1. WHAT IS SQL?

A. Structured Query Language (SQL) is a programming language used to interact with databases.

2. WHAT IS DATABASE?

A. Database is a system that allows user to store and organise data.

3. TYPES OF DATABASES:

A. **Relational database**: Data stored in the form of tables.

B. **Non-Relational database**: Data stored in the form of key and value pairs.

4. EXCEL USE?

- Small amount of data.
- One-time analysis.
- Quick Chart/Graph.
- Untrained Person.

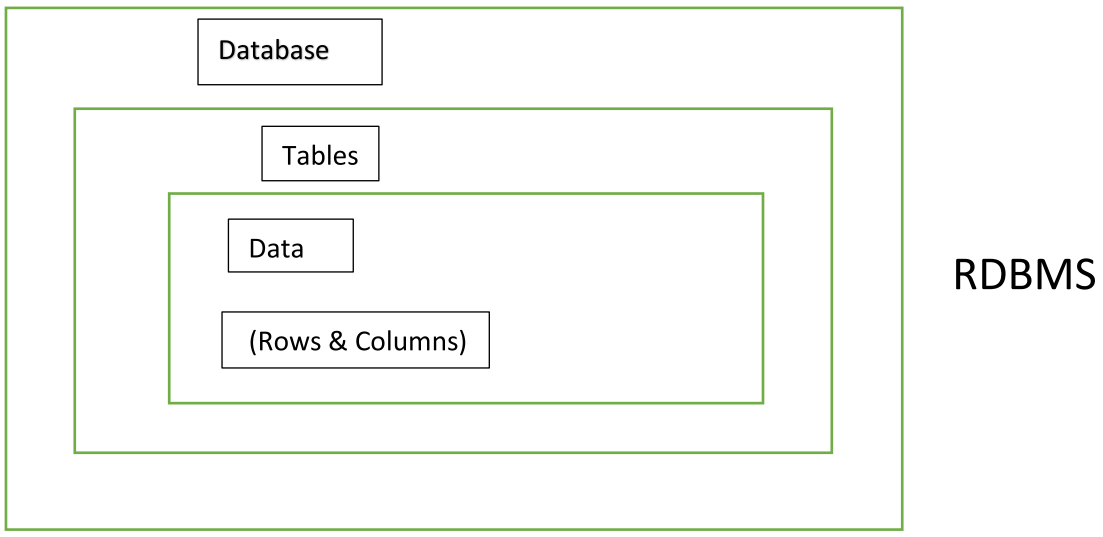
5. DATABASE USE?

- Large amount of data.
- Store Real time data from Website/Apps.
- Easily combine with different datasets.
- Automate steps and can re-use.
- Easy and Safe access.
- Data Integrity.
- Deep search capabilities.

6. DATABASE PLATFORMS:

- MySQL
- SQLite
- SQL Server
- PostgreSQL
- Oracle PL/SQL
- Snowflake
- Amazon REDSHIFT.

7. SQL STRUCTURE



8. DATA TYPES

- Data type of a column defines what value the column can store in table.
- Defined while creating tables in databases.
- Data types mainly classified into three categories + mostly used

- **String**: char, varchar, etc.
- **Numeric**: int, float, bool, etc.
- **Date and time**: date, datetime, etc.

Commonly Used data types in SQL:

- **int**: used for the integer value.
- **float**: used to specify a decimal point number.
- **bool**: used to specify Boolean values trues and false.
- **char**: fixed length string that can contain numbers, letters and special characters.
- **varchar**: variable length string that can contain numbers, letters and special characters.
- **date**: date format YYYY-MM-DD.
- **datetime**: date & time combination, format is YYYY-MM-DD.

9. Primary and Foreign Keys:

Primary Key (PK):

- A PK is a unique column we set in a table to easily identify and locate data in queries.
- A table can have only one PK, which should be UNIQUE and NOT NULL.

Foreign Key (FK):

- A FK is a column used to link two or more tables together.
- A table can have any number of FK, can contain DUPLICATE and NULL values.

10. Constraints:

- Constraints are used to specify rules for data in a table.4
- This ensures the accuracy and reliability of the data in the table.
- Constraints can be specified when the table is created with the CREATE TABLE statement, or
- After the table is created with the ALTER TABLE statement.
- Syntax:

```
CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
        ------
);
```

Commonly used constraints in SQL:

- **NOT NULL**- Ensures that a column cannot have a NULL value.
- **UNIQUE**- Ensures that all values in a column are different.

- **PRIMARY KEY**- A combination of a NOT NULL and UNIQUE.
- **FOREIGN KEY**- Prevents actions that would destroy links between tables (used to link multiple tables together)
- **CHECK**- Ensures that the values in a column satisfies a specific condition.
- **DEFAULT**- Sets a default value for a column if no value is specified.
- **CREATE INDEX**- Used to create and retrieve data from the database very quickly.

11. Create Database and Table

CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

- Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
);
```

CREATE DATABASE

- Syntax:

```
CREATE DATABASE database_name
```

1>**CREATE TABLE** statement is used to create a new table in a database. You should specify the name and data types of each column.

```
  CREATE TABLE table_name(
  Column1 datatype,
  Column2 datatype,
  Column3 datatype
);
```

```
create table arpandb.Hello
(
        PersonID int,
   firstname varchar(255),
   lastname varchar(255),
   city varchar(255),
   salary int
);
```

2>**INSERT INTO** statement is used to insert new record in a table

```
  INSERT INTO table_name(Column1, Column2, Column3)
  VALUES(value1, value2, value3);
```

```
insert into
arpandb.hello(personid,firstname,lastname,city,salary)
values(1,"Arpan","Panigrahi","ABC 1", 3000);

SELECT * FROM arpandb.hello;

insert into
arpandb.hello(personid,firstname,lastname,city,salary)
values(2,"Lokesh","Yadav","XYZ 2", 1000);

insert into
arpandb.hello(personid,firstname,lastname,city,salary)
values(1,"Adarsh","Singh","MNO 3", 10000);

insert into
arpandb.hello(personid,firstname,lastname,city,salary)
values(1,"Arpan","Panigrahi","ABC 1", 3000);

insert into
arpandb.hello(personid,firstname,lastname,city,salary)
values(4,"Satyam","Shukla","WRE 5", 2000);

SELECT * FROM arpandb.hello;
```

3>**<u>AND Operator</u>** displays a record if both the first and second condition is true.

<u>OR Operator</u> displays a record if either the first or the second condition is true.

SELECT column1,column2,---

FROM table_name

WHERE condition1 AND condition2 AND condition3 --;

Select * from arpandb.hello;

Select * from arpandb.hello where firstname = "Arpan" AND lastname = "Panigrahi";

select * from arpandb.hello where firstname = "Lokesh" OR lastname = "Shukla";

4>**<u>WHERE Clause</u>** is used to extract only those records that fullfill a specified criterion.

SELECT column_name from

table_name WHERE column_name

operator value;

select * from arpandb.hello where personid = 4;

5>**ORDER BY** Keyword is used to sort the result set by specified column.

1> The ORDER BY Keyword sort the records in ascending order by default.

2> If you want to sort the records in a descending order, you can use DESC keyword.

SELECT column_name from table_name ORDER BY column_name ASC|DESC;

select * from arpandb.hello order by salary desc;

6>In a table, some of the columns may contain duplicate values. This is Not a problem, sometimes you will want to list only different values in a table.

SELECT **DISTINCT** column_name

from table_name;*/

Select * from arpandb.hello;

Select distinct(firstname) from arpandb.hello;

7>The **DELETE Statement** is used to delete rows in a table.

NOTE:- If you omit the WHERE clause all records will be deleted. Be careful while deleting records. You cannot undo this statement.

```
DELETE FROM table_name where

condition;


delete from arpandb.hello where personid = 4;
```

```
use arpandb; /* Select * from arpandb.hello; */


select * from hello;
```

8>**DATE TIME**=>

```
SELECT NOW(),CURDATE(),CURTIME();


select now(),curdate(),curtime();
```

9> **Functions**:

AVG()- SELECT AVG(column_name) from table_name;

COUNT()- SELECT COUNT(column_name) from table_name;

LCASE()- SELECT LCASE(column_name) from table_name;

MAX()- SELECT MAX(column_name) from table_name;

MIN()- SELECT MIN(column_name) from table_name;

SUM()- SELECT SUM(column_name) from table_name
WHERE condition;

ROUND()- SELECT column_name,
ROUND(column_name,decimals) from table_name;

SUBSTRING()-> is used to get part of a String

SELECT LastName, SUBSTR(FirstName,1,1) AS
Initial from Persons;

UCASE()- SELECT UCASE(column_name) from table_name;

REPLACE()- SELECT REPLACE(CustomerName,'Brown','Hello')
from Orders;

SELECT AVG(salary) from hello ;

SELECT SUM(salary) from hello;

Select UCASE(firstname) from hello;

Select LCASE(lastname) from hello;

Select MAX(salary) from hello;

Select MIN(salary) from hello;

SELECT lastname, SUBSTR(firstname,1,1) from hello;

SELECT COUNT(salary) from hello;

SELECT city, ROUND(lastname,2) from hello;

10>**GROUP BY** Statement is used in conjunction with the aggregate functions to group the result set by one or more columns.

SELECT column_name(s), FROM table_name

WHERE condition GROUP BY column_name(s)

create table orders(

o_id int,

orderprice int,

customer varchar(255)

```
);

insert into orders(o_id,orderprice,customer)
values(1,1200,"john");
insert into orders(o_id,orderprice,customer)
values(2,1500,"brown");
insert into orders(o_id,orderprice,customer)
values(3,300,"john");
insert into orders(o_id,orderprice,customer)
values(4,200,"taylor");
insert into orders(o_id,orderprice,customer)
values(5,1000,"john");
insert into orders(o_id,orderprice,customer)
values(6,2000,"Maria");
insert into orders(o_id,orderprice,customer)
values(6,2000,"Thomas");
insert into orders(o_id,orderprice,customer)
values(7,3000,"Antonio");
insert into orders(o_id,orderprice,customer)
values(8,4000,"ABCDEF");
insert into orders(o_id,orderprice,customer)
values(9,5000,"kate");
```

select * from orders;

----- Find the total sum(total orders) from each customer; ---

select customer,sum(orderprice) from orders group by customer;

11> The **HAVING Clause** was added to SQL bcoz the Where Keyword cannot be used with aggregate functions

SELECT Customer, Sum(OrderPrice) from

Orders

Group BY Customer Having

SUM(OrderPrice)<4000;

-- // we want to find that if any of the customers have

-- // a total order of more than 2000

select customer,sum(orderprice) from orders

group by customer having sum(orderprice)>2000;

12> **ALTER TABLE** statement is used to add,delete,or modify columns in an existing table.

1> **To Add a Column in a table** -> (ALTER TABLE table_name ADD column_name datatypes;)

2> **To delete a column in a table** -> (ALTER TABLE table_name drop column column_name;)

3> **To change the data types of a column in a table** -> (ALTER TABLE table_name modify column column_name datatypes;).

select * from orders;

ALTER TABLE orders ADD location varchar(255);

ALTER TABLE orders drop column location;

ALTER TABLE orders modify column orderprice int;

ALTER TABLE orders modify column orderprice varchar(255);

13> **SQL ALIAS** - you can give a table or a column another name by using alias.

SELECT Column_name as alias_name from table_name;

SELECT customer as customer_name from orders;

14> **SQL DROP DATABASE** -> The DROP DATABASE is used to drop an existing SQL Database.

SQL DROP TABLE -> The DROP TABLE Statement is used to drop(delete) an existing table in a Database.

DROP DATABASE Database_name;

```
DROP TABLE Table_name;
```

```
drop table orders;

drop database arpandb;
```

15> **<u>BETWEEN Operator</u>** is used in a where clause to select a range of data between two values.

Begin and end value are included.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
create database arpandb;
use arpandb;
select * from orders;

SELECT * FROM orders WHERE orderprice BETWEEN 1000
AND 1500;
```

16> **<u>IN Operator</u>** allows you to specify multiple values in a where clause. The number of values in the parenthesis can be one or more, with each values separated by comma.

SELECT * From Persons where LASTNAME IN('JAMES','ARPAN');

SELECT * From orders where customer IN('john','brown');

select orderprice from orders where customer in ('john','brown');

17> **<u>SQL Like Operator</u>** is used in a Where CLAUSE to search for a specified pattern in a column.

- WHERE CustomerName LIKE 'a%' => Finds any values that start with "a"
- WHERE CustomerName LIKE '%a' => Finds any values that end with "a"
- WHERE CustomerName LIKE '%or%' => Finds any values that have "or" in any position
- WHERE CustomerName LIKE '_r%' => Finds any values that have "r" in the second position
- WHERE CustomerName LIKE 'a_%' => Finds any values that start with "a" and are at least 2 characters in length.
- WHERE CustomerName LIKE 'a__%' => Finds any values that start with "a" and are at least 3 characters in length.
- WHERE CustomerName LIKE 'a%o' => Finds any values that start with "a" and ends with "o"

```
select * from orders where customer like "j%";
select * from orders where customer like "%n";
select * from orders where customer like "%ow%";
```

18> The **SQL TRUNCATE TABLE** command is used to delete complete data from an existing table.

```
TRUNCATE TABLE table_name;

TRUNCATE TABLE orders;
```

19> **The UPDATE Statement** is used to update records in a table.

```
UPDATE table_name
   SET column1 = value1, columnn2 = value2, ----
   WHERE condition;

update orders
set orderprice=800
where o_id = 4;
```

20> **CONSTRAINT** => SQL Constraints are used to specify rules for the data in a table.

- **NOT NULL** => Ensures that a column cannot have a NULL value.
- **UNIQUE** => Ensures that all values in a column are different.
- **PRIMARY KEY** => A combination of a NOT NULL and UNIQUE. Uniquely Identifies each row in a table.
- **FOREIGN KEY** => Prevents actions that would destroy links between tables.
- **CHECK** => Ensures that the values in a column satisfies a specific condition.
- **DEFAULT** => Sets a default value for a column if no value is specified.

21> **NOT NULL** => By default a table column can hold NULL Values.

The NOT NULL Constraint enforces a column to NOT accept NULL values.

```
                    CREATE TABLE Persons(
                        ID int NOT NULL,
         LastName varchar(255) NOT NULL,
         FastName varchar(255) NOT NULL,
         Age int
                    );
```

22> **UNIQUE** => Ensures that all values in a column are different.

```
CREATE TABLE Persons(
  ID int NOT NULL UNIQUE,
  LastName varchar(255) NOT NULL,
  FastName varchar(255),
  Age int
);
```

23> **CHECK** => It is used to limit the value range that can be placed in a column.

If you define a CHECK Constraint on a column it will allow only certain values for this column.

```
CREATE TABLE Persons(
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FastName varchar(255),
  Age int,
CHECK(Age>=18)
);
```

drop table persons; /*To delete the complete table*/

```
use arpandb;
------ NOT NULL --- UNIQUE ---- CHECK ------
create table persons
(
        id int not null unique,
    personname varchar(255),
    location varchar(255),
    age int,
    check(age>=18)
);

insert into persons(id,personname,location,age)
values(2,"arpan","XYZ",20);

select * from persons;
```

24>The **PRIMARY KEY (UNIQUE + NOT NULL)** constraint uniquely identifies each record in a table.

Primary Keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key.

```
                        CREATE TABLE Persons(
            ID int NOT NULL,

            LatName varchar(255) NOT NULL,

            FirstName varchar(255),

            Age int,

            PRIMARY KEY(ID)

                        );
```

25> The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

```
                        CREATE TABLE Orders(

             OrderID int NOT NULL,

             OrderNumber int NOT NULL,

             PersonID int,

             PRIMARY KEY(ORDERID),

             FOREIGN KEY(PersonID) REFERENCE
Persons(PersonID)

                        );
```

1. Primary key - A table can have only one primary key (unique + not null)

2. Foreign key - to make relationship between two or more than two tables

3. One table contain primary key and other table contain foreign key.

4. A common column in both the tables (common column should have same datatype)

5. primary key (parent table) + foreign key (child table)

```
use arpandb;

create table orderss

(

order_id int primary key,

ordernumber int,

location varchar(255)

);


insert into orderss(order_id,ordernumber,location)

values(10,123456,"XYZ");

insert into orderss(order_id,ordernumber,location)

values(20,4567896,"MNO");
```

```
insert into orderss(order_id,ordernumber,location)
values(30,5643278,"ABC");

SELECT * FROM ORDERSS;

-------------------------------------------


create table personss
(
personid int,
personname varchar(255),
order_id int,
foreign key(order_id) references orderss(order_id)
);

insert into personss(personid,personname,order_id)
values(1,"arpan",10);
insert into personss(personid,personname,order_id)
values(8,"satyam",40); -- ERROR ---

SELECT * FROM PERSONSS;
----------------------------------------------------------------------------
```

26> **JOIN OPERATION** => A JOIN clause is used to combine rows of two or more tables based on a related column between them

- **INNER JOIN / JOIN** -> Return rows that have matching values in both tables.
- **LEFT JOIN** -> Return all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN** -> Return all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN** -> Returns rows when there is a match in one of the tables.

```
create table personsss(
p_id int,
firstname varchar(255),
lastname varchar(255),
Address varchar(255),
city varchar(255)
);

insert into personsss(p_id,firstname,lastname,Address,city)
values(1,"Arpan","Panigrahi","Street 15","Sander");
insert into personsss(p_id,firstname,lastname,Address,city)
values(2,"Lokesh","Yadav","Green 68","Sander");
```

```
insert into personsss(p_id,firstname,lastname,Address,city)
values(3,"Suyash","Singh","Street 10","Stave");

select * from personsss;



create table ordersss
(
o_id int,
order_no int,
p_id int
);

insert into ordersss(o_id,order_no,p_id) values(1,23452,3);
insert into ordersss(o_id,order_no,p_id) values(1,12345,3);
insert into ordersss(o_id,order_no,p_id) values(1,87654,1);
insert into ordersss(o_id,order_no,p_id) values(1,903657,1);
insert into ordersss(o_id,order_no,p_id) values(1,982353,10);

select * from personsss;
select * from ordersss;
```

27>**SYNTAX**: JOIN / INNER JOIN => SELECT table1.column1, table2.column2... FROM table1 INNER JOIN table2

ON

table1.common_field = table2.common_field;

SELECT ordersss.p_id, personsss.firstname, personsss.lastname,ordersss.order_no

FROM personsss INNER JOIN ordersss ON personsss.p_id = ordersss.p_id;

select personsss.firstname,personsss.lastname,ordersss.order_no,ordersss.p_id

from personsss left join ordersss on personsss.p_id = ordersss.p_id;

select personsss.firstname,personsss.lastname,ordersss.order_no,ordersss.p_id

from personsss right join ordersss on personsss.p_id = ordersss.p_id;

28> **INCREMENT** => Auto Increment allows a unique number to be generated when a new record is inserted into a table.

```
    CREATE TABLE Persons (
        Personid int NOT NULL AUTO_INCREMENT,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        PRIMARY KEY(Personid)
         );
```

```
create table hello
(
p_id int primary key auto_increment,
personname varchar(255),
salary int
);

insert into hello(personname,salary)values("arpan",2000);
insert into hello(personname,salary)values("saytam",3000);

select * from hello;
```

29> **TOP Clause** is used to specify the number of records to return.

```
    Select * from persons limit 5;
```

```
select * from personsss;
```

```
select * from personsss limit 2; -- It will show only top 2 records from the table --
```

30> **SQL COMMAND** =>

- **DDL** -> Data Definition Language
- **DQL** -> Data Query Language
- **DML** -> Data Manipulation Language
- **DCL** -> Data Control Language

- DDL -> CREATE, DROP, ALTER, TRUNCATE.
- DML -> INSERT, DELETE, UPDATE.
- DQL -> SELECT
- TCL -> COMMIT, SAVEPOINT, ROLLBACK
- DCL -> GRANT, REVOKE

DDL – DR. CAT

DML – UDI

DQL – SELECT

TCL – CSR

DCL – RG