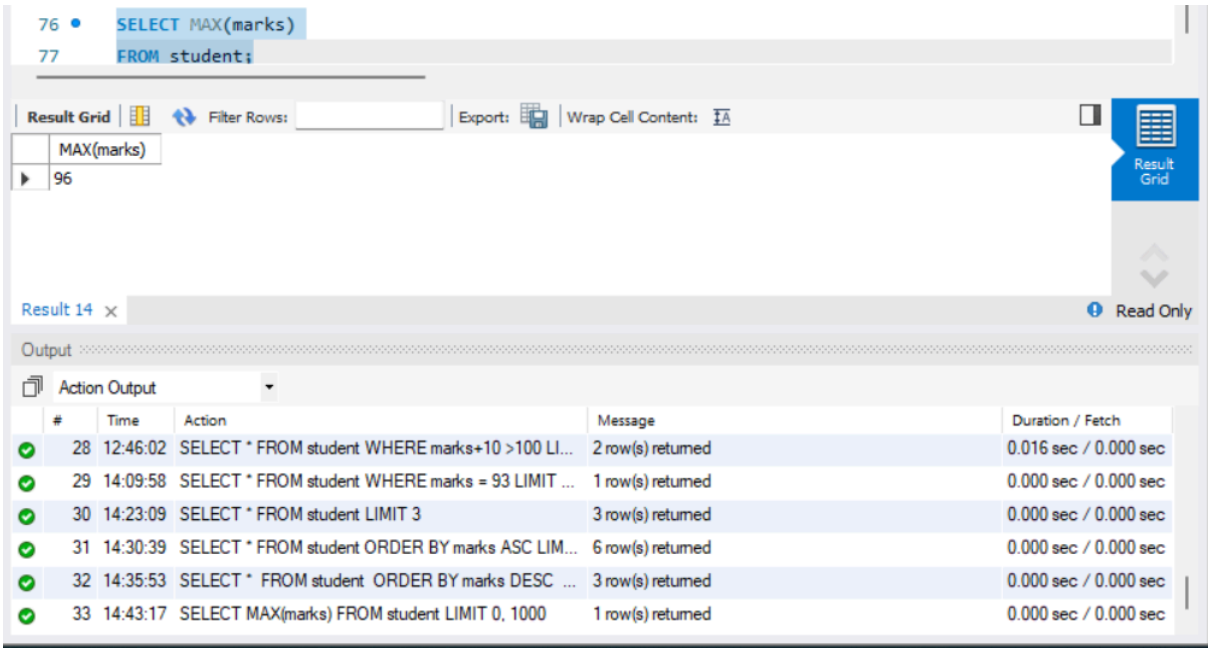
We got top 3 students in class .

Aggregate Functions:-

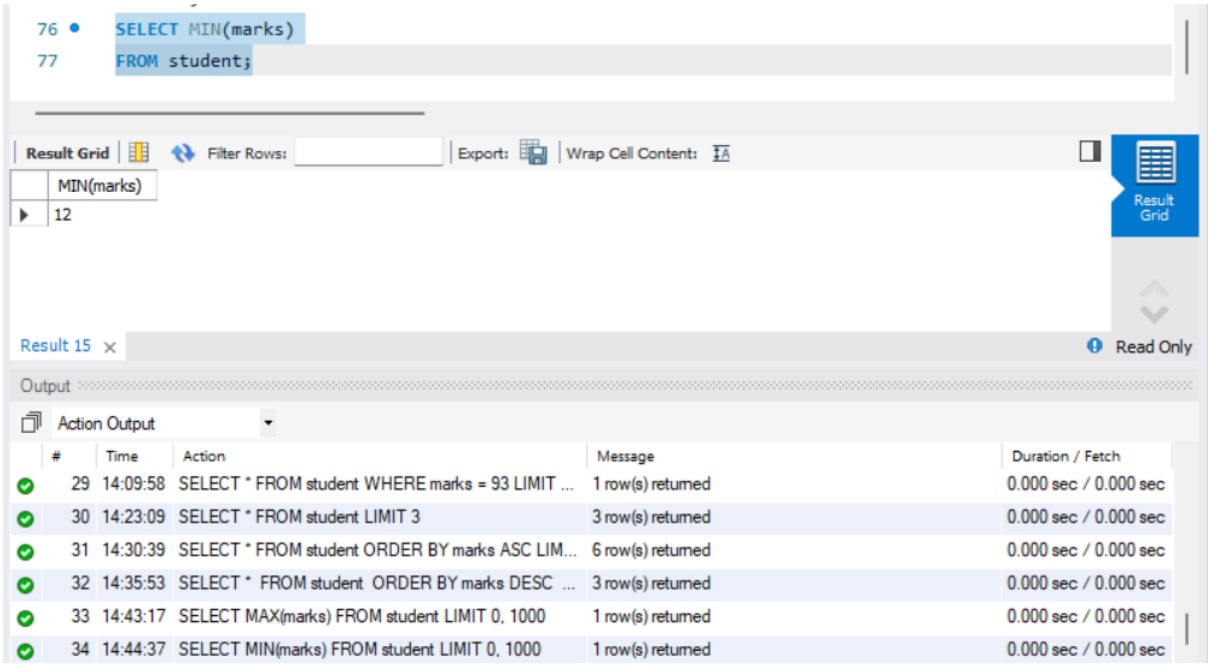
Functions are basically a structure with logic , and it perform the operations and gives us output.
Aggregate means combining things together , So aggregate functions take i/p more than 1 , and return a single value .
Here are some popular aggregate functions



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



Similarly we can check **minimum marks** ,



Average of marks

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

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

AVG(marks)
74.3333

Result 16 | Read Only

Output

Action Output

#	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

Here we got the class average …

Now Suppose we want to count number of students ,

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

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

COUNT(rollno)
6

Result 17 | Read Only

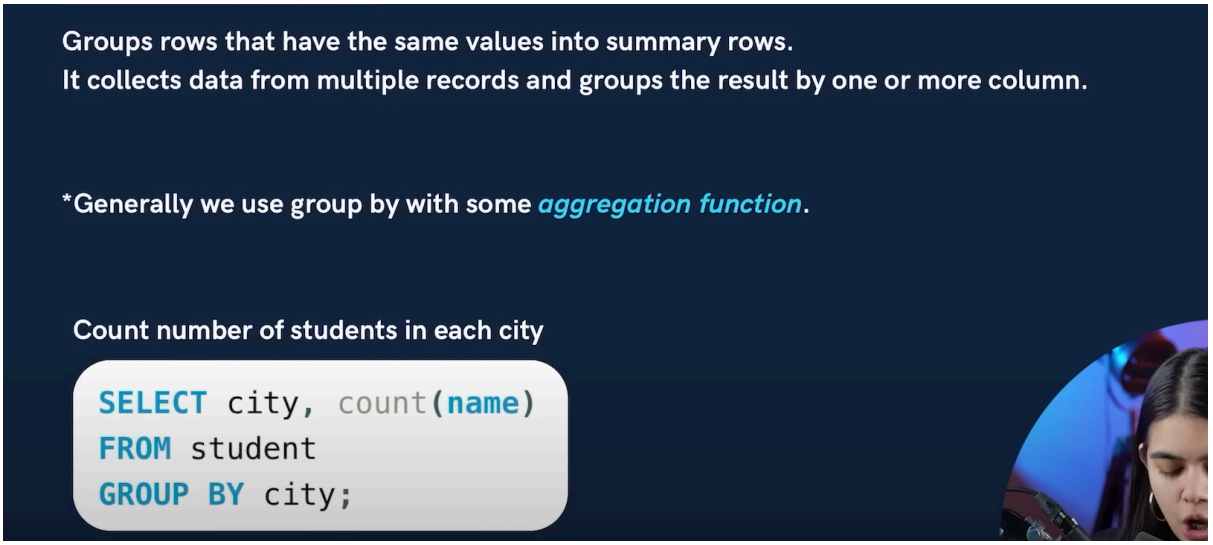
Output

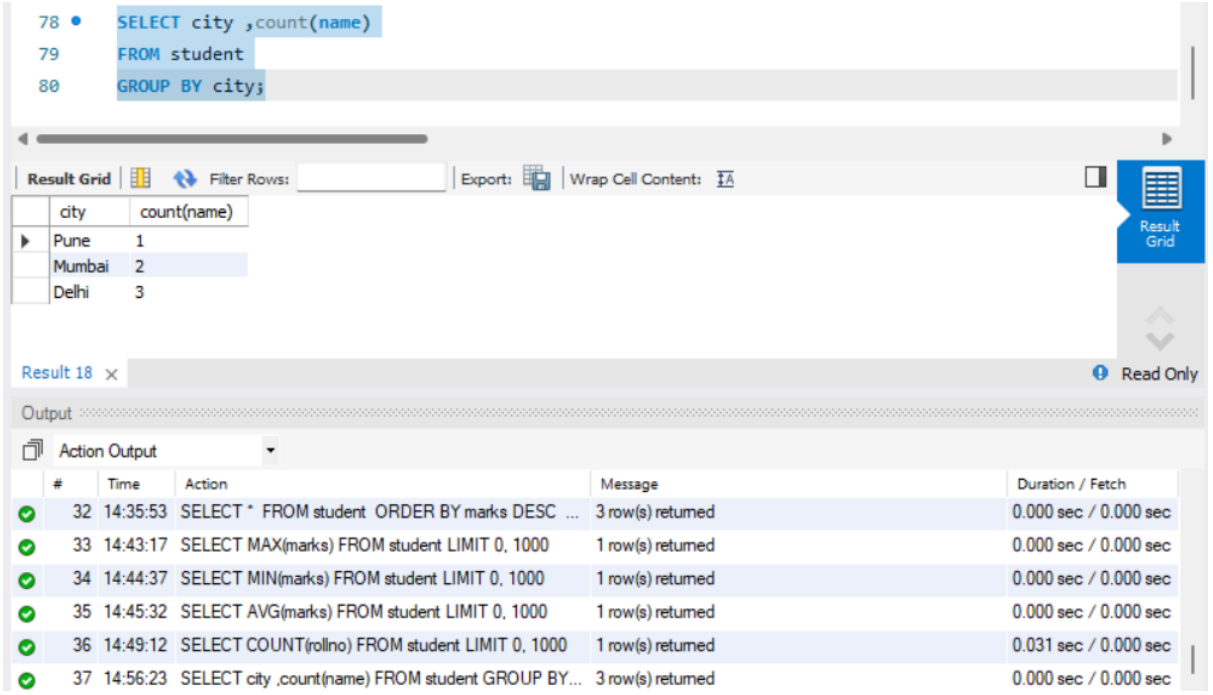
Action Output

#	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

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

GROUP BY Clause :-





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 .**

To exit full screen, press Esc

Practice Qs

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;
```

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

city	AVG(marks)
Delhi	63.3333
Mumbai	89.0000
Pune	78.0000

Result Grid

Result 19 ×

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 synta...	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 .**

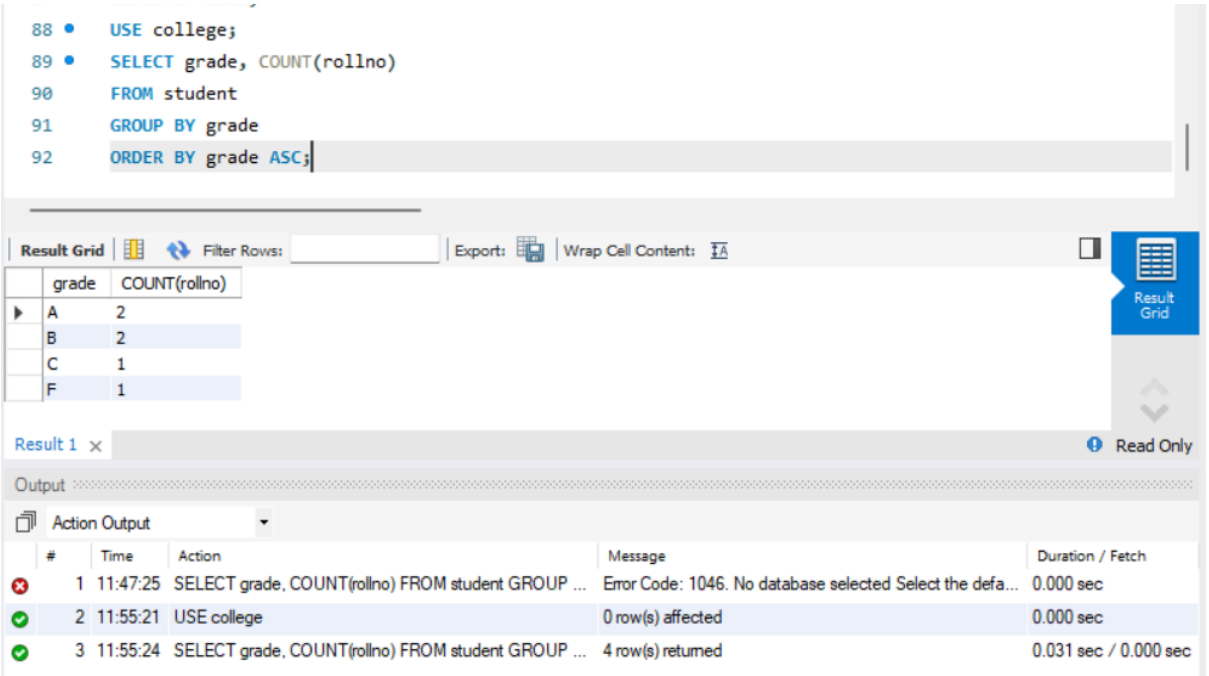
SELECT mode, count(customer)
FROM payment
GROUP BY mode;

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..

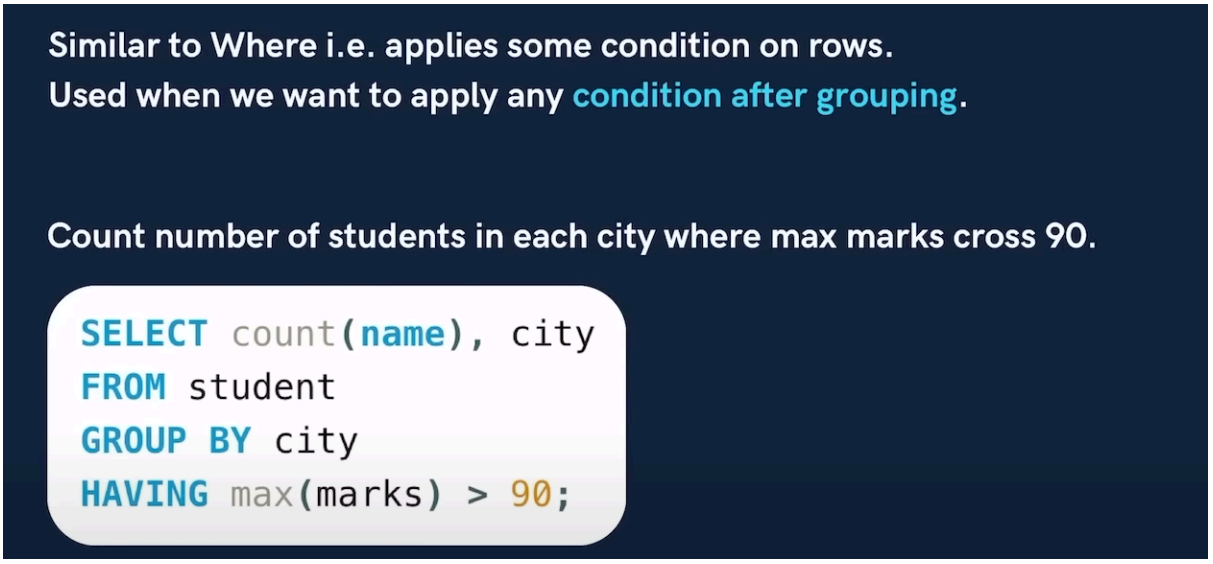
SELECT grade, count(rollno)
FROM student
GROUP BY grade
ORDER BY grade ASC;

SO here we have counted student **rollno**, we can also count **name**
It doesnt matter , we just have to count the students .
Since **rollno** is unique , so we mostly prefer that for counting .

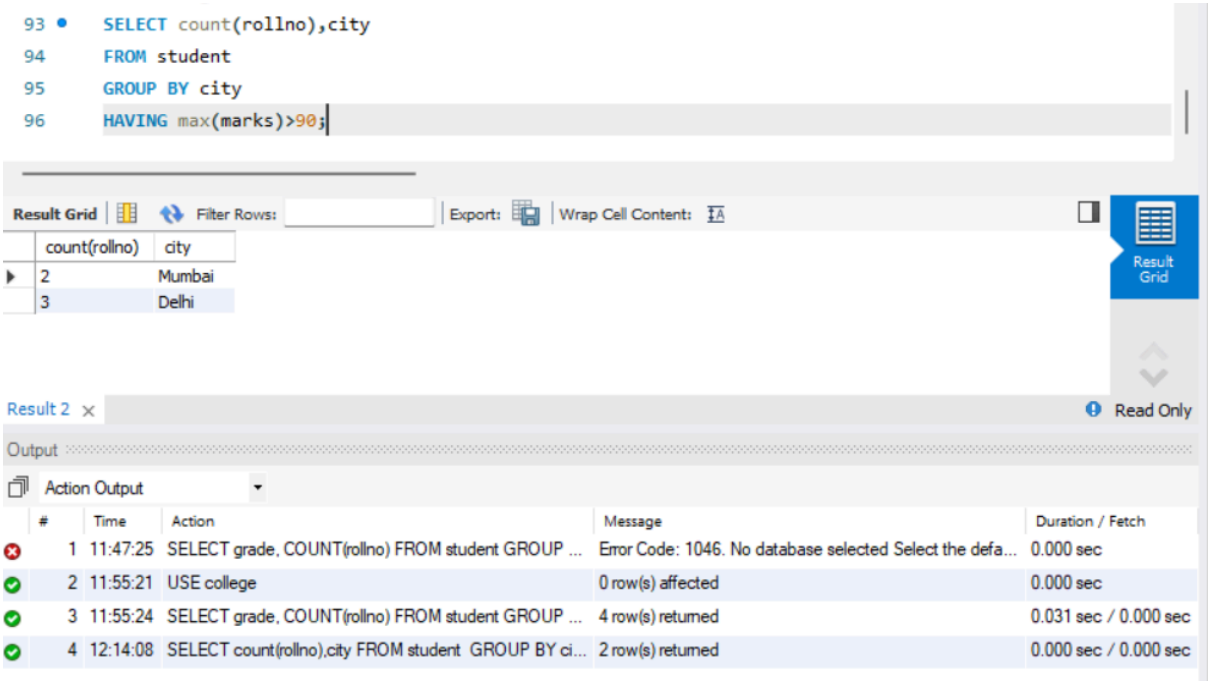


HAVING Clause

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

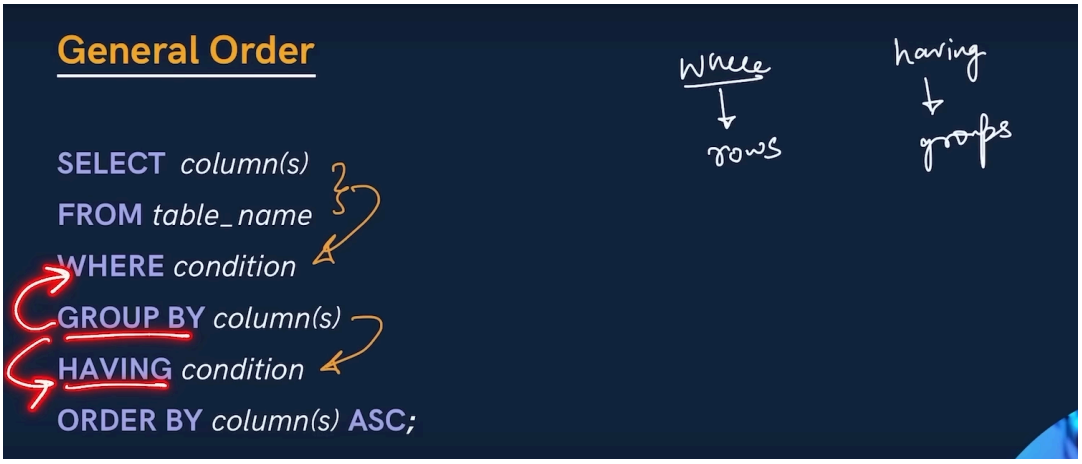


Sometimes we have situation where we have to write command in SQL where we cant write WHERE clause , **WHERE** is use for to apply conditions on **normal rows** , **HAVING** clause is used to condition on **GROUPS** .



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

Now lets 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 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:

city
Delhi
Mumbai

Result Grid

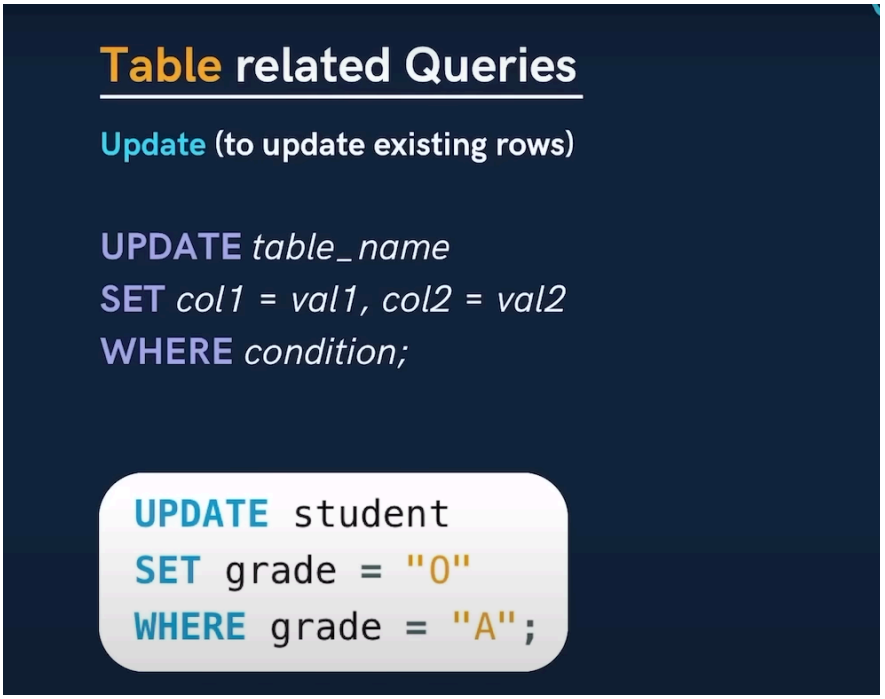
student 3 × Read Only

Output

Action Output

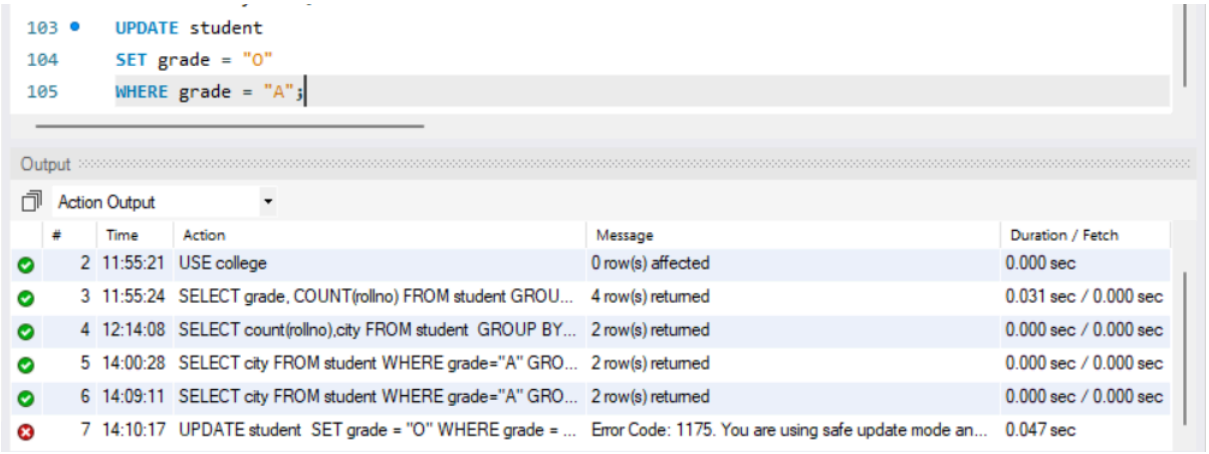
#	Time	Action	Message	Duration / Fetch
1	11:47:25	SELECT grade, COUNT(rollno) FROM student GROUP ...	Error Code: 1046. No database selected Select the defa...	0.000 sec
2	11:55:21	USE college	0 row(s) affected	0.000 sec
3	11:55:24	SELECT grade, COUNT(rollno) FROM student GROUP ...	4 row(s) returned	0.031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



Example,
earlier we were giving grades as “A”, “B”,”C”,... and so on, Suppose college now decided to grades as , above 90 -> ”Outstanding”. I.e. we have to change “A” grade to “O” grade.
For that will use UPDATE

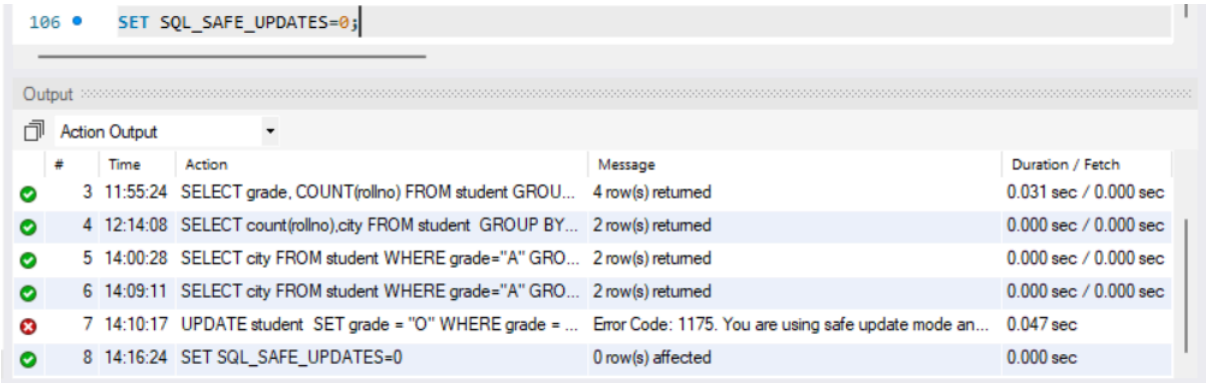
UPDATE student
Set grade = “O”
WHERE grade = “A”;



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

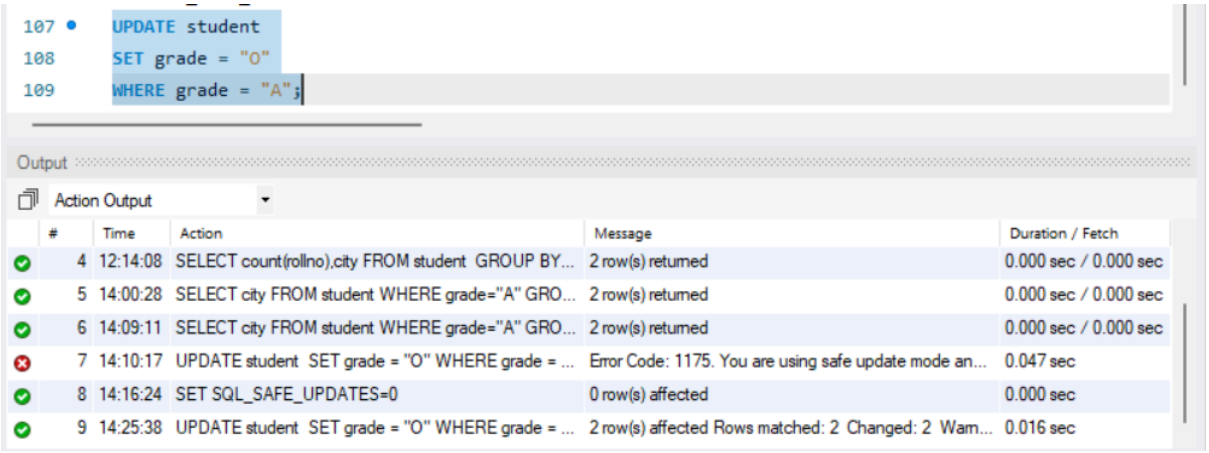
Now we can update tables , because safe mode is off .



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 …



Now lets check the data using SELECT

```
110 •  SELECT * FROM student;
```

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

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 | Apply

Output

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

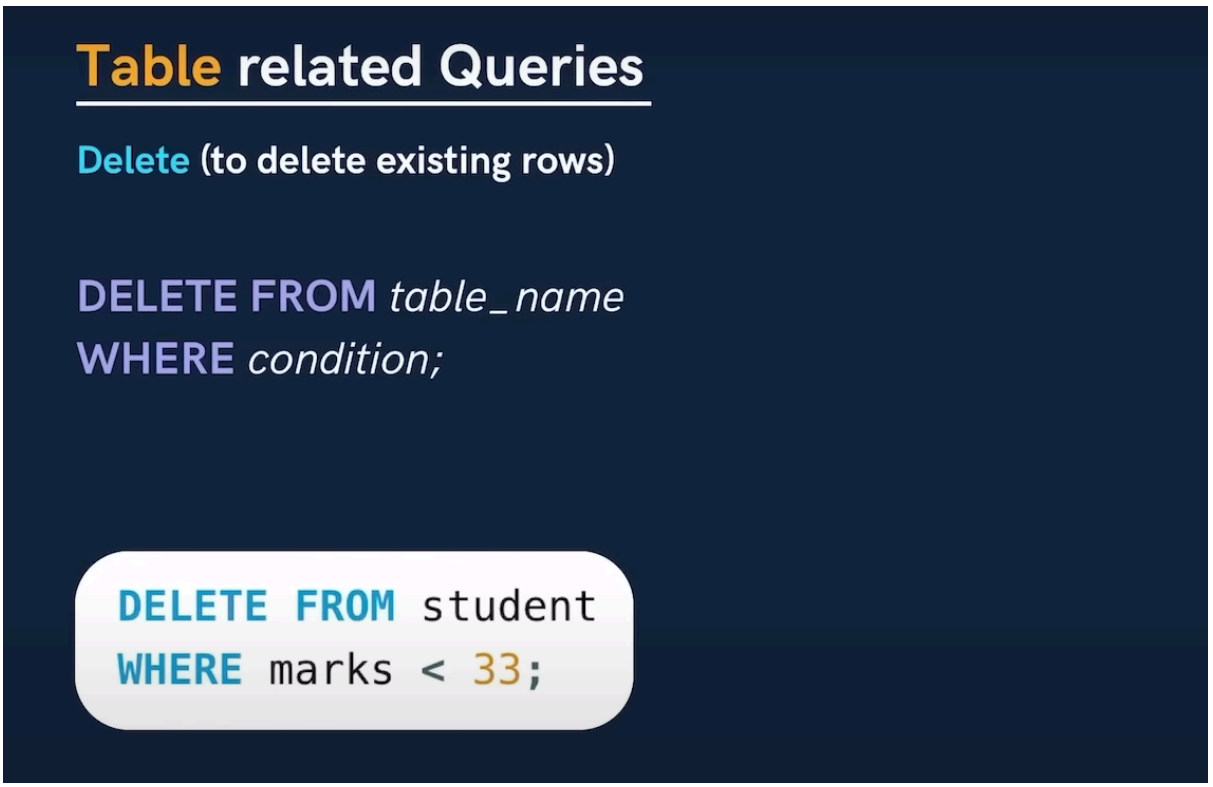
rollno	name	marks	grade	city
101	anil	79	C	Pune
102	bhumika	94	O	Mumbai
103	chetan	86	B	Mumbai
104	dhruv	97	O	Delhi
105	emanuel	13	F	Delhi

student 6 | Apply

Output

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



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

Action 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 | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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

student 7 | Apply

Output

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 Wam...	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 Wam...	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;
```

Output

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 Wam...	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 Wam...	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 | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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 8 | Apply

Output

Action Output

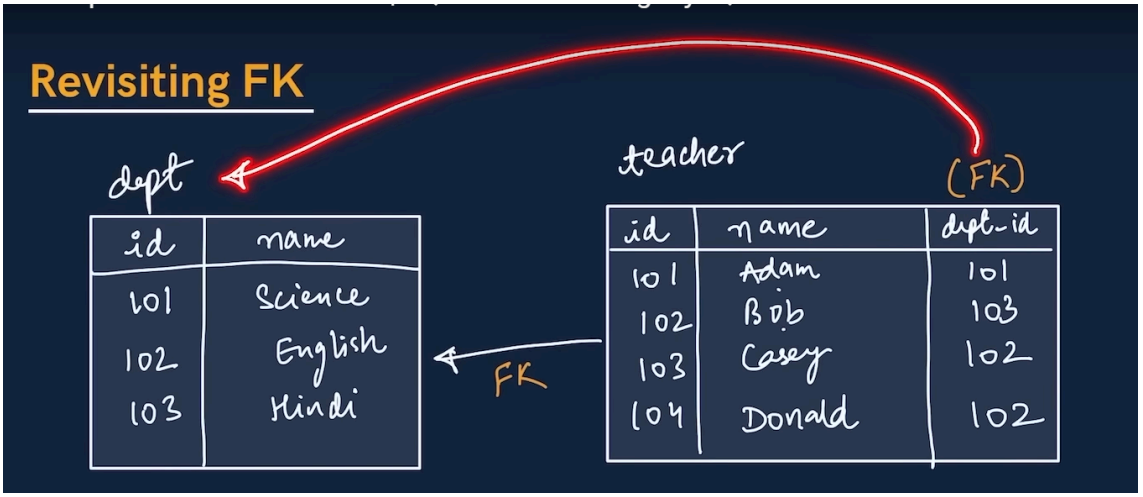
#	Time	Action	Message	Duration / Fetch
12	14:32:23	select * FROM studnet LIMIT 0, 1000	Error Code: 1146. Table 'college.studnet' doesn't exist	0.016 sec
13	14:32:30	select * FROM student LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
14	14:35:48	UPDATE student SET marks=12 WHERE rollno=105	1 row(s)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

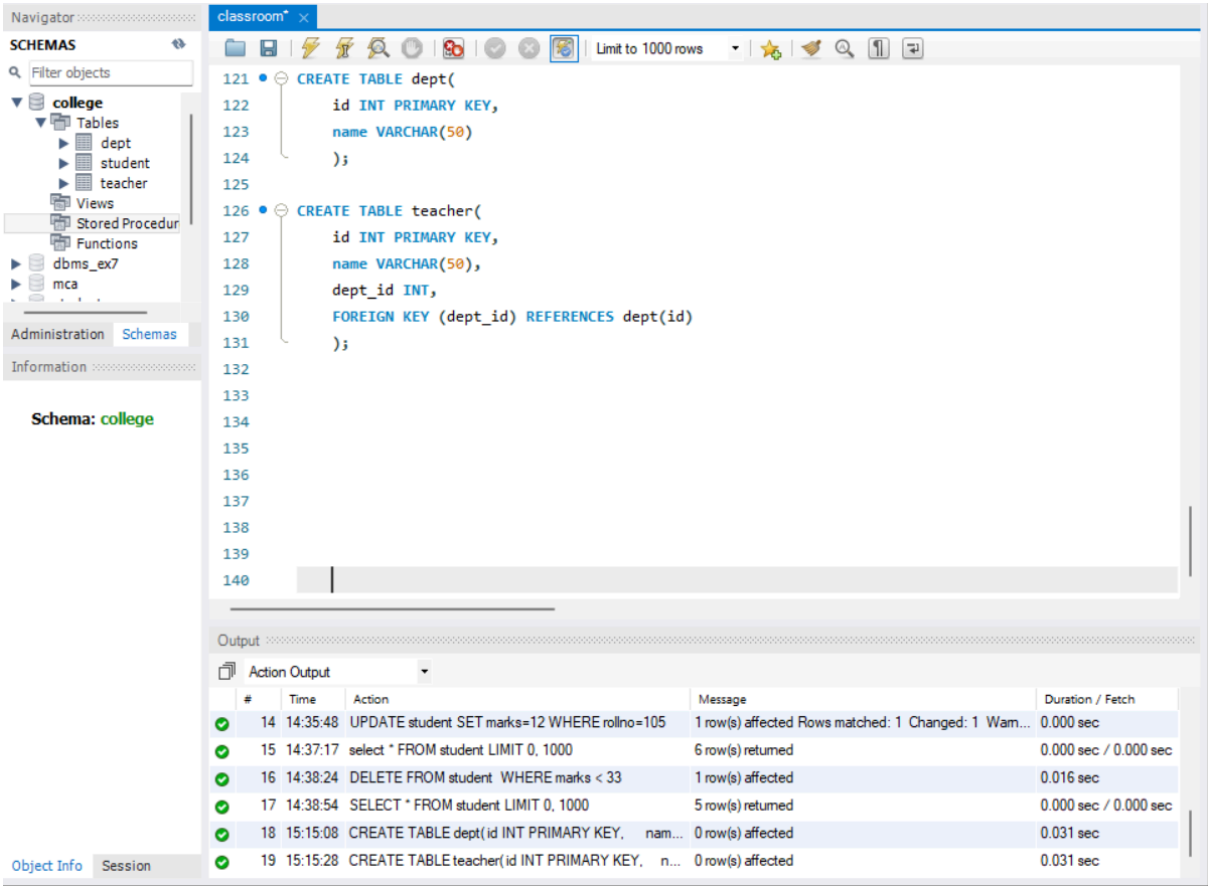
Use it carefully ,
Just **DELETE FROM student ;** can format the data of the **student** table.

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 ..

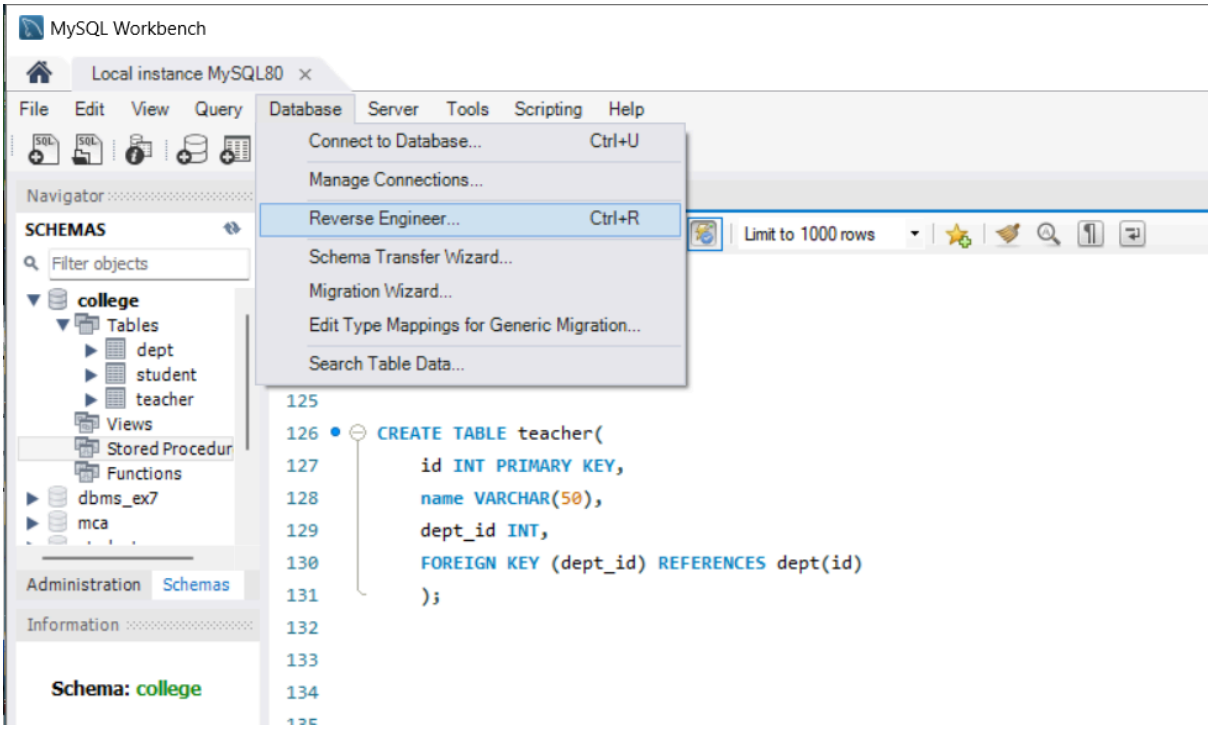


Department table is linked with teacher table , The foreign key is referencing to department table . i.e **FOREIGN KEY(dept_id) REFERENCES dept(id)**

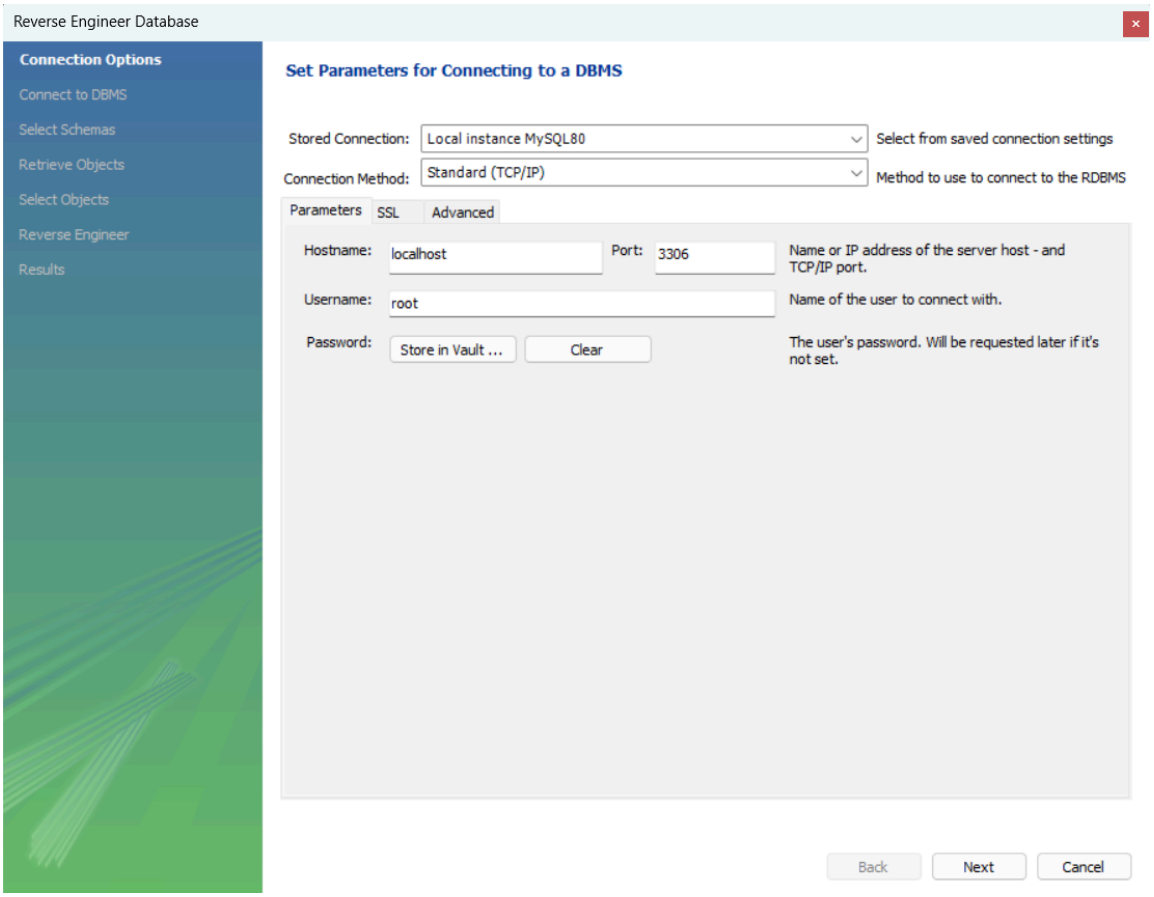


Lets see our ER diagram .

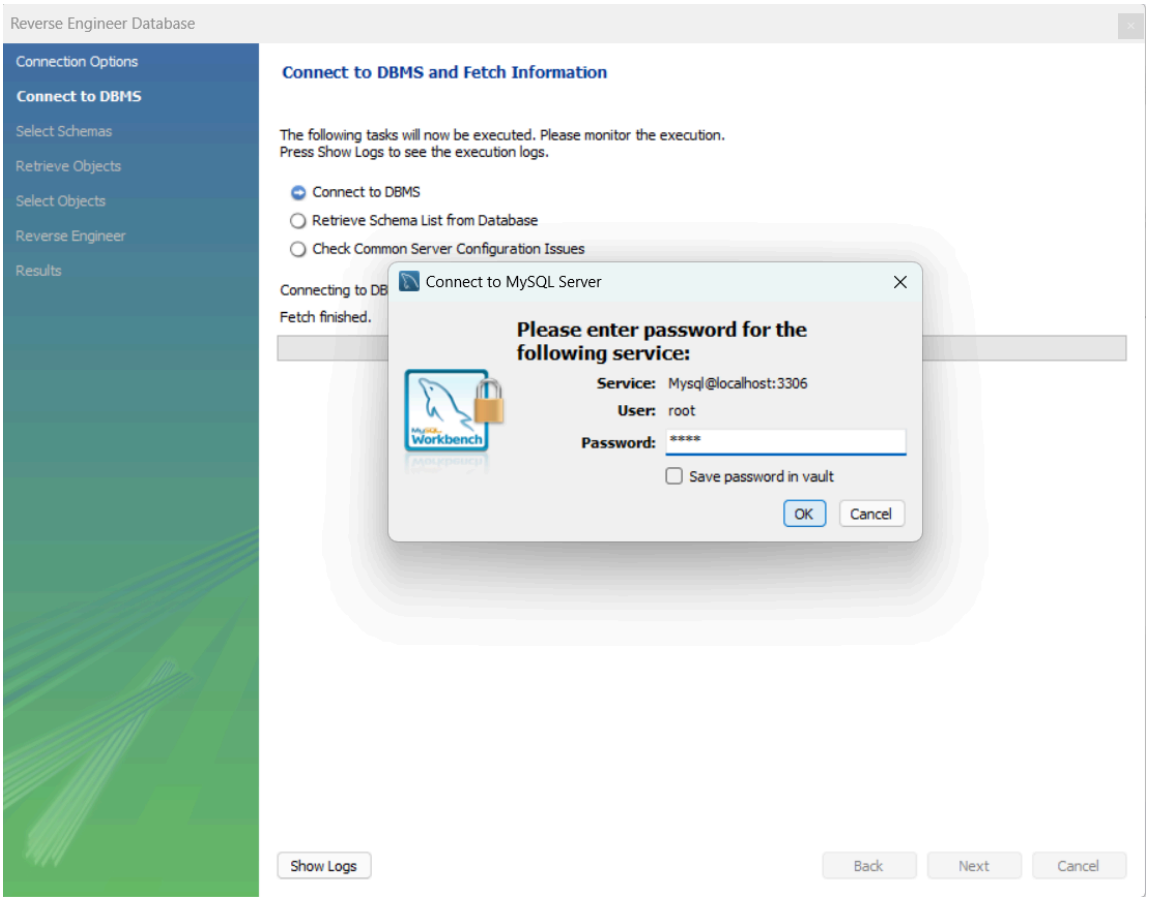
1. CLICK on **Reverse Engineer** under **database** option



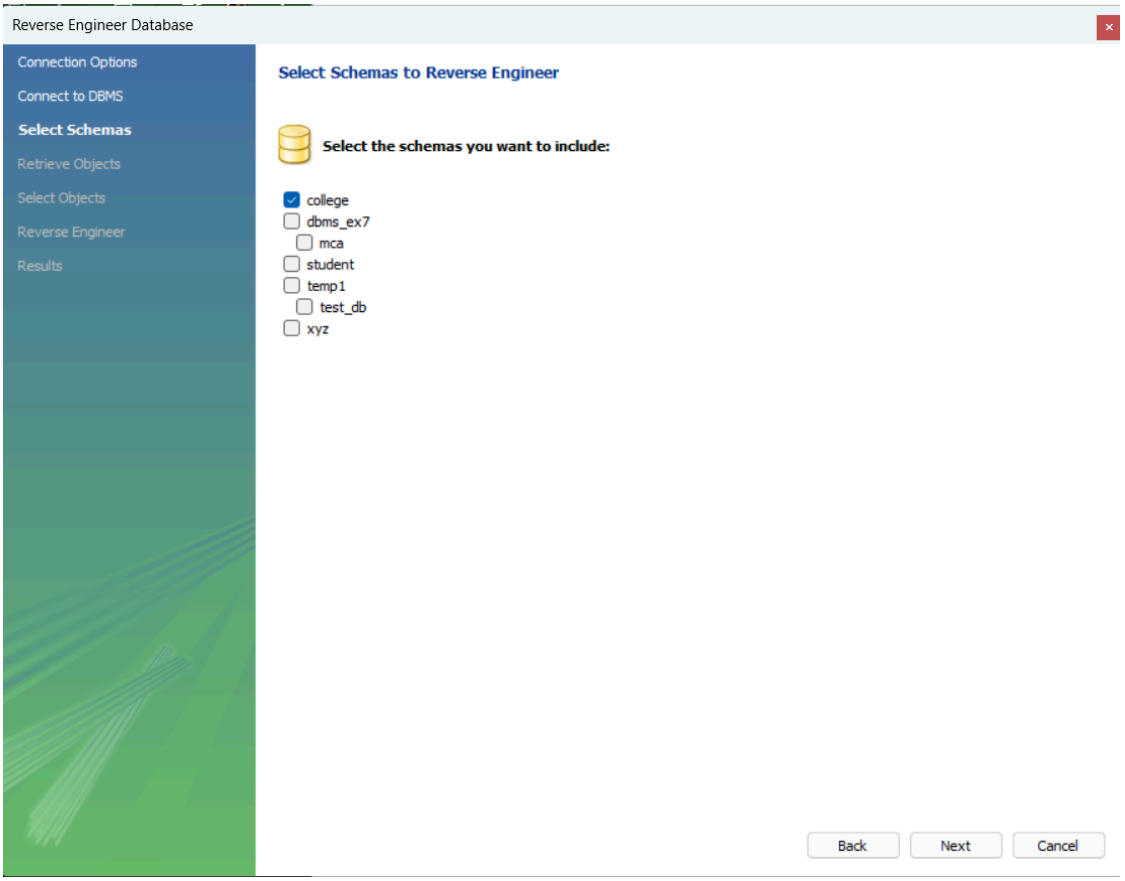
2. Then select **stored connection** and CLICK on **next**



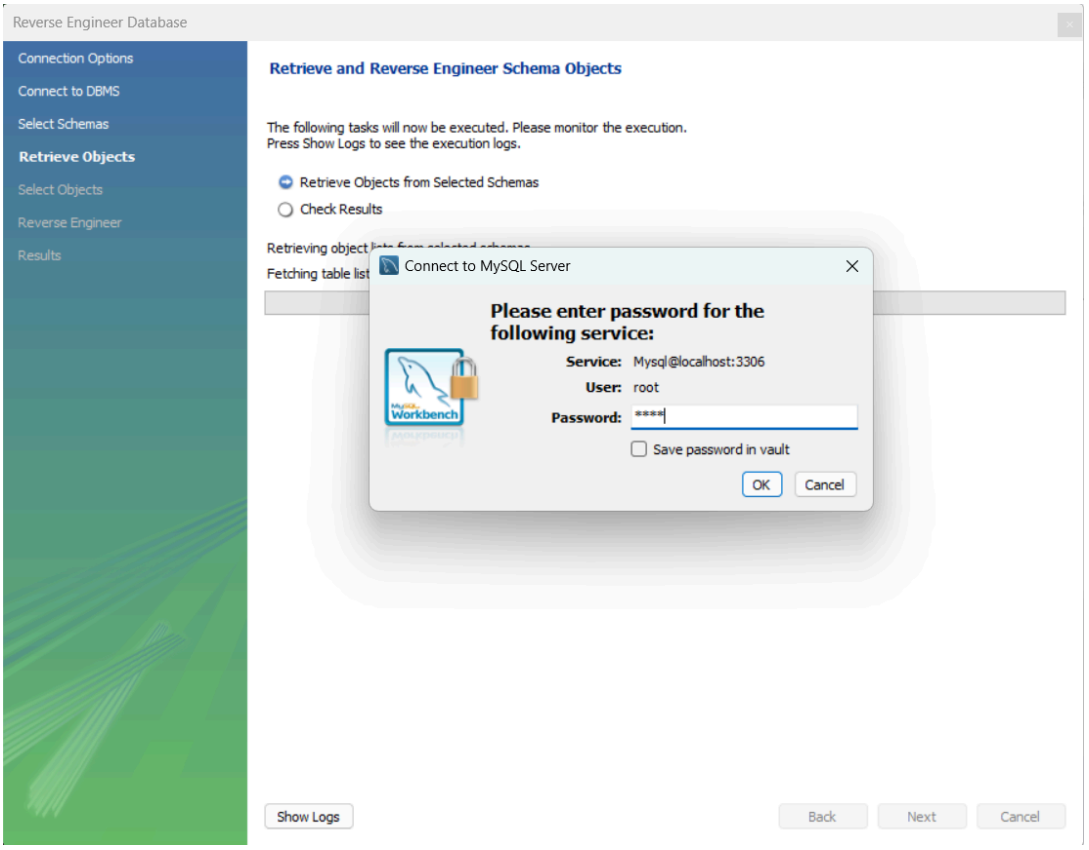
3. Enter password and click on **ok-> next.**



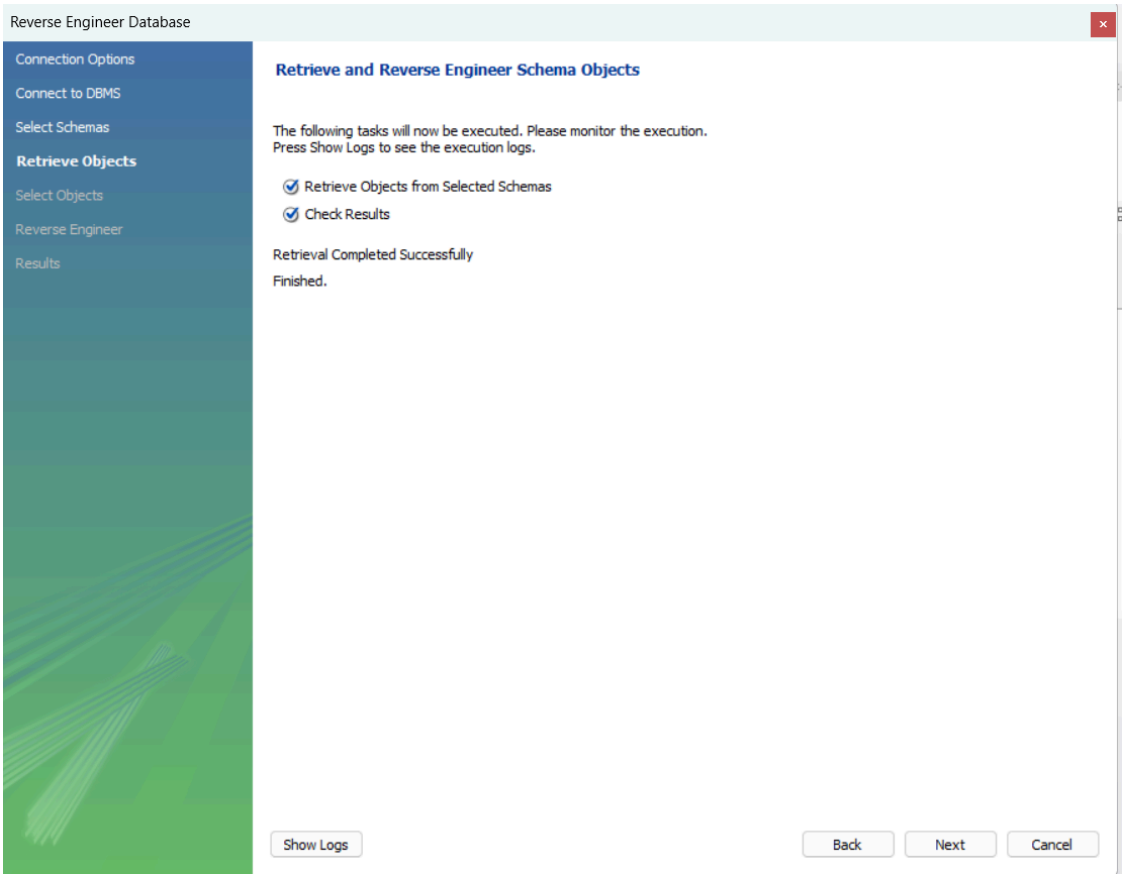
4. Select your schema and CLICK **next .**



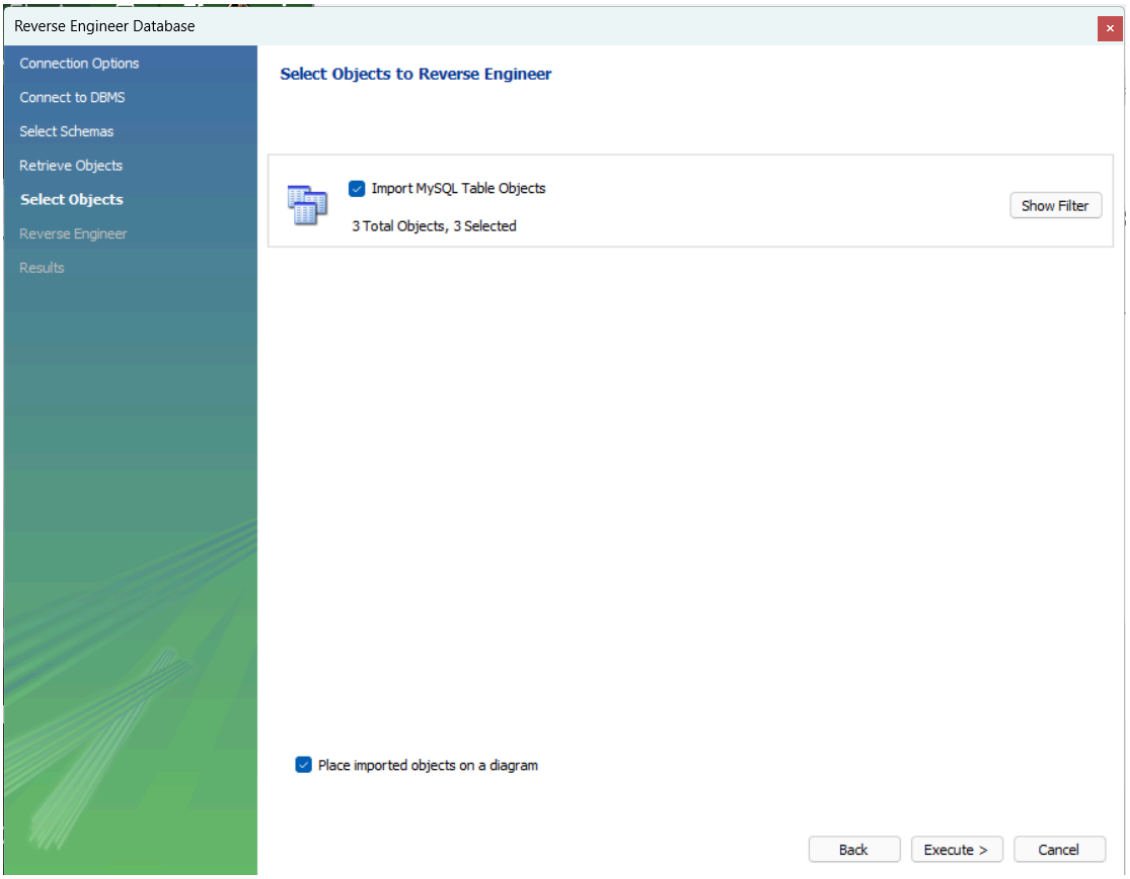
5. Again enter **password**



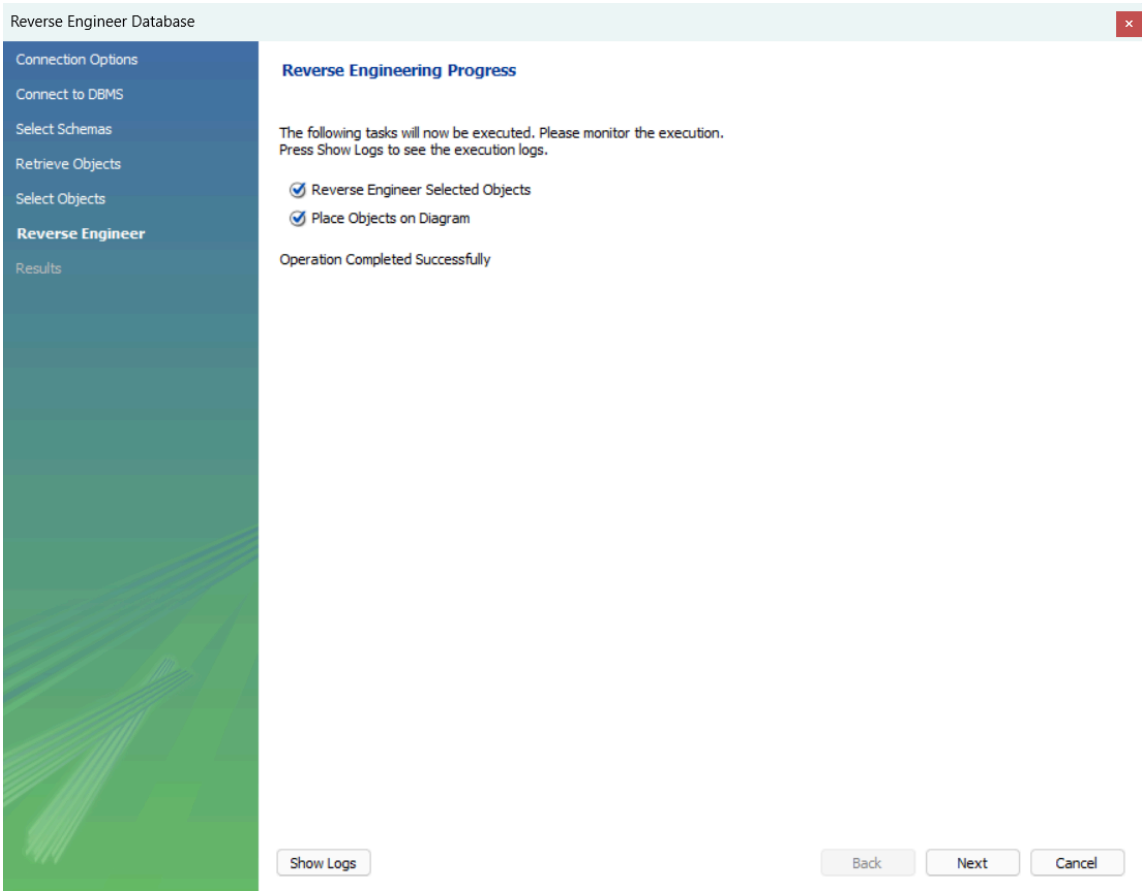
6. Now its done …

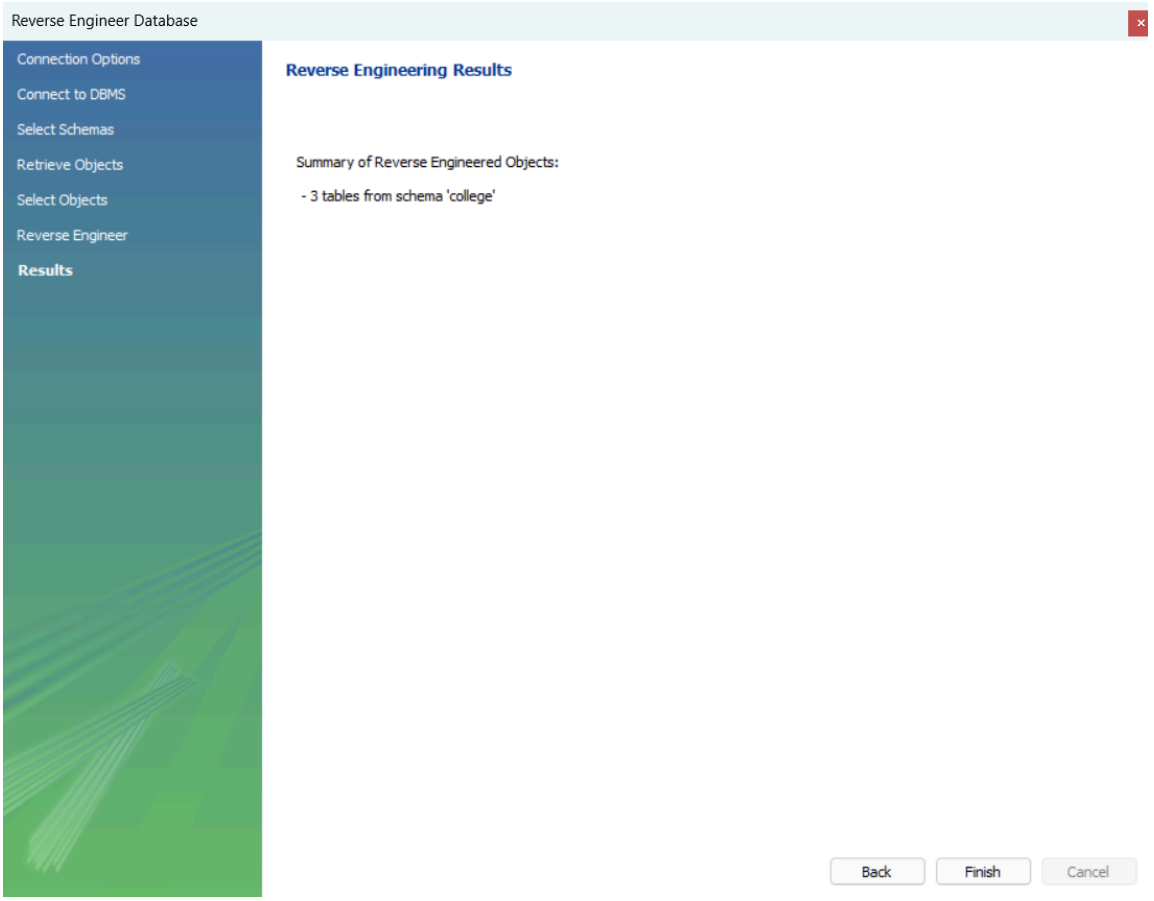


7. Click on **execute**

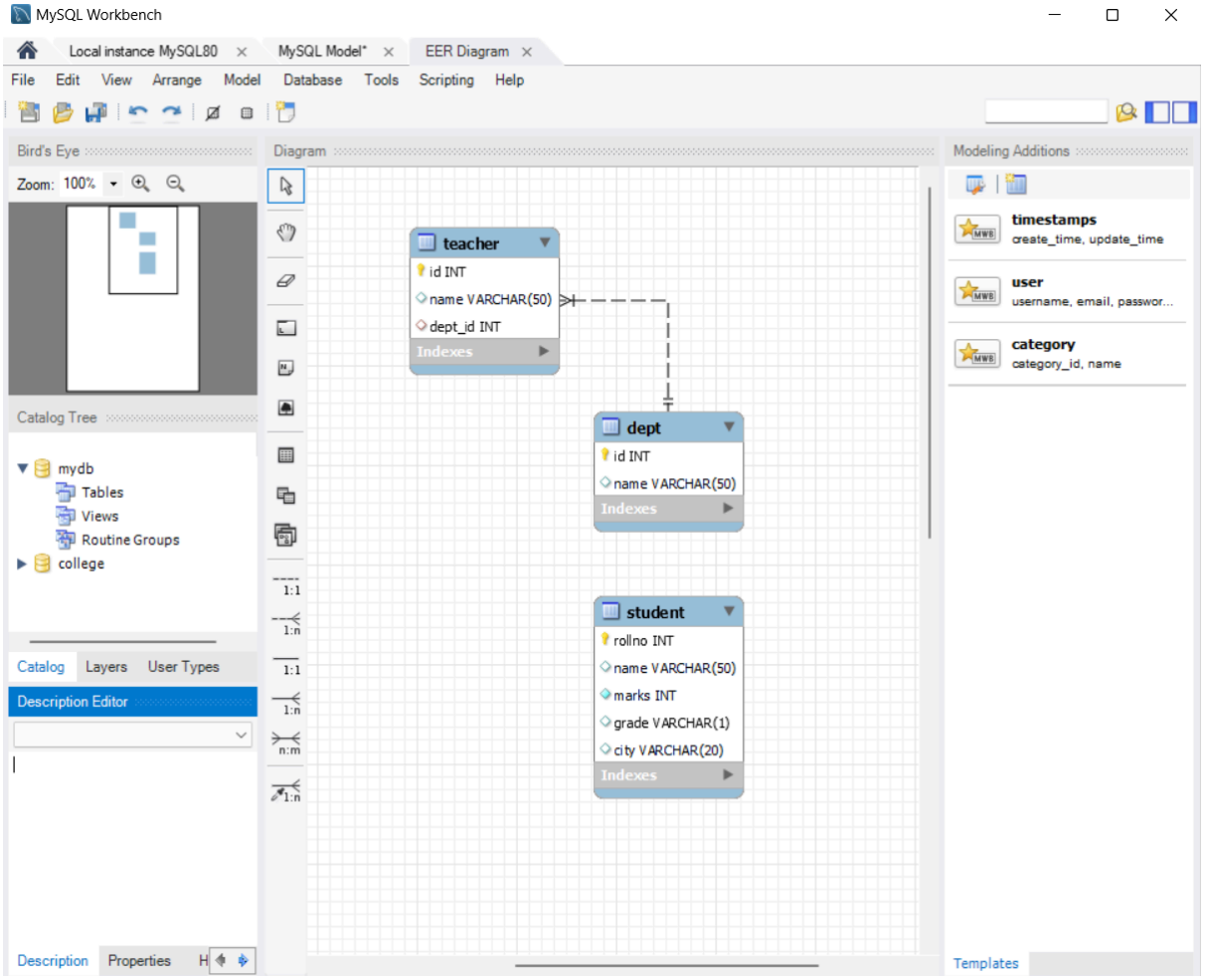


8. Some more steps…

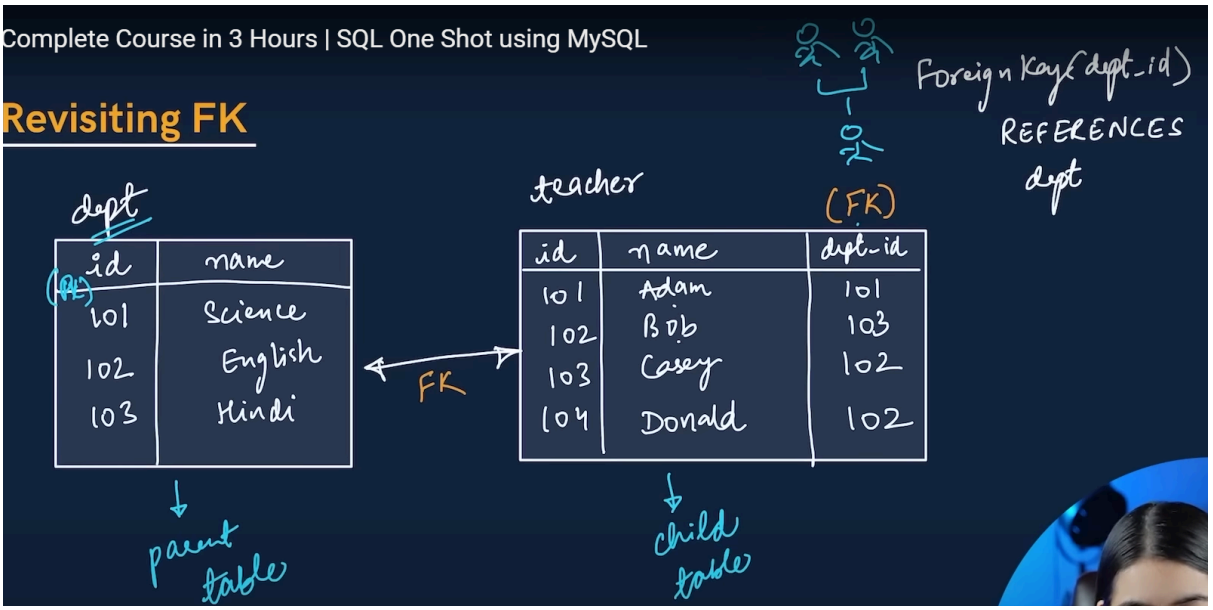




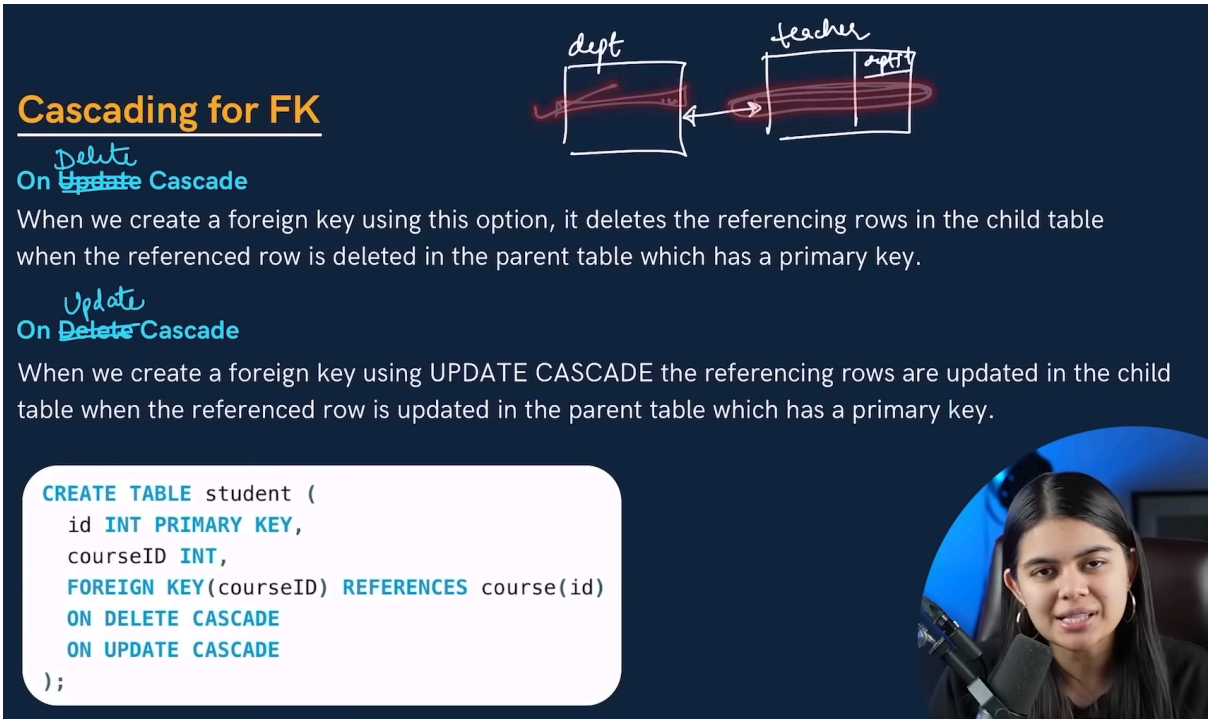
CLICK on **finish ..**Now we can see our ER diagram



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 .



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 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 DELETE 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 techer;
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 .

```
136 • INSERT INTO dept
137   VALUES
138   (101,"english"),
139   (102,"IT");
140
```

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;
```

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

id	name
101	english
102	IT
NULL	NULL

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;
```

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

id	name	dept_id
101	Adam	101
102	Eve	102
NULL	NULL	NULL

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.
Lets update and display the data

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

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

id	name
101	english
103	IT
NULL	NULL

dept 11 | Apply

Output

Action Output

#	Time	Action	Message	Duration / Fetch
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
27	16:12:53	UPDATE department SET id = 103 WHERE id = 102	Error Code: 1146. Table 'college.department' doesn't e...	0.000 sec
28	16:13:13	UPDATE dept SET id = 103 WHERE id = 102	1 row(s) affected Rows matched: 1 Changed: 1 Warn...	0.047 sec
29	16:13:46	SELECT * FROM dept LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

Now lets check the data of **teacher** table

```
155 •  SELECT * FROM teacher;
```

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

id	name	dept_id
101	Adam	101
102	Eve	103
NULL	NULL	NULL

teacher 13 | Apply

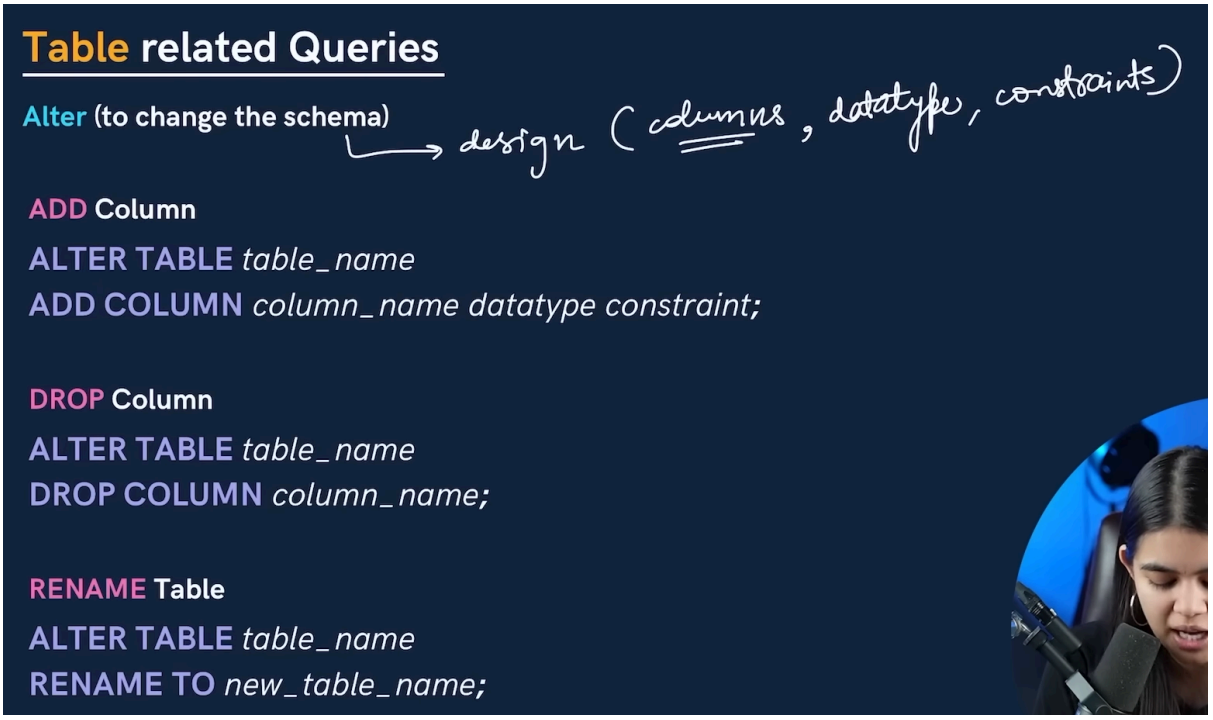
Output

Action Output

#	Time	Action	Message	Duration / Fetch
26	16:07:56	SELECT * FROM teacher LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
27	16:12:53	UPDATE department SET id = 103 WHERE id = 102	Error Code: 1146. Table 'college.department' doesn't e...	0.000 sec
28	16:13:13	UPDATE dept SET id = 103 WHERE id = 102	1 row(s) affected Rows matched: 1 Changed: 1 Warn...	0.047 sec
29	16:13:46	SELECT * FROM dept LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
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

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 .



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:

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

student 15

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:

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

Output

Action Output

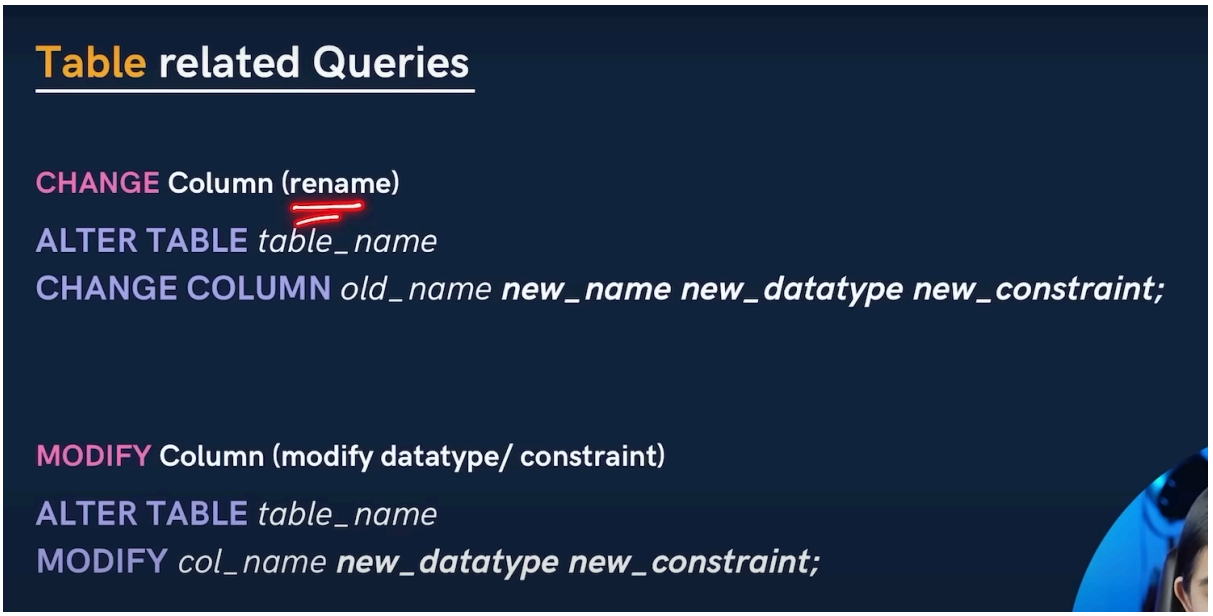
#	Time	Action	Message	Duration / Fetch
32	18:03:11	desc college	Error Code: 1146. Table 'college.college' doesn't exist	0.000 sec
33	18:03:39	desc student	5 row(s) returned	0.000 sec / 0.000 sec
34	18:04:24	ALTER TABLE student ADD COLUMN age INT	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	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;



Similarly we can also CHANGE the COLUMN NAME by

ALTER TABLE table_name

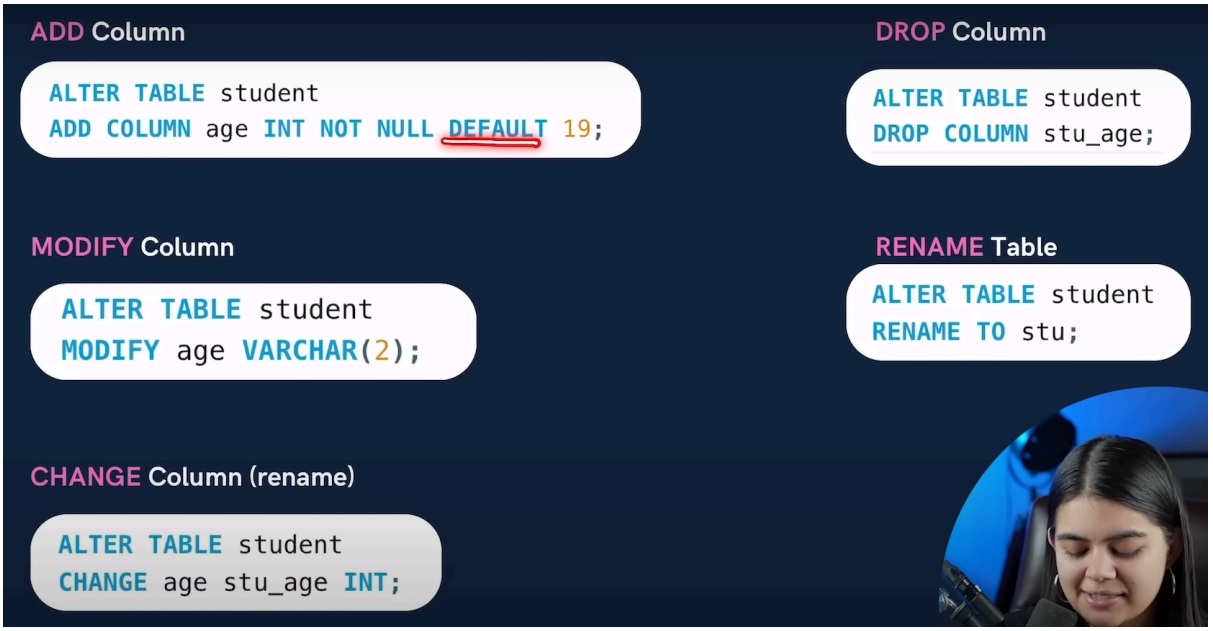
CHANGE COLUMN old_name new_name new_datatype new_constraint;

We can modify the DATATYPE or CONSTRAINT by ,

ALTER TABLE table_name

MODIFY col_name new_datatype new_constraint;

Lets try some examples …



1. **ADD Column**

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

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

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 ×

Output

Action 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
```

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

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	

Result 19 | Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
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
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;
```

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

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	NULL

student 20 | Apply

Output

Action Output

#	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:

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	NULL

Result Grid | Form Editor | student 21 | Apply

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

SCHEMAS

Filter objects

- Tables
  - dept
  - stu
  - teacher
- 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)

classroom*

Limit to 1000 rows

```
177 •  desc STUDENT;
178 •  ALTER TABLE student
179    CHANGE age stu_age INT;
180 •  SELECT * FROM student;
181 •  ALTER TABLE student
182    DROP COLUMN stu_age;
183 •  SELECT * FROM student;
184 •  ALTER TABLE student
185    RENAME TO stu;
186 •  SELECT * FROM stu;
```

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

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	NULL

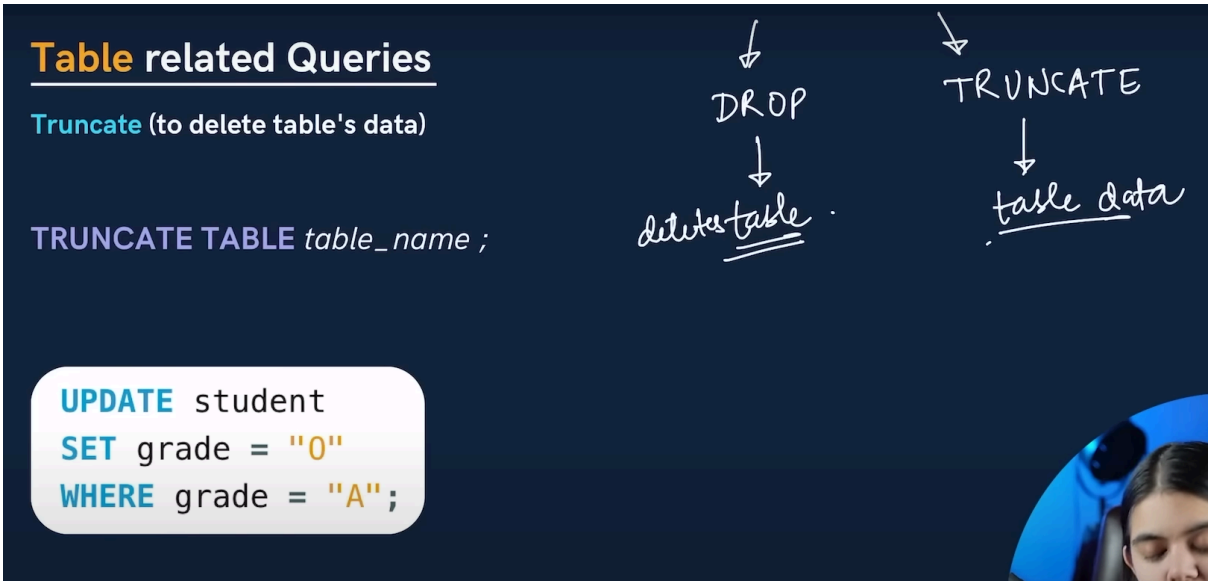
Result Grid | Form Editor | stu 22 | Apply

Output

Action Output

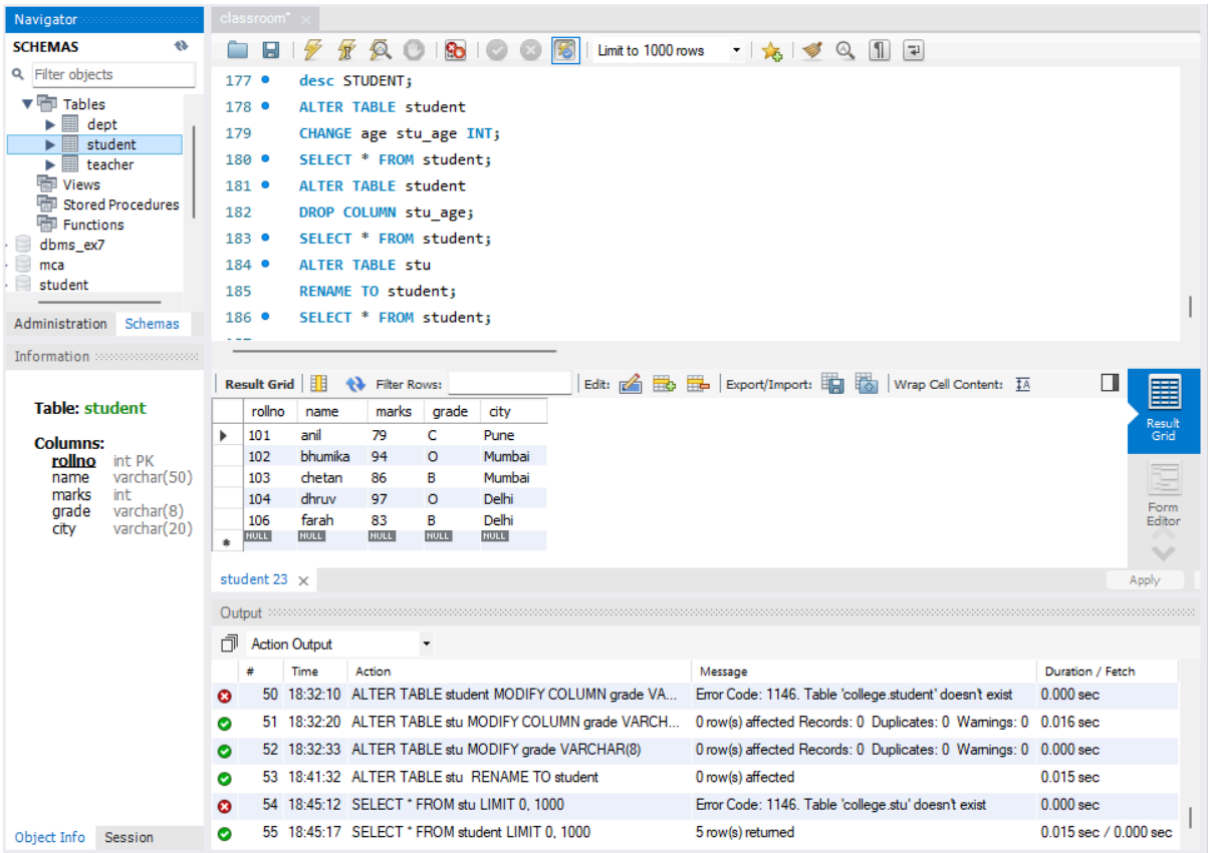
#	Time	Action	Message	Duration / Fetch
44	18:25:56	ALTER TABLE student CHANGE age stu_age INT	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.047 sec
45	18:26:19	SELECT * FROM student LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
46	18:27:24	ALTER TABLE student DROP COLUMN stu_age	0 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

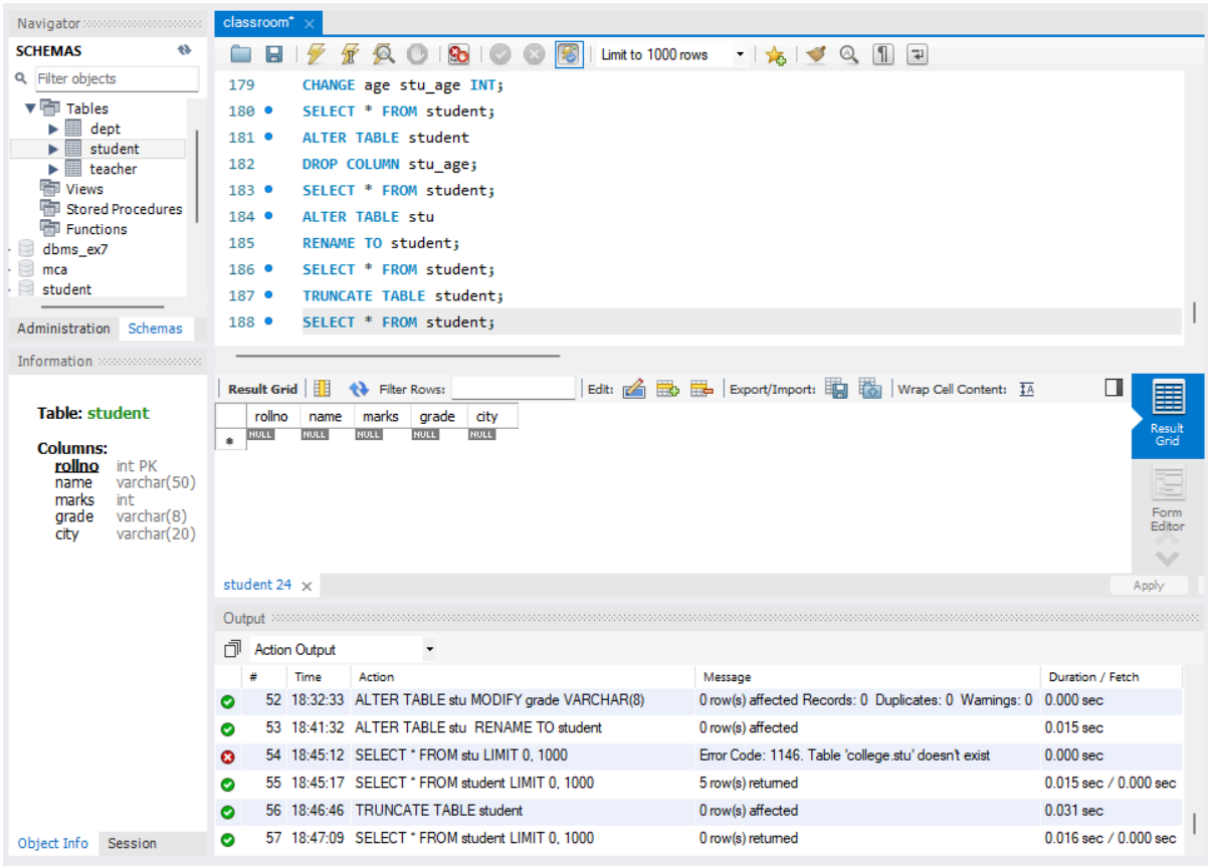


The main difference between DROP and TRUNCATE is that ,
DROP deletes the **table** and TRUNCATE deletes the **table data.**

Lets delete whole **student** table data ,
Before deletion

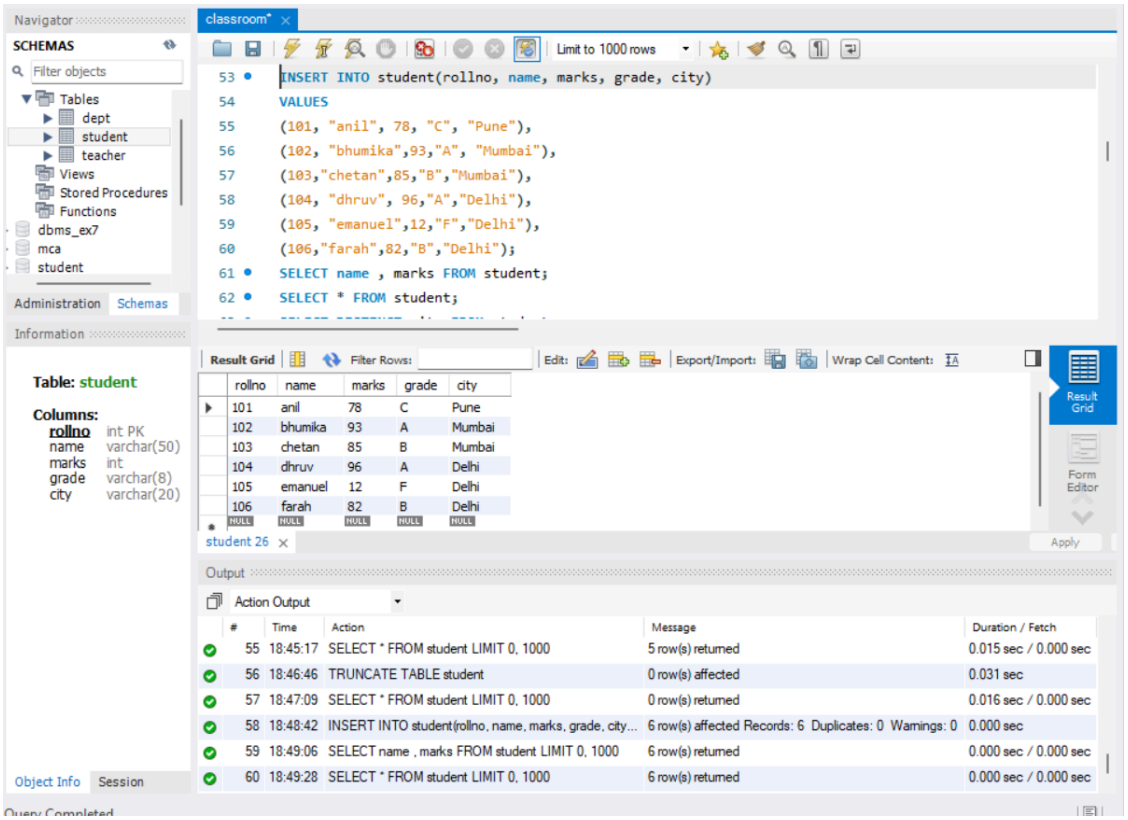


After deletion ,



Whole table data has been deleted ….

Again lets insert the data



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 :

a. Change the name of column "name" to "full_name".

b. Delete all the students who scored marks less than 80.

c. Delete the column for grades.

a)->

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

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

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	NULL

Result Grid | Form Editor | Apply

student 27

Output

Action 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;
```

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

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

Result Grid | Form Editor | Apply

student 28 ×

Output

Action Output

#	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;
```

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

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

Result Grid | Form Editor | Apply

student 29 ×

Output

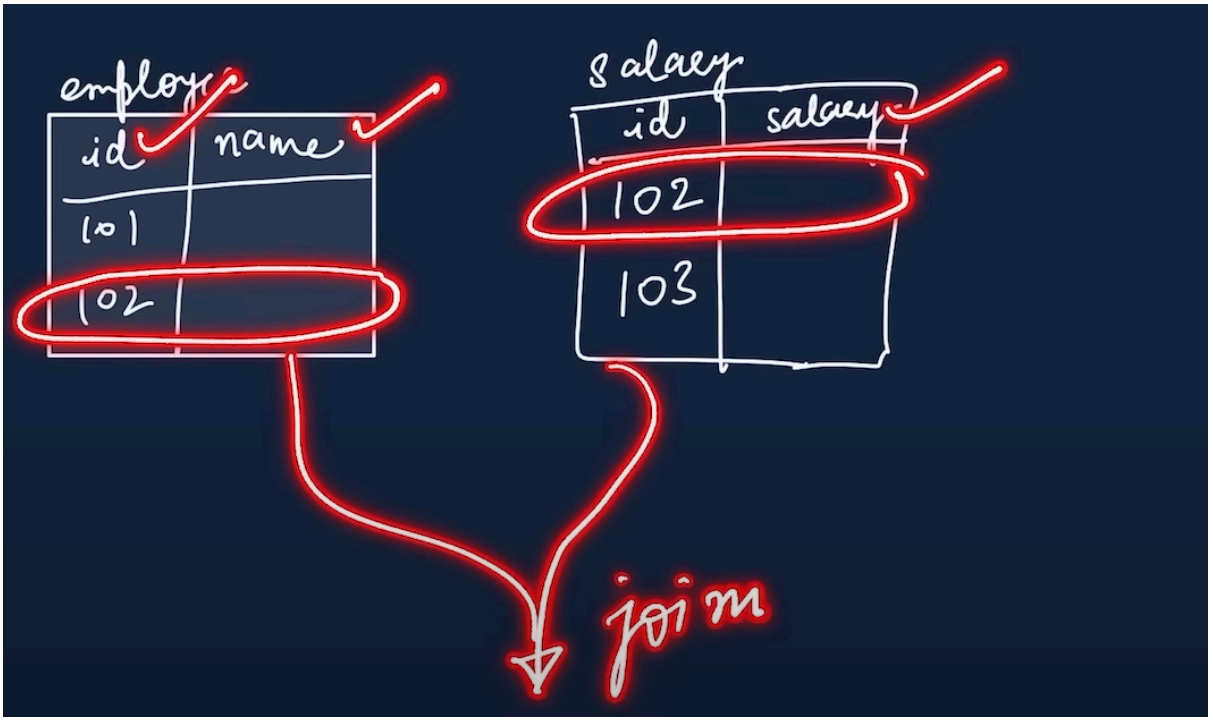
Action Output

#	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 synta...	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

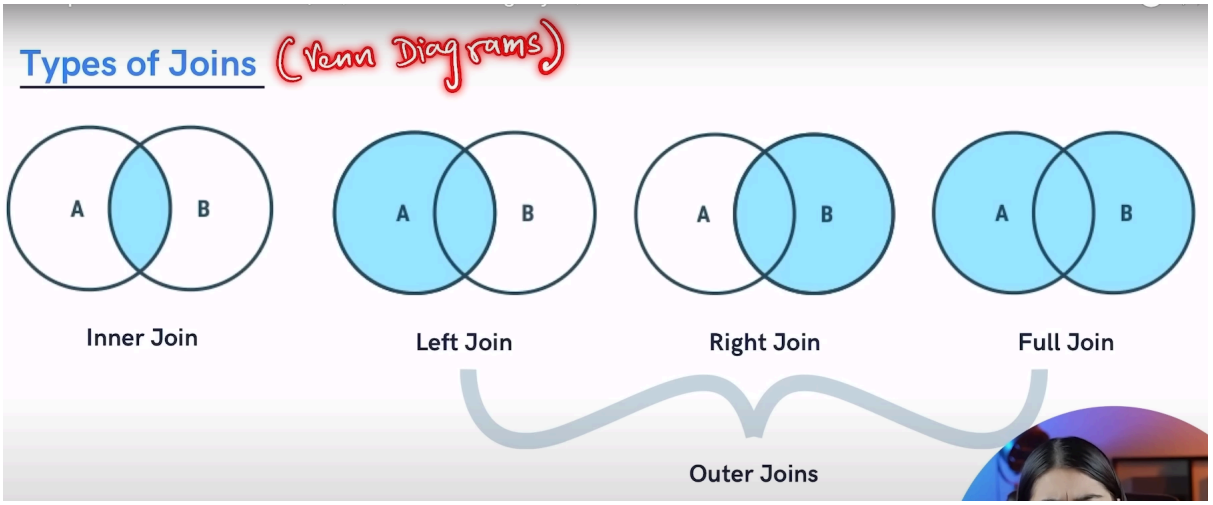
Join is used to combine rows from two or more tables , based on a **related column** between them.

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 .



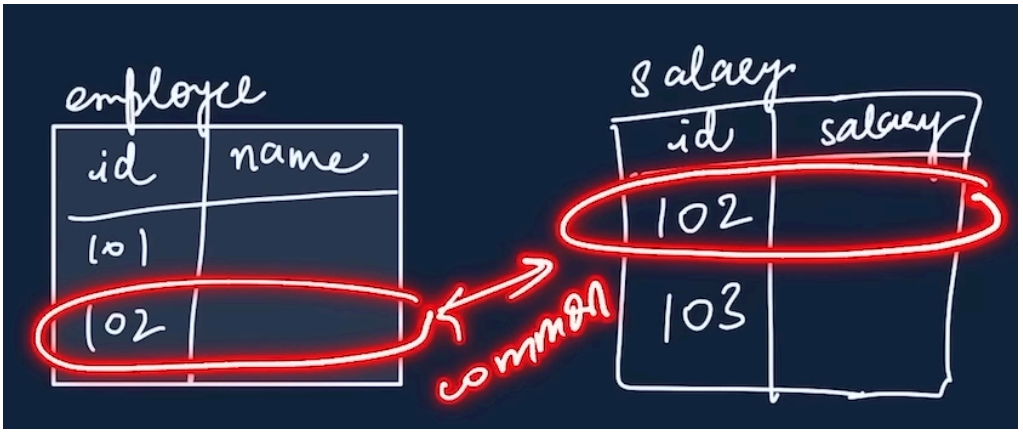
Now we want the data of employee which are in both tables i.e. here as we can see 102 is in both tables , so will get whole data of 102 (id , name , salary).
So we can JOIN both of the table to get the COMMON INFORMATION.
Since **id** are **related column / common column** in both of the table

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

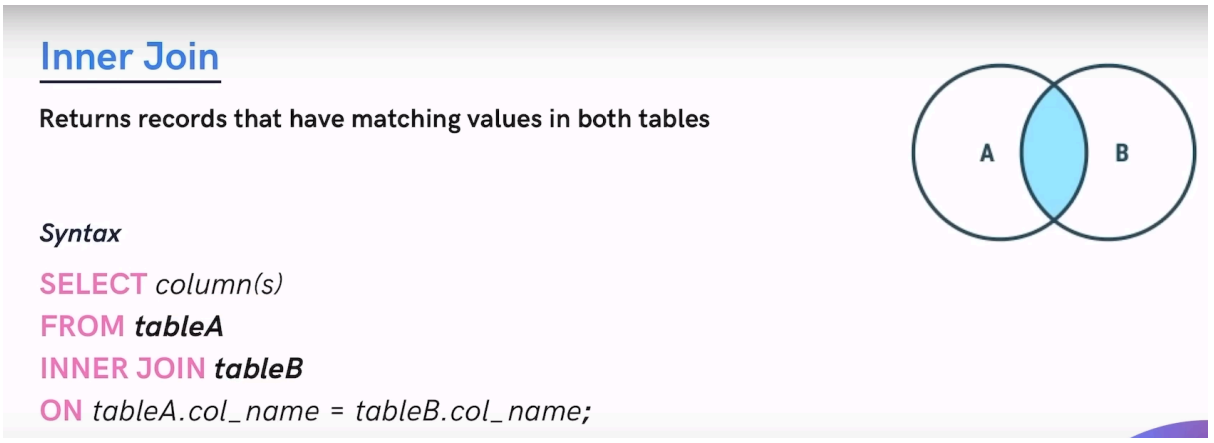


1. Inner Join :-

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 ,

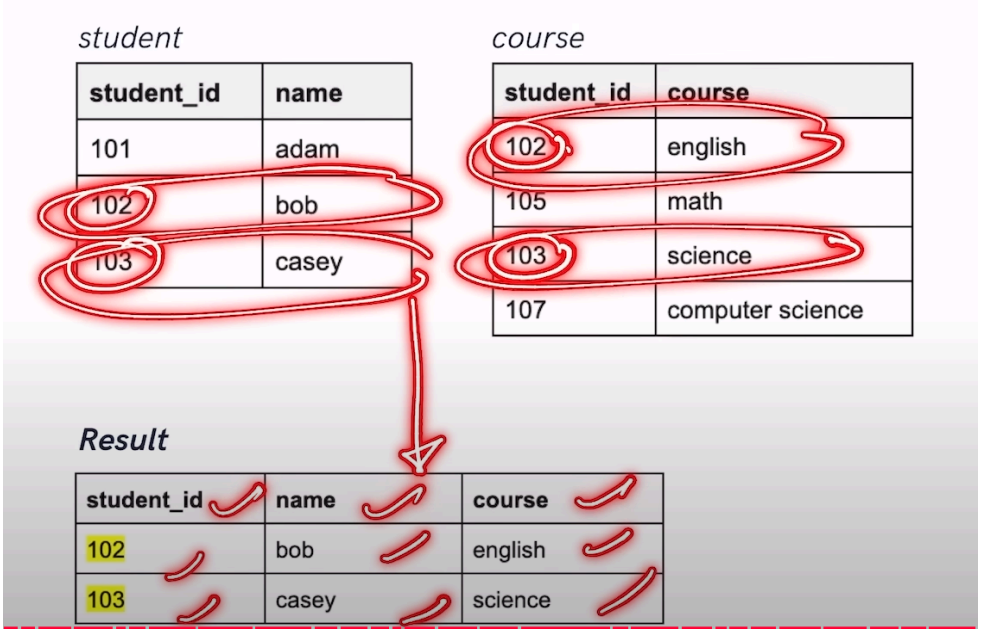


Here matching value will be in both columns .



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,"engli...	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;
```

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

id	name
101	adam
102	bob
103	casey
NULL	NULL

Result Grid | Form Editor

student 32 ×

Apply | Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
73	22:30:17	INSERT INTO course(id, course) VALUES (102,"engli...	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:

id	course
102	english
103	science
105	math
107	computer science
NULL	NULL

course 33 × 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:

id	name	id	course
102	bob	102	english
103	casey	103	science

Result 34 × 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

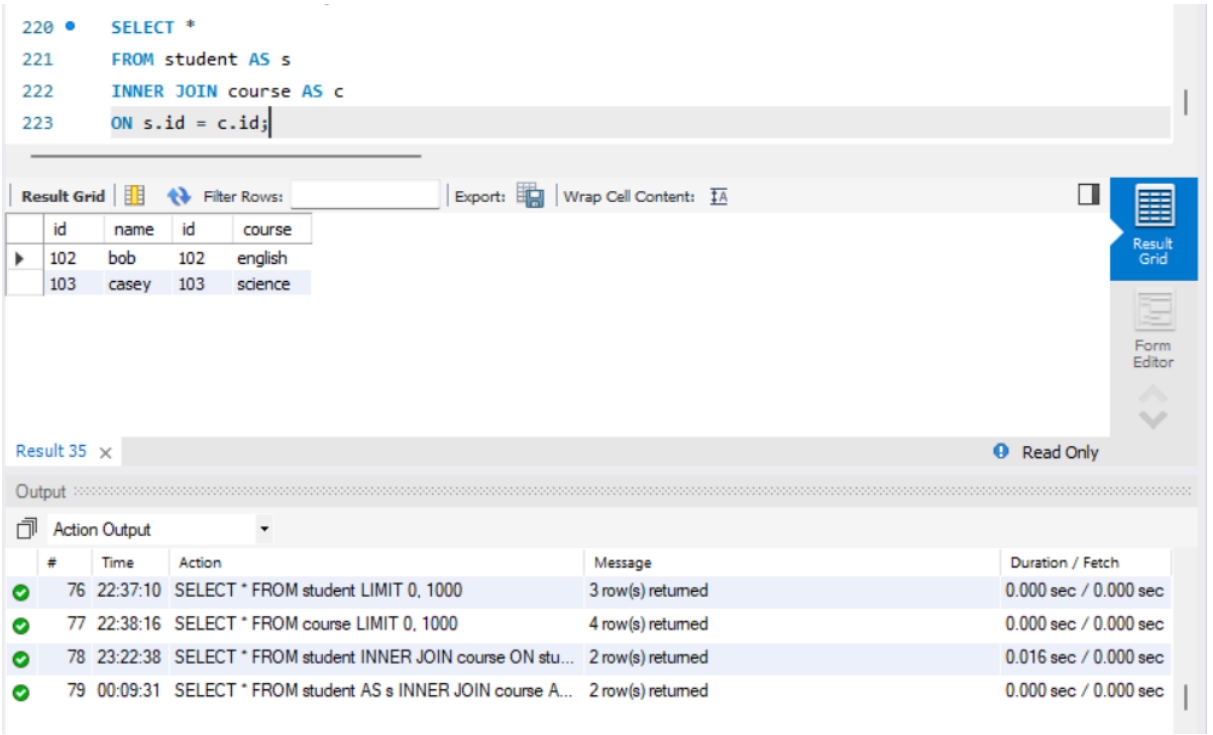
OUTPUT we can see a table in which we got COMBINE COMMON data in one SINGLE TABLE.

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 ;



We got same result as earlier

2. Outer Join :-

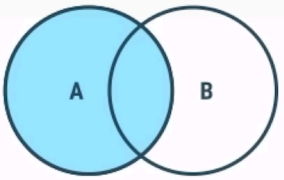
There are three types of outer joins -> **Left Join, Right Join** and **Full Join.**

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

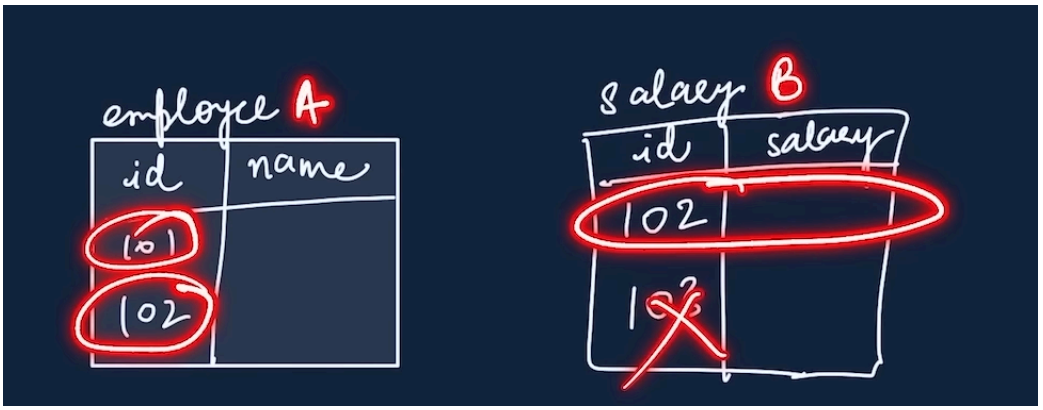
Returns all records from the left table, and the matched records from the right table



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**

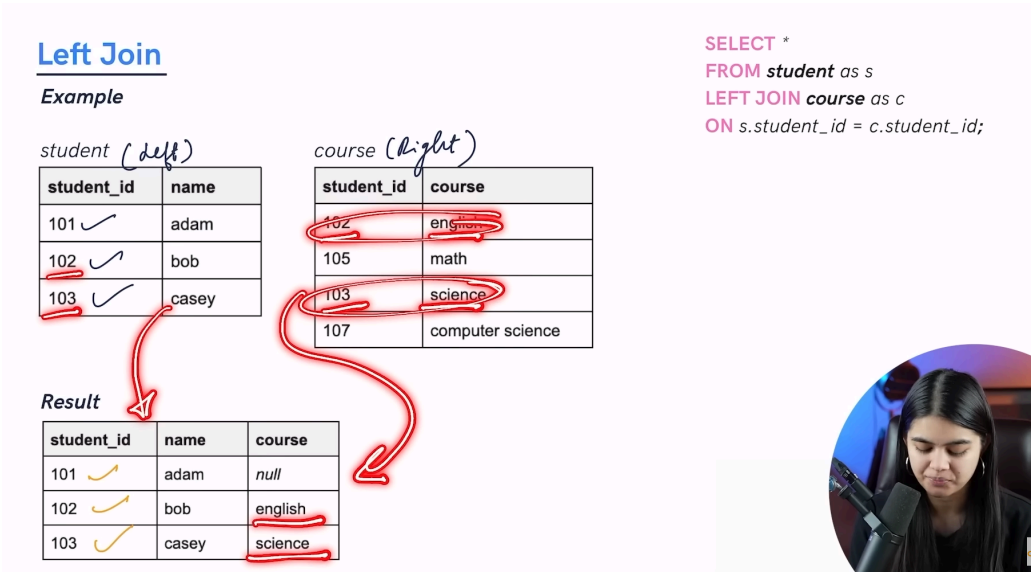


So will get A data (left) , and overlap or COMMON data .

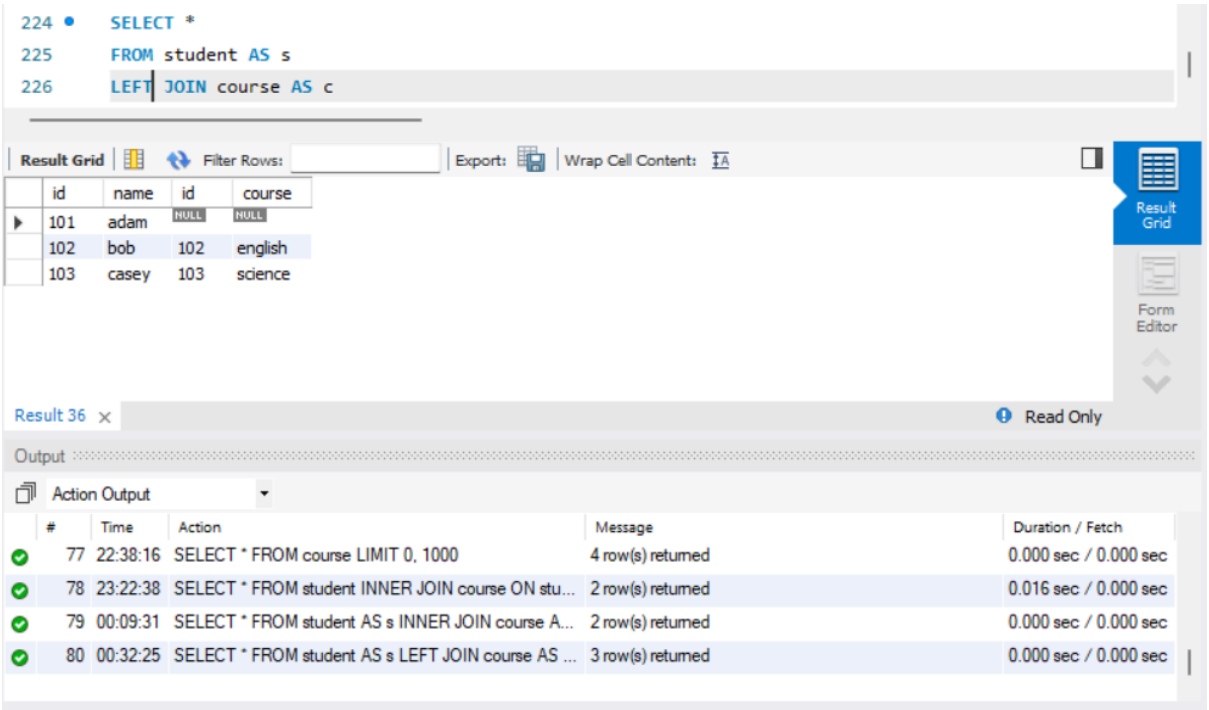
SELECT column(s)
FROM **tableA** …… **LEFT TABLE**
LEFT JOIN **tableB** …….. **RIGHT TABLE**
ON tableA.col_name = tableB.col_name

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 ,



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



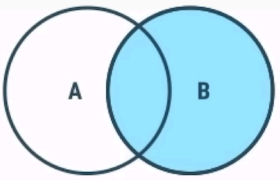
Here as we can see , we got the LEFT table and COMMON data of both .

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

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

student

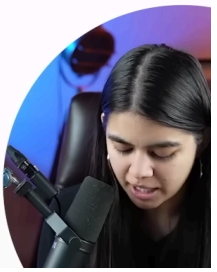
student_id	name
101	adam
102	bob
103	casey

course

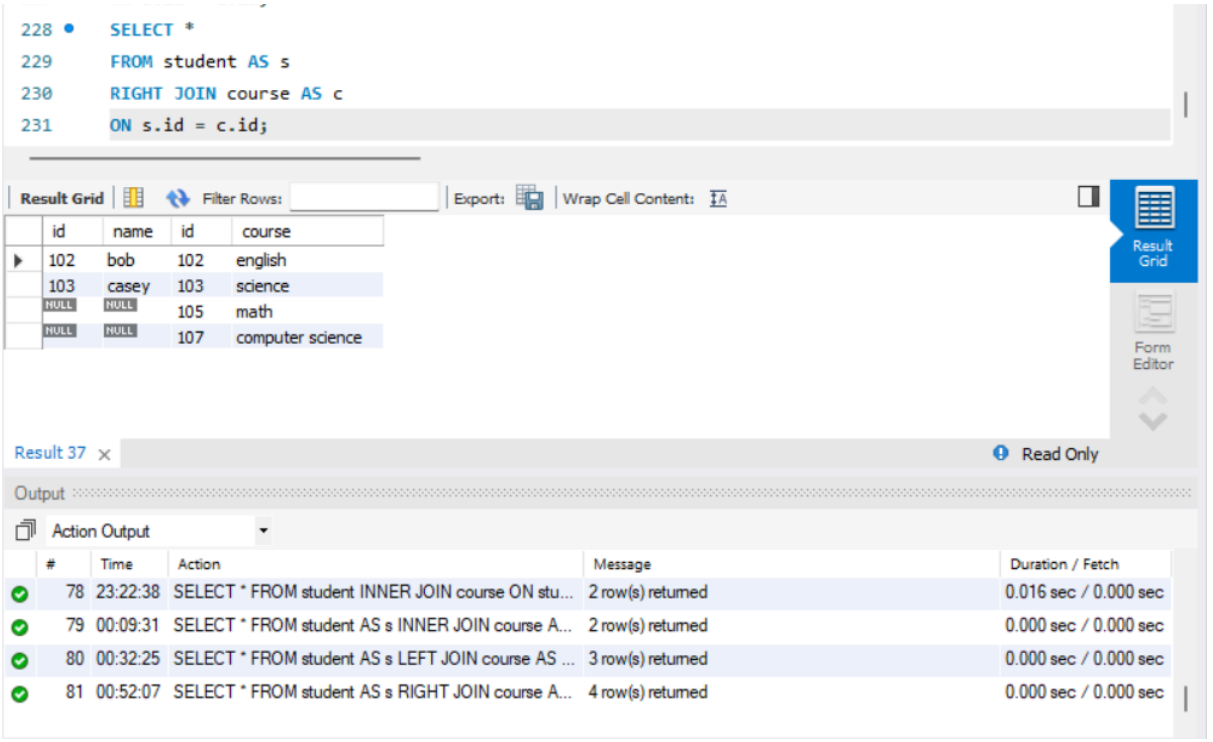
student_id	course
102	english
105	math
103	science
107	computer science

Result

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



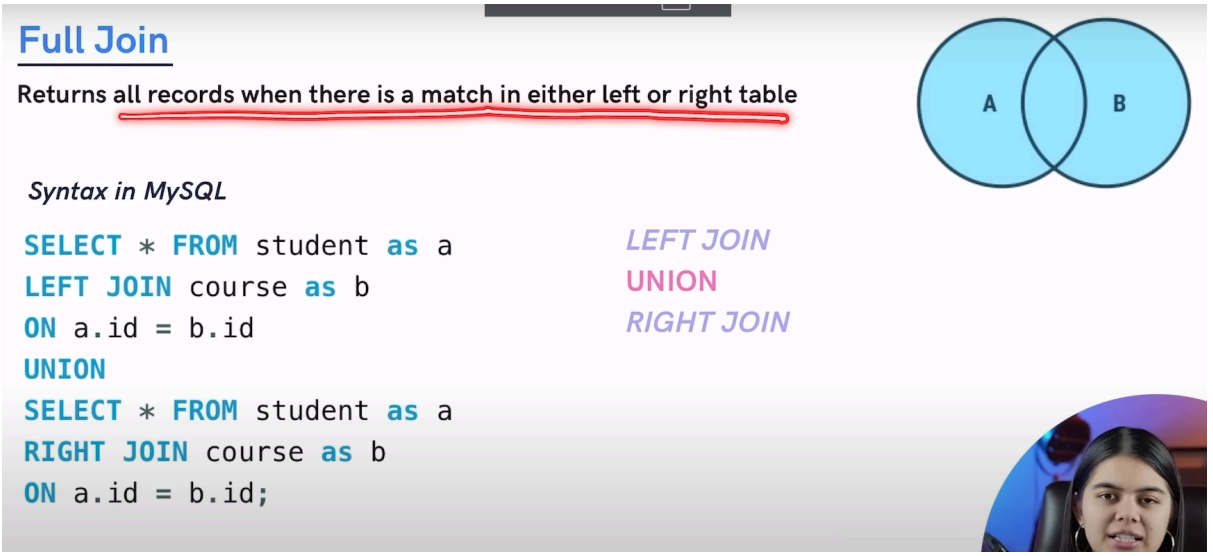
102 and 103 are common , and will get RIGHT side of data .



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

c) Full Join:-

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



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    UNION
238    SELECT *
239    FROM student AS s
240    RIGHT JOIN course AS c
241    ON s.id = c.id;
```

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

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 | Read Only

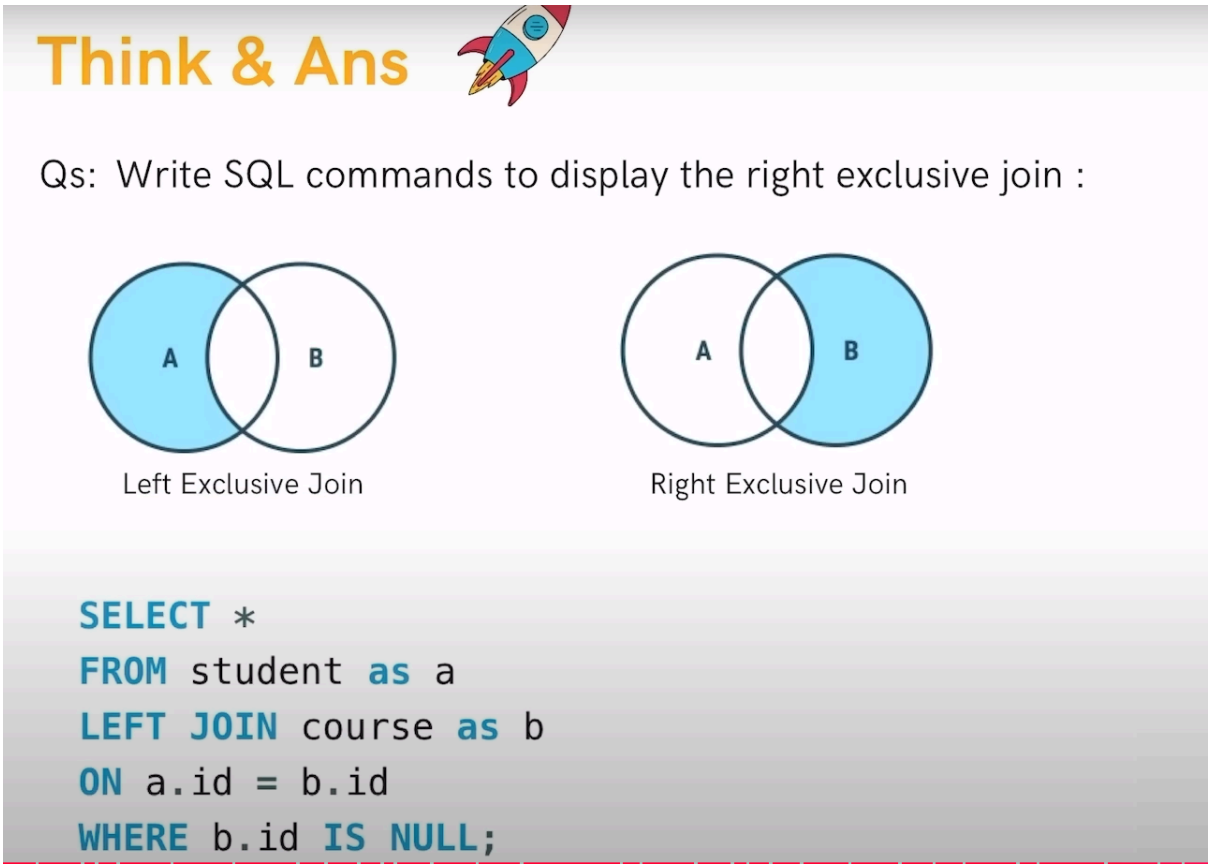
Output

Action Output

#	Time	Action	Message	Duration / Fetch
79	00:09:31	SELECT * FROM student AS s INNER JOIN course A...	2 row(s) returned	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
82	01:00:02	SELECT * FROM student AS s LEFT JOIN course AS ...	5 row(s) returned	0.015 sec / 0.000 sec

---

Now will do an activity where will learn more JOINS …



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

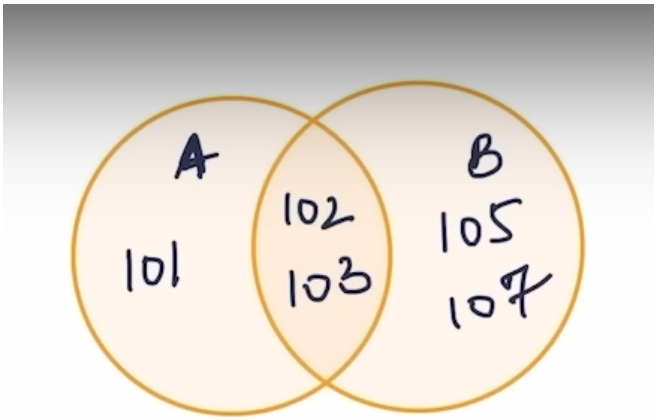
student_id	name
101	adam
102	bob
103	casey

course

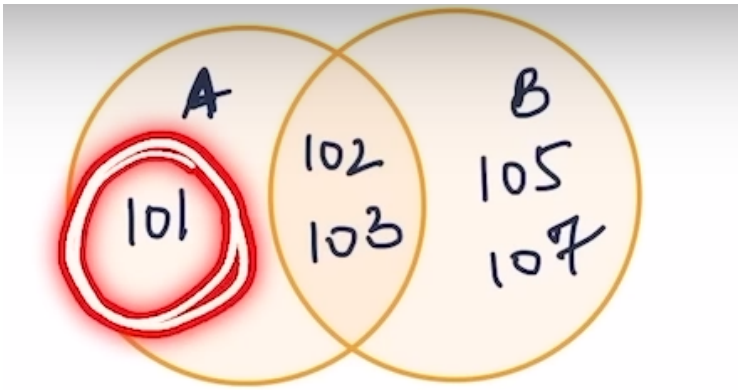
student_id	course
102	english
105	math
103	science
107	computer science

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 chcek ,

```
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:

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 synta...	0.000 sec
84	01:36:50	WHERE b.id IS NULL	Error Code: 1064. You have an error in your SQL synta...	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:

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 synta...	0.000 sec
84	01:36:50	WHERE b.id IS NULL	Error Code: 1064. You have an error in your SQL synta...	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
UNION                       ======= FULL JOIN=========
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          ……. EXCIUSIVE 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 Grid | Form Editor

Result 41 | Read Only

Output

Action Output

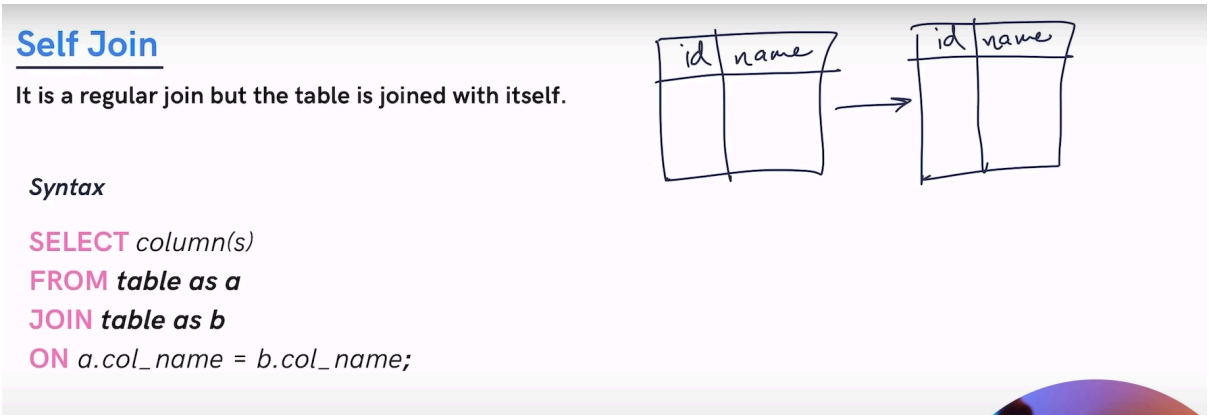
#	Time	Action	Message	Duration / Fetch
84	01:36:50	WHERE b.id IS NULL	Error Code: 1064. You have an error in your SQL synta...	0.000 sec
85	01:37:00	SELECT * FROM student as a LEFT JOIN course as ...	1 row(s) returned	0.016 sec / 0.000 sec
86	01:39:53	SELECT * FROM student as a RIGHT JOIN course as...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

There is one more join known as ,

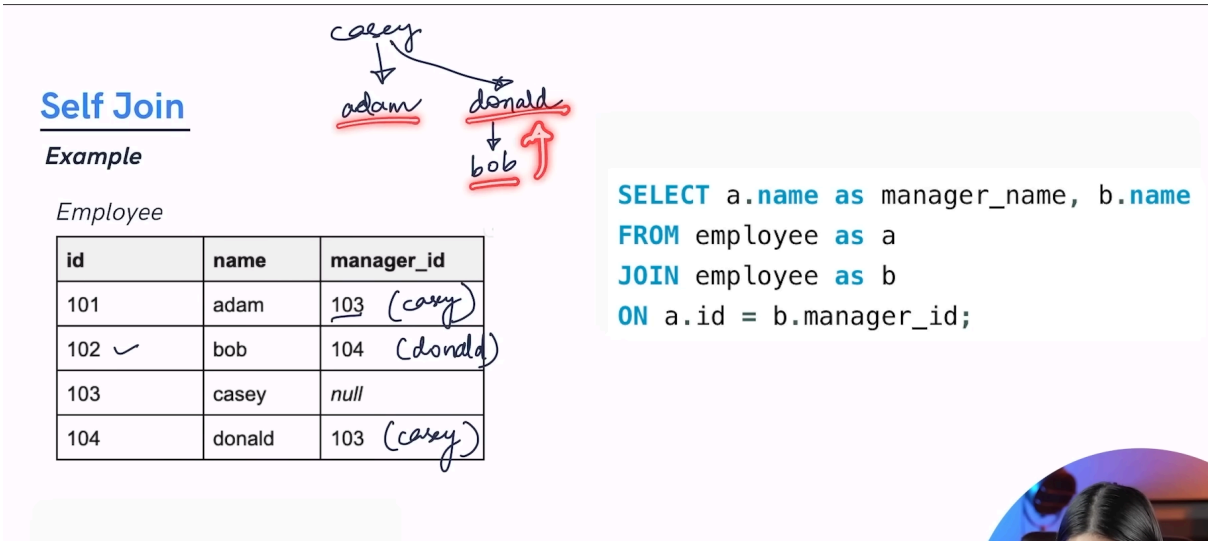
SELF JOIN:-

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



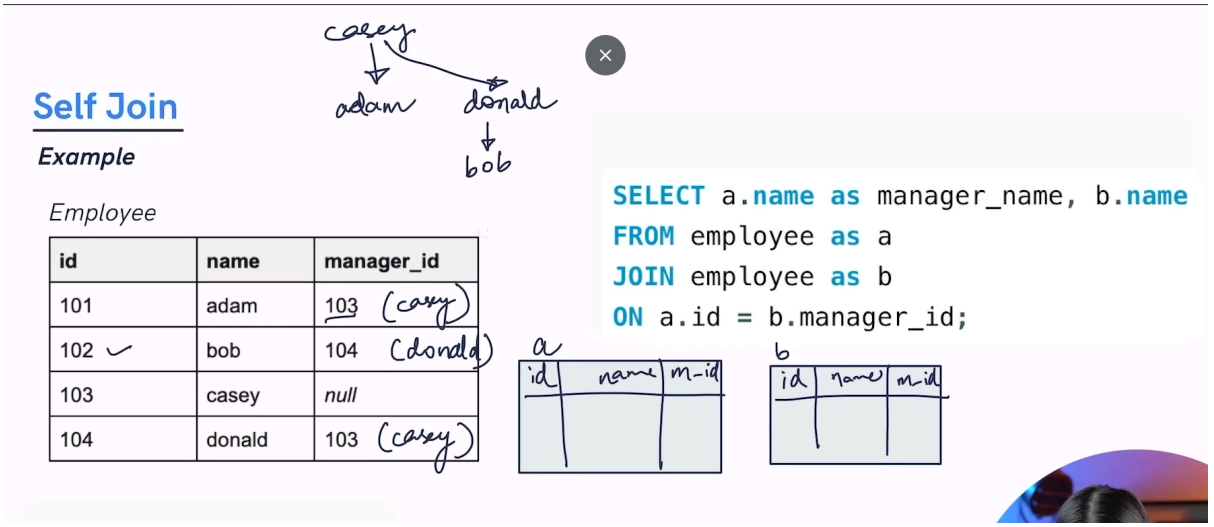
When we want to join two same tables …..

Here will just write **JOIN** in code ..
Example for usecase ,
Lets suppose we have table **Employee**



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,



Lets create employee table

```sql
CREATE TABLE employee(
  id INT PRIMARY KEY,
  name VARCHAR(50),
  manager_id INT
);

INSERT INTO employee (id, name, manager_id)
VALUES
(101, "adam", 103),
(102, "bob", 104),
(103, "casey", NULL),
(104, "donald", 103);

SEL
```

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. Unkn
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), (1...	4 row(s) affected Records: 4

Lets display it ,

```sql
SELECT * FROM employee;
```

130% 24:40

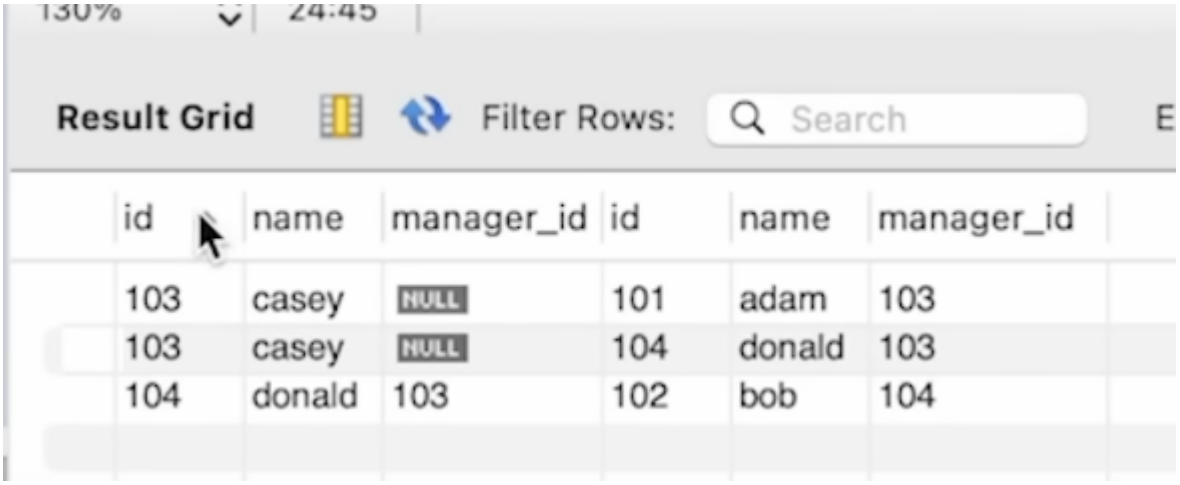
Result Grid Filter Rows: Search Edit: Export/Import:

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 ,

```sql
SELECT *
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

Output ,



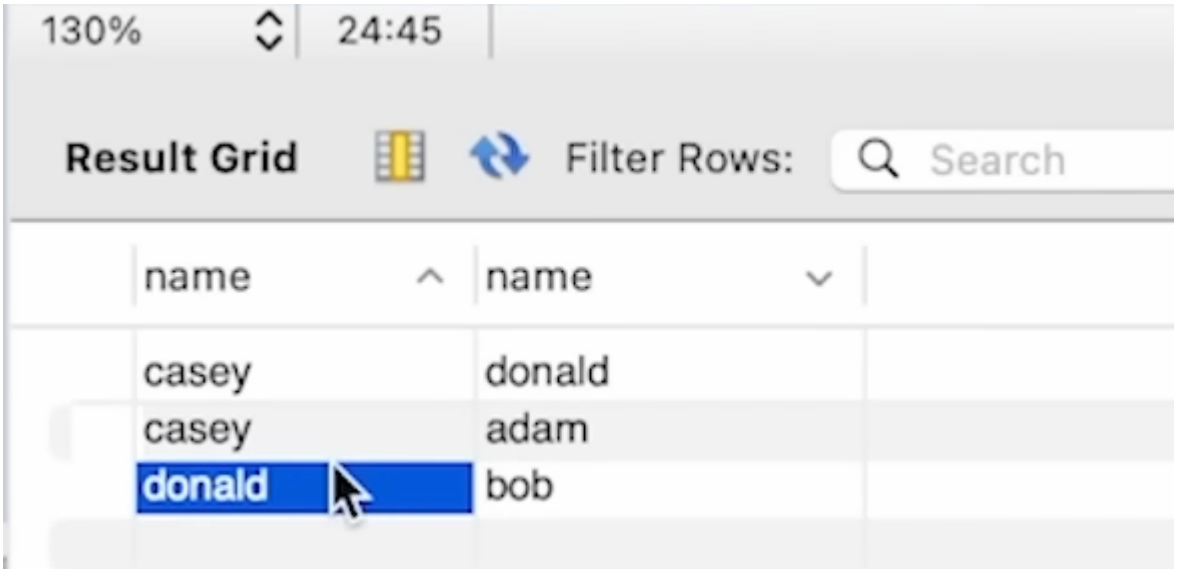
Let's read from RIGHT to LEFT , because query executes from right to left

adam ‘s manager is casey , donald manager is also casey aand bob has donald as manager .

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

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

Output is ,



Now both are displaying as **name.** Lets print LEFT table as **manager_name**

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

Output is ,

130% 24:45

Result Grid Filter Rows: Search Export:

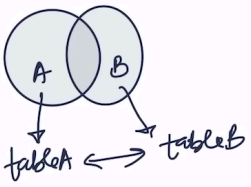
manager_name	name
casey	adam
donald	bob
casey	donald

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 store .
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.

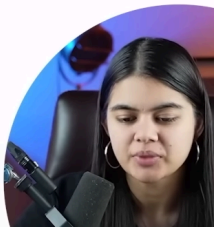


To use it :

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

Syntax

```sql
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
donald

;

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

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

Output,

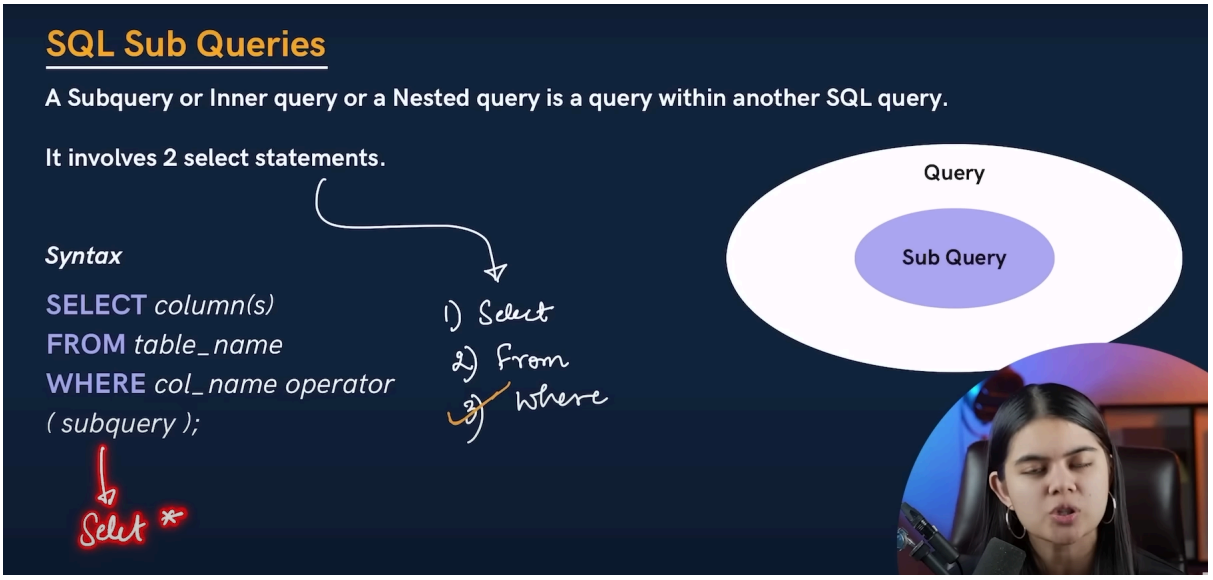
name
adam
bob
casey
donald
adam
bob
casey
donald

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

SQL Sub Queries:-

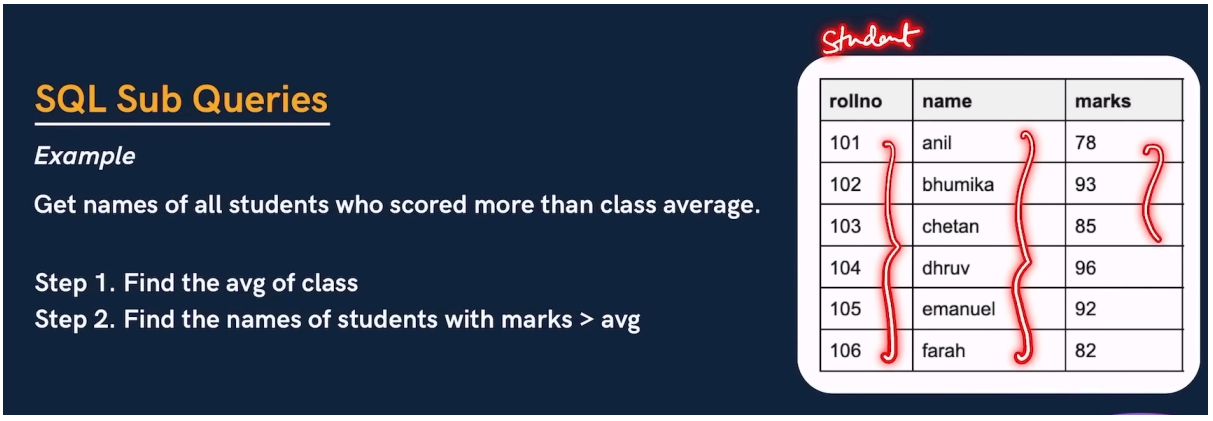
There are 3 methods to write sub queries …

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



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 ,



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:

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

Result Grid | Form Editor | student 43 | Apply | Revert

Output

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:

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;
```

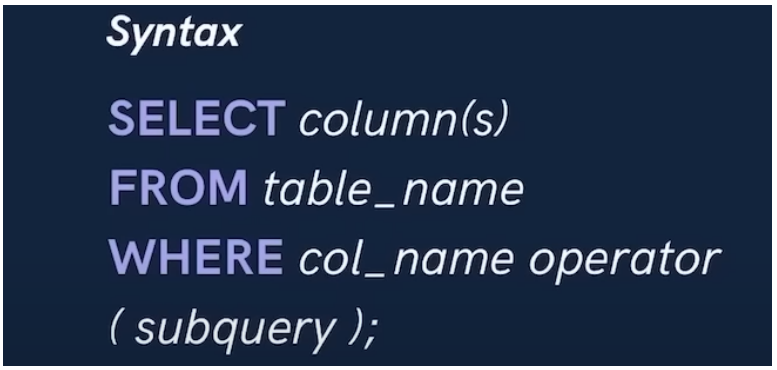
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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.




```
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 ) ;
```

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

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

Result Grid | Form Editor

student 47 × | Read Only

Output

Action Output

#	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 ,

SQL Sub Queries

Example

Find the names of all students with even roll numbers.

Step 1. Find the even roll numbers
Step 2. Find the names of students with even roll no

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;
```

Result Grid | Filter Rows: | Edit:

rollno
102
104
106
NULL

STEP 2:-

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

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

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);
```

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

name
bhumika
dhruv
farah

Result Grid | Form Editor

student 50 | Read Only

Output

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:

name	rollno
bhumika	102
dhruv	104
farah	106
NULL	NULL

Result Grid | Form Editor

student 51 | Apply | Revert

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

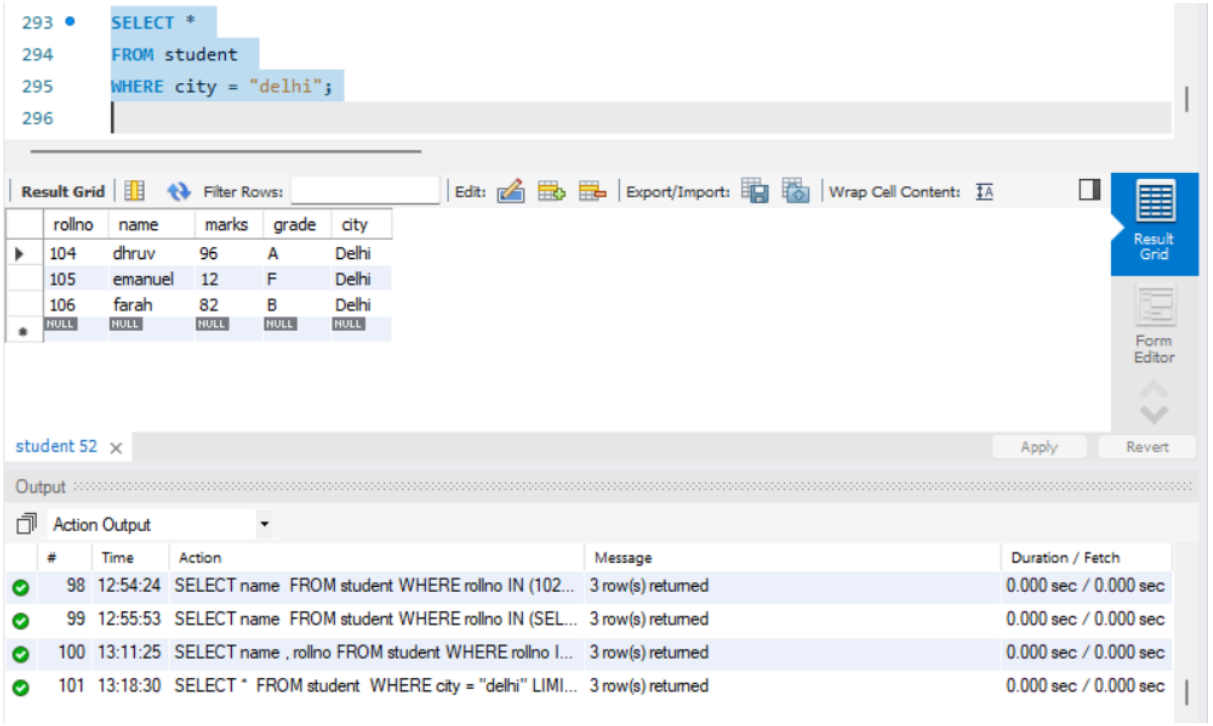
Example with FROM

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

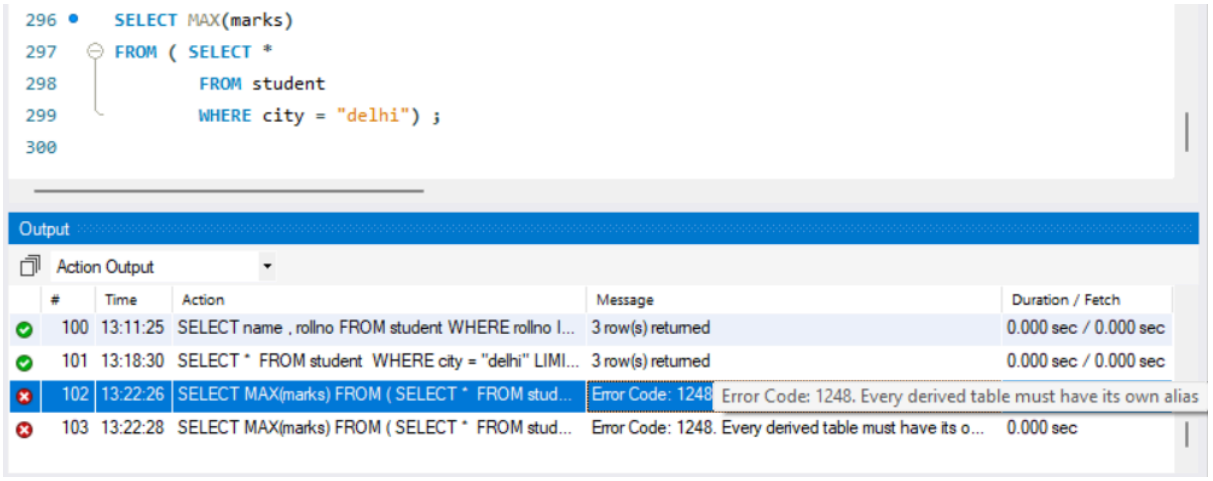
rollno	name	marks	city
101	anil	78	Pune
102	bhumika	93	Mumbai
103	chetan	85	Mumbai
104	dhruv	96	Delhi
105	emanuel	92	Delhi
106	farah	82	Delhi

STEP 1:-



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.**



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:

MAX(marks)
96

Result Grid | Form Editor

Result 53 | Read Only

Output

Action 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 Grid | Form Editor

Result 54 | Read Only

Output

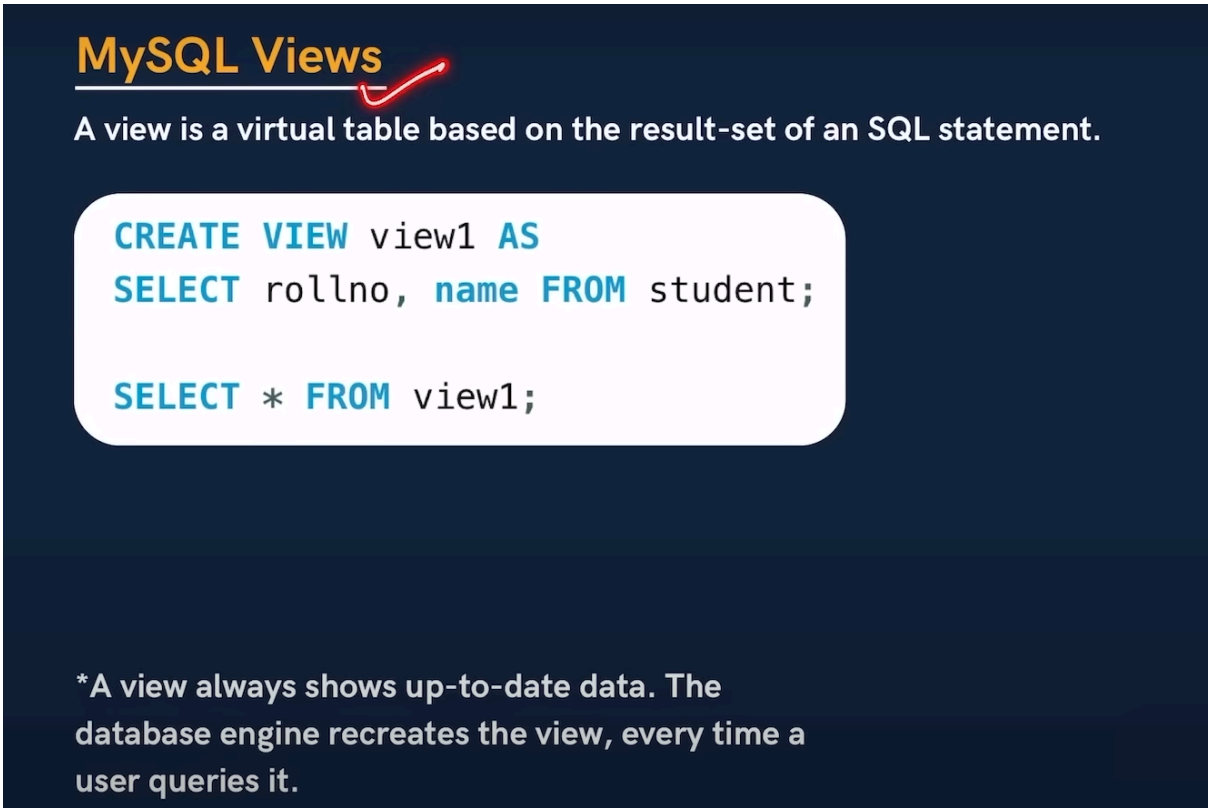
Action Output

#	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

---

WE rarely use SELECT for Sub query ,

MySQL Views



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;
```

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

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

Result Grid | Form Editor

view1 55 × 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;
```

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

rollno	name	marks
102	bhumika	93
104	dhruv	96

Result Grid | Form Editor

view1 56 × 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;**