Date...../...../..... Page. No: .....

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
- Database Models
- RDBMS
- SQL etc.

Database concepts :

→ Database :

- 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 System :

- shortly called as DBMS
- Software that stores data in databases in an organized way to make it easier to create, retrieve, update etc.
- Examples : DBase, FoxPro, MySQL, Oracle, MangoDB, MariaDB, SQLite, Cassandra. and many more.

→ Data Models

- Defines how related data is connected to each other and stored inside the Database.

Date...../...../..... Page. No: .....

- Types of Data Models :
  - → Hierarchical Model
  - → Network Model
  - → Entity - Relationship Model
  - → Relational Model
    - → Data is stored in the form of tables
    - → Tables organize the data in the form of rows and columns
    - → It is a popular and widely used by most of the DBMS software.

→ RDBMS :

- DBMS using Relational Data models are known as RDBMS.
- Examples of RDBMS software:
  1. Oracle
  2. MS-SQL server
  3. DB2
  4. MySQL
  5. MS-ACCESS
  6. 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

Date...../...../..... Page. No: .....

Practice SQL without installing any software

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. Existing tables with the required data is in place.
4. Start by Restoring the Database. If we lose any data by our operations and want it back, we can restore it.
5. Executing a sample SQL statement

```
Select * from Customers;
```

Select Command

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

distinct Keyword

1. For unique values to be retrived, we have to use distinct keyword with select command.
2. without distinct:
   → `select Country from Customers;`

3. With distinct :
   1. select distinct country from Customers;
   2. Select distinct Country, PostalCode from customers;
      → combination of country and PostalCode should be unique.

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;

Semicolon

1. Mandatory in some RDBMS software
2. without semicolon
3. Why semicolon → separating Multiples SQL statements.

Where clause

1. Purpose is to filter records based on some conditions
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. Further, using other operators with where clause will be explained.
4. We have used where clause with only select statements. Further, using where clause with other statements like update, Delete etc. will be explained.

Date...../...../..... Page. No: .....

Using Relational operators in Where clause condition

1. Different Relational operators we can use in Where clause condition

→ Equal operator (=)

```
select * from Products where Price = 40;
```

→ Greater Than Operator (>)

```
Select * from Products where Price > 40;
```

→ Less Than Operator (<)

```
Select * from Products where Price < 40;
```

→ Greater Than or Equal to Operator (>=)

```
Select * from Products where Price >= 40;
```

→ Less Than or Equal to Operator (<=)

```
Select * from Products where Price <= 40;
```

→ Not Equal To operator (<>)

```
Select * from Products where Price <> 40;
```

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
   - AND
   - OR
   - NOT
3. Practical Demonstration:

1. AND

→ `select * from customers;`

→ `select * from Customers where country = 'Mexico';`

→ `Select * from Customers where country = 'Mexico' AND customerID > 3;`

Date...../...../..... Page. No: .....

→ Select * from Customers where Country = 'Mexico' AND ContactName <> 'Francisco Chang' AND CustomerID > 3;

2. OR

→ Select * from Customers;
→ Select * from Customers where country = 'Germany';
→ Select * from Customers where Country = 'Germany' OR City = 'London';
→ Select * from Customers where Country = 'Germany' OR City = 'London' OR CustomerID > 90;

3. NOT

→ Select * from Customers;
→ Select * from Customers where Country = 'Germany';
→ Select * from Customers where NOT country = 'Germany';
→ select * from Customers where NOT Country = 'Germany' AND City = 'London'.

Between Operator

1. Purpose is to filter records based on some range.
2. Practical Demonstration:
→ Select * from Products;
→ Select * from Products where Price Between 10 And 20;

OR

Select * from Products where Price >= 10 And Price <= 20;

→ Select * from Products where Price NOT Between 10 And 20;

OR

Select * from Products where Price < 10 OR Price > 20;

Date...../...../..... Page. No: .....

Order By Clause

1. Purpose is to order the retrieved records in ascending or descending order.
2. Practical Demonstration
   - → SELECT * FROM customers;
   - → SELECT * FROM customers Order By country;
   - → SELECT * FROM customers Order By country ASC;
   - → SELECT * FROM Customers Order By country DESC;
   - → SELECT * FROM Customers Order By country ASC, City ASC;
   - → SELECT * FROM Customers Order By country DESC, City DESC;
   - → SELECT * FROM Products order By Price;
   - → SELECT * FROM Products Order By Price ASC;
   - → SELECT * FROM Products Order By Price DESC;

Between operator with Text

1. Previously, we have discussed about how to use Between operator with Numbers.
2. Practical Demonstration:
   - → SELECT * from Customers;
   - → SELECT * from customers order By country;
   - → SELECT * from Customers where country Between 'Canada' And 'Finland' Order By country;
   - → SELECT * from Customers where country NOT Between 'Canada' And 'Finland' Order By country.

Date...../...../..... Page. No: .....

In Operator

1. Simplifies providing multiple values in where clause, when all the values are from the same column.
2. Practical Demonstration :
   - → SELECT * from Products;
   - → select * from Products where Price = 18 OR Price = 30 OR Price = 10;
   - → Select * from Products where Price in (18, 30, 10);
   - → select * from Customers;
   - → Select * from Customers where Country = 'USA' OR Country = 'Canada' OR country = 'UK';
   - → Select * from Customers where Country in ('USA', Canada, 'UK');

Like operator and wildcard characters

1. We can use like operator and wildcard character for pattern matching needs.
2. We can use them in the where clause conditions
3. Practical Demonstration :
   - → select * from Customers;
   - → select * from Customers where Country like '%a';
   - → select * from Customers where Country like '%F';
   - → Select * from Customers where Country like 'G%Y';
   - → select * from Customers where Country like 'Mex%';
   - → select * from Customers where Country like '%nez%';
   - → Select * from Customers where Country like '%Can%';
   - → select * from Customers where country like '-weden';
   - → Select * from Customers where Country like 'U_';
   - → Select * from Customers where Country like '__A';
   - → Select * from Customers where Country like 'M_x_C_';
   - → Select * from Customers where Country like 'Fin_a%';

% → Multi char.
_ → Single char.

Date...../...../..... Page. No: .....

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 columns names.
2. Practical Demonstration:
   - → Select * from Categories;
   - → Select CategoryID, CategoryName from Categories;
   - → Select CategoryID as ID, CategoryName as Name from Categories;
   - → As is optional - Select CategoryID ID, CategoryName Name from Categories;
   - → Select CategoryID as [Category ID], CategoryName as [Category Name] from Categories;
     (has space; has space)

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, thereby improving the performance.
3. Practical Demonstration:
   - → Select * from customers;
   - → select * from Customers Limit 3;
   - → Select * from Customers where Country = 'USA';
   - → Select * from Customers where Country = 'USA' Limit 5;
   - → select * from customers Limit 2, 6;
     (After 2nd record, display 6 records.)

Date: ...../...../..... Page No: .....

Breaking the lengthy SQL statement into multiple lines

1. We can break the lengthy SQL statement into multiple lines for better understanding.
2. Practical demonstration:
   - → Select * from Customers;
   - → Select CustomerID, CustomerName, Country, City from Customers where Country = 'France';
   - →
```
Select
CustomerID, CustomerName, Country, City
from Customers
where Country = 'France';
```

MySQL Built in Functions

1. Built-in functions and RDBMS software
2. There are several Built in function 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 Function:
- upper()
- lower()
- length()
- trim()
- instr()
- substr()
- concat()
- Many More

2. Numeric function:
- abs()
- sqrt()
- mod()
- power()
- truncate()
- greatest()
- least()
- Many More

Date...../...../..... Page. No: .....

3. Date and Time function
   - current_date()
   - current_time()
   - now()
   - sysdate()
   - month()
   - year()
   - day()
   - Many More

4. Aggregate function
   - avg()
   - sum()
   - min()
   - max()
   - count()
   - Many More

upper() MySQL String Function

1. One of the built-in function of MySQL
2. Converts the text under the specified column data to Upper case.
3. Practical Demonstration:

```
→ Select upper ('Arun Motoori');
→ Select upper ('Arun Motoori') AS FullName;
→ select * from Customers;
→ Select Country from Customers;
→ Select upper (country) from Customers;
→ select upper (country) AS country from customers;
→ Select upper (country) AS Country, City from Customers;
→ Select upper (country) AS Country, upper (City) from Customers;
→ Select upper (country) AS Country, upper (city) AS City from
  customers;
```

lower() MySQL String function

1. One of the built-in functions of MySQL
2. Converts the text under the specified column data to lower case.
3. Practical Demonstration:

```
→ select lower ('Arun Motoori');
→ select lower ('Arun Motoori') As FullName;
```

→ select * from customers;
→ Select country from Customers;
→ select lower(country) from customers;
→ select lower (country) As country from customers;
→ select lower(country) AS country, City from Customers;
→ Select lower (country) AS country, lower (City) from customers;
→ Select lower (country) AS country, lower (City) AS city from Customers;

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:
→ Select 'Arun Motoori';
→ Select length ('Arun Motoori');
→ Select length ('Arun Motoori') AS GivenNameSize;
→ select * from customers;
→ Select country from Customers;
→ Select length(country) from customers;
→ select length (country) AS SIZE from Customers;
→ select country, length (country) AS SIZE from Customers;
→ Select * from Customers where lenght (country) = 6;

instr() MySQL string Function

1. One of the built-in functions of MySQL.
2. Finds the position of the given text in the data of the specified column.
3. Practical Demonstration:
→ select 'Arun Motoori';
→ Select instr('Arun Motoori', 'n');

Date...../...../..... Page. No: .....

→ Select instr ('Arun Motoori', 'n') AS Position ;
→ Select instr ('Arun Motoori', 'ri') AS Position;
→ select * from customers;
→ select country from customers;
→ Select instr (country, 'e') from customers;
→ Select instr (country, 'e') AS Position from customers;
→ Select country, instr (country, 'e') AS Position from Customers;

substr() MySQL string function

1. One of the built-in functions of MySQL.
2. Retrieves a position of text from the data of the specified column.
3. Practical Demonstration :
→ Select 'Arun';
→ Select substr ('Arun', 2, 3); (Index (start from 1); No. of characters to be retrived)
→ Select substr ('Arun', 2, 3) AS Portion;
→ Select substr ('Arun', -3, 2);
→ Select substr ('Arun', -3, 2) AS Portion;
→ Select * from customers;
→ select country from Customers;
→ select substr (country, 2, 4) from customers;
→ Select substr (country, 2, 4) AS countryPortion from Customers;
→ Select country, substr (country, 2, 4) AS countryPortion from Customers;

concat() MySQL String Function

1. One of the built-in function of MySQL
2. Adds Two or More Table Column data together.
3. Practical Demonstration
→ Select 'Arun';
→ Select concat ('Arun', ' ', 'Motoori');

→ Select concat('Arun',' ', Motoori') AS FullName;
→ Select * from Employees;
→ Select Concat(FirstName,' ',LastName) from Employees;
→ Select Concat(FirstName,' ', LastName) AS FullName from Employees;
→ Select FirstName, LastName, Concat(FirstName,' ', LastName) AS FullName from Employees;
→ Select Concat('My full name is',' ',FirstName,' ',LastName) AS FullName from Employees;

trim() MySQL String Function

1. One of the built-in function of MySQL.
2. Removes the Leading and trailing spaces of the Column data
3. Practical Demonstration:
   → Select 'Arun';
   → Select ' Arun ';
   → Select length(' Arun ');
   → Select length(trim(' Arun '));
   → Select trim(ColumnName) from TableName;
4. Other MySQL String Category functions.

abs() MySQL Numeric function

1. One of the built-in function of MySQL.
2. Return the positive values irrespective of the given positive or negative number data in the specified column.
3. Practical Demonstration:
   → Select 9;
   → Select -9;
   → Select abs(9);
   → Select abs(-9);
   → Inserting a new record into Products table and applying abs on Price.

Date...../...../..... Page. No: .....

mod() MySQL Numeric function

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

greatest() and least() MySQL Numeric Function

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

truncate() MySQL Numeric function

1. Built-in function of MySQL
2. truncate() - Return the numerical values with the allowed number of digits after decimal point.
3. Practical Demonstration:
   - → Select truncate (123.4567, 2); → 123.45
   - → Select truncate (123.4567, 3); → 123.456
   - → Select truncate (123.4567, 8); → 123.45670000
   - → Select truncate (123.4567, 0); → 123
   - → Select truncate (123.4567, -2); → 100
   - → Select truncate (Price, 5) FROM Products;

Date...../...../..... Page. No: .....

power() and sqrt() MySQL Numeric Functions

1. Practical Demonstration:
   - → Select power(3,4);
   - → Select sqrt(2);
2. Many other MySQL Number Functions;

current_date(), curdate(), current_time(), curtime(), now(), and sysdate() MySQL Date&Time function

- → current_date()-Return the current date in a string format - '2020-04-25'.
- → curdate() - Return the current date in a string format - '2020-04-25'.
- → current_time() - Return the current time - HH:MM:SS.
- → curtime() - Return the current time - HH:MM:SS
- → now() - Return both current date and time.
- → sysdate() - Return both current date and time.

Practical Demonstration:

- → Select current_date();
- → Select curdate();
- → Select current_time();
- → Select curtime();
- → Select now();
- → Select Sysdate();

Date...../...../..... Page. No: .....

year(), month(), day(), monthname(), dayname() MySQL Date and Time functions

1. Practical Demonstration:
   - → Select year('2020-04-25'); → Return 2020
   - → select month('2020-04-25'); → Return 4
   - → Select day('2020-04-25'); → Return 25
   - → Select monthname('2020-04-25'); → Return April
   - → Select dayname('2020-04-25'); → Return Saturday
   - → Select * From orders where month(OrderDate) = 4;
   - → Select * from orders where monthname(OrderDate) = 'May';
2. Many more Date and Time functions.

avg(), max(), min(), count() and sum() MySQL Aggregate functions.

1. Practical Demonstration
   - → Select avg(Price) from Products;
   - → Select min(Price) from Products;
   - → select max(Price) from Products;
   - → Select count(*) from Products;
   - → select sum(price) from Products;

2. Many more Aggregate functions;

Date...../...../..... Page. No: .....

Arithmetic Operator

1. The below are the different Arithmetic operators we can use in the SQL statements:
   - Addition (+)
   - Substraction (-)
   - Multiplication (*)
   - Division (/)
   - Modulus (%)
2. Practical Demonstration:
   - Select 5+4;
   - Select 5-4;
   - Select 5*4;
   - Select 5/4;
   - Select 5%4;
   - Select Price, Price +10 from Products;
   - Select Price, Price - 10 from Products;
   - Select Price, Price *10 from Products;
   - Select Price, Price/10 from Products;
   - Select Price, Price %10 from Products;

Installing MySQL Server and Workbench Client for Practising SQL

1. Database components
   - Client - Runs the SQL queries and communicates with the server. Generally installed in the Local machines.
   - Server - Stores the Data into DB. Generally installed in Remote Server Machines.
2. So far we have practiced SQL statements on the DB hosted by W3schools.
3. Now, lets install our own DB in our machine for practising SQL from Scratch.

Date...../...../..... Page. No: .....

4. There are several RDBMS software available in the market and MySQL is an open source and available for free of cost.
5. Installing MySQL RDBMS software, will install both Client and Server.

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/"
   3. Click on 'Go to Download 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 'Custom' option is selected.
   8. Move MySQL server and MySQL Workbench (From Available products to Products to be installed)
   9. Click on 'Next' until you reach 'Accounts and Roles' screen.
   10. Give root and root as password and repeat password.
   11. Click on 'Add user' button
   12. Give admin and admin as username and password.
   13. Click 'OK' and Click on 'Next' button.
   14. Click on 'Next' button until you see 'Execute' button.
   15. Click on 'Execute' button.
   16. click on 'Finish' button
   17. Click on 'Next' and 'Finish' buttons until it asks for password.
   18. Click on 'Next' and 'finish' and observe that the MySQL Workbench will open.

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'.

Creating, Deleting, viewing and using Databases

1. Practical Demonstration

- Show Databases; → viewing Database
- Create Databases TestDB; → creating the Database
- Refresh to see the Database Reflected
- Observe that the DB will be created without any below objects:
  1. Tables
  2. Views
  3. Stored Procedures
  4. etc.
- Drop Database TestDB; → Deleting the Database
- Refresh to see the Database deleted.
- Create database QAFox;
- Select * from actor;
- select * from sakila.actor;
- Use Sakila;
- Select * from actor;

Date...../...../..... Page. No: .....

Creating, viewing, Describing and Deleting Tables

1. Table is an object of a Database.
2. Practical Demonstration
   - → use world; → use a particular database
   - → show tables; → viewing table in a particular Database
   - → use QAFox;
   - → Create table Employees (id int, name Varchar(15), experience int);
   - → Describe Employees; → display metadata about columns, indexes & data partition of tables or views
   - → Select * from employees; ↳ creating a table

Imp:
   - → Create table Emp as Select id, name from Employees;
   - → drop table Emp;
     ↳ deleting the tables

Insert into statements (For inserting data into Tables)

1. Practical Demonstration
   - → use qafox;
   - → show tables
   - → Select * from employees
   - → Describe employees;
   - → insert into employees values (1, 'Arun', 12);
   - → Select * from employees;
   - → insert into employees values (2, 'Varun', 5);
   - → Select * from employees;
   - → insert into employees values (3, 'Tharun', 7);
   - → Select * from employees;
   - → insert into employees values (4, 'Alice'); → will get an error
   - → insert into employees (id, name) values (4, 'Alice');
     ↳ Null value will be inserted in place on non-inserted column.

Date...../...../..... Page. No: .....

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 excepted in the specified table.
4. Different Data types:

MySQL allows us to use the below data types

1. String Data Types:
   - → varchar(size)
2. Numeric Data Types:
   - → int
   - → double
3. Data and Time Data types:
   - → DATE - YYYY-MM-DD
   - → TIME - hh:mm:ss
   - → DateTime - YYYY-MM-DD HH:MM:SS
   - → YEAR - YYYY

5. Practical Demonstration:

→ use qafox;
→ Create table xyz (a int);
→ Select * from xyz;
→ insert into xyz value (9);
→ Select * from xyz;
→ insert into xyz value ('Arun'); → Error
→ Select * from xyz;
→ insert into xyz value (1.5); → Insert 2 (rounded up - nearest integer value)
→ Select * from xyz;
→ drop table xyz;

Date...../...../..... Page. No: .....

```
create table xyz (a double);
select * from xyz;
insert into xyz values (9); → accepted
select * from xyz;
insert into xyz value (9.5);
select * from xyz;
drop table xyz;
create table xyz (a varchar(15));
select * from xyz;
insert into xyz values ('Arun');
select * from xyz;
drop table xyz;
create table xyz (a date);
select * from xyz;
insert into xyz values ('1992-12-03');
select * from xyz;
drop table xyz;
create table xyz (a time);
select * from xyz;
insert into xyz values ('11:12:13');
select * from xyz;
drop table xyz;
create table xyz (a datetime);
select * from xyz;
insert into xyz value ('1992-12-03 11:12:13');
drop table xyz;
create table xyz (a year);
select * from xyz;
insert into xyz values ('1992');
drop table xyz;
```

Date: ...../...../..... Page. No: .....

Null value, IS Null operator and Is Not Null operator

1. Practical Demonstration

→ use qafox;
→ show tables;
→ select * from employees;
→ insert into Employees values (5, 'Kiran'); → Error (It has 3 column)
→ insert into Employees(id, name) values (5, 'kiran');
→ select * from employees; ↳ 3rd column will be null
→ select * from employees where experience IS NULL;
→ select * from employees where experience IS NOT NULL;

Delete Statement (For Deleting the Records from Table)

↳ It is used to delete existing records in a table

1. Practical Demonstration:

→ use qafox
→ Select * from employees;
→ delete from employees where id = 5; → deleting specific record
→ delete from employees where name = 'Tharun';
→ delete from employees;
↳ all records from that particular table will be deleted and table will be empty

Update statement and Set Keyword (For updating the table Records)

1. Practical Demonstration ↳ It is used to modify the existing records in a table

→ use qafox;
→ select * from employees;
→ insert into employees values (1, 'Arun', 12);
→ insert into employees values (2, 'Varun', 5);
→ insert into employees values (3, 'Tharun', 7);
→ Update employees set name = 'Kiran' where id = 3;
→ update employees set name = 'Tharun', id = 4 where experience = 5;

Date...../...../..... Page. No: .....

→ update employees set name = 'Akhilesh';

↳ If we don't specify where class condition then all the records will be updated with the new details.

Rename Statement along with To Keyword (For Renaming Table Name)

1. For renaming the table name.
2. Practical Demonstration:

→ use qafox;
→ show tables;
→ rename table employees to emp;
→ rename table emp to employees;

Alter Statement - Add, Modify column, Rename column and Drop Column

1. Alter table Statement is used to add, delete, or modify columns in an existing table.
2. Practical Demonstration:

→ use qafox;
→ show tables;
→ describe employees;
→ alter table employees add location varchar(15); ↳ To add a column in a table.
→ describe employees;
→ alter table employees modify column location varchar(20); ↳ To change the data type of column in a table.
→ describe employees;
→ alter table employees rename column location to loc; ↳ To rename a column in a table.
→ describe employees;
→ alter table employees drop column loc;
→ describe employees; ↳ To delete a column in a table.

Date...../...../..... Page. No: .....

Set Autocommit

1. The auto-commit statement is used to avoid the explicit mention of the commit command after each SQL statement.

set autocommit = 0

- commit won't happen in a automatic way
- The changes you make to the table will become temporary

2. Practical Demonstration :

- use qafox;
- show tables;
- Select * from employees;
- insert into employees values (9, 'Dinesh', 3);
- Select * from employees;
- Restart the workbench client and observe that the inserted data is stored.
    1. use qafox;
    2. Select * from employees;
- set autocommit = 0;
- insert into employees value (11, 'Isha', 6);
- Select * from employees;
- Restart the workbench client and observe that the inserted data is not stored.
    1. use qafox;
    2. Select * from employees;
- set autocommit = 1;
- insert into employees values (11, 'Isha', 6);
- Select * from employees;
- Restart the workbench client and observe that the inserted data is sorted
    1. use qafox;
    2. Select * from employees

Date...../...../..... Page. No: .....

Commit Statement

1. Commit in SQL is a transaction control language that is used to permanently save the changes done in the transaction in tables / databases.
2. Practical Demonstration:

→ use qafox;
→ show tables;
→ set autocommit = 0;
→ select * from employees;
→ insert into employees values (9, 'Dinesh', 3);
→ select * from employees;
→ Restart the workbench client and observe that the inserted data is not permanently stored.

1. use qafox;
2. Select * from employees;

→ insert into employees values (9, 'Dinesh', 3);
→ commit;
→ select * from employees;
→ Restart the workbench client and observe that the inserted data is stored permanently.

1. use qafox;
2. select * from employees;

→ set autocommit = 1;

Rollback Statement

1. Revert the temporary changes done on a particular table.
2. Rollback in SQL is a transactional control language that is used to permanently save the changes done in the transaction in tables/ databases.

Date...../...../..... Page. No: .....

3. Perform Demonstration:

→ use qafox;
→ show tables;
→ set autocommit = 0;
→ select * from employees;
→ delete from employees;
→ select * from employees;
→ rollback;
→ delete from employees;
→ commit;
→ rollback;
→ set autocommit = 1;

Truncate Statement

set autocommit = 0;
↳ inserting
  deleting
} Temporary change

All the records will be deleted
↓
delete from employees;
↓
Delete the records in the table (Temporary change)

All the records will be deleted
↓
truncate table employees;
↓
Permanently delete the records from the table.

1. Practical Demonstration

→ use qafox;
→ show tables;
→ select * from employees;
→ insert into employees values (1, 'Arun' 12);

→ commit;
→ set autocommit = 0;
→ delete from employees;
→ rollback;
→ Select * from employees;
→ truncate table employees;
→ rollback;
→ select * from employees;
→ set autocommit = 1;

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:
   → To explain the underlying statements to make it understandable.
   → To hide the statement from execution.
3. Types of comments in SQL
   1. Single Line comments
   2. Multi Line comments
4. Practical Demonstration:
   → -- The below SQL statement is used for selecting the Database for further operation.
   → use qafox;
   → /* The below SQL statements are used for finding the list of tables in the above selected database and to retrieve the records from the specified table */
5. show tables;
6. select * from employees;

Date...../...../..... Page. No: .....

Group by clause

1. To group the retrieved records according to the specified column.
2. Practical Demonstration:

```
→ use world;
→ show tables;
→ select * from country;
→ Select name from country;
→ select count(name) from country;
→ Select count(name) from country group by continent;
→ select continent, count(name) from country group by
  continent;
```

Having clause

1. Having clause is like a where clause for Group By clause.
2. In case of providing condition for Group By clause, we have to use Having clause.
3. Practical Demonstration

```
→ use world;
→ show tables;
→ select * from country;
→ select name from country;
→ select count(name) from country;
→ select count(name) from country where surfacearea>100;
→ Select continent, count(name) from country group by continent;
→ select continent, count(name) from country group by continent
  having count(name)<10;
```

Date...../...../..... Page. No: .....

Sequence of using where, group by, having and order by clauses

1. where > group by > having > order by
2. Practical Demonstration
   → use world;
   → select continent, count(name) from country where surfacearea>300 group by continent having count(name) > 20 order by count(name) ASC;

Set Operators

1. We can retrieve the data from different tables at a time using the set operators.
   → Select * from TableOne set Operator Select * from TableTwo;
2. The below are the different set operators
   - Union
   - Union All
   - Intersect
   - Minus

Union Operator

1. Union operator is one of the operators in the four set operators.
2. Purpose of union operator: The union operator combines the results of two or more queries into a distinct single result set that includes all the rows that belong to all queries in the union.
3. Practical Demonstration:
   → use qafox;
   → create table empone (id int);
   → insert into empone value (1);
   → insert into empone value (3);
   → insert into empone value (5);

Date...../...../..... Page. No: .....

→ select * from empone;
→ create table emptwo (num int)
→ insert into emptwo values (2);
→ insert into emptwo values (3);
→ insert into emptwo values (4);
→ select * from emptwo;
→ Select * from empone union select * from emptwo;
→ Demonstrate with the two column table
    1. Combination of column records will be considered for comparison and elimination of duplicates.
→ Rules:
    1. Select statements of each table must retrieve the same number of column.
    2. Column provideed 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.

These two will work and we don't get any error but we will not get expected result.

Union All operator

1. Union All operator is one of the set operator.
2. Unlike union operator, union all won't eliminate the duplicates.
3. Practical Demonstration:
→ use qafox;
→ create table tone (id int);
→ insert into tone values (1);
→ insert into tone values (1);
→ insert into tone values (5);
→ select * from tone;
→ create table ttwo (num int);
→ insert into ttwo (2);
→ insert into ttwo (3);

Date...../...../..... Page. No: .....

→ insert into ttwo(4);
→ select * from ttwo;
→ Select id from tone union select num from ttwo;

Intersect Operator

1. Intersect operator is one of the set operator.
2. Intersect operator will retrieve the common records from the given tables.
3. Intersect operator is not supported by MySQL RDBMS.
   → We have to use Oracle SQL RDBMS
   → Google search for 'Try Oracle SQL Online Practice'.
4. Practical Demonstration:
   → use qafox;
   → create table tone (id int);
   → insert into tone values (1);
   → insert into tone values (3);
   → insert into tone values (5);
   → select * from tone;
   → create table ttwo (num int);
   → insert into ttwo (2);
   → insert into ttwo (3);
   → insert into ttwo (4);
   → select * from ttwo;
   → select id from tone intersect select num from ttwo;

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.
   → We have to use Oracle SQL RDBMS.
   → Google search for 'Try Oracle SQL online Practice'.

4. Practicle Demonstration:

→ use qafox;
→ create table tone (id int);
→ insert into tone values (1);
→ insert into tone values (3);
→ insert into tone values (5);
→ select * from tone;
→ create table ttwo (num int);
→ insert into ttwo (2);
→ insert into ttwo (3);
→ insert into ttwo (4);
→ select * from ttwo;
→ select id from tone minus seleet num from ttwo;
→ seleet num from ttwo minus select id from tone;

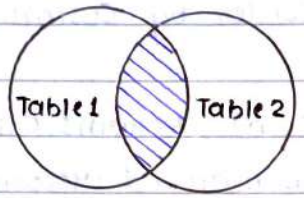
Alias for Table

→ Example 1: select * from customers AS Persons;
→ Example 2: select o.OrderID, o.OrderDate, c.CustomerName
from Customers AS c, Orders AS o
Where c.CustomerName = 'Around the Horn' And
c.CustomerID = o.CustomerID;
→ Example 3: seleet o.id, o.firstname, t.lasthame from
empdetailsone o, empdetailstwo t where o.id = t.id;
→ Example 4: Select Orders.OrderID, Orders.OrderDate,
Customers.CustomerName From Customers, Orders
Where Customers.CustomerName = 'Around the Horn'
And Customers.CustomerID = Orders.CustomerID;

Date...../...../..... Page. No: .....

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.
2. The Two tables should have a common column.
3. Types of Joins:
   - Inner Join (Equi Join or Simple Join): Returns records that have matching values in both tables.



Inner Join

Syntax:

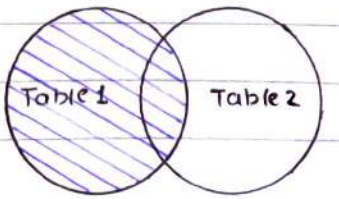
```
Select column_name(s) from table1 Inner Join table2 ON
table1.column_name = table2.column-name;
```

Example:

→
```
Select * from empdetailsone inner join empdetailstwo
on empdetailsone.id = empdetailstwo.id;
```

→
```
Select Products.ProductID, Products.ProductName,
Categories.CategoryName From Products Inner Join
Categories ON Products.CategoryID = Categories.CategoryID;
```

   - Left Join (Left Outer Join): Returns all records from the left table, and the matched records from the right table.



Left Join

Date...../...../..... Page. No: .....

Left Join Syntax:

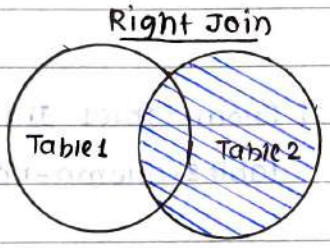
Select column_name(s) from table1 Left Join table2
ON table1.column_name = table2.column_name;

Example:

→ Select * from empdetailsone left join empdetailstwo
on empdetailsone.id = empdetailstwo.id;

→ Select Customers.CostomerName, Orders.OrderID
From Customers Left Join Order ON Customers.CustomerID
= Order.CostomersID Order by Customers.CustomerName;

- Right Join (Right Outer Join): Returns all records from the right table, and the matched records from the left table.



Right Join syntax:

Select column_name(s) From table1 Right Join table2
on table1.column_name = table2.column_name;

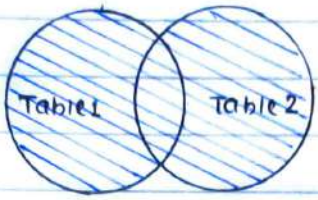
Example:

→ select * from empdetailsone right join empdetailstwo
on empdetailsone.id = empdetailstwo.id;

→ Select Orders.OrderID, Employees.LastName, Employees.FirstName
from Order right join Employees on Order.EmployeeID
= Employees.EmployeeID Order by Order.OrderID;

Date...../...../..... Page. No: .....

- Full Join (Full Outer Join) - Returns all records when there is a match in either left or right table.



Full outer Join

Syntax:

Select column_name(s) from table1 Full outer Join table2 ON table1.column_name = table2.column_name where condition;

Example:

→ Select * from empdetailsone full join empdetailstwo on empdetailsone.id = empdetailstwo.id;

→ Select Customers.CustomerName, Orders.OrderID from Customers full outer join order on Customers.CustomerID = Order.CustomerID order by customers.CustomerName;

- Self Join: A self join is a regular join, but the table is joined with itself (only one table required).

Example:

select * from empdetails e, empdetails t where e.id = t.deptid;

Syntax:

Select column_name(s) from table t1, table t2 where condition;

Date...../...../..... Page. No: .....

Sub Query (Explaining Single Row Sub Query by solving different SQL Problems)

1. Sub Query is a query inside another query.
2. Understanding sub query:
   - → Outer Query
   - → 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
   2. Multi Row Sub Query
4. Practical Demonstration:

Single Row Sub Query

1. Find all the customers in the Customers table who are from the same city of 'Hari Kumar'
   - → First we have to find the city of Hari Kumar in the customer table.
     - Select City from Customers where ContactName = 'Hari Kumar';
   - → We have to make the above query a sub-query (Inner query) in the where clause of Outer query.
     - Select * from Customers where City = (Select City from Customers where ContactName = 'Hari Kumar');
2. Finding the second maximum price of the products.
   - → Select Price from Products;
   - → Select max(Price) from Products;
   - → Select Price from Products < (Select max(Price) from Products);
   - → Select Price from Products where Price < (Select max(Price) from Products);
   - → Select max(Price) from Products where Price < (Select max(Price from Products);

Date...../...../..... Page. No: .....

3. Finding the third maximum Price of the Products.
   → Select max(Price) from Product where Price < (select max(Price) from Product where Price < (Select max(Price) from Product));
4. Display the product sold at the least price.
   → select min(Price) from Products;
   → select ProductName from Products where Price = (select min(Price) from Products);

Multi row sub query

1. We will practically demonstrate the multi-Row sub query in upcoming sessions.
2. We have to use in, any, all and exits operators.

In operator

1. The IN operator allows you to specify multiple values in a WHERE clause. It allows us to provide multiple values in the Where clause condition.

Syntax:

```
Select column_name(s) from table_name where column_name
IN (value1, value2, ...);
```

2. Practical Demonstration:
   → Select * from Customers where Country = 'USA';
   → Select * from Customers where Country IN ('USA', 'UK', 'Italy', 'France', 'Spain');
   → Select * from Customers where Country NOT IN ('USA', 'UK', 'Italy', 'France', 'Spain');
   → Select * from Customers where CustomerID IN (Select CustomerID from Orders);

→ Select * from Customers Where CustomerID NOT IN (select CustomerID from Orders);

3. In Operator is mainly used with Multi Row Sub Queries.

Using in operator with Multi Row Sub Query

1. Multi Row Sub Query is a Query inside another Query and it provides multi records as input to the outer query.
2. Different operators which can be using in Multi Row Sub Query.
   - in
   - any
   - all
   - exists
3. Practical Demonstration
   1. Find the different products from the specified categories

```
Select * from Product where CategoryID in (Select CategoryID
from Categories where CategoryName like 'C%');
```

Using Any operator with Multi Row sub Query

1. Allows you to perform a comparison between a single column value and a range of other values.

The ANY Operator:
- returns a boolean value as a result
- return TRUE if any of the Subquery values meet the condition.

ANY means that the conditions will be true if the operation is true for any of the values in the range.

Syntax:
Select column_name(s) from table-name where column-name operator ANY (select column-name from table-name where condition);

2. Practical Demonstration
→ Select * from Products where ProductID < any (select ProductID from OrderDetails where Quantity = 1);

Using All operator with Multi-Row Sub Query

1. Allow you to perform comparison between a single column value and a range of other values.

The All operation:
- return a boolean value as a result
- return TRUE if all of the subquery values meet the condition
- is used with select, where and Having statements.

ALL means that the condition will be true only if the operation is true for all values in the range.

Syntax:
Select column_name(s) from table-name where column-name operator ALL (select column-name from table-name where condition);

2. Practical Demonstration
→ Select * from Products where ProductID < all (select ProductID from OrderDetails where Quantity = 1);

Date...../...../..... Page. No: .....

Exist Operator

1. The Exists operator is used to test for the existence of any record in a subquery.
The Exists operator return TRUE if the subquery return one or more records.
Exist operator is used with sub queries.

Syntax:
```
select column_name(s) from table_name where exists
(select column_name from table_name where condition);
```

2. Practical Demonstration
→ `Select * from order where exists (select * from customers where customerID >91);`

Using sub Queries for retrieving the records from multiple tables

1. Practical Demonstration
→ `Select city, (Select country from country where country.country_ID = city.country_id) country from City;`
↳ will get the records

→ `Select country, (select city from city where country.country_id = city.country_id) city from country;`
↳ will get an error
↳ sub query resulting in multiple records, will give an error

Date...../...../..... Page. No: .....

Using Multiple sub Queries in a single SQL statement

1. Practical Demonstration
   → List out the films having the length less than the maximum length and having the rental duration equal to the minimum rental duration.
      1. Finding the maximum length of the films
         → select max(lenght) from film;
      2. finding the minimum rental duration of the films.
         → 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.
         → select title, length, rental_duration from film where length < (select max(length) from film) and rental_duration = (select min(rental_duration) from film);

Integrity constraints

1. 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
   2. Unique
   3. Primary Key
   4. Foreign Key
   5. Check
   6. Default
4. Example: Create table employees (sno int not null, name varchar(10), experience int);

Date...../...../..... Page No: .....

Syntax:

Create table table_name (column1 datatype constraint, column2 datatype constraint, column3 constraint, ---);

Not Null (Integrity constraint)

1. Not Null when specified to a column will not allow the column to allow null values.
2. Not Null is an Integrity constraint
3. Practical Demonstration:
   - → create table emp (id int not null, name varchar(15), experience int);
   - → Alter table Persons Modify column Age int NOT NULL;
     - ↳ Not Null on ALTER TABLE

Unique (Integrity constraint)

1. Unique is an Integrity constraint.
2. Unique when specified to a column will not allow the column to allow duplicates value.
3. Practical Demonstration
   - → create table emp (id int unique, name varchar(15), experience int);
   - → create table Persons (id int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, Unique(ID));
   - → create table emp (id int, name varchar(15), experience int, unique (id, name));
     - ↳ unique constraint on multiple columns.
   - → Alter Table Persons Add unique (ID)
   - → Alter Table Persons Add constraint Uc_Person Unique (ID, LastName);
   - → Alter Table Persons Drop Index Uc_Person;

     OR

     Alter Table Persons DROP constraint Uc_Person;

Date...../...../..... Page. No: .....

4. Practical Demonstration:

→ use qafox;

→ create table emp (id int, name varchar(15), experience int, unique (id, name));

→ select * from emp;

→ insert into emp values (1, 'Arun', 12);

→ insert into emp values (1, 'Varun', 8) → will NOT given an error as combination of both id and Name (1 & Varun) is unique.

→ insert into emp values (1, 'Arun', 9)

↓

will given an error as combination is not unique

Primary Key (Integrity Constraint)

1. Primary key is an Integrity constraint
2. Primary Key = NOT NULL + Unique
3. Primary key must contain Unique values, and cannot contain NULL values
4. A table can have only one primary key; and in the table, this primary key can consist of single or multiple columns (fields).
5. Practical Demonstration:

→ create table emp (id int not null unique, name varchar(15), experience int);

→ create table emp (id int Primary Key, name varchar(15), experience int);

→ create table emp (id int, name varchar(15), experience int, Primary Key(ID));

→ create table emp (id int, name varchar(15), experience int, Primary Key (Id, name));

→ Alter table Persons Add Primary Key (ID);

→ Alter table Persons Add Constraint PK_Person Primary Key (ID, LastName);

→ Alter Table Persons Drop Primary key;

OR

Alter table Person Drop constraint PK_Person;

Date...../...../..... Page No: .....

6. Practical Demonstration:

→ use qatox;

→ create table emp (id int, name varchar(15), experience int, primary key (id, name));

→ select * from emp;

→ insert into emp values (1, 'Arun', 12);

→ insert into emp values (1, 'Varun', 7); → will not give an error, combination is unique (1 and Varun)

→ insert into emp values (1, 'Arun', 9); ↓ will give an error as combination is not unique.

→ insert into emp(name, experience) values ('Tharun', 10); ↳ Error, id is NULL.

→ insert into emp(id, experience) values (3, 10); → Error, name is NULL.

Foreign Key (Integrity constraint)

1. Foreign key is one of the Integrity constraints.
2. The foreign key constraint is used to prevent action that would destroy links between tables.
   A foreign key is a field (or collection of fields) in one table, that refers to the Primary key in another table.
3. We need to create two tables having a common column.
   - Parent table (Reference Table) - should have a Primary key (candidate key) specified for common column.
   - Child Table - should have the foreign key specified for common column.
4. Child Table depends on the parent Table:
   → Inserting the records in child table should match with the Parent table common column.
   → Deleting the records in Parent table is not possible if there are dependent records in the child Table.
   ↳ Use ON Delete Cascade for this

Date...../...../..... Page. No: .....

5. Practical Demonstration:

→ create table employee (id int primary key, name varchar(15), experience int);

→ create table salary (id int, sal int, foreign key (id) refrence employee (id));

→ Alter table Order Add foreign key (Person ID) References Persons (PersonID); → creating foreign key on PersonID column when 'order' table is already created.

→ Alter table Order Drop foreign key FK_PersonOrder;

On Delete Cascade: ON DELETE CASCADE clause in MySQL is used to automatically remove the matching records from the child table when we delete the rows from the parent table.

→ create table salary (id int, sal int, foreign key (id) references employee (id) on delete cascade);

Check (Integrity constraint)

1. check is an integrity constraint.
2. The check constraint is used to limit the value range that can be placed in a column.
3. Practical Demonstration:

→ create table empone (id int, name varchar(15), experience int check (experience > 5));

→ Create table Persons (id int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, City varchar(255), Constraint CHK_Person check (Age >= 18 AND City = 'sandnes'));

→ Alter Table Persons Add check (Age >= 18); ↳ on multiple column.

→ Alter Table Persons Add Constraint CHK_PersonAge check (Age >= 18 AND city = 'Sandnes');

→ Alter Table Persons Drop check CHK_PersonAge;

→ create table emptwo (id int, location varchar(15) check (location in ('India', 'USA', 'UK')));

Date...../...../..... Page. No: .....

Default (Integrity constraint)

1. Default is an integrity constraint
2. The Default constrain is used to set a default value for a column.
3. Practical Demonstration:
   - → create table empone (id int, experience int default 5);
   - → create table emptwo (id int, location varchar(15) default 'India');
   - → create table empthree (id int, DateOfJoining date default '2007-08-15');
   - → Alter Table Persons Alter City Set Default 'Sandnes';
   - → Alter Table Persons Alter City Drop Default;

auto-increment

1. auto-increment will increment the values of the table columns by one, if we are not inserting any data. → auto-increment should always specify with primary key.
2. Practical Demonstration:
   - → create table empOne (id int primary key auto-increment, name varchar(15));
     - insert into empone values(5, 'Arun'); → 5 Arun
     - insert into empone(name) values ('Varun'); → 6 Varun (+1)
     - insert into empone values(9, 'Tharun'); → 9 Tharun
     - insert into empone(name) values ('Dinesh'); → 10 Dinesh (+1)
   - → create table emptwo (id int primary key auto-increment, name varchar(15));
     - insert into empone(name) values('Arun'); → 1 Arun
     - insert into empone(name) values ('Varun'); → 2 Varun
   - → create table emptwo (id int primary key auto-increment, name varchar(15)) auto-increment = 100;

Date...../...../..... Page. No: .....

- insert into empone(name) values ('Arun'); → 100 Arun
- insert into empone(name) values ('Varun'); → 101 Varun

Insert Into

1. Insert into statement is used for copying the records from a table into another table having same column names and data types.

Syntax:

Specify both the column names and the values to be inserted:

- Insert into table_name (column1, column2, column3, ...) values (value1, value2, value3, ...);

- Specify only values to be inserted:
  Insert into table_name values (value1, value2, value3, ...);

2. Practical Demonstration:

→ Insert into newcity select * from city;
→ Insert into citytwo (id, name) select id, name from city;
→ Insert into citythree (id, name) select id, name from city where CountryCode = 'AFG';

↳ copying the record from a table into another table

AS Keyword

1. AS keyword is used to create a new table with the records of an existing table.
2. Practical Demonstration
   → Create Table NewPlace AS Select * from Place;
   → Create Table PlaceTwo AS select name, district from Place;

Date...../...../..... Page. No: .....

IfNull() function

1. IfNull() function is used to provide an alternate value when the column value is null.
2. Practical Demonstration:
   → Create table empone (id int, salary int);
   → Select (ifnull(salary, 0) + 100000) newsalary from empone;
   → Select (ifnull(salary, 100) + 100000) newsalary from empone;

Case, When, Then and End Keywords

1. The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it return the value in the ELSE clause.
   If there is no ELSE part and no conditions are true, it return NULL.

Case syntax:

```
case
    When condition1 Then result1
    When condition2 Then result2
    When conditionN Then resultN
     Else result
END;
```

2. Case, when, Then and End keywords can be used in select statement.
3. Practical Demonstration:
   → Select OrderID, Quantity
   case
   When Quantity > 30 Then 'The quantity is greater that 30'

```
        when Quantity = 30 Then 'The quantity is 30'
        Else 'The quantity is under 30'
END AS QuantityText
From OrderDetails;
```

→
```
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,
 Unit
 from Products;
```

→
```
Select * from Customers
 where city =
 (Case
      When Country in ('USA', 'UK') Then City
      When Country Not in ('USA', 'UK') Then 'Berlin'
 End);
```

→
```
select * from Customers
 Order By
 (case
    When Country in ('USA', 'UK') Then Country
    When Country NOT in ('USA', 'UK') Then City
End);
```

Date...../...../..... Page. No: .....

Delimiter

1. Default Delimiter is ;
2. We can change the delimiter to other symbol
3. Practical Demonstration :

```
→ select * from city;
→ delimiter //
→ select * from city //
```

4. Different symbols can be used instead of //
5. Delimiters are generally used with stored Procedures.

Delimiter Usage in Stored Procedures

1. Stored Procedures belong to PL/SQL.
2. PL/SQL is a out of topic and is not required for testers too.
3. We are going to learn 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 statement.
6. Practical Demonstration :

```
Delimiter //
Create procedure getAllCities()
Begin
    Select * from city;
End //
call getAllCities();
```

Date...../...../..... Page. No: .....

Views

1. The main purpose of views is to create different varieties of the same table or tables (Virtual Tables)
2. We can create views by customizing the same table columns or by selecting the columns from multiple tables.
3. Hide the Database implementation of the actual tables and show the desired virtual views to the Users.
4. Practical Demonstration:

```
→ create table empone (id int, name varchar(15), country varchar(15));
→ insert into empone values (1, 'Arun', 'India');
→ insert into empone values (2, 'Varun', 'UK');
→ insert into empone values (3, 'Tharun', 'Spain');
→ insert into empone values (4, 'Dinesh', 'New Zealand');
→ select * from empone;
→ create table emptwo (id int, experience int);
→ insert into emptwo values (1, 12);
→ insert into emptwo values (2, 7);
→ select * from emptwo;
→ create view emponeviewa as select * from empone;
→ create view emponeviewb as select id, name from empone;
→ create view emponeviewc as select id, country from empone;
→ create view emponeviewd as select country, name, id from
  empone;
→ create view empviewb as select empone.id, empone.name,
  emptwo.experience from empone, emptwo where empone.id
  = emptwo.id;
→ Drop view empviewb;
```

5. Performing operations on the view will effect the original tables and vice versa.

Date ...../...../..... Page. No: .....

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.
   → Insertions and update will show down if you create indexes.
   → Select operations need indexes for speeder retrieval.
4. Practical Demonstration:
   → use qafox;
   → create table empone (id int, name varchar(15), experience int, country varchar(15));
   → insert into empone values (1, 'Arun', 12, 'India');
   → insert into empone values (2, 'Varun', 7, 'USA');
   → insert into empone values (3, 'Tharun', 9, 'UK');
   → select * from empone;
   → select * from empone where country = 'USA';
   → show indexes from empone;
   → create index empcountry on empone (country);
   → show indexes from empone;
   → drop index empcountry on empone;
   → create table emptwo (id int primary key, name varchar(15));
      - Primary index will be automatically created.
      - show indexest from emptwo;
   → create table emptwee (id int unique, name varchar(15));
      - id index will be automatically created.
      - show indexes from emptwee;

Date...../...../..... Page. No: .....

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 :
   - → mysql -u root -p
   - → Give the password
   - → Then perform the operations.

Types of SQL statements

1. DQL (Data Query Language) :
   - → Select

2. DML (Data Manipulation Language) :
   - → Insert
   - → update
   - → Delete

3. DDL (Data Defination Language) :
   - → Create
   - → Drop
   - → Alter
   - → Truncate

4. TCL (Transaction Control Language):
   - → commit
   - → rollback

5. DCL (Data Control language)
   - → grant
   - → revoke

Date...../...../..... Page. No: .....

Grant and Revoke SQL statements

1. Grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, INDEX, CREATE, ALTER, DROP, GRANT OPTION OR ALL.
2. Once you have granted previleges, you may need to revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, OR ALL.
3. Create user in MySQL server
4. Grant and Revoke Permission:
   - → use qafox;
   - → grant select on empone to 'userone';
   - → revoke select on empone from 'userone';
   - → grant create, delete, drop, seleet, insert, update, alter on empone to 'userone', 'usertwo';
   - → revoke select, delete, drop, create, alter, insert, update on empone from 'userone', 'usertwo';

Temporary tables

1. There may be some situations where the Applications has to create a table for a purpose and there after using the table it has delete it.
2. Practical Demonstration:
   - → use qafox;
   - → create table normaltable (id int, name varchar(15));
   - → create temporary table temptable (id int, name varchar(15));
   - → insert into temptable values (1, 'Arun');
   - → insert into temptable values (2, 'Varun');
   - → insert into temptable values (3, 'Tharun');
   - → select * from temptable;

Date...../...../..... Page. No: .....

show columns, show Indexes, show Privileges and show Grant statements

1. show colomns from tablename;
2. show indexes from tablename;
3. show privileges
4. show grants for 'userone';
5. show grants for 'usertwo';

Inserting Null

1. Two ways in which we can insert null values into the Tables
   - Direct way - Null
   - No value - Null
2. Practical Demonstration:
   - → use qafox;
   - → create table empone (id int, name varchar(15));
   - → insert into empone values (1, 'Arun');
   - → insert into empone values (2, null);
   - → insert into empone(name) values ('Tharun');
   - → select * from empone;

using trim() for trimming the corner characters of the specified Table values

1. Practical Demonstration:
   - → use qafox;
   - → select trim ('a' from 'arun'); → run
   - → select trim ('a' from 'rana'); → ran
   - → select trim ('a' from 'aruna'); → run
   - → create table empone (name varchar(15));
   - → insert into empone values ('arun');
   - → insert into empone values ('naresh');

→ insert into empone values ('varun');
→ insert into empone values ('tharun');
→ insert into empone values ('nandan');
→ select * from emptwo;
→ select trim('n' from name) from emptwo;

using wild-cards as normal characters

1. Practical Demonstration:

→ use qatox;
→ select * from empone;
→ insert into empone values ('dines_');
→ Select * from empone where name like '%_';
→ select * from empone where name like '%$_' escape '$';

Database Objects

1. The things that we can create under any database using the create statements in SQL are nothing but database objects.
   - Tables
   - Views
   - Index
   - Stored Procedures
   - etc.
2. Practical Demonstration:

→ create table empone (id int, name varchar(15));
→ create index idindex on empone(id);
→ create view empview as select * from empone;
→ Delimiter //

```
create procedure getAllEmpdetails()
Begin
    Select * from empone;
    Slect id from empone;
    Select name from empone;
End //
```

Date...../...../..... Page. No: .....

Savepoint

1. Using savepoint we can save the different states of the table and rollback to the desired state of the table when needed.
2. Practical demonstration:

→ set autocommit = 0;
→ create table
→ start transaction;
→ select * from table
→ savepoint zero;
→ insert into table values
→ savepoint one;
→ insert into table values
→ savepoint two;
→ rollback to two;
→ rollback to one;
→ rollback to zero;

Uninstalling MySQL server completely from windows machine

→ Uninstall from control panel.
→ Remove MySQL from Program Files.
→ Remove MySQL from Program files (x86).
→ Remove MySQL from User data hidden folder.
→ Remove MySQL from C > Users > username > AppData > Roaming > MySQL.