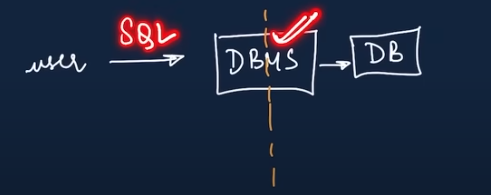
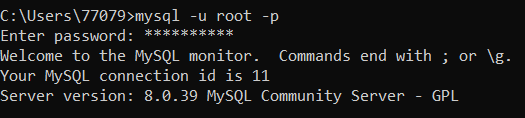
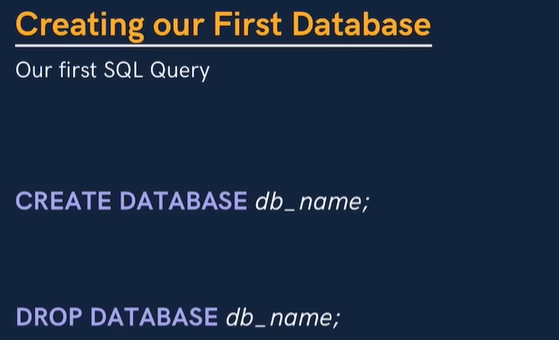
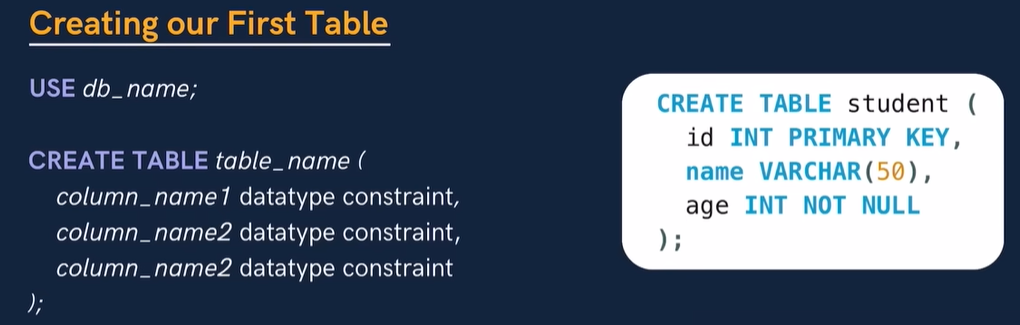
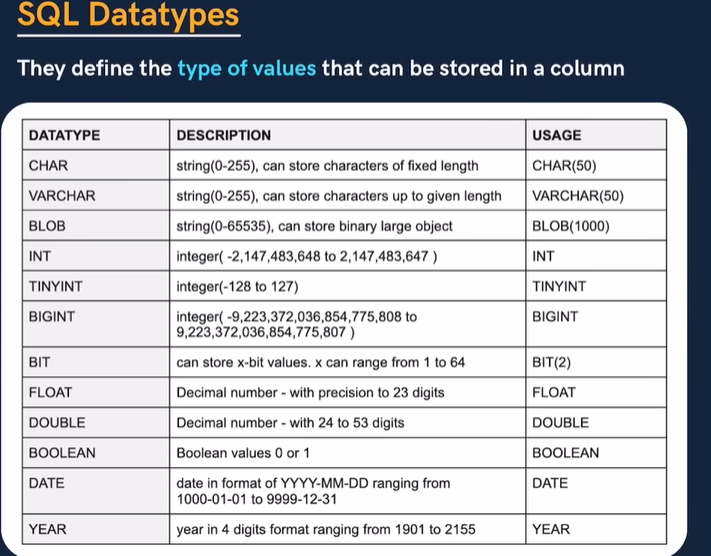
**Database –** Database is a collection of data in a format that can be easily accessed.  
A software application used to manage our DB is called DBMS (Database Management System).  
 **Type of Databases -**   
1 – Relational 🡪 Data store in tables. eg 🡪 MySQL , Oracle , PostgreSQL , SQLServer , IBM DB2  
2 – Non-relational (NoSQL) 🡪 data does not store in tables. eg 🡪 MongoDB , Cassandra **\*\* We use SQL to work with relational database only \*\*.  
  
SQL –** **Structured Query Language**It used to perform the operation like CRUD (Create, Read, Update, Delete) on the data which are stored in relational databases.   
Here using SQL User is trying to perform the operation on DB so he will write a query for that and DBMS will process in backend from DB and return the result to user. **Difference between SEQUEL and SQL 🡪** SQL was initially called SEQUEL (Structured English Query Language). It was developed by IBM in the early 1970s as part of their System R project which aimed to create a relational database model but Due to a trademark issue with a UK-based company that had rights to the name "SEQUEL," IBM had to change the name to SQL (Structured Query Language) so nowit's officially known as SQL. You can call it either SEQUEL or SQL.

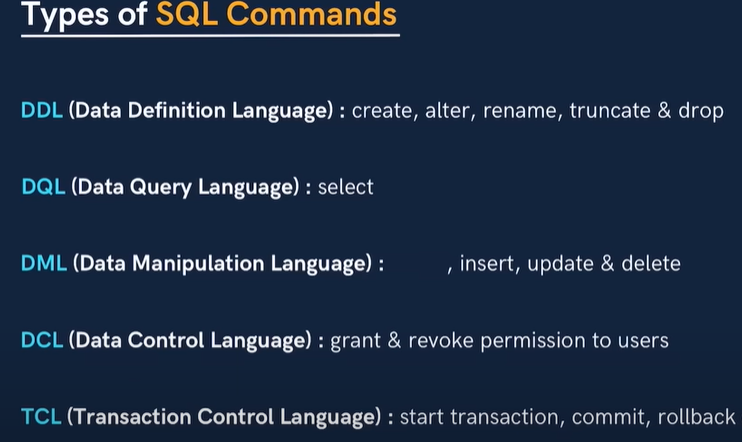
**How to install MySQL DB – Go to 🡪** [**https://www.mysql.com/downloads/**](https://www.mysql.com/downloads/) **🡪 Now at bottom Click on 🡪** [MySQL Community (GPL) Downloads »](https://dev.mysql.com/downloads/) 🡪 Now click on 🡪 [MySQL Community Server](https://dev.mysql.com/downloads/mysql/) 🡪 Now click on 🡪 **Windows (x86, 64-bit), MSI Installer Download 🡪 Now just click on 🡪** [**No thanks, just start my download.**](https://dev.mysql.com/get/Downloads/MySQL-9.0/mysql-9.0.1-winx64.zip) **Now install Workbench to manage server 🡪 Go to same link 🡪** [**https://www.mysql.com/downloads/**](https://www.mysql.com/downloads/) 🡪 then click on 🡪 [MySQL Workbench](https://dev.mysql.com/downloads/workbench/) 🡪 click on download and choose just start my download option to avoid sign In / login process.  
  
OR --------------🡪  
  
To avoid separate-separate download workbench and MySQL server better to choose option 🡪  
[MySQL Installer for Windows](https://dev.mysql.com/downloads/windows/) 🡪 choose 🡪 **Windows (x86, 32-bit), MSI Installer (303.6 MB)**

**🡨 ------------------- OR ---------🡪  
Follow this Youtube link 🡪** [**https://www.youtube.com/watch?v=jYMKQIzkMyk**](https://www.youtube.com/watch?v=jYMKQIzkMyk)To check MySQL installed or not setup environment variable 🡪 go to 🡪 **C:\Program Files\MySQL\MySQL Server 8.0\bin** 🡪 copy this path and open environment variable 🡪 in path add new this path and 3 times OK. 🡪 Now open cmd 🡪 type below command 🡪 It will show version details.  
 **To start 🡪 type mysql –u root –p 🡪 enter password MySQL@2024 🡪 now you can use this.  
  
Here –u is for User which is root –p is for password.**

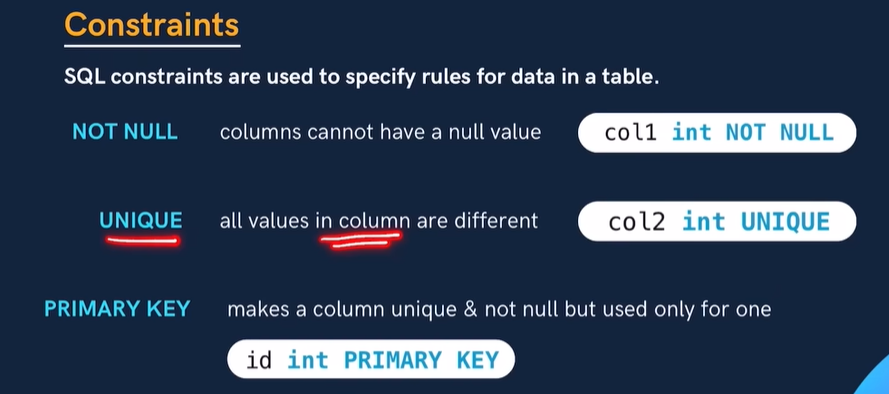
**Follow this tutorial -** [**https://www.youtube.com/watch?v=hlGoQC332VM**](https://www.youtube.com/watch?v=hlGoQC332VM) **How to create Database 🡪  
  
  
It is not case sensitive. You can write query in upper case as well as lower case.  
  
  
Datatypes 🡪  
  
  
BIT(1) => 0 or 1 so BIT(2) means 00 , 01 or 10 , 11  
BLOB 🡪 Used to store bigger string   
  
  
Difference between Signed and Unsigned data types 🡪  
  
Signed** 🡪 means that the data type can store both **positive** and **negative** values. For example, a SIGNED INT can hold values from **-2,147,483,648 to 2,147,483,647  
  
Unsigned** 🡪 means that the data type can only store **non-negative** values (i.e., positive numbers and zero). For example, an UNSIGNED INT can hold values from **0 to 4,294,967,295** (the upper limit doubles, but it can't hold negative values). **Difference b/w CHAR and VARCHAR 🡪**The main difference between CHAR and VARCHAR in SQL is how they handle storage and performance for string data.  
 **CHAR (Fixed-Length)**:

* **Fixed size**: CHAR will always take up the defined length of space, regardless of the actual length of the string stored.
* For example, CHAR (10) will always use 10 bytes, even if you store the string 'abc'. The remaining space is padded with spaces.

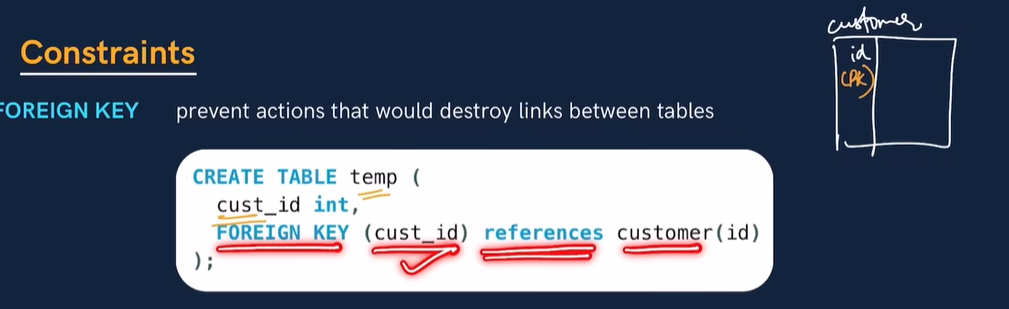
**VARCHAR (Variable-Length)**:

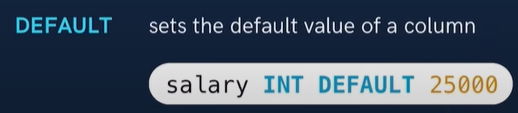
* **Variable size**: VARCHAR only uses as much space as needed to store the string, plus 1 or 2 bytes for length information.
* For example, VARCHAR (10) will store 'abc' using just 3 bytes (plus overhead for the length) instead of 10. **  
    
  IF NOT EXISTS 🡪** It will check existence of database and return warning if exist. **IF EXISTS 🡪** It will also give warning if DB already exist.

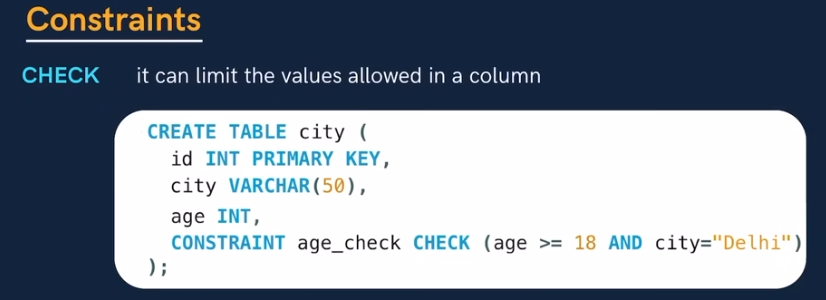
**------------------------------------ Keys 🡪**

**1 - Primary Key 🡪** It is a column in a table that uniquely identifies each row (a unique id) and It would be only one in entire table and It should not be null. **2 - Foreign Key 🡪** A foreign key is a column in a table that refers to primary key of another table. There can be multiple foreign keys. It can have duplicate and null value.  
  
  
**------------------🡪 Constraints** 🡪 Constraint are used to specify rules for data in a table.  


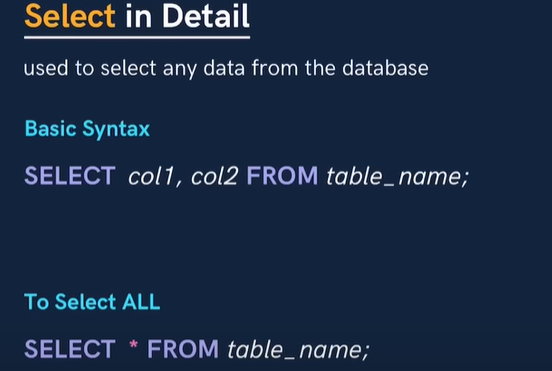
**1 - NOT NULL** 🡪 Column cannot have a null value. Eg – col1 int not null; i.e. col1 cannot be null.  
**2 - UNIQUE** 🡪 All values in column are different. Eg- id int unique. ID cannot have duplicate value.  
**3 - PRIMARY KEY** 🡪 Makes a column unique and cannot be null. Eg – id int primary key;  
 **4 - FOREIN KEY** 🡪 It link the two table of primary key. Here you can see there are two table – temp and customer and both table consist cust\_id and cust\_id is a primary key of 2nd table customer.

  
So you can see cust\_id column of first table we are making as foreign key by giving references for first column of 2nd table (customer) ID which is primary key.

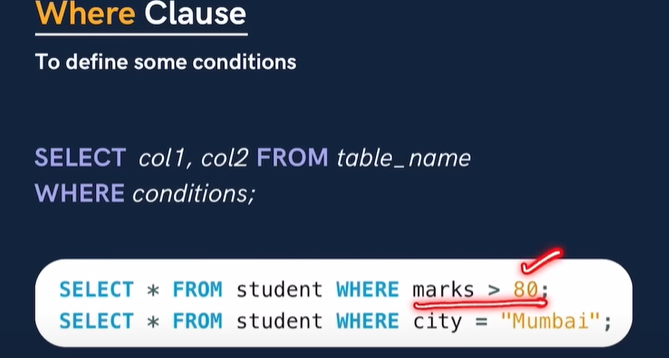
**5 - DEFAULT 🡪** sets the default value of a column if not added by user.  
for exp – salary **INT DEFAULT** 25000  


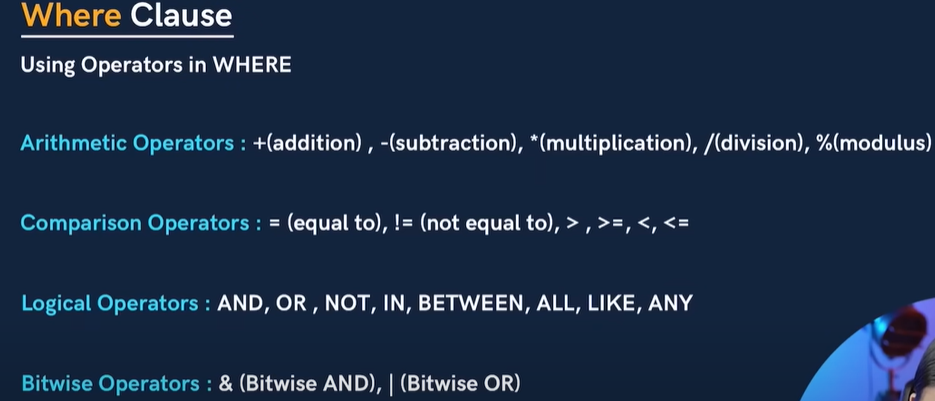
**6 - CHECK 🡪** it check the condition whether it is true or not.  
  
  
===========================================================================

------------------------------------ **SELECT** --------------------------------------------------



------------------------------------- **WHERE CLAUSE** ------------------------------------------------



--------------------------- **WHERE CLAUSE WITH OPERATORS** -------------------------------------  


### ------------------------------ LOGICAL OPERATOR ---------------------- ****1.**** ALL ****Operator****

The ALL operator compares a value to all the values in a subquery. It ensures that a condition is true for **all rows** in the subquery.

### **Example**: Find students who scored higher than all students in the PUNE CITY . ****2.**** LIKE ****Operator****

The LIKE operator is used for pattern matching with strings. You can use % for zero or more characters and \_ for a single character.

#### **Example**: Find students whose names start with "A".

### ****3.**** ANY ****Operator****

The ANY operator compares a value to **any** value in a subquery. It is true if the condition is satisfied by **at least one** row in the subquery.

#### **Example**: Find students who scored higher than at least one student in the Math subject.

------------------------------------------ **BITWISE OPERATOR** ----------------------------

#### **1. Bitwise AND (**&**)**

Returns 1 only if both corresponding bits are 1  
  
SELECT 5 & 3 AS Result;  
Binary representation:

* 5 = 101
* 3 = 011

Bitwise AND:

* 101 & 011 = 001

Result: 1

#### **2. Bitwise OR (**|**)**

Returns 1 if at least one corresponding bit is 1.

Copy code

SELECT 5 | 3 AS Result;

* Binary representation:
  + 5 = 101
  + 3 = 011
* Bitwise OR:
  + 101 | 011 = 111
* **Result**: 7

#### **3. Bitwise XOR (**^**)**

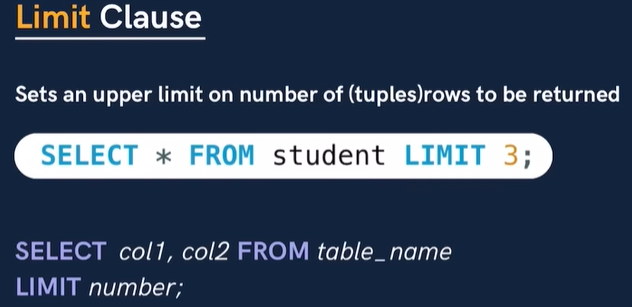
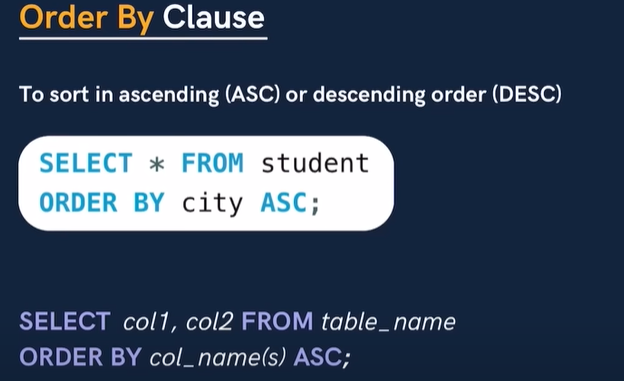
Returns 1 if the corresponding bits are different.

Copy code -

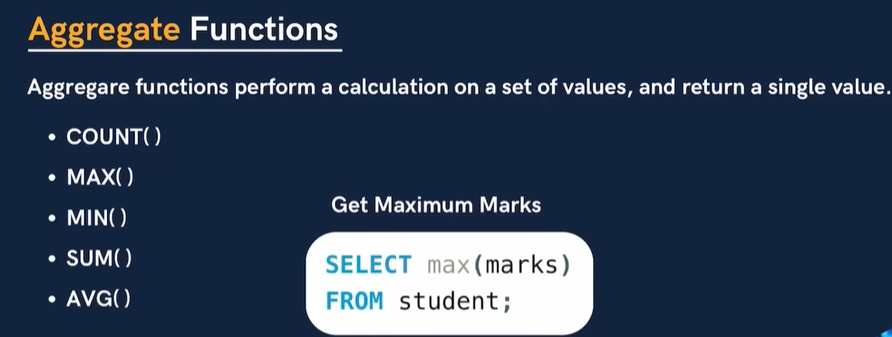
SELECT 5 ^ 3 AS Result;

* Binary representation:
  + 5 = 101
  + 3 = 011
* Bitwise XOR:
  + 101 ^ 011 = 110
* **Result**: 6

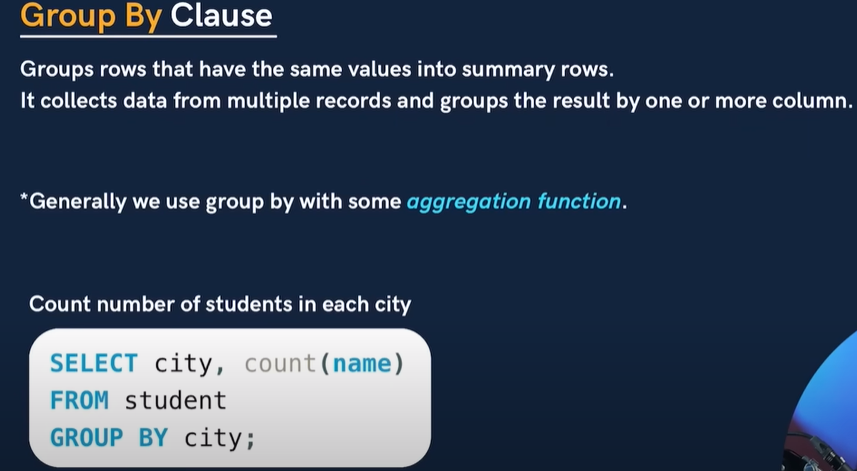
\*\*\*\*\*\*\*\*\*\*-------------------------------- **LIMIT CLAUSE** ------------------------\*\*\*\*\*\*\*\*\*\*\*

  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **ORDER BY CLAUSE** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

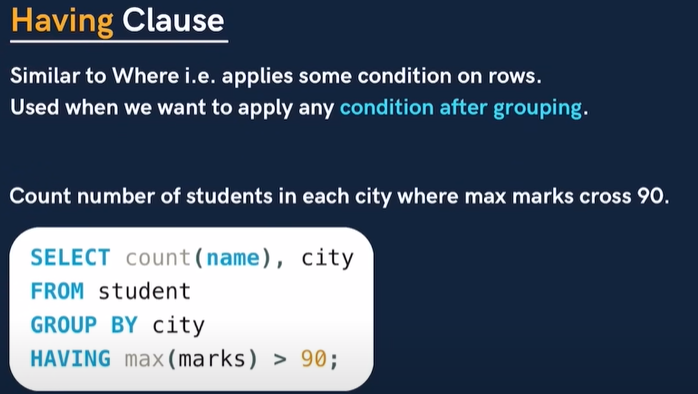
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **AGGREGATE FUNCTIONS** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



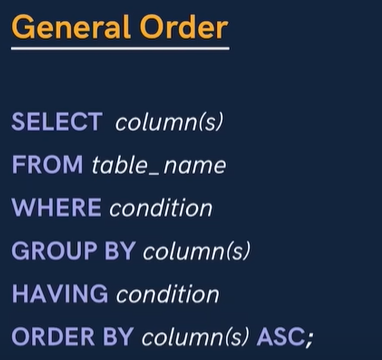
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **GROUP BY CLAUSE** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



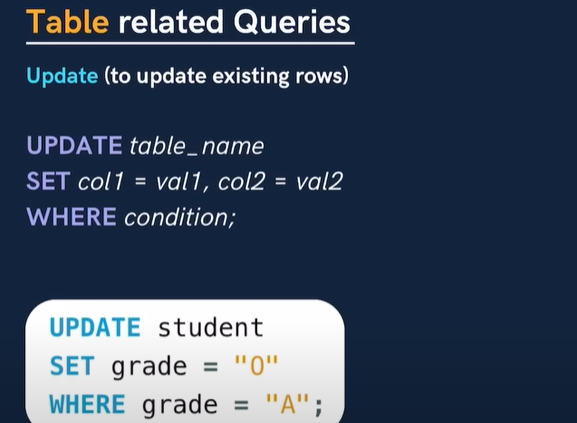
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **HAVING CLAUSE** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **GENERAL ORDER TO USE SYNTAX** \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

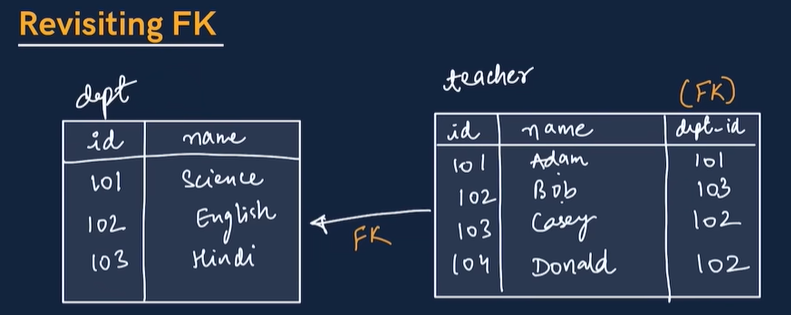


-------------------------------- **TABLE RELATED QUERIES - DML** ------------------------------------------





====================== **FOREIGN KEY** =================================

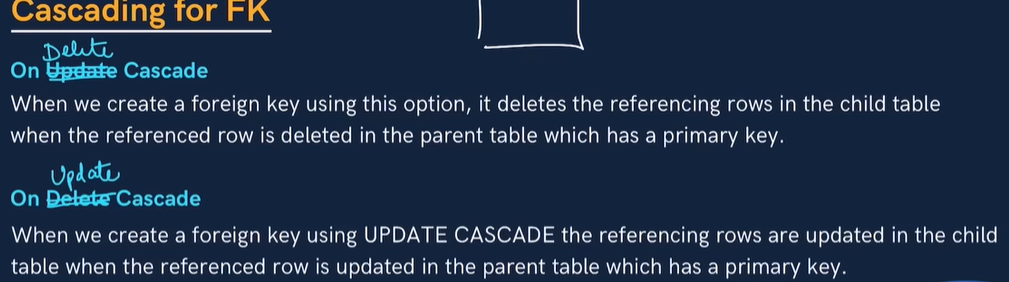
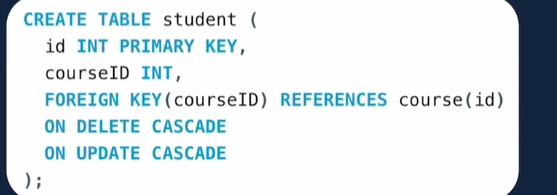


Primary key of 1st table having correlation with 2nd table with dept-id. That is called Foreign key.

You can check this table relationship into EER (Entity Relationship Diagram)..

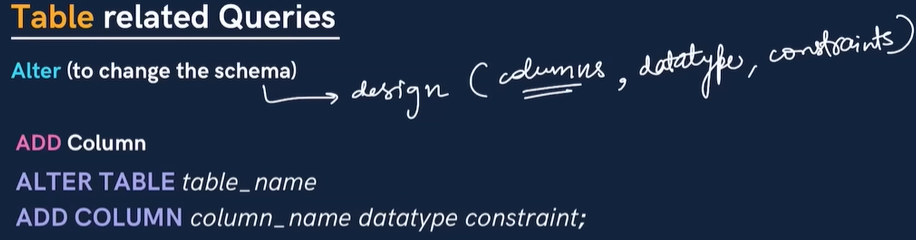
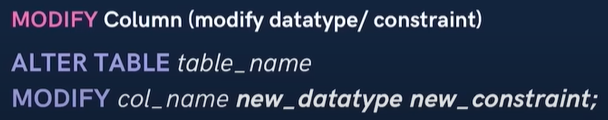
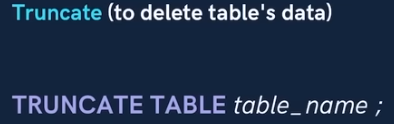
**Go to – Database -> Reverse Engineer 🡪 Next 🡪 Next 🡪 choose your DB (which having table of primary key and foreign key 🡪 Next 🡪 Next till execute 🡪 Execute 🡪 Finish**

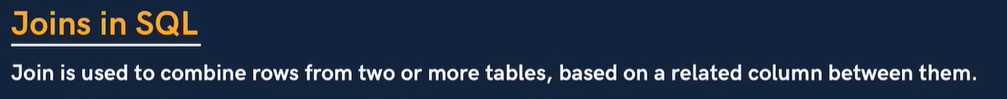
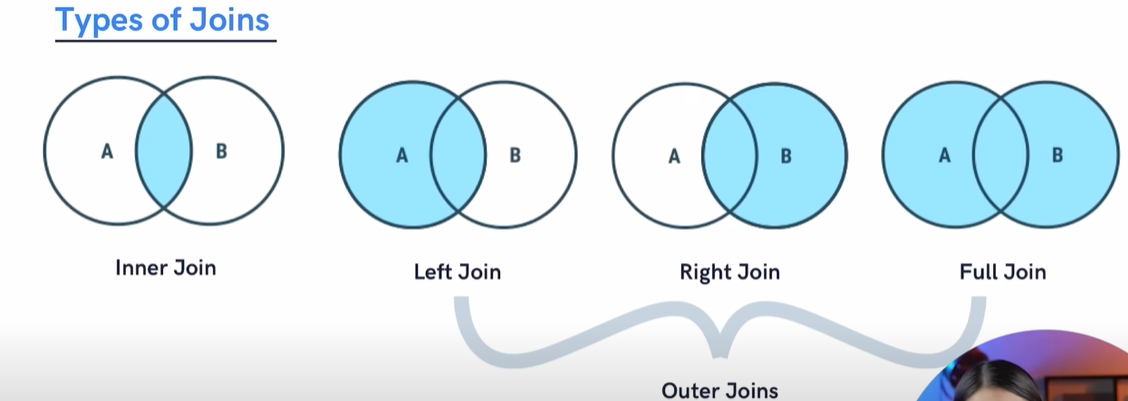
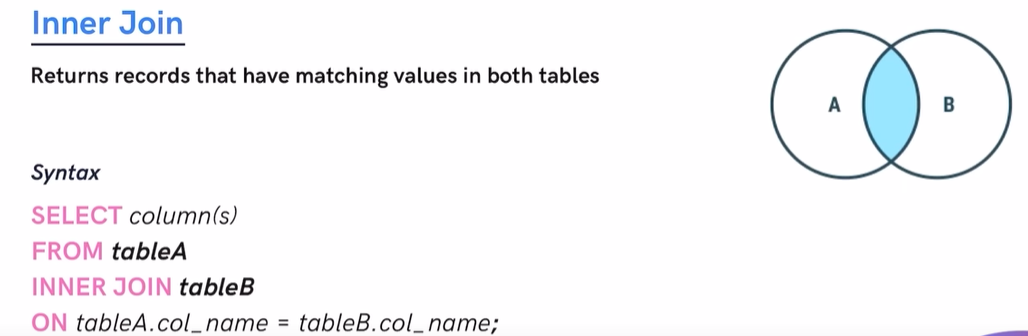
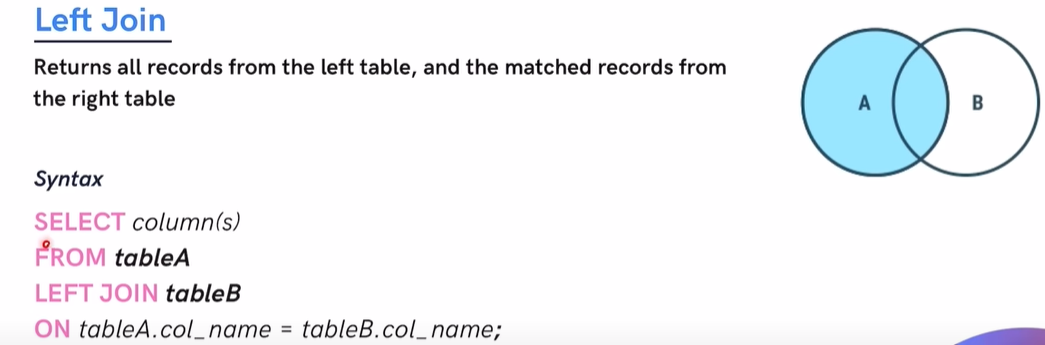
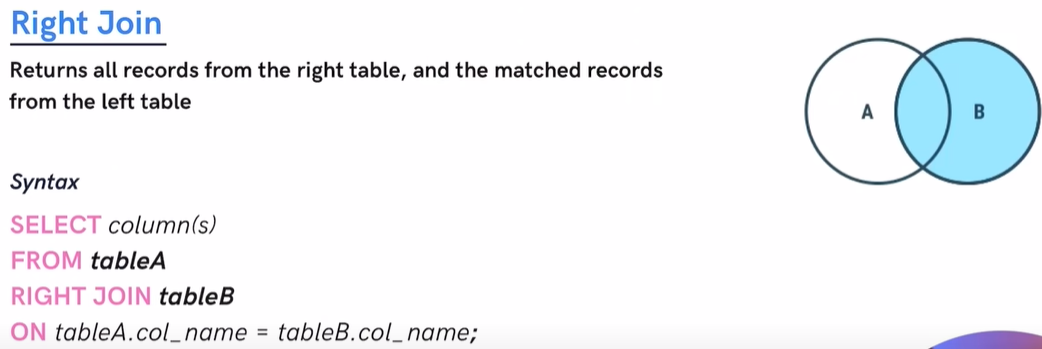
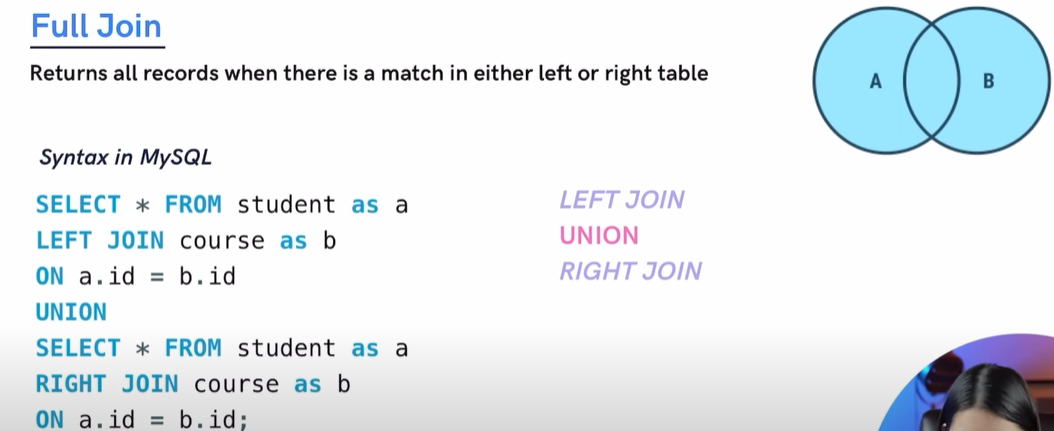
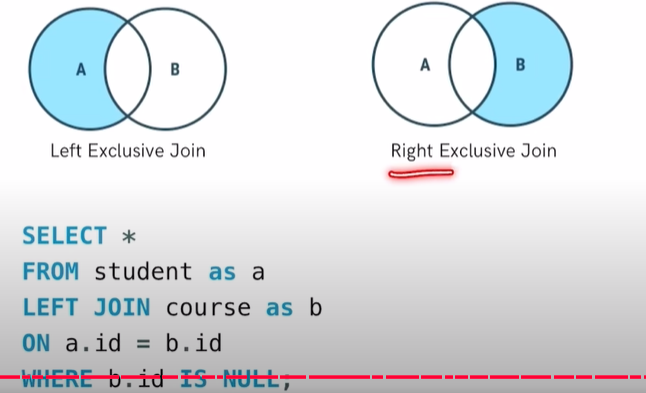
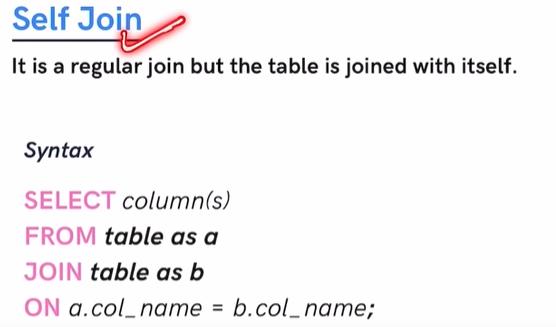
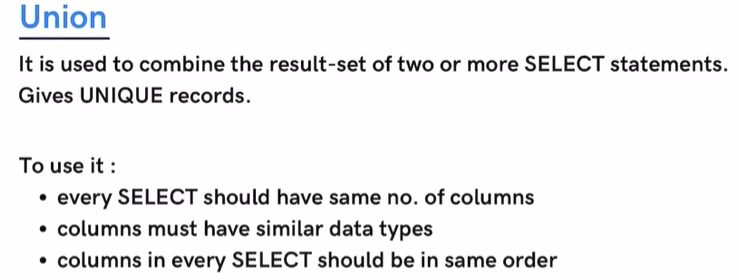
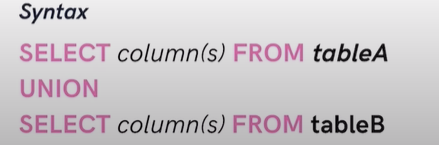
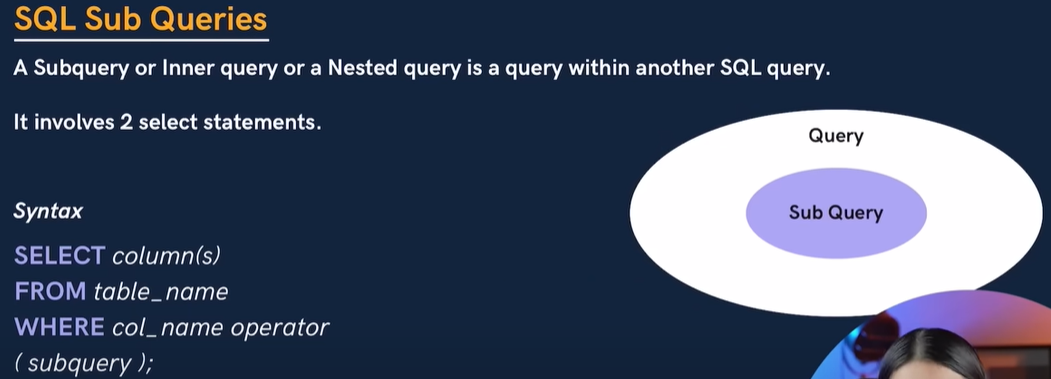
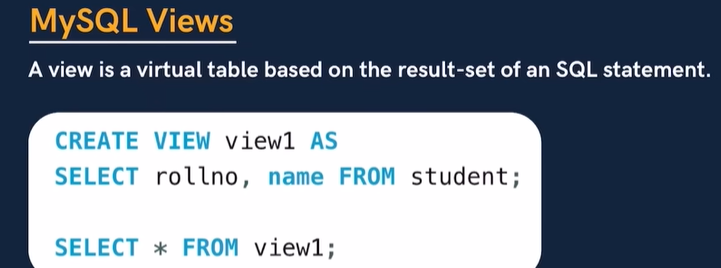
**1st table is called 🡪 Parent Table , 2nd table is called 🡪 Child Table**

**  
**

**ON DELETE CASCADE 🡪 Any deletion in parent table will effect in child table also.**

**ON UPDATE CASCADE 🡪 Any Changes in parent table will update in child table also.**

**  
  
  
  
  
**

**  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**

Here’s a step-by-step guide to upload all your MySQL queries (e.g., .sql files) to your GitHub account: 🡪 **Notes taken from Chat GPT. So in case any doubt just asked to chatgpt**.

### ****1. Prepare Your MySQL Query Files****

* Ensure all your MySQL queries are saved as .sql files in a folder on your local computer. For example:

Like day1.sql , day2.sql etc.

### ****2. Create a Repository on GitHub****

1. Log in to your [GitHub account](https://github.com).
2. Click on the **+** icon in the top-right corner and select **New Repository**.
3. Fill in the repository details:
   * **Repository Name**: For example, mysql-queries.
   * **Description**: Add a brief description (optional).
   * Choose between **Public** or **Private**.
4. Click **Create Repository**.

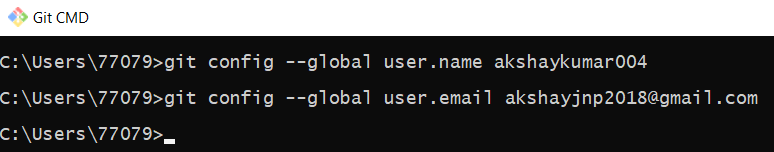
### ****3. Set Up Git on Your Local Machine****

1. If you don’t have Git installed, download and install it from [git-scm.com](https://git-scm.com/).
2. Configure Git with your GitHub account: **see below image**

GIT CMD -

Copy code -🡪

git config --global user.name "Your GitHub Username"

git config --global user.email [your-email@example.com](mailto:your-email@example.com)  
  


### ****4. Initialize a Git Repository****

1. Open a terminal or command prompt.
2. Navigate to the folder containing your .sql files:

bash

Copy code

cd /path/to/my-mysql-queries

1. Initialize the folder as a Git repository:

bash

Copy code

git init

### ****5. Add Files and Commit****

1. Add all .sql files to the staging area:

bash

Copy code

git add .

1. Commit the changes with a message:

bash

Copy code

git commit -m "Initial commit of MySQL queries"

### ****6. Link Your Repository to GitHub****

1. Copy the **repository URL** from GitHub (e.g., https://github.com/yourusername/mysql-queries.git).
2. Add the URL as the remote origin:

bash

Copy code

git remote add origin https://github.com/yourusername/mysql-queries.git

### ****7. Push Files to GitHub****

1. Push the files to the GitHub repository:

bash

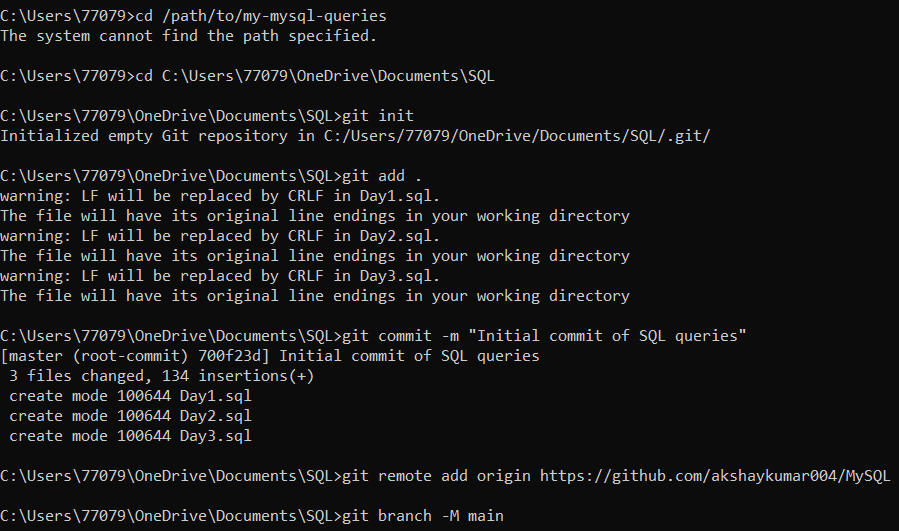
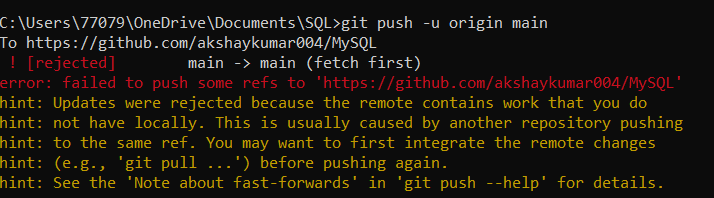
Copy code

git branch -M main

git push -u origin main

### ****8. Verify on GitHub****

* Go to your GitHub repository URL in a browser.
* You should see all your .sql files uploaded successfully.

**Note 🡪 Eg – See below image**   
  
  
  
  
  
  
  
After this command **git push -u origin main** If you’re getting below error then follow below image instruction  
  
  
The error you're encountering occurs because the remote repository on GitHub already has some changes or files, and your local branch is out of sync. To resolve this issue, you need to sync your local repository with the remote repository. Here's how to fix it:

### ****Steps to Resolve the Error****

#### **1. Pull the Remote Changes**

Before pushing, fetch and integrate the remote changes into your local branch:  
**git pull origin main --rebase**

* origin: Refers to the remote repository.
* main: Refers to the main branch.
* --rebase: Applies your changes on top of the pulled changes, keeping the commit history linear.

### Now in below image you can see I rebased it and push the code in main branch and it worked. Tips to Avoid This in the Future

* Always pull changes before making or pushing updates:

Copy code

**git pull origin main**

* Regularly sync your local repository with the remote to avoid large differences.

**Now Since you've already set up the repository, you don’t need to follow all the steps again for new .sql files. Here’s a simplified process for adding new files to the same repository:**

**Steps to Add a New .sql File to Your GitHub Repository**

1. **Place the New File in Your Local Folder**
   * Save your new .sql file in the same local folder where your Git repository is located.
2. **Stage the New File**
   * Add only the new file to the staging area:

Copy code

git add newfile.sql

*(Replace newfile.sql with the name of your file.)*

1. **Commit the New File**
   * Commit the change with a message:  
     Copy code  
     git commit -m "Add new SQL file: newfile.sql"
2. **Push the Changes to GitHub**
   * Push the update to the remote repository:

Copy code

git push origin main

**No Need to Repeat:**

* Initializing the repository (git init).
* Adding the remote URL (git remote add origin).
* Setting the branch to main (git branch -M main).

These steps were one-time setups and don’t need to be repeated.