# Part 1 - Database Concepts (Database, DBMS, Data Models, RDBMS and SQL)

* In-order to understand SQL, we have to first understand different Database Concepts like:
  + Database
  + DBMS
  + Data Models
  + RDBMS
  + SQL etc.
* Database Concepts:
  + Data: It’s a raw information that might now make a lot of sense. Eg: Rohit, UP65, 21
  + Related Data: When data makes sense, like Name: Rohit, Age: 21, Vehicle No: UP65
  + Databases
    - The place we can store the related data and later retrieve the data is known as Database
      * Storing the related Data like Employee ID, Employee Name, Employee Salary, Department etc. can be stored
      * Retrieving the data based on some conditions
  + Database Management Systems (DBMS)
    - System that stores data in databases in an organized way to make it easier to manipulate data, i.e create, retrieve, update etc.
    - Examples: DBase, FoxPro, MySQL(Free & Open-source), Oracle, MangoDB, MariaDB, SQLite, Cassandra and many more
  + Data Models
    - Defines how related data is connected to each other and stored inside the database.
    - Types of Data Models:
      * Hierarchical Model (Vehicle->Cars, Bikes, Cars->Coupe, Sedans, etc.)
      * Network Model (Nodes connected in a computer network)
      * Entity-Relationship Model (Employee is an entity and it has eID, eName, eSalaray, etc.)
      * Relational Model:
        + Data is stored in the form of tables
        + Tables organize the data in the form of rows and columns
        + It is most popular and widely used by most of the DBMS Software
  + RDBMS
    - DBMS using Relational Data Models are known as RDBMS
    - Examples for RDBMS Software:
      * Oracle
      * MS-SQL Server
      * DB2
      * MySQL
      * MS-ACCESS
      * etc.
  + SQL
    - Query language for Relational Databases
    - Stands for Structured Query Language
    - The following things can be performed on Database using SQL:
      * Inserting Data
      * Retrieving Data
      * Updating Data
      * Deleting Data
      * And many more

# Part 2 - Practice SQL without installing any Software (Using W3 Schools)

1. Kick start learning or practicing SQL without installing any RDBMS software
2. Google Search for 'W3Schools Try SQL'
3. We don't need to create any DB or tables to get started
   1. Existing tables with the required data is in place
4. Start by Restoring the Database
   1. If we lose any data by our operations and want it back, we can restore it
5. Executing a sample SQL statement
   1. Select \* from Customers;

# Part 3 - Select Command

1. The purpose this SQL command is to retrieve the data from the tables
2. Retrieving all the data from Customers table
   1. Syntax: Select \* from TableName;
   2. Select \* from Customers;
   3. Select \* means select all columns.
3. Retrieving specific column data from Customers table
   1. Select CustomerName from Customers;
   2. Select CustomerName,Country,City,PostalCode from Customers;

# Part 4 - distinct keyword

1. For unique values to be retrieved, we have to use distinct keyword with Select command
2. Without distinct:
   1. Select Country from Customers;
3. With distinct:
   1. Select distinct Country from Customers;
   2. Select distinct Country,PostalCode from Customers;
      1. Combination of Country and PostalCode should be unique, i.e Mexico 4410 is different from Mexico 3392

# Part 5 - SQL is not case sensitive

1. All the below statements will work same irrespective of their case
   * Select CustomerName,PostalCode from Customers;
   * seLect customerNaMe,pOstalcode from customeRs;
   * SELECT CUSTOMERNAME,POSTALCODE FROM CUSTOMERS;

# Part 6 - Semicolon

1. Mandatory in some RDBMS Software
2. Without Semicolon
3. Why Semicolon
   1. Separating Multiple SQL Statements

# Part 7 - Where Clause

1. Purpose is to filter records based on some condition
2. Practice Where Clause:
   1. Select \* from Customers where City='London';
   2. Select \* from Customers where CustomerID=9;
   3. Select CustomerName,Country,City where City='London';
3. We have used only one operator i.e. Equal Operator in these examples
   1. In the upcoming videos, using other operators with Where Clause will be explained.
4. We have used Where Clause with only Select statements
   1. In the upcoming videos, using Where Clause with other statements like Update, Delete etc. will be explained

# Part 8 - Using Relational Operators in Where Clause Condition

1. Used usually with select statement, and is used to filter the data based on certain conditions.
2. Different Relational Operators we can use in Where Clause Condition
   1. Equal Operator (=)
      1. Select \* from Products where Price=40;
   2. Greater Than Operator (>)
      1. Select \* from Products where Price>40;
   3. Less Than Operator (<)
      1. Select \* from Products where Price<40;
   4. Greater Than Or Equal To Operator (>=)
      1. Select \* from Products where Price>=40;
   5. Less Than Or Equal To Operator (<=)
      1. Select \* from Products where Price<=40;
   6. Not Equal To Operator (<>) ,(!= also works)
      1. Select \* from Products where Price<>40;

# Part 9 - Using Logical Operators (AND, OR, NOT)

1. Purpose: To filter the records based on Multiple Conditions
2. Different Logical Operators we can use in Where Clause Condition
   1. AND
   2. OR
   3. NOT
3. Practical Demonstrations:
   1. AND
      1. Select \* from Customers;
      2. Select \* from Customers where Country='Mexico';
      3. Select \* from Customers where Country='Mexico' AND CustomerID>3;
      4. Select \* from Customers where Country='Mexico' AND CustomerID>3  AND ContactName<>'Francisco Chang';
   2. OR
      1. Select \* from Customers;
      2. Select \* from Customers where Country='Germany';
      3. Select \* from Customers where Country='Germany' OR City='London';
      4. Select \* from Customers where Country='Germany' OR City='London' OR CustomerID>90;
   3. NOT
      1. Select \* from Customers;
      2. Select \* from Customers where Country='Germany';
      3. Select \* from Customers where NOT Country='Germany';
      4. Select \* from Customers where NOT Country='Germany' AND City='London';

# Part 10 - Between Operator

1. Purpose is to filter records based on some Range inclusive.
2. Practical Demonstration:
   1. Select \* from Products;
   2. Select \* from Products where Price Between 10 And 20; (Includes 10 and 20)
      1. Select \* from Products where Price>=10 And Price<=20;
   3. Select \* from Products where Price NOT Between 10 And 20;
      1. Select \* from Products where Price<10 OR Price>20;

# Part 11 - Order By Clause (ASC, DESC)

1. Purpose is to order the retrieved records in ascending or descending order if nothing is given out of ASC or DESC, by default ASC is performed.
2. Practical Demonstration:
   1. SELECT \* FROM Customers;
   2. SELECT \* FROM Customers Order By Country;
   3. SELECT \* FROM Customers Order By Country ASC;
   4. SELECT \* FROM Customers Order By Country DESC;
   5. SELECT \* FROM Customers Order By Country ASC,City ASC;
   6. SELECT \* FROM Customers Order By Country DESC,City DESC;
   7. SELECT \* FROM Products Order By Price;
   8. SELECT \* FROM Products Order By Price ASC;
   9. SELECT \* FROM Products Order By Price DESC;
3. If used on multiple columns like Select \* from products order by country,city asc;, then firstly country will be ordered after that in each country the city will be ordered and so on.

# Part 12 - Using Between Operator with Text

1. In one the previous video, I have explained how to use Between operator with Numbers
2. Practical Demonstration:
   1. Select \* from Customers;
   2. Select \* from Customers Order By Country;
   3. Select \* from Customers where Country Between 'Canada' And 'Finland' Order By Country;
   4. Select \* from Customers where Country NOT Between 'Canada' And 'Finland' Order By Country;
3. If between is used for strings then the strings are arranged lexicographically then between is checked.

# Part 13 - In Operator

1. Simplifies providing multiple values in Where Clause, when all the values are from the same column
2. Practical Demonstration:
   1. Select \* from Products;
   2. Select \* from Products where Price=18 OR Price=30 OR Price=10;
   3. Select \* from Products where Price in (18,30,10);
   4. Select \* from Customers;
   5. Select \* from Customers where Country='USA' OR Country='Canada' OR Country='UK';
   6. Select \* from Customers where Country in ('USA','Canada','UK');

Part 14 - Like Operator and Wildcard Characters

1. We can use Like Operator and Wildcard Characters for Pattern matching needs
2. We can use them in the Where Clause Conditions
3. When we use Wildcard, we must use Like operator
4. We don’t use = instead use ‘like’
5. Wildcards are of 4 types:

* % for 0 or more characters
* \_ for 1 character
* [ ] for given characters
* [^] for NOT given characters

1. Practical Demonstration:
   1. Select \* from Customers;
   2. Select \* from Customers Where Country Like '%a'; //last letter should be a
   3. Select \* from Customers Where Country Like 'F%'; //First letter should be F
   4. Select \* from Customers Where Country Like 'G%Y';
   5. Select \* from Customers where Country like 'Mex%';
   6. Select \* from Customers where country like '%nez%';
   7. Select \* from Customers where country like '%Can%';
   8. Select \* from Customers where country like '\_weden'; //Only 1 char then weden
   9. Select \* from Customers where country like 'U\_';
   10. Select \* from Customers where country like '\_\_A';
   11. Select \* from Customers where country like 'M\_x\_c\_';
   12. Select \* from Customers where Country like 'Fin\_a%';
   13. Select \* from Customers where Country like ‘h[oa]t’; //can be hot, hat
   14. Select \* from Customers where Country like ‘h[^oa]t’; //cannot be hot, hat. Ie hit, etc

# Part 15 - Aliases for Table Column Names (AS Keyword)

1. While the records are retrieved these alias names provided for Column names will be displayed temporary in place of Original Column names
2. Practical Demonstration:
   1. Select \* from Categories;
   2. Select CategoryID,CategoryName from Categories;
   3. Select CategoryID as ID,CategoryName as Name from Categories;
   4. As is optional - Select CategoryID ID,CategoryName Name from Categories;
   5. Select CategoryID as [Category ID],CategoryName as [Category Name] from Categories (If alias has space in between, can use quotes too)

# Part 16 - Limit Keyword

1. When a Table has 1000's of records, the Application will slow down when trying to display them.
2. Using Limit keyword we can decide how many records needs to be displayed on the page irrespective of number of records available, there by improving the performance
3. Practical Demonstration:
   1. Select \* from Customers;
   2. Select \* from Customers Limit 3;
   3. Select \* from Customers where Country='USA';
   4. Select \* from Customers where Country='USA' Limit 5;
   5. Select \* from Customers Limit 2,6; This means after displaying 2 records show next 6 records i.e total of 8

# Part 17 - Breaking the Lengthy SQL Statement into multiple lines

1. We can break the lengthy SQL statement into multiple lines for better understanding. Achieved by literally pressing the Enter key lol.
2. Practical Demonstration
   1. Select \* from Customers;
   2. Select CustomerID,CustomerName,Country,City from Customers where Country='France';

# Part 18 - MySQL Built-in Functions

1. Built-in Functions can be different in different RDBMS Software
2. There are several Built-in Functions in MySQL using which we can perform different operations on the different Table data like Text,Number,Date and Time
3. MySQL Built-in Functions can be categorized into:
   1. String Functions
      1. Upper()
      2. Lower()
      3. Length()
      4. Trim()
      5. Instr()
      6. Substr()
      7. Concat(), etc.
   2. Numeric Functions
      1. Abs()
      2. Sqrt()
      3. Mod()
      4. Power()
      5. Truncate()
      6. Greatest()
      7. Least(), etc.
   3. Date and Time Functions
      1. Current\_date()
      2. Current\_time()
      3. Now()
      4. Sysdata()
      5. Month()
      6. Year()
      7. Day(), etc.
   4. Aggregate Functions
      1. Average()
      2. Sum()
      3. Min()
      4. Max()
      5. Count(), etc.

# Part 19 - upper() MySQL String Function

1. One of the built-in functions of MySQL
2. Converts the whole text under the specified Column(s) data to Upper Case
3. Practical Demonstration
   1. Select upper('Arun Motoori');
   2. Select upper('Arun Motoori') AS FullName; (Good alias because by default it gives stupid name to the new column as “Upper(Customer)”
   3. Select \* from Customers;
   4. Select Country from Customers;
   5. Select upper(Country) from Customers;
   6. Select upper(Country) AS Country from Customers;
   7. Select upper(Country) AS Country,City from Customers;
   8. Select upper(Country) AS Country,upper(City) from Customers;
   9. Select upper(Country) AS Country,upper(City) AS City from Customers;

# Part 20 - lower() MySQL String Function

1. One of the built-in functions of MySQL
2. Converts the whole text under the specified Column(s) data to Lower Case
3. Practical Demonstration
   1. Select lower('Arun Motoori');
   2. Select lower('Arun Motoori') AS FullName;
   3. Select \* from Customers;
   4. Select Country from Customers;
   5. Select lower(Country) from Customers;
   6. Select lower(Country) AS Country from Customers;
   7. Select lower(Country) AS Country,City from Customers;
   8. Select lower(Country) AS Country,lower(City) from Customers;
   9. Select lower(Country) AS Country,lower(City) AS City from Customers;

# Part 21 - length() MySQL String Function

1. One of the built-in functions of MySQL
2. Finds the size of the data under the specified Column
3. Practical Demonstration
   1. Select 'Arun Motoori';
   2. Select length('Arun Motoori');
   3. Select length('Arun Motoori') AS GivenNameSize;
   4. Select \* from Customers;
   5. Select Country from Customers;
   6. Select length(Country) from Customers;
   7. Select length(Country) AS SIZE from Customers;
   8. Select Country, length(Country) AS SIZE from Customers;
   9. Select \* from Customers where length(Country)=6;

# Part 22 - instr() MySQL String Function

1. One of the built-in functions of MySQL
2. Finds the position of first occurrence of the given text in the data of the specified Column
3. Practical Demonstration
   1. Select 'Arun Motoori';
   2. Select instr('Arun Motoori','n');
   3. Select instr('Arun Motoori','n') AS Position; //Alias Position it’s 4
   4. Select instr('Arun Motoori','ri') AS Position;
   5. Select \* from Customers;
   6. Select Country from Customers;
   7. Select instr(Country,'e') from Customers;
   8. Select instr(Country,'e') AS Position from Customers;
   9. Select Country, instr(Country,'e') AS Position from Customers;

# Part 23 - substr() MySQL String Function

1. One of the built-in functions of MySQL
2. Retrieves a portion of text from the data of the specified column
3. Practical Demonstration
   1. Select 'Arun';
   2. Select substr('Arun',2,3);
   3. Select substr('Arun',2,3) AS Portion;
   4. Select substr('Arun',-3,2);
   5. Select substr('Kartikey',-3,-2);
   6. Select substr('Arun',-3,2) As Portion;
   7. Select \* from Customers;
   8. Select Country from Customers;
   9. Select substr(Country,2,4) from Customers;
   10. Select substr(Country,2,4) AS CountryPortion from Customers;
   11. Select Country, substr(Country,2,4) AS CountryPortion from Customers;

# Part 24 - concat() MySQL String Function

1. One of the built-in functions of MySQL
2. Adds Two or More Table Column data together
3. Practical Demonstration
   1. Select 'Arun';
   2. Select concat('Arun',' ','Motoori', ‘is’, ‘Chomu’);
   3. Select concat('Arun',' ','Motoori') AS FullName;
   4. Select \* from Employees;
   5. Select Concat(FirstName,' ',LastName) from Employees;
   6. Select Concat(FirstName,' ',LastName) AS FullName from Employees;
   7. Select FirstName,LastName,Concat(FirstName,' ',LastName) AS FullName from Employees;
   8. Select Concat('My full name is',' ',FirstName,' ',LastName) AS FullName from Employees;

# Part 25 - trim() MySQL String Function

1. One of the built-in functions of MySQL
2. Removes the leading and trailing spaces of the Column data
3. Practical Demonstration
   1. Select 'Arun';
   2. Select '   Arun   ';
   3. Select length('   Arun   '); //10
   4. Select length(trim('   Arun   ')); //4
   5. Select trim(ColumnName) from TableName;
4. Insertion command:

Insert into Customers value(92,” Kartikey “, “Davey”, “Sigra”, “Varanasi”, 221010, “India”

This will add in database but spaces will not be shown in Kartikey but still length will give output as 14, so always use length(trim(Column)) to get correct answer.

# Part 26 - abs() MySQL Numeric Function

1. One of the built-in functions of MySQL
2. Returns the positive values irrespective of the given positive or negative number data in the specified column
3. Practical Demonstration
   1. Select 9;
   2. Select -9;
   3. Select abs(9);
   4. Select abs(-9);
   5. Inserting a new record into Products table and applying abs on price.

# Part 27 - mod() MySQL Numeric Function

1. One of the built-in functions of MySQL
2. Returns the reminder value of the numeric data of the specified column
3. Practical Demonstration
   1. Select 9;
   2. Select mod(9,4); //1
   3. Select \* from OrderDetails;
   4. Select Quantity,mod(Quantity,3) from OrderDetails;

# Part 28 - greatest() and least() MySQL Numeric Functions

1. One of the built-in functions of MySQL
2. Returns the greatest and least values of the given numeric values
3. Practical Demonstration
   1. Select greatest(88,64,123,97,3,100); //123
   2. Select least(88,64,123,97,3,100); //3
   3. Select greatest('Arun','Varun','Tharun'); //Varun
   4. Select least('Arun','Varun','Tharun'); //Arun

# Part 29 - truncate() MySQL Numeric Function

1. Built-in function of MySQL
2. truncate() - Returns the numerical values with the allowed number of digits after decimal point and does NOT approximate.
3. Practical Demonstration
   1. Select truncate(123.4567,2); //123.45
   2. Select truncate(123.4567,3);
   3. Select truncate(123.4567,6); //123.456700
   4. Select truncate(123.4567,0); //123
   5. Select truncate(123.4567,-2); //100 (destroyed 2 digits)
   6. SELECT truncate(Price,5) FROM Products;

# Part 30 - power() and sqrt() MySQL Numeric Functions

1. Practical Demonstration
   1. Select power(3,4);
   2. Select sqrt(2);
2. Many other MySQL Numeric Functions

# Part 31 - current\_date(), curdate(), current\_time(), curtime(), now() and sysdate() MySQL Date Time Functions

1. Practical Demonstration
   1. Select current\_date(); //YYYY-MM-DD
   2. Select curdate(); // same as above
   3. Select current\_time(); //HH:MM:SS (GMT in W3 Server)
   4. Select curtime(); // same as above
   5. select now(); //YYYY-MM-DD HH:MM:SS
   6. select sysdate(); //same as above

# Part 32 year(),month(),day,monthname(),dayname() MySQL Date Time Functions

1. Practical Demonstration
   1. Select year('2020-04-25'); //2020 will be printed
   2. Select month('2020-04-25'); //4
   3. Select day('2020-04-25'); //25
   4. Select monthname('2020-04-25'); //April
   5. Select dayname('2020-04-25'); //Saturday
   6. SELECT \* FROM Orders where month(OrderDate)=4;
   7. SELECT \* FROM Orders where monthname(OrderDate)='May';
2. Many more Date and Time functions look on their documentation.

# Part 33 - avg(), max(), min(), count() and sum() MySQL Aggregate Functions

1. Practical Demonstration
   1. Select avg(price) from Products; //28.866363636363637 (15decimal digits)
   2. Select min(price) from Products; //2.5
   3. Select max(price) from Products; //263.5
   4. Select count(\*) from Products; //77
   5. Select count(price) from products; //same as above
   6. Select sum(price) from Products; //2222.71
2. Many more other Aggregate functions

# Part 34 - Arithmetic Operators

1. The below are the different Arithmetic Operators we can use in the SQL Statements:
   1. Addition (+)
   2. Subtraction (-)
   3. Multiplication (\*)
   4. Division (/)
   5. Modulus (%)
2. Practical Demonstration:
   1. Select 5+4; //9
   2. Select 5-4; //1
   3. Select 5\*4; //20
   4. Select 5/4; //1 (integer)
   5. Select 5%4; //1
   6. Select Price, Price+10 from Products;
   7. Select Price, Price - 10 from Products;
   8. Select Price, Price \* 10 from Products;
   9. Select Price, Price / 10 from Products;
   10. Select Price as OriginalPrice, Price % 10 as ModifiedPrice from Products;

# Part 35 - Installing MySQL Server and Workbench Client for Practicing SQL

1. Database Components
   1. Client - Runs the SQL queries and communicates with the Server
      1. Generally installed in the Local machines
   2. Server - Stores the Data into DB
      1. Generally installed in RemoteServer Machine (NOT in same machine as Client)
2. So far, we have practiced SQL statements on the DB hosted by W3Schools
3. Now, let’s install our own DB in our machine for practicing SQL from Scratch
4. There are several RDBMS Software available in the market and MySQL is an open source and available for free of cost while Oracle isn’t.
5. Installing MySQL RDBMS Software, will install both Client and Server
6. The below are the steps for installing MySQL Server and Workbench Client Software:
   1. Installing MySQL on Windows
      1. Google Search for 'download mysql on windows'
      2. Click on this link '<https://dev.mysql.com/downloads/mysql/>" from the search results
      3. Click on 'Go to Downloader Page'
      4. Select to download the 'mysql-installer-community-Version.msi'
      5. Login or Register to Download
      6. Click on 'Download Now' button
      7. Click on 'Next' when the 'Developer Default' option is selected
      8. Don't select anything in the 'Check Requirements' and click on 'Next'
      9. Click on 'Yes' on the displayed message
      10. Click on 'Execute' Button
      11. Click on 'Next' until you reach 'Accounts and Roles' screen
      12. Give root and root as username and password
      13. Click on 'Add User' button
      14. Give admin and admin as username and password
      15. Click 'ok' and click on 'Next' button
      16. Click on 'Next' button until you see 'Execute' button
      17. Click on 'Execute' button
      18. Click on 'Finish' button
      19. Click on 'Next' and 'Finish' buttons until it asks for password
      20. Provide password as 'root' and click on 'Check' button
      21. Click on 'Next' and 'Execute' button
      22. Click on 'Finish'
      23. Click on 'Next' and 'Finish' and observe that the MySQL Client GUI and command line will open
      24. Close them
   2. Using GUI version of Client - MySQL Workbench
      1. Search for Workbench in windows and open the Software
      2. Select Database > Manage Connections
      3. Select Local Instance and change the name to your desired name say 'MySQL\_Server', test and close
      4. Select Database > Connect to Database
      5. Click on 'Ok'
      6. Click on 'Schemas' tab in the displayed window and observe that it will show you all the databases already created
      7. We can expand these Databases and see what is inside
      8. In order to see the data inside any Table, right click on the table and select 'Select Rows - Limit 1000'
7. Select \* from sakila.actor;

Select first\_name, last\_name from sakila.actor;

# Part 36 - Creating, Deleting, viewing and using Databases

1. Practical Demonstration
   1. Show Databases; //DB aka Schema
   2. Create Database TestDB;
   3. Refresh to see the Database Reflected
   4. Observe that the DB will be created without any below objects:
      1. Tables
      2. Views
      3. Stored Procedures
      4. Functions.
   5. Drop Database TestDB;
   6. Refresh to see the Database deleted
   7. Create Database QAFox
   8. Select \* from actor; //this will not work as there are in multiple DBs
   9. Select \* from sakila.actor;
   10. Use sakila; //to avoid always writing sakila.actor
   11. Select \* from actor; //now this will work after doing use sakila;

# Part 37 - Creating, Viewing, Describing and Deleting Tables

1. Table is one of the objects of a Database like (Views, Procedures, Functions and so on…)
2. Practical Demonstration
   1. use world;
   2. show tables;
   3. use QAFox;
   4. Create table Employees(id int, name varchar(15),experience int);
   5. Describe Employees;
   6. Select \* from employees;
   7. Create table Emp as Select id, name from Employees; //another table emp created having only 2 column id and name from Employees table
   8. drop table Emp;

# Part 38 - Insert Into Statements (For inserting data into Tables)

1. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. Select \* from employees;
   4. Describe employees; //describes the each column and its data type and other info
   5. insert into employees values(1,'Arun',12);
   6. Select \* from employees;
   7. insert into employees values(2,'Varun',5);
   8. Select \* from employees;
   9. insert into employees values(3,'Tharun',7);
   10. Select \* from employees;
   11. insert into employees values(4,'Alice');
       1. Will get an error
   12. insert into employees(id,name) values(4,'Alice');
       1. Null value will be inserted in place on non-inserted column

# Part 39 - Data Types

1. Every column in a Table has a Name and a data type
2. While creating a table, we have to decide the name and type for each and every column
3. Type of the Column decides the data that is expected in the specified table
4. Different Data Types
   1. MySQL allows us to use the below data types ([Click here](https://www.w3schools.com/sql/sql_datatypes.asp))
      1. String Data Types
         1. Varchar(Size)
      2. Numeric Data Types
         1. int
         2. double
      3. Data and Time Data Types
         1. DATE - YYYY-MM-DD
         2. Time - hh:mm:ss
         3. DateTime - YYYY-MM-DD HH:MM:SS
         4. YEAR - YYYY
5. Practical Demonstration:
   1. use qafox;
   2. create table xyz(a int);
   3. Select \* from xyz;
   4. insert into xyz values(9);
   5. Select \* from xyz;
   6. insert into xyz values('Arun');
   7. Select \* from xyz;
   8. insert into xyz values(1.5);
   9. Select \* from xyz;
   10. drop table xyz;
   11. create table xyz(a double);
   12. insert into xyz(a double);
   13. Select \* from xyz;
   14. insert into xyz values(9);
   15. Select \* from xyz;
   16. insert into xyz values(9.5);
   17. drop table xyz;
   18. create table xyz(a varchar(15));
   19. Select \* from xyz;
   20. insert into xyz values('Arun');
   21. Select \* from xyz;
   22. drop table xyz;
   23. create table xyz(a date);
   24. Select \* from xyz;
   25. insert into xyz values('1992-12-03');
   26. Select \* from xyz;
   27. drop table xyz;
   28. create table xyz(a time);
   29. Select \* from xyz;
   30. insert into xyz values('11:12:13');
   31. Select \* from xyz;
   32. drop table xyz;
   33. create table xyz(a datetime);
   34. Select \* from xyz;
   35. insert into xyz values('1992-12-03 11:12:13');
   36. drop table xyz;
   37. create table xyz(a year);
   38. Select \* from xyz;
   39. insert into xyz values('1992');
   40. drop table xyz;

# Part 40 - Null Value, Is Null Operator and Is Not Null Operator

1. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. select \* from employees;
   4. insert into Employees values(5,'Kiran');
      1. Error
   5. insert into Employees(id,name) values(5,'Kiran');
   6. select \* from employees;
   7. Select \* from employees where experience IS NULL;
      1. Select \* from employees where experience = null WILL NOT WORK
   8. Select \* from employees where experience IS NOT NULL;

# Part 41 - Delete Statement (For Deleting the Records from Table)

1. Practical Demonstration
   1. use qafox;
   2. select \* from employees;
   3. delete from employees where id=5;
   4. delete from employees where name='Tharun';
   5. delete from employees; //Everything gets deleted

Unlike drop database DBname or drop table tableName for records we literally use the word delete

# Part 42 - Update Statement and Set Keyword (For Updating the Table Records)

1. Practical Demonstration
   1. use qafox;
   2. select \* from employees;
   3. insert into employees values(1,'Arun',12);
   4. insert into employees values(2,'Varun',5);
   5. insert into employees values(3,'Tharun',7);
   6. update employees set name='Kiran' where id=3;
   7. update employees set name='Dinesh',id=4 where experience=5;
   8. update employees set name=’Akhilesh’; //all table’s name column will be Akhilesh

# Part 43 - Rename statement and To Keyword (For Renaming Table Name)

1. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. rename table employees to emp;
   4. rename table emp to employees;

# Part 44 - Alter Statement, Add, Modify Column, Rename Column and Drop Column Keywords

1. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. describe employees;
   4. alter table employees add location varchar(15);
   5. describe employees;
   6. alter table employees modify column location varchar(20);
   7. describe employees;
   8. alter table employees rename column location to loc;
   9. describe employees;
   10. alter table employees drop column loc;
   11. describe employees;

# Part 45 - Set Autocommit

1. By default it is 1
2. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. Select \* from employees;
   4. insert into employees values(9,'Dinesh',3);
   5. Select \* from employees;
   6. Restart the workbench client and observe that the inserted data is stored
      1. use qafox;
      2. Select \* from employees;
   7. set autocommit=0;
   8. insert into employees values(11,'Isha',6);
   9. Select \* from employees;
   10. Restart the workbench client and observe that the inserted data is not stored i.e temporary.
       1. use qafox;
       2. Select \* from employees;
   11. set autocommit=1;
   12. insert into employees values(11,'Isha',6);
   13. Select \* from employees;
   14. Restart the workbench client and observe that the inserted data is stored
       1. use qafox;
       2. Select \* from employees;

# Part 46 - Commit Statement

1. When autocommit=0 so whatever we do is temporary so to save our work we have to explicitly tell to commit the work using commit statement.
2. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. set autocommit=0;
   4. Select \* from employees;
   5. insert into employees values(9,'Dinesh',3);
   6. Select \* from employees;
   7. Restart the workbench client and observe that the inserted data is not permanently stored
      1. use qafox;
      2. Select \* from employees;
   8. insert into employees values(9,'Dinesh',3);
   9. commit;
   10. Select \* from employees;
   11. Restart the workbench client and observe that the inserted data is stored permanently
       1. use qafox;
       2. Select \* from employees;
   12. set autocommit=1;

# Part 47 - Rollback Statement

1. Purpose is to revert back the changes done temporarily on the table. i.e opposite of commit
2. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. set autocommit=0;
   4. Select \* from employees;
   5. delete from employees; //deletes all content of employees table and NOT the table itself
   6. Select \* from employees;
   7. rollback;
   8. delete from employees;
   9. commit;
   10. rollback; //can’t rollback now because damage was commited
   11. set autocommit=1;

# Part 48 - Truncate Statement

1. It is used to permanently delete the records from table irrespective of autocommit=1 or 0.
2. Delete from employees; and truncate table employees; are exactly same but delete depends on autocommit while truncate permanently deletes the record.
3. Practical Demonstration
   1. use qafox;
   2. show tables;
   3. Select \* from employees;
   4. insert into employees values(1,'Arun',12);
   5. commit;
   6. set autocommit=0;
   7. delete from employees;
   8. rollback;
   9. Select \* from employees;
   10. truncate table employees;
   11. rollback; //useless now
   12. Select \* from employees;
   13. set autocommit=1;

# Part 49 - Single Line and Multi-Line Comments

1. Comments are the Statements which won't be executed like general SQL statements
2. Can be used for any of the below reasons:
   1. To explain the underlying statements to make it understandable
   2. To hide the statement from execution
3. Types of Comments in SQL
   1. Single Line Comments
   2. Multi-Line Comments
4. Practical Demonstration
   1. -- The below SQL statement is used for selecting the Database for further operations
   2. use qafox;
   3. /\* The below SQL statements are used for finding the list of tables in the above selected
   4. database and to retrieve the records from the specified table \*/
   5. show tables;
   6. select \* from employees;

# Part 50 - Group By Clause

1. Purpose is two group the retrieve records according to the specified column.
2. Practical Demonstration
   1. use world;
   2. show tables;
   3. select \* from country;
   4. Select name from country;
   5. select count(Name) from country; //normal retrival
   6. select count(name) from country group by Continent; //grouping them
   7. select continent,count(name) from country group by Continent;

# Part 51 - Having Clause

1. For normal conditions we can use ‘where’ clause but having clause helps when the records are grouped.
2. Having Clause is like a Where Clause for Group By Clause
3. In case of providing condition for Group By Clause, we have to use Having Clause
4. Practical Demonstration
   1. use world;
   2. show tables;
   3. select \* from Country;
   4. select name from country;
   5. select count(name) from country;
   6. select count(name) from country where surfacearea>100;
   7. select continent,count(name) from country group by continent;
   8. select continent,count(name) from country group by continent having count(name)<10;

# Part 52 - Sequence of using where, group by, having and order by clauses

1. where > group by > having > order by //i.e when having multiple clauses first use where then group by then having and so on…
2. Practical Demonstration
   1. use world;
   2. select continent,count(name) from country where surfacearea>300 group by continent having count(name)>20 order by count(name) ASC;

# Part 53 - Set Operators

1. Select \* from Table1; select \* from Table2; will work but retrieval is done 1 by 1.
2. We can retrieve the data from different tables at the same time using the Set Operators
   1. Select \* from TableOne **Set Operator** Select \* from TableTwo;
3. The below are the different Set Operators
   1. Union (common is not counted twice)
   2. Union All (common is duplicated)
   3. Intersect (only common is retrieved)
   4. Minus (Values in Table1 but not in Table2)
4. Consider Table1 (1,3,5) and Table2 (2,3,4)
   1. Union (1,2,3,4,5)
   2. Union All (1,2,3,3,4,5)
   3. Intersect (3)
   4. Minus (1,5)

# Part 54 - Union Operator

1. Union Operator in one of the Operators in the four Set Operators
2. Purpose of Union Operator is to retrieve records from multiple tables.
3. Practical Demonstration
   1. use qafox;
   2. create table empone(id int);
   3. insert into empone values(1);
   4. insert into empone values(3);
   5. insert into empone values(5);
   6. select \* from empone;
   7. create table emptwo(num int);
   8. insert into emptwo values(2);
   9. insert into emptwo values(3);
   10. insert into emptwo values(4);
   11. select \* from emptwo;
   12. select \* from empone union select \* from emptwo;
   13. Demonstrate with the two column table
       1. Combination of column records will be considered for comparison and elimination of duplicates 3 Rohit is different from 3 Richard, but say we have 2 columns are we’re only concerned with ID so 3 will not be repeated.
   14. Rules:
       1. For select \* statement to work each table must have the same number of columns
       2. Columns provided in the select statements should have the similar data type to work properly
       3. The order of the columns provided in the select statements should be same to work properly

# Part 55 - Union All Operator

1. Union All Operator is one of the Set Operators
2. Unlike Union Operator, Union All won't eliminate the duplicates
3. Practical Demonstration
   1. use qafox;
   2. create table tone(id int);
   3. insert into tone values(1);
   4. insert into tone values(3);
   5. insert into tone values(5);
   6. select \* from tone;
   7. create table ttwo(num int);
   8. insert into ttwo(2);
   9. insert into ttwo(3);
   10. insert into ttwo(4);
   11. select \* from ttwo;
   12. select id from tone union all select num from ttwo;

# Part 56 - Intersect Operator

1. Intersect Operator is one of the Set Operators
2. Intersect Operator will retrieve the common records from the given tables
3. Intersect Operator is not supported by MySQL RDBMS
   1. We must use Oracle SQL RDMBS
   2. Google search for 'Try Oracle SQL Online Practice'
4. Practical Demonstration
   1. use qafox;
   2. create table tone(id int);
   3. insert into tone values(1);
   4. insert into tone values(3);
   5. insert into tone values(5);
   6. select \* from tone;
   7. create table ttwo(num int);
   8. insert into ttwo(2);
   9. insert into ttwo(3);
   10. insert into ttwo(4);
   11. select \* from ttwo;
   12. select id from tone intersect select num from ttwo;

# Part 57 - Minus Operator

1. Minus Operator is one of the Set Operators
2. Minus Operator will retrieve the records in the first table and that are not available in the second table
3. Minus Operator is not supported by MySQL RDBMS
   1. We have to use Oracle SQL RDMBS
   2. Google search for 'Try Oracle SQL Online Practice'
4. Practical Demonstration
   1. use qafox;
   2. create table tone(id int);
   3. insert into tone values(1);
   4. insert into tone values(3);
   5. insert into tone values(5);
   6. select \* from tone;
   7. create table ttwo(num int);
   8. insert into ttwo(2);
   9. insert into ttwo(3);
   10. insert into ttwo(4);
   11. select \* from ttwo;
   12. select id from tone minus select num from ttwo;
   13. select id from two minus select num from tone;

# Part 58 - Tables and Aliases

1. Previously we used aliases for table columns but here we’ll use for table itself.
2. Practical Demonstration (o and t are aliases for tables)
   1. Select id from emplistone, emplisttwo; //Both have ‘id’ as a column but it’ll still give an error because it’s ambiguous which id we’re talking about.
   2. Select o.id,o.firstname,t.lastname

from empdetailsone o,empdetailstwo t

where o.id=t.id; //select where the first id is = second’s id

# Part 59 - Joins (Inner Join, Left Join, Right Join, Full Join and Self Join)

1. The purpose of the Joins is to Join two tables while retrieving the records using the common column, advantage is that we can retrive records from 2 table instead of 1 tradationally.
2. The two tables must have a common column
3. Types of Joins:
   1. Inner Join (Equi Join or Simple Join)
   2. Left Join (Left Outer Join)
   3. Right Join (Right Outer Join)
   4. Full Join (Full Outer Join) - Not supported by MySQL
   5. Self Join
4. Practical Demonstrations
   1. Consider 2 Tabels empdetail1 and empdetail 2:

Diagram

Description automatically generated

* 1. Inner Join: Select \* from empdetail1 inner join empdetail2 on empdetail1.id=empdetail2.id, the common stuff between them will be retrived i.e 1 Arun 1 Motoori and 2 Varan 2 Kumar.
  2. **NOTE: In case of joins we don’t use ‘where’ instead we use ‘on’.**
  3. Left Join: Select \* from empdetail1 left join empdetail2 on empdetail1.id=empdetail2.id, the common stuff will be retrived but importance given to left(1st) table i.e 1 Arun 1 Motoori, 2 Varun null null, 4 Tharun 4 Kumar.
  4. Right Join: Select \* from empdetail1 right join empdetail2 on empdetail1.id=empdetail2.id, similar to above, importance is given to right table (2nd ) i.e 1 Motoori 1 Arun, 4 Kumar 4 Tharun, 5 Rathod null null.
  5. **NOTE: In left/right join the left/right table will come fully but in respect to the common stuff between them the other table’s content will come.**
  6. Full Join: It’s not supported in MySQL but supported in Oracle RDBMS(Online works unlike MySQL) Full join means retriving all complete records. i.e 1 Arun 1 Motoori, 2 Varun null null 4 Tharun 4 Kumar 5 Rathod null null (more like null null 5 Rathod).
  7. Self Join: It’s different from rest of the joins as all previous requires 2 tables, but here we operate on same table two times. Since we’re using same table twice we’ve to use alias name. eg: Select \* from empdetails o, empdetails t where o.id=t.deptid;

Text, table

Description automatically generated

We’ll get output: 1 Arun Motoori 1 1 Arun Motoori 1, 3 Tharun Rathod 3 3 Tharun Rathod 3.

# Part 60 - Sub Query (Explaining Single Row Sub Query by solving different SQL problems)

1. Sub Query is a Query inside another Query aka Inner Query or Nested Query and output of inner query will be input for outer query.
2. Understanding Sub Query:
   1. Outer Query
   2. Inner Query (Sub Query)
3. Based on the number of records retrieved by the Inner Query, we can categorize Sub Queries into:
   1. Single Row Sub Query (Subquery gives single record)
   2. Multi Row Sub Query (Subquery gives multiple records i.e many rows)
4. Practical Demonstrations:
   1. Single Row Sub Query
      1. Find all the Customers in the Customers table who are from the same city of 'Hari Kumar' (Output is London in w3school editor, and which is a single row)
         1. First we have to find the city of Hari Kumar in the Customer Table
            1. Select City from Customers where ContactName='Hari Kumar';
         2. We have to make the above query a sub-query (Inner Query) in the where clause of Outer Query
            1. Select \* from Customers where City=(Select City from Customers where ContactName='Hari Kumar');
      2. Finding the second maximum price of the Products
         1. Select max(Price) from Products where Price < (Select max(price) from Products);
      3. Finding the third maximum price of the Products
         1. Select max(Price) from Products where Price < (Select max(Price) from Products where Price < (Select max(price) from Products));
      4. Display the products sold at the least price
         1. Select ProductName from Products where price = (Select min(price) from Products);
   2. Multi Row Sub Query
      1. I will practically demonstrate the Multi-Row Sub Query in the upcoming sessions
         1. We have to use in,any,all and exits operators

# Part 61 - In Operator

1. Equals to Anything in the List
2. Allows us to provide multiple values in the Where Clause Condition
3. Practical Demonstration
   1. Select \* from Customers where Country='USA';
   2. Select \* from Customers where Country In ('USA','UK','Italy','France','Spain');
   3. SELECT \* FROM Customers where Country Not In ('USA','UK','Italy','France','Spain');
4. In Operator is mainly used with Multi Row Sub Queries

# Part 62 - Using In Operator with Multi Row Sub Query

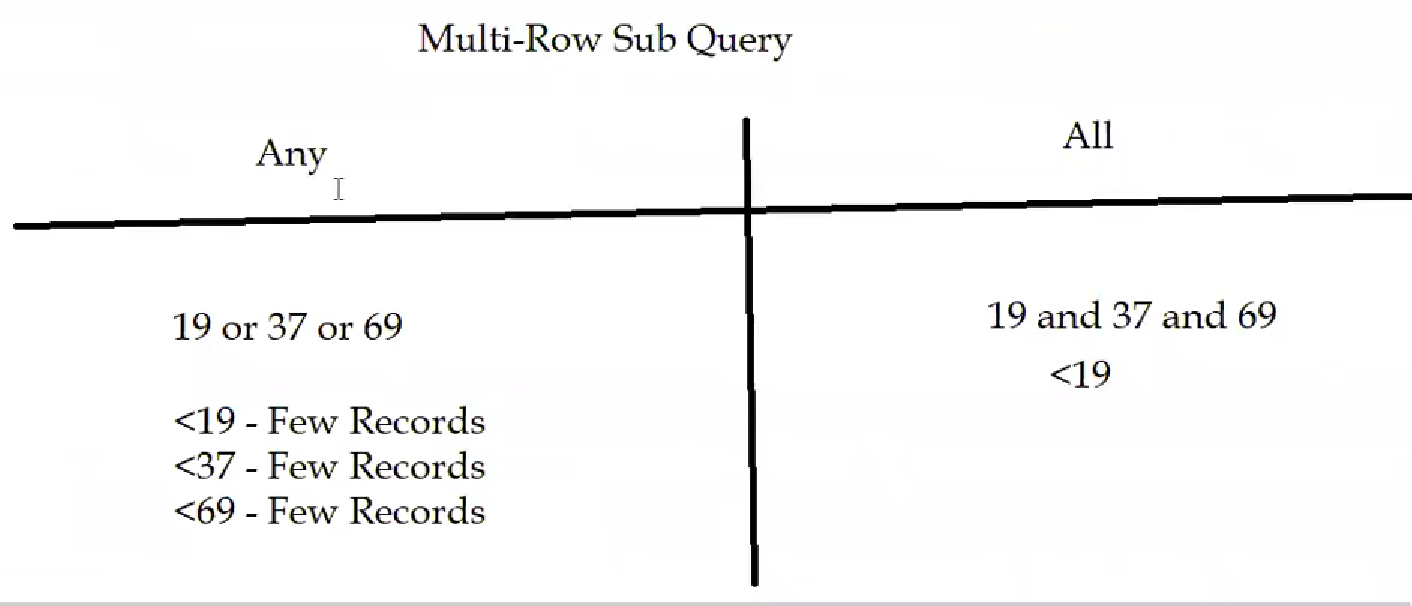
1. Equals to Anything in the List
2. Multi Row Sub Query is a Query inside another Query and it provides multi records as input to the outer query
3. Different Operators which can be used in Multi Row Sub Query
   1. In (we’re here)
   2. any
   3. all
   4. exists
4. Practical Demonstration
   1. Find the different Products from the specified Categories
      1. Select \* from Products where CategoryID **in** (Select CategoryID from Categories where CategoryName like 'c%');

# Part 63 - Using Any Operator in Multi Row Sub Query

1. Compares Value to Each Value Returned by the Sub Query, sometimes also known as SOME.
2. Multi Row Sub Query is a Query inside another Query and it provides multi records as input to the outer query
3. Different Operators which can be used in Multi Row Sub Query
   1. in
   2. any (we’re here)
   3. all
   4. exists
4. Practical Demonstration
   1. In operator probhits using >,=,< symbols beause it mean equal itself but with any we can use any of them.
   2. Select \* from Products where ProductID < any (SELECT ProductID FROM OrderDetails where Quantity=1);

# Part 64 - Using All Operator in Multi Row Sub Query

1. Compares Value to Every Value Returned by the Sub Query.
2. Practical Demonstration
   1. Select \* from Products where ProductID < all (SELECT ProductID FROM OrderDetails where Quantity=1);



# Part 65 - Exists Operator

1. It works as true or false if the subquery returns one or more rows then the outer query will execute if there exists no records of subquery the outer query will not execute.
2. Exists Operator is used with Sub Queries
3. Practical Demonstration
   1. Select \* from Orders where exists (Select \* from Customers where CustomerID>91); // In W3 school there are 91 records so this query will not execute

# Part 66 - Using Sub Queries for retrieving the records from multiple tables

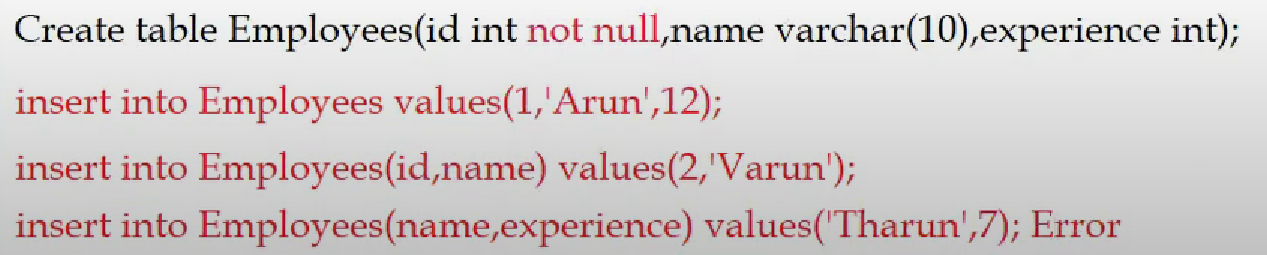
1. We can use various types of joins for retrieving records from multiple tables but we can also use sub queries as columns(must be single row)
2. Practical Demonstration
   1. Select city,(Select country from country where country.country\_id=city.country\_id) country from City;
      1. Will get the records
   2. Select country,(Select city from city where country.country\_id=city.country\_id) city from Country;
      1. Will get an error
      2. Sub Query resulting in multiple records, will give an error
   3. This is because a city has only 1 country but one country can have multiple cities. That’s why 2nd demonstration failed.

# Part 67 - Using Multiple Sub Queries in a single SQL statement

1. Practical Demonstration
   1. Problem: List out the films having the length less than the maximum length and having the rental duration equal to the minimum rental duration (Inside MySQL’s sakila’s film table)
      1. Finding the maximum length of the films
         1. select max(length) from film;
      2. Finding the minimum rental duration of the films
         1. select min(rental\_duration) from film;
      3. Now list out the films having the length less than the maximum length and having the rental duration equal to the minimum rental duration
         1. select title,length,rental\_duration from film
         2. where length < (select max(length) from film)
         3. and rental\_duration = (select min(rental\_duration) from film);

# Part 68 - Integrity Constraints

1. They’re conditions we can apply on the Table Column Data
2. We can apply this Integrity Constraints to the Table Columns, while using Create and Alter SQL statements
3. Types of Integrity Constraints
   1. Not Null (Entity Integrity Constraint)
   2. Unique (Key Constraint)
   3. Primary Key
   4. Foreign Key (Refrential Entity Integrity Constraint)
   5. Check
   6. Default
4. Example: create table employees(ID int **not null**, name varchar(10), experience int);



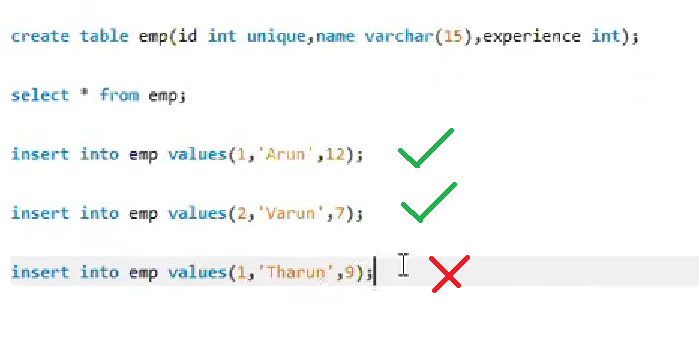
1. Generally ID is primary key and we don’t want that to be null (Entity Integrity Constraint)

# Part 69 - Not Null (Integrity Constraint)

1. Not Null is an Integrity Constraint
2. Not Null when specified to a column will not allow the column to allow null values
3. Practical Demonstration
   1. create table emp(id int not null,name varchar(15),experience int);

# Part 70 - Unique (Integrity Constraint)

1. Unique is an Integrity Constraint
2. Unique when specified to a column will not allow the column to allow duplicate values
3. Practical Demonstration
   1. create table emp(id int unique,name varchar(15),experience int);
4. Unique values can be used at Table and Column levels
   1. create table emp(id int unique,name varchar(15),experience int);
   2. create table emp(id int,name varchar(15),experience int,unique(id,name));

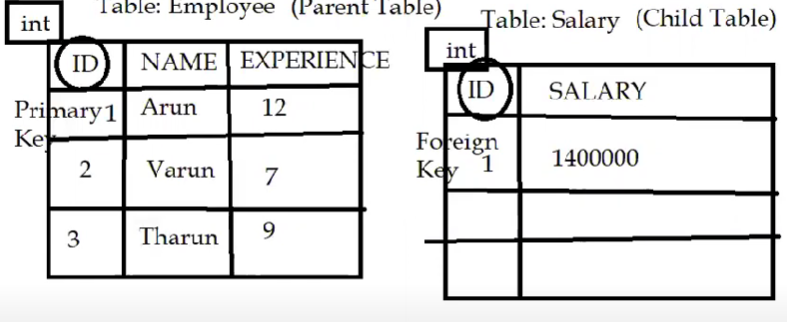


# Part 71 - Primary Key (Integrity Constraint)

1. Primary Key is an Integrity Constraint
2. **Primary Key = Not Null + Unique;**
3. Practical Demonstration
   1. create table emp(id int not null unique,name varchar(15),experience int);
   2. create table emp(id int primary key, name varchar(15),experience int);
   3. create table emp(id int, name varchar(15),experience int, primary key(id));
   4. create table emp(id int, name varchar(15),experience int, primary key(id,name));
4. Primary Key at Column Level and Table Level

# Part 72 - Foreign Key (Integrity Constraint)

1. Foreign Key is one of the Integrity Constraints
2. We need to create two tables having a common column having same data type. Name can be different i.e both don’t have to be of same name say ID.
   1. Parent Table (Reference Table)
      1. **Should have a Primary Key specified for Common Column**
   2. Child Table (Depends on parent table)
      1. Should have the Foreign Key Specified for Common Column
3. Child Table Depends on the Parent Table following the rules given:
   1. Inserting the records in Child table should match with the Parent Table common column
   2. Deleting the records in Parent Table is not possible if there are dependent records in the Child Table
4. On Delete Cascade (automatic deletion of depended record if parent record is deleted)
5. Practical Demonstration
   1. create table employee(id int primary key,name varchar(15),experience int);
   2. **create table salary(id int,sal int,foreign key(id) references employee(id));**
   3. create table salary(id int,sal int,foreign key(id) references employee(id) on delete cascade);



1. Interesting cases:
   1. Say we want to add ID 7 and Salary 150000 in Salary table, it is not possible to do so.
   2. Say we want to delete 1 Arun 12 from Employee, it is not possible because 1 1400000 in Salary table depends on it, so to do that we first have to delete the dependent data from salary table first then the intended data in Employee.

# Part 73 - Check (Integrity Constraint)

1. Check is an Integrity Constraint
2. Practical Demonstration
   1. create table empone(id int,name varchar(15),experience int check(experience >5));
      1. Here if we do insert into empone values(1,’Mohit’,3); it will not work
   2. create table emptwo(id int,location varchar(15) check(location in ('India','USA','UK')));

# Part 74 - Default (Integrity Constraint)

1. Default is an Integrity Constraint
2. It’s applied on table columns and if no value is inserted into that column the provided default value is inserted instead of null.
3. Practical Demonstration
   1. create table empone(id int, experience int default 5);
      1. insert into empone(id) values(2); will set ID 2 and experience 5 by default.
   2. create table emptwo(id int,location varchar(15) default 'India');
   3. create table empthree(id int,DateOfJoining date default '2007-08-15');

# Part 75 - auto\_increment

1. auto\_increment will increment the values of the specified table columns by one, if we are not inserting any data
   1. Very useful when we have something like Serial No. where if we start with 1, we usually put second serial no. as 2,3,4 and so on…
2. Practical Demonstration
   1. create table empone(id int primary key auto\_increment,name varchar(15));
      1. insert into empone values(5,'Arun');
      2. insert into empone(name) values('Varun'); //Instead of null 6 is inserted
      3. insert into empone values(9,'Tharun'); //explicit 9
      4. insert into empone(name) values('Dinesh'); //10 Dinesh
   2. create table emptwo(id int primary key auto\_increment,name varchar(15));
      1. insert into empone(name) values('Arun'); //1 Arun inserted
      2. insert into empone(name) values('Varun');
   3. create table emptwo(id int primary key auto\_increment,name varchar(15)) auto\_increment=100;
      1. insert into empone(name) values('Arun'); //100 Arun
      2. insert into empone(name) values('Varun'); //101 Varun
3. **CAN ONLY BE APPLIED ON PRIMARY KEY**

# Part 76 - Insert Into

1. Insert Into Statement is used for copying the records from one table into another table having same column names and data types
2. Practical Demonstration
   1. Use describe city to get data type details of city table and use same data type for newcity.
   2. Create table newcity(nid int, name char(35),country code char(3), district char(20), population int);
   3. insert into newcity Select \* from city; (copying all records from city table to newcity)
   4. insert into newcity2(id,name) select id,name from city; //copying id and name only from city to newcity2. The non filled places will have null value.
   5. We can also use where clause

# Part 77 - AS Keyword

1. AS Keyword is used to create a new table with the records of an existing table, i.e copy of table
   1. Similar job as insert into that we discussed previously
2. Practical Demonstration
   1. Create Table NewPlace AS Select \* from Place;
   2. Create Table PlaceTwo AS Select name,district from Place;

# Part 78 - IfNull() function

1. IfNull() function is used to provide an alternate value when the column value is null
2. Practical Demonstration
   1. create table empone(id int,salary int);
   2. Select (ifnull(salary,0)+100000) newsalary from empone;

# Part 79 - Case, When,Then and End Keywords

1. Case,When,Then and End Keywords can be used in Select statements
2. Practical Demonstration
   1. Example One:
      1. Select ProductName,Price,

Case

When Price > 10 Then 'The Price is greater than 10'

When Price = 10 Then 'The Price is equal to 10'

When Price < 10 Then 'The Price is less than 10'

End As PriceDetails, //Alias

Unit

from Products;

IT has 4 columns ProductName, Price, Text(looks bad use alias ) and Unit

* 1. Example Two:
     1. SELECT \* FROM Customers
     2. where City =
     3. (Case
        1. when Country In ('USA','UK') Then City
        2. When Country Not In ('USA','UK') Then 'Berlin'
     4. End);
  2. Example Three
     1. Select \* from Customers
     2. Order By
     3. (Case
        1. When Country In ('USA','UK') Then Country
        2. When Country Not In ('USA','UK') Then City
     4. End);

# Part 80 - Delimiter

1. Default Delimiter is ;
2. We can change the delimiter to other symbol
3. Practical Demonstration
   1. Select \* from city;
   2. Delimiter //
   3. Select \* from city//
4. Different symbols can be used instead of //
5. Delimiters are generally used with Stored Procedures

# Part 81 - Delimiter usage in Stored Procedures

1. Store Procedures belong to PL/SQL they’re similar to that of functions.
2. PL/SQL is a out of topic for this video course and is not required for Testers too.
3. But I am going to explain about Stored Procedures with respect to Delimiter usage
4. Stored Procedures are like functions in programming languages
5. We create stored procedures for executing the repetitive block of SQL statements
6. Practical Demonstration

Delimiter // (Without delimiter the below code will not run)

create procedure getAllCities()

Begin

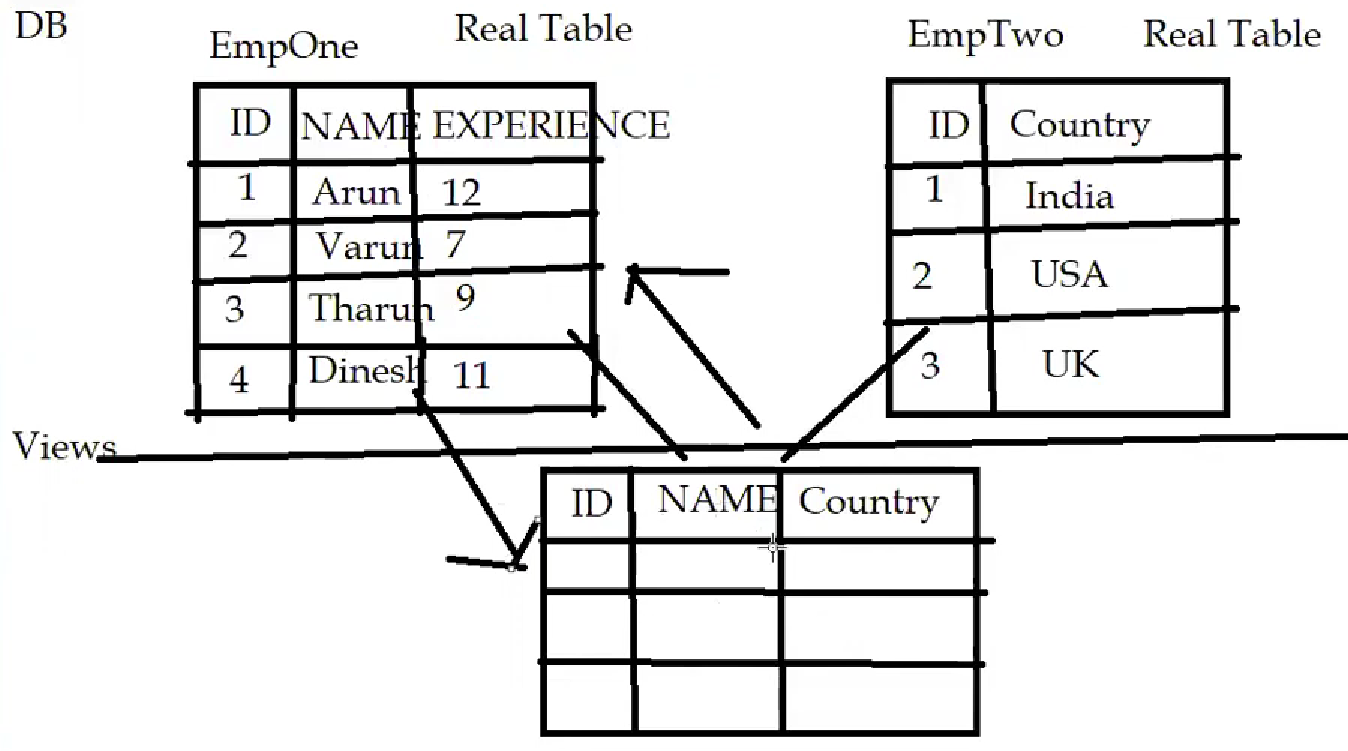
Select \* from City; (Can also use select \* from city// )

End //

1. Under Stored Procedures getAllCities will be created.
2. To call that procedure we can use **call getAllCities();** in SQL to run the whole code.
3. **PL/SQL (Programming Language in SQL) is a way in which we can write programs that executes SQL, like any other programming languages it has variables, constants, functions, statements, control flow, constructors, etc.**
   1. BEGIN
   2. dbms\_output.put\_line (‘Hello World');
   3. END;
   4. /

# Part 82 - Views

1. Problem earlier was that whenever we want to make changes in original table no matter how small that change is we have to create a new table. This is overcome by Views concept.
2. The main purpose of views is to create different varieties of the same table or tables (Virtual Tables)
3. We can create Views by customizing the same table columns or by selecting the columns from multiple tables
4. Hide the Database implementation of the actual tables and show the desired virtual views to the Users
5. Practical Demonstration
   1. create table empone(id int,name varchar(15),country varchar(15));
   2. insert into empone values (1,'Arun','India');
   3. insert into empone values (2,'Varun','UK');
   4. insert into empone values (3,'Tharun','Spain');
   5. insert into empone values (4,'Dinesh','New Zealand);
   6. select \* from empone;
   7. create table emptwo(id int,experience int);
   8. insert into emptwo values(1,12);
   9. insert into emptwo values(2,7);
   10. select \* from emptwo;
   11. create view emponeviewa as Select \* from empone;
   12. create view emponeviewb as Select id,name from empone;
   13. create view emponeviewc as Select id,country from empone;
   14. create view emponeviewd as Select country,name,id from empone;
   15. create view empviewb as Select empone.id,empone.name,emptwo.experience from empone,emptwo where empone.id=emptwo.id;



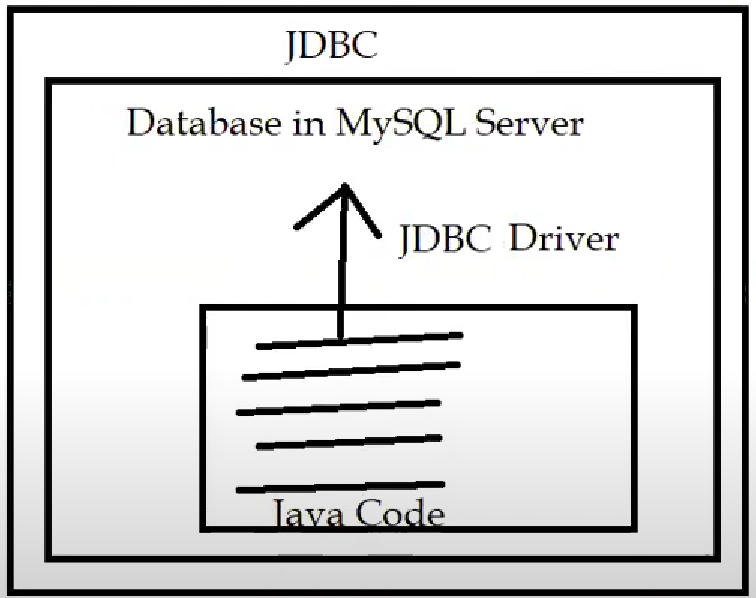
1. **Performing operators on the Views will effect the original tables and vice versa**

# Part 83 - Indexes

1. Indexes when implemented for a table, will increase the performance of the Application by retrieving the records at faster speed
2. Indexes are like Table of Contents or Index in a Book, which are provided for faster access of the required topics
3. Indexes should be implemented for retrieval operations
   1. Insertions and Updates will slow down if you create indexes so only use it when needed.
   2. Select operations need indexes for speeder retrieval
4. Practical Demonstration
   1. use qafox;
   2. create table empone(id int,name varchar(15),experience int,country varchar(15));
   3. insert into empone values(1,'Arun',12,'India');
   4. insert into empone values(2,'Varun',7,'USA');
   5. insert into empone values(3,'Tharun',9,'UK');
   6. Select \* from empone;
   7. Select \* from empone where country='USA';
   8. show indexes from empone;
   9. create index empcountry
   10. on empone(country);
   11. show indexes from empone;
   12. drop index empcountry on empone;
   13. create table emptwo(id int primary key,name varchar(15));
       1. primary index will be automatically created if there exist a primary key
       2. show indexes from emptwo;
   14. create table empthree(id int unique,name varchar(15));
       1. id index will be automatically created
       2. show indexes from empthree;

# Part 84 - JDBC (Conneting to MySQL Database from Java Programs)

1. Basic Java knowledge is required for understanding this session
2. If you don't know Java, then try to understand the concept behind JDBC
3. So far we have used Client Tools like MySQL Workbench for connecting and performing operations on the Databases in MySQL Server
4. We can also connect to Server and perform Operations on the DB in the Server from Java Programs
5. JDBC stands for Java Database Base Connectivity
6. How to Connect to MySQL Database from Java Programs?
   1. Install the below requirements:
      1. Java JDK
      2. Eclipse IDE for Java Enterprise Developer



1. Follow the below steps
   1. Use MySQL Workbench to connect to MySQL qafox DB and create a table

use qafox;

create table empone(id int,name varchar(15));

insert into empone values(1,'Arun');

insert into empone values(2,'Varun');

insert into empone values(3,'Tharun');

Select \* from empone;

* 1. Create a Maven Project in Eclipse IDE (maven-archetype-quickstart
  2. Configure the Project with MySQL Java Connectivity Driver
     1. Every DBMS software like MySQL, Oracle has their own drivers
  3. Create a new Demo Class
  4. Create a Connection
     1. Check how do you make a connection to MySQL Server using any Client say MySQL Workbench

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/qafox","root","root");

if(!connection.isClosed()) {

System.out.println("Connection is successfull");

}

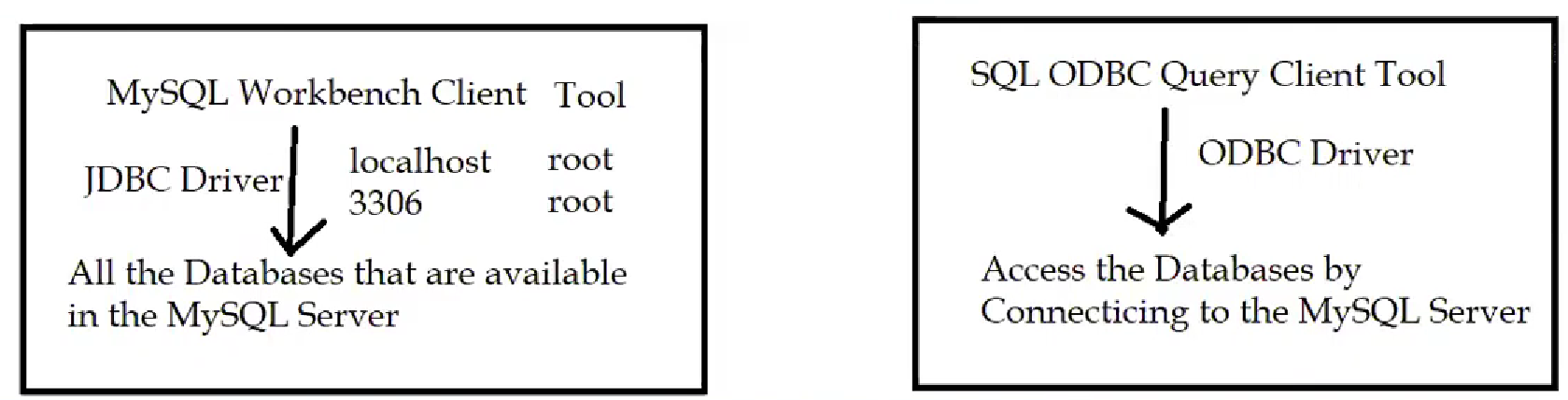
* 1. Create and Execute the Statement from Java

Statement statement = connection.createStatement();

statement.execute("insert into empone values(4,'Dinesh')");

* + 1. Execute the program and check in Workbench to see if the statement executed changes are reflected
    2. Also, repeat the above for
       1. statement.execute("delete from empone where id=2");
    3. Reading all the records from the table using executeQuery()
       1. ResultSet resultset = statement.executeQuery("Select id,name from empone");
    4. while(resultset.next()) {
       1. int eid = resultset.getInt("id");
       2. String ename = resultset.getString("name");
       3. System.out.println(eid+" "+ename);
    5. }

# Part 85 - ODBC

1. MySQL Workbench Client internally uses JDBC driver (Not only used by Java Code) for connecting to the Databases in MySQL Server (generally runs on localhost:3306)
2. SQL ODBC (Object DataBase Connectivity) Query Client Tool requires ODBC driver for connecting to the Database in MySQL Server
3. 
4. Downloading and Installing MySQL ODBC Driver
   1. Windows Search ODBC and select 64bit (Already installed when installing MySQL)
5. Creating DataSource Name (System DSN)
   1. Select ANSI Driver and Finish
   2. Give some name to it
   3. TCP/IP will be localhost and port 3306
   4. User: root and password: root
   5. Select Database from list (world, sakila, etc.)
   6. Click test to find if connection was successful or not.
6. Downloading SQL ODBC Query Tool using ODBC
   1. QTODBC shortcut will be created on desktop now use it as your new workbench
7. JDBC and ODBC are the popular drivers for connecting to the MySQL DB

# Part 86 - MySQL Shell Command Line Tool

1. MySQL Workbench Client Tool is GUI application so if we want to use command line we can install Shell Command.
2. MySQL Shell Command Line Tool for connecting to the MySQL Server
3. Window Search for 'MySQL Shell’
4. We can Google MySQL commands to learn more
   1. \sql
   2. \connect root@localhost:3306
   3. Show databases;
   4. Use world;
   5. Show tables;
   6. Select \* from city;
   7. Blah blah…

# Part 87 - MySQL Command Line Client

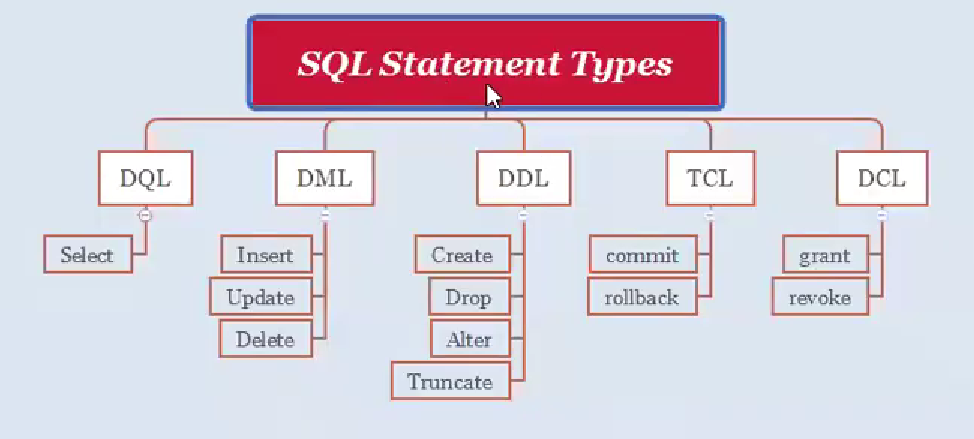
1. It is exactly same as shell
2. It comes by default with MySQL DB Software
3. Windows Search > MySQL and launch 'MySQL Command line client'
4. Give password as root
5. Perform required operations

# Part 88 - Using default Windows Command Prompt for connecting to MySQL Server

1. Go to C:\Program Files\MySQL\MySQL Server 8.0\bin
2. Type cmd
3. Execute the below commands:
   1. mysql -u root -p
   2. Give the password
   3. Then perform the operations
   4. Then use it like shell

# Part 89 - Types of SQL Statements

1. DQL (Data Query Language)
2. DML (Data Manipulation Language)
3. DDL (Data Definition Language)
4. TCL (Transation Control Language)
5. DCL (Data Control Language)



# Part 90 - Grant and Revoke SQL Statements

1. They fall under DCL category type
2. Create User in MySQL Server (MySQL Workbench>Server>User and Privilages>Add Account), we’ve to add password here too.
3. Grant and Revoke Permissions
   1. use qafox;
   2. grant select on empone to 'userone';
   3. revoke select on empone from 'userone';
   4. grant create,delete,drop,select,insert,update,alter on empone to 'userone','usertwo';
   5. revoke select,delete,drop,create,alter,insert,update on empone from 'userone','usertwo';
4. We can login from userone by: Workbench>Database>Connect to Database>Username:userone>Ok then it’ll ask for password and then this user can only access the granted stuff.

# Part 91 - Temporary Tables

1. Useful when we have to create a table for particular purpose and after achieving that purpose we no longer need it and hence we delete it.
2. They only exits as long as the session exists.
3. Normally: create table joey(id int, name varchar(15));
4. Temporary: create temporary table joe(id int, name varchar(18));
5. Close the workbench, reopen it it’ll be gone.

# Part 92 - Show Columns,Show Indexes,Show Privileges and Show Grants Statements

1. Show Tables and Databases
   * Show databases;
   * Use world;

Show tables;

1. Show Columns and Show Indexes
   * show columns from tablename;
   * show indexes from tablename;
   * show privileges;
   * show grants for 'userone';
   * show grants for 'usertwo';

# Part 93 - Inserting Null

1. Ways of inserting null:

Use world;

Create table emp(id int,name varchar(15));

* 1. Insert into emp values(1,null);
  2. Insert into emp(name) values(‘Rohit’);

# Part 94 - Using trim() for trimming the corner characters of the specified Table values

1. Practical Demonstration
   1. use qafox;
   2. Select trim('a' from 'arun'); //run
   3. Select trim('a' from 'rana'); //ran
   4. Select trim('a' from 'aruna'); //run
   5. create table empone(name varchar(15));
   6. insert into empone values('arun');
   7. insert into empone values('naresh');
   8. insert into empone values('varun');
   9. insert into empone values('tharun');
   10. insert into emptwo values('nandan');
   11. Select \* from emptwo;
   12. Select trim('n' from name) from emptwo; //aru, aresh, varu, tharu, anda
       1. This is although a temprorary change and not commited to table

# Part 95 - Using wild-cards as normal characters

1. Problem can arise when table has wild-card characters.
2. Use sakila;
3. Create table emp(id int, name varchar(15));
4. Insert into emp values(1,’mohan’);
5. Insert into emp values(1,’Gajju\_’);
6. Insert into emp values(1,’Davey’);
7. Insert into emp values(1,’Shailendra\_’);
8. What will happen if we use
   1. Select \* from emp where name like ‘%\_’; will it give only Gajju\_ and Shailendra\_?
   2. No, it will give all records coz (%){ \_ } means (anything or nothing){singleCharacter} and all records satify this condition.
   3. So, to retrive Gajju\_ and Shailendra\_ use backslash ( \ ) to convert wildcard into character.
      1. Select \* from emp where name like ‘%\\_’;
9. Escape keyword helps us to change the wild-card converter ( \ by default)
   1. Select \* from emp where name like ‘%$\_’ escape ‘$’;
      1. This will work exactly the same as ‘%\\_’;

# Part 96 - Database Objects

1. The things that we can create under any database using the create statements in SQL are nothing but the database objects.
   * Table
   * Views
   * Indexes
   * Stored Procedures (requires delimiter)