# SQL Learning Notes

**Database:** Database is a structured collection of data that is stored, managed, and accessed electronically. It is designed to efficiently store, retrieve, update, and manage data in a way that ensures integrity, consistency, and security**.**

Database management system

Your APP

(Amazon)

Database

**Database Management System:** A **Database Management System (DBMS)** is **software that interacts with databases** to perform tasks such as **storing, retrieving, updating, and managing data** efficiently while ensuring **data integrity, security, and concurrency control**.

**MySQL, PostgreSQL, SQLite, etc., are not databases themselves, but rather Database Management Systems (DBMS)** that provide the tools and functionalities to interact with, manage, and manipulate data stored in databases.

## Creating Databases and Tables

* SHOW DATABASES;
* It will show the databases which is available in the system.
* CREATE DATABASE yethishwar\_rao;
* It will create a database.
* DROP DATABASE yethishwar\_rao;
* It will delete the database permanently.

NOTE: using snake\_case is a good practice naming databases, tables, column\_names etc…

* USE database\_name;
* Switches the active database to the specified one.
* Just use the above query to switch to the working database and work on it.
* SELECT DATABASE();
* Displays the currently selected database.

Inside a Database there will be Tables this is the area where we actually focus on and manipulate the things. A table consists of rows and columns, almost all SQL flavoured languages deals with tables, other than MangoDB and some more.

### Data Types

We have many numerical, character and date time data types, among all some are frequently used like VARCHAR(n), CHAR(n), INT, FLOAT, BIGINT etc…

To create tables use the below query

* CREATE TABLE methodist\_collage

( students VARCHAR(100),

roll\_no INT,

grade CHAR(10)

);

#To see the structure of the table

* DESCRIBE collages\_in\_hyderabad;

-- OR --

* SHOW COLUMNS FROM gurunanak;

-- OR --

* SHOW TABLES;

-- To delete a table permanamtly

* DROP TABLE gurunanak;

-- Insert values into the Table

* INSERT INTO malla\_reddy(students, roll\_no, grade)
* VALUES('Siddhartha\_rao', 90, 'S++');
* INSERT INTO malla\_reddy(students, roll\_no, grade)
* VALUES('Sai Kiran Rao',89,'S++');

We can also interchange the values like this👇… the order does not matter but make sure to tally the column names and values.

* INSERT INTO malla\_reddy(roll\_no, grade, students)
* VALUES(89,'S++','Sai Kiran Rao');

-- Insert multiple rows at a time with single insert command

* INSERT INTO malla\_reddy(students,roll\_no)
* VALUES('Kiara',33),
* ('preethi',24),
* ('Sushil',12),
* ('Rushi',22)
* ;

-- To see the values in a table

* SELECT \* FROM malla\_reddy;

-- By specifying NOT NULL while creating a table... The table dooesn't support any null values

* CREATE TABLE population(

state VARCHAR(20) NOT NULL,

population INT NOT NULL);

If we try to insert values like this, 👇 it will throw error

* INSERT INTO population(state)
* VALUES('TELANGANA')

ERROR: ERROR 1364 (HY000): Field 'age' doesn't have a default value

So to overcome this error we need to specify a default value which is suitable for that data type like below… DEFAULT value **only applies when a column is entirely omitted** from the INSERT statement, **not when NULL is explicitly inserted**. 👇

* create table people2(
* -> name varchar(20) DEFAULT 'No value here',
* -> age int default 0);

If we insert null values also it just show the default value instead of ‘NULL’ when we select the table.

If you want MySQL to **always** replace NULL with the default value, you **must** use NOT NULL in the table👇

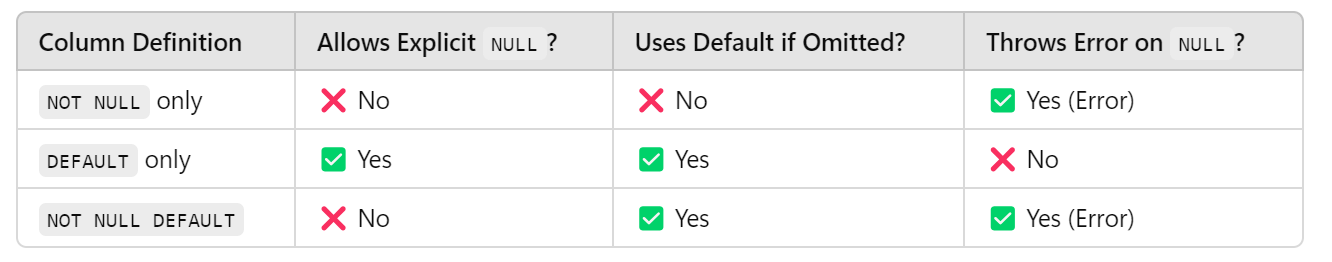
* create table people2(
* -> name varchar(20) NOT NULL DEFAULT 'No value here',
* -> age int NOT NULL default 0);

If you explicitly inserts a null value like this👇 it will take NULL value unless you mention NO NULL wile creating

* insert into people2(name, age)
* -> values(NULL,NULL);

-- If we want double codes in the middle of the name just mention back slash like thisit will ignore the next symbol and proceeds with no ERROR.👇

* INSERT INTO people2(name)
* VALUES('Yethishwar\'s Family');



### #Primary Key

A **Primary Key** is a unique identifier for each record in a database table. It ensures that no two rows have the same value in the primary key column and that the value cannot be **NULL**. This helps maintain data integrity and allows efficient data retrieval.

This query ensures no null values allowed and two different rows will contain unique values.

* create table unique\_people(
* aadhar bigint not null primary key,
* name varchar(100),
* age int);

You can also specify it at the end like this

* PRIMARY KEY (aadhar)

If you try to insert identical values like this it will throw an error like below

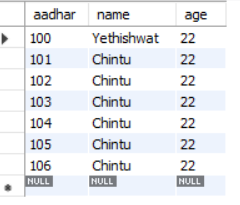
* insert into unique\_people(aadhar,name,age)
* values(1,'Yethishwar',21),
* (1,'Yethishwar',21);

ERROR 1062 (23000): Duplicate entry '1' for key 'unique\_people.PRIMARY'

### #AUTO\_INCREMENT

he **AUTO\_INCREMENT** constraint is used in SQL (specifically in MySQL and some other databases) to automatically generate a unique, incrementing value for a **Primary Key** column.

- If we you want the primary key values to be auto incremented please specify auto\_increment liek this

* CREATE TABLE people4(
* aadhar BIGINT auto\_increment,
* name VARCHAR(100),
* age INT,
* PRIMARY KEY(aadhar)
* );
* INSERT INTO people4(aadhar,name,age)
* VALUES(100,'Yethishwat',22);
* select \* from people4;
* INSERT INTO people4(name,age)
* VALUES('Chintu', 22),
* ('Chintu', 22),
* ('Chintu', 22),
* ('Chintu', 22),
* ('Chintu', 22),
* ('Chintu', 22);

MySQL picks the next available number after the highest existing value in the column, not the lowest missing number.

### FINAL ASSIGNMENT

* create table employee(
* id int primary key auto\_increment,
* last\_name varchar(100) not null,
* first\_name varchar(100) not null,
* middle\_name varchar(100),
* age int not null,
* current\_status varchar(100) not null default 'Employed');

## CRUD(Create, Read, Update, Delete) BASICS

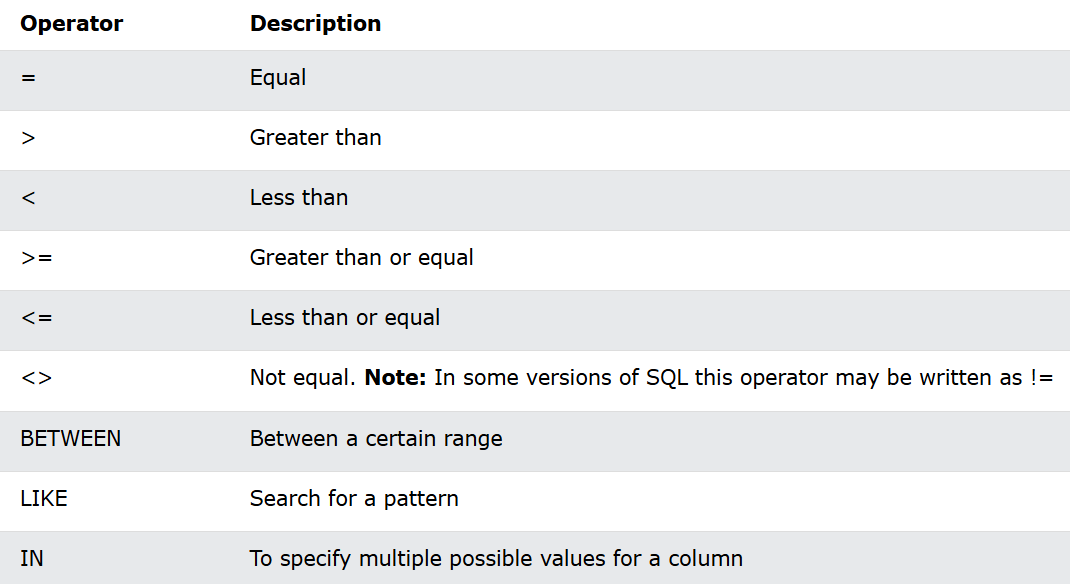
### READ

In **MySQL**, a **read operation** refers to any query that retrieves data from the database without modifying it.

Lets take a cats table and work on it

* SELECT \* FROM cats;
* SELECT name FROM cats;
* SELECT name, breed FROM cats;
* SELECT breed, name, age, age FROM cats;

### WHERE



* SELECT name FROM cats
* WHERE breed = 'Tabby';
* SELECT name,age FROM cats
* WHERE age <= 10;
* SELECT \* FROM cats
* WHERE breed != 'Tabby';
* SELECT \* FROM cats
* WHERE age BETWEEN 5 and 10;
* SELECT name FROM cats
* WHERE breed LIKE 'T%';
* SELECT \* FROM cats
* WHERE breed IN ('Tabby','Sphynx');

### ALIASES

* Aliases are used to give a table, or a column in a table, a temporary name.
* Aliases are often used to make column names more readable.
* An alias only exists for the duration of that query.
* An alias is created with the AS keyword.

If the query is big and a column name is used multiple times, using an **alias** can make the query **shorter, more readable, and easier to maintain**.

* SELECT cat\_id AS id, breed AS b,name AS n FROM cats;

### UPDATE

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

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

* UPDATE cats
* SET name = 'Preeti'
* WHERE age >13;
* UPDATE cats
* SET name = ‘MISTY’, breed = ‘DHUNNA’
* WHERE age = 13;

**A Good Rule of Thumb** **👉 Always SELECT the data before performing an UPDATE or DELETE.because once you done update you cannot undo it.**

### DELETE

The DELETE statement is used to delete existing records in a table by applying a where condition.

* DELETE FROM cats
* WHERE age > 10;
* DELETE FROM cats; #Total values will be deleted but remains structure.

## String Operations

NOTE:- source filename.sql -> executes the commands present in that file.

### CONCAT(parameters)

Used to combine text and give custom column name

* SELECT CONCAT('Hi', 'Yethishwar') as Lovely;
* SELECT CONCAT(author\_fname, ' ', author\_lname) FROM books;
* SELECT CONCAT(author\_fname, ' ', author\_lname) AS author\_fullname FROM books;

### CONCAT\_WS(‘separator’, parameters…)

It just works like concat but it combines with a given separator

* SELECT CONCAT\_WS(' ~ ',author\_fname, author\_lname, pages, released\_year) as loverboy FROM books;



### SUBSTRING() OR SUBSTR()

* SELECT SUBSTRING(author\_fname, 2) FROM books;
* Extract a substring from a string (start from the position 2 onwards till at the last)
* SELECT SUBSTRING(author\_fname, 1, 3) FROM books;
* Extract a substring from a string (start at position 1, extract 3 characters)
* SELECT SUBSTRING(author\_fname, -1),author\_fname FROM books;
* Extract the last letter of the string
* SELECT SUBSTRING('Yethishwar', -4,2);
* Extract a substring from a string (start from the end, at position -4, extract 2 characters):

### COMBINING CONCAT(SUBSTRING(), ‘ ’)

The query starts executing from nested function and moving to outer function and filtering the stuff.

* SELECT
* CONCAT(SUBSTRING(author\_fname, 1, 5),
* ' ',
* SUBSTRING(author\_lname, 1, 3),
* ' ',
* '!!!!!') AS combined\_string
* FROM
* books;
* SELECT
* SUBSTRING(author\_fname, 1, 1), SUBSTRING(author\_lname, 1, 1)
* FROM
* books;
* SELECT
* CONCAT(SUBSTRING(author\_fname, 1, 1),
* ' . ',
* SUBSTRING(author\_lname, 1, 1))
* FROM
* books;

### REPLACE()

The REPLACE() function replaces all occurrences of a substring within a string, with a new substring.

* SELECT REPLACE('Ye&\*th&\*is&\*hw&\*ar', '&\*', '');
* SELECT REPLACE(title, ' ', '-') from books;
* REPLACE(string, substring, new\_string)
* SELECT
* CONCAT(SUBSTRING(author\_fname, 1, 5),
* ' ',
* SUBSTRING(author\_lname, 1, 3),
* ' ',
* '!!!!!', ' ',
* REPLACE(title, ' ', '-')
* ) AS combined\_string
* FROM
* books;

### REVERSE()

The REVERSE() function reverses a string and returns the result.

* SELECT REVERSE(‘yethishwar’);
* SELECT REVERSE(author\_fname) from books;

### CHAR\_LENGHTH()

It prints the number of charaters of specified string

* SELECT CHAR\_LENGTH('Yethishwar'); -> 10
* SELECT CHAR\_LENGTH('Yethishwar') AS length , title FROM books;

### LENGTH()

It prints the number of bites of specfied string

* SELECT LENGTH('字字典'); -> 9

### UPPER() & LOWER() or UCASE() & LCASE()

Converts into upper case and lower case

* SELECT UPPER(title) FROM books;
* SELECT LOWER(title) FROM books;
* SELECT CONCAT('I LOVE ', UPPER(title), '!!!') FROM books;

### INSERT()

The INSERT() function inserts a string within a string at the specified position and for a certain number of characters.

* SELECT INSERT('HELLO WORLD',6,5, ' YETHISHWAR ');

### LEFT() & RIGHT()

Return the left and right most charaters with the specified length

* SELECT LEFT('Yethishwar',5);
* SELECT RIGHT('Yethishwar rao', 3);

### REPEAT()

Repeats the string upto specified count times

* SELECT REPEAT('Yethishwar Rao ' , 5);
* SELECT REPEAT(author\_fname, 3) FROM books;

### TRIM()

Trims specified charaters at the beginning or ending.

* SELECT TRIM(' yETHISHWAR ');

**yETHISHWAR**

* SELECT TRIM(LEADING '\*' FROM '\*\*\*\*\*\*\*\*\*\*yETHISHWAR\*\*\*\*\*\*\*\*');

**yETHISHWAR\*\*\*\*\*\*\*\***

* SELECT TRIM(BOTH '\*' FROM '\*\*\*\*\*\*\*\*\*\*yETHISHWAR\*\*\*\*\*\*\*\*');

**yETHISHWAR**

## Refining Selections

### DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values.

* SELECT DISTINCT title FROM books;
* SELECT DISTINCT author\_fname FROM books;
* SELECT DISTINCT(CONCAT(author\_fname, ' ', author\_lname)) FROM books;
* SELECT DISTINCT author\_fname, author\_lname FROM books;

### ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

* SELECT \* FROM books
* ORDER BY pages DESC;

For string values the ORDER BY keyword will order alphabetically

* SELECT \* FROM books
* ORDER BY title ASC, pages DESC;

we can also use number to select column by specified its location starting from 1.

* SELECT title, author\_fname, author\_lname,pages FROM books ORDER BY 4;
* SELECT author\_fname, released\_year, book\_id FROM books
* ORDER BY author\_fname, released\_year, book\_id;
* SELECT CONCAT(author\_fname, author\_lname) AS author FROM books ORDER BY author;

### LIMIT 1

If we want top 10 rows or from specific location to these many rows we can do it using LIMIT

* SELECT book\_id, author\_fname, author\_lname, released\_year FROM books ORDER BY released\_year limit 5;
* SELECT book\_id, author\_fname, author\_lname, released\_year FROM books ORDER BY released\_year limit 2,5;

### LIKE

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

* SELECT book\_id, author\_fname, author\_lname FROM books WHERE author\_fname LIKE '%Da%' AND author\_lname LIKE 'E%';
* SELECT author\_fname FROM books WHERE author\_fname LIKE '\_\_\_\_\_';
* SELECT author\_fname FROM books WHERE author\_fname LIKE '\_a\_';

IF we want to search ‘%, \_’ these items in the data we need to use backslash(\) to tell query to this is to be search

* SELECT title FROM books WHERE title LIKE '%\%%';
* SELECT title FROM books WHERE title LIKE '%\\_%';

## Aggregate Functions

### COUNT()

* SELECT COUNT(\*) FROM books;
* SELECT COUNT(DISTINCT author\_fname) FROM books;
* SELECT COUNT(DISTINCT released\_year) FROM books;
* SELECT COUNT(title) FROM books
* WHERE title LIKE '%THE%';

### GROUP BY

* SELECT \* FROM books;
* SELECT author\_lname,COUNT(\*) AS books\_written FROM books
* GROUP BY author\_lname ORDER BY books\_written;

All non-aggregated columns in SELECT must be in GROUP BY

* SELECT released\_year, COUNT(book\_id) AS total\_books
* FROM books
* GROUP BY released\_year
* ORDER BY total\_books;

### MIN()/MAX()

* SELECT MIN(released\_year) FROM books;
* SELECT MAX(pages) FROM books;
* SELECT MAX(author\_lname), MIN(author\_lname) FROM books;

### SUBQUERY

---- TO find the book which has highest pages ---------

* SELECT title FROM books
* WHERE pages = 634;
* SELECT title, pages FROM books
* WHERE pages = (SELECT MAX(pages) FROM books);
* INSERT INTO books(title, pages)
* VALUES('Drag me to the Hell', 634);
* SELECT title, released\_year FROM books
* WHERE released\_year = (SELECT MIN(released\_year) FROM books);
* SELECT title FROM books
* ORDER BY pages DESC LIMIT 4;

### GROUP BY MULTIPLE COLUMNS

* SELECT \* FROM books;
* SELECT CONCAT(author\_fname, ' ', author\_lname) AS author\_fullname, COUNT(pages) FROM books GROUP BY author\_fullname;
* SELECT author\_fname, author\_lname, COUNT(pages) FROM books GROUP BY author\_lname, author\_fname;

### GROUP BY WITH MIN/MAX

-------- Find the year each author published their first book --------

* SELECT
* CONCAT(author\_fname, ' ', author\_lname) AS author,
* MIN(released\_year) AS earliest\_release,
* MAX(released\_year) AS recent\_released,
* COUNT(\*) AS books\_written
* FROM
* books
* GROUP BY author;

### SUM()

* SELECT
* released\_year, SUM(pages) AS page\_count
* FROM
* books
* GROUP BY released\_year
* ORDER BY page\_count DESC;
* SELECT
* author\_lname, SUM(pages) AS page\_count
* FROM
* books
* GROUP BY author\_lname
* ORDER BY page\_count;
* SELECT SUM(author\_lname) FROM books;

### AVG()

* SELECT AVG(stock\_quantity) AS stock, author\_lname FROM books GROUP BY released\_year ORDER BY stock;