What is Data?
Data is a collection of distinct small of piece of information. It can be of any type like Text, Number, Special Characters etc.

What is Database & its advantages?
Database is a collection of organized data that can be stored and retrieved using different database modelling and design approaches.

Ex: Google Chrome's Search Engine can be taken as 'Database' because it stores the dearch keywords that are frequently used by the users all over the workd.

Purpose:
1. We are converting raw data ---> Information--->Knowledge to understand & analaysis ---> Putting the analysis we did into actions to improve our business.
2. Database will help in reducing the data redundancy and inconsistencies in data.
3. Database will allow multiple users to access the data stored in database from multiple locations.
4.Database will avoid Security issues on your data and you will be having backup option to restore your organizational data.
5.Database will provide the feature of data integrity.
6. Having database will reduce Application development time.

Applications that are using Databases in real time:
1.Railway Ticket Booking Systems
2.Banking Applications
3.Educational System etc

What are all types of Databases & which one is most used?
1. Centralized Databases
2. Distributed Databases
3. Relational Database
4.NoSQL Database
5. Cloud Database etc

Relational Database model is the most used type of Database in business world.

What is DBMS & What can we do with it?
DBMS stands for Database Management System that will act as interface between user & database to make a communication Bridge.
DBMS is nothing but a software tool which is responsible for creation, updation, Retrival and management of database that we stored.

For Example:Google Chrome ---> DBMS

What is RDBMS? & Properties of RDBMS?
 RDBMS stands for Relational Database Management system. RDMS is a relational model which is nothing but the data is stored in the
 form of relation/table.

 Properties of RDBMS:
 ACID Stands for Atomicity - Consistency - Isolation-Durability.

 Atomicity: Atomcity property will give us information about the Transaction/data operation success or failure status. It follows "all or nothing" strategy.
 Consistency: If we perform any data operation, its before value and after value should be stored. For ex: The account balance before and after the transaction should be accurately preserved.
 3.Isolation: There can be multiple users who are accessing data at a sametime from the databse but the database/data should remain isolated.

For example:Multiple Transactions that occur at the same time should not be affecting other's transactions.
4.Durability:It ensures that once the operation on data is completed and you committed your changes, the data changes should remain permanent.

What is Table & few terminologies in RDBMS?
A table is an organized collection of data that can be stored in the form of rows and columns.

Terminologies:
Rows - are also known as Tuple/record.
Column - Are also known as Attribute/field.
Degree of the table - Number/Count of Columns in a table is referred to as 'Degree' of the table.
Cardinality - Number/Count of rows in a table is referred to as 'Cardinality' of the table.

Properties of a Relation/Table:
1.Each relation/Table should have unique column to identify each and every row.
2.Table does not contain any duplicate rows.
3.All the columns in a table should be having atomic value --->Each and Every single column should contain only single value in it.

Adv & Disadv of DBMS:
1.Data Sharing among multiple users.
2.Large data can be stored for longer period.
3.Data integrity will be maintained.
4.Data redundancy can be controlled.
5.It will give us backup options to restore our database whenevr server got failed and also we can avoid security issues.
6. Easy to access and fetch the data we need.
7. Query capabilities without keeping much complexity.

Disadvantage:
Size & cost.

What is Database Normalization:
Normalization is the process of organizing data in the database. It can be used to break the large table/relation into smaller ones to keep the data in well-structured format. It is used to minimize the data redundancy,data duplication, data inconsistencies and eliminate the three types of anamolies like Insertion, Updation and deletion.
Normalization can link the smaller tables in further stages of development by allowing us to create the relationship between them.

Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
Updatation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normalization:
1.1NF
2.2NF
3.3NF
4.BCNF

5.4NF
6.5NF

First Normal Form(1NF):
1. Whenever a table has atomic value in it then that table is said to be in 1NF.
2.Each and every single cell in column/attribute of a table must contain only single value

Second Normal Form(2NF):
1. In the 2NF, table must be in 1NF.
2.A relation is said to be in 2NF when the Non-primary key attribute /columns are fully dependent on the primary attribute.

Third Normal Form:
1.In the 3NF, that table/relation should follow 2NF.
2.A relation is said to be in 3NF when it is not having any transitive partial dependency.

A=B
B=C
A=C ---> C is indirectly dependent on A.

SQL:

SQL stands for Structured Query Language. We can write queries to fetch the data whatever we want from DB.

Processing of SQL query:
1.We will write SQL Commands/SQL Query.
2.There will be a SQL language processor which can convert SQL query(Human Language) to bytes.
3.DBMS Engine will carry the bytes information request to Physical Database that we stored.
4.DB will get hit by the request and gives the required data to us.

SQL Commands:
1.CREATE Commands
2.UPDATE Command
3.DELETE Command
4.SELECT Command
5.DROP Command
6.INSERT Command

Data Types:
1.Numeric Data Type:

Numeric Data Type includes Integer(INT), Decimal(DEC or DECIMAL dependents on the DBMS tool that we use), float (FLOAT) etc. Numeric Data type will hold
numeric values/figures like Amount, Profit, Emp_ID, Emp_Age etc.

2.String Data Type:
String Data types will hold the Alphabets, texts, variable text with special characters etc. String data types are like CHAR, VARCHAR,LONGTEXT etc.

3.Date & Time Data Type:
This data type will represent date, time, datetimestamp, and year.

SQL Constraints:

Constraints are nothing but the conditions/retrictions/limitations/rule/instruction that we should apply on top of columns in a table.

1.NOT NULL:It will make sure that column cannot have NULL(Blank) values.
2.DEFAULT:If no value is specified for a column then default value should be assigned.
3.UNIQUE:To keep the values unique in a column, UNIQUE can be used.
4.CHECK:This constraint will make sure that all the values on a column of a table are satisfying the given condition/criteria that we had set while creating table.
5.Primary Key:To specify the column of a table as a unique tthis constraint can be used. In other words, Primary key is used to identify each and every row in a table uniquely.
6.Foreign Key:It refers to a common column that is existing in one or more tables.
7.AUTO Increment: This constraint will help the column to automatically assign the value even though there is no data in the columns

CREATE Database:

Using this command we can create database.

SYNTAX:

```
CREATE DATABASE databasename;
```

Ex:

```
CREATE DATABASE Employee;
```

**********How to connect to particular database which we have created?

```
USE databasename;
```

CREATE Table:

Using this command we can create tables in database.

SYNTAX:

```
CREATE Table tablename (Col1name data type, Col2name data type,....,ColNname Data type);
```

Ex:

```
CREATE Table Employee (Emp_ID INT(2), Emp_name CHAR(50),Emp_mailid VARCHAR(20), DOB DATE);
```

To check the above table select * from Employee;

INSERT INTO:
This command will help us to load or insert the data into the table.

SYNTAX:

```
INSERT INTO tablename(Col1name,Col2name,...,ColNname)
VALUES (Col1Value,Col2value,...,Colnvalue);
```

Ex:

```
 INSERT INTO Emp_PersonalInfo (Emp_ID, Name, DOB)
VALUES ('2', 'Nikhitha','1995-03-21'),
('3','Pooja','1995-08-22'),
('4','Akshay','1996-09-23');
```

DROP Table:
Using this command we can delete entire table or database.

SYNTAX:For table
DROP table tablename;

Ex:
DROP TABLE Emp_PersonalInfo;

SYNTAX:For Database
DROP database Databasename;

Ex:
DROP database nikhitha_data;

ALTER Table:

Alter statement can be used to add new column, modify existing columnname/data type in an existing table.

SYNTAX for Adding New Column in existing table:

ALTER TABLE tablename ADD new_Columnname data type;

Ex:
ALTER TABLE Emp_PersonalInfo ADD Emp_Email VARCHAR(30);

SYNTAX for Modifying Existing Column datatype in an existing table:
ALTER TABLE tablename MODIFY COLUMN Columnname data type;

Ex: ALTER TABLE Emp_PersonalInfo MODIFY COLUMN Emp_ID bigint;

SYNTAX for Modifying Existing Columnname in an existing table:
ALTER TABLE tablename RENAME COLUMN Columnname_old to Columnname_new;
Ex:

ALTER TABLE Emp_PersonalInfo RENAME COLUMN Name to Emp_Name;

Syntax for dropping/deleting an existing column in a table:

ALTER Table tablename DROP COLUMN Columnname;

Ex: ALTER Table emp_personalinfo DROP COLUMN DOB;

DESCRIBE Command:

Describe command will show the table structure along with data type information of a column.

SYNTAX:

DESCRIBE tablename;

***** Group of tables or databases is called as 'Schema'.

TRUNCATE COMMAND:

Truncate will delete all the data in a table but the structure if the table remain same.

SYNTAX:

Truncate tablename;

Types of SQL Commands:

1.Data Definition Langugae(DDL): These DDL Statements deals with database schema & their descriyptions of how the data should reside in the database.

----> Create: To create a database & its objects like table, view, index, stored procedures, functions & triggers.
----> ALTER: Alters the structure of existing database/its table.
----> DROP: Delete objects from database.
---->TRUNCATE: Removes all records from the table, including all spaces allocated for records are removed.

2. Data Manipulation Langugae(DML): It will deals with data manipulation and it is used to store, modify,retrieve, delete and update data in a database.

----> INSERT: We cvan insert data into the table.
----> SELECT: It will retrievethe data from database.
----> UPDATE: Updates existing data with a new value in a table.
---->DELETE: Delete records from table.

3.Data Control Langugae(DCL): It deals with righ permissions and other controls that a user a can have on the database system.
---->GRANT: Allows users with privileges to access the database.
---->REVOKE: Withdraw users access privileges given by the Admin.

4.Transaction Control Langugae(TCL): It deals with a transaction/operation within a database.

---->COMMIT: Commits a transaction.
----> ROLLBACK: Undo the transaction/changes in case of any errors.
----> SET TRANSACTION: It will specify some characteristics of the transaction.

DELETE Command:
This will delete the records from a table.

SYNTAX: DELETE from tablename where condition(Optional)

Ex: DELETE from emp_personalinfo where Name = '1'

UPDATE Command:

It will help us in updating any existing table in terms of data value in a column.

SYNTAX:

UPDATE tablename SET Columnname1 = NewValue1, ColumnName2 = New Value2, ... where Condition;

Ex: UPDATE payment_methods SET Name = 'Card Payment'
where payment_method_id = '1';

SELECT Command:

It is used to retrieve the data from tables or database.

Ex:

Select * from invoices; ----> To see entire data

Select invoice_id, Invoice_total from invoices; ----> To see specific column data.

SQL Clause:

1.Where
2.From
3.Order by
4.Group by
5.Having

1. Where Clause:

Where clause can be used with SELECT, INSERT, UPDATE and DELETE Statement in order to filter the data set results.

SYNTAX:

Select * from tablename where condition;

Ex: Select * from Clients where City = 'San Francisco';
Ex:Select invoice_id, Invoice_total,Payment_total from invoices where Client_id = '5';

2.From:
From Clause is used to select records from/it refers to a tablename that we need.

SYNTAX:
Select * from tablename;

3.Order By:
ORDER BY clause is used to sort the records in either ascending or descending or Alphabetical order.

SYNTAX:

Select * from tablename where condition Order by Columnname ASC|DESC;

Ex:Select * from Clients order by Name DESC;

4.GROUP BY:

This is used to collect the data from multiple records with same value and group the result by one or more columns.
Group by statement is always used with aggregate funjctions like SUM(),MIN(),MAX(),AVG(),COUNT() etc.

SYNTAX:

Select * from tablename where condition Order by Columnname ASC|DESC Group by Columnname;

Ex:
What is the total invoice amount that has to be paid by each client?
Select Client_id, Sum(Invoice_total) from invoices Group by Client_id;

How many no of bills that are pending from each client?
select Count(invoice_id),Client_id from invoices Group by Client_id

What is the avg salary by Office_id?
Select Office_id, AVG(Salary) from employees group by office_id

What is the minimum payment done by each client?
select Client_id,Min(Payment_total) from invoices Group by Client_id

What is the maximum payment done by each client?
select Client_id,Max(Payment_total) from invoices Group by Client_id

5.HAVING:
Having clause should always be used after Group by cluase & it will be useful to perform filtering operation on aggregate functions.

SYNTAX:
Select * from tablename where condition Order by Columnname ASC|DESC Group by Columnname having Condition;

Ex:
Find out the employees who worked more than
their own avg working hours '6' on which date?
Select Emp_Name, Working_hours, Avg(Working_hours), Working_Date
from Working_hours group by Emp_Name
Having Avg(Working_Hours) > '6'

Find out the employees who worked more than
their own avg working hours on which date?
Select Emp_Name, Working_hours, Avg(Working_hours), Working_Date
from Working_hours group by Emp_Name
Having Working_Hours > Avg(Working_hours)

---->MySQL Distinct:

Distinct will give you the unique records of data.

SYNTAX:
Select distinct(Colname) from tablename;

Ex:
Select distinct Emp_ID, Emp_Name from Working_hours
Select distinct Emp_ID, Emp_Name, Working_hours from Working_hours

MySQL Aggregate functions:

1.MySQL Count:
Count function will give you the total no. of rows/records value in a table

SYNTAX: Select Count(Columnname or Expression) from tablename;

Ex:
Select Count(Emp_ID) from Working_hours;
Select Count(distinct(Emp_ID)) from Working_hours;

2.MySQL Sum();
3.MySQL MIN();
4.MySQL MAX():
5.MySQL AVG():
6.MySQL LIMIT:
LIMIT function will restrict the no. of rows to be displayed from the result set.

SYNTAX:

Select * from Tablename LIMIT Number;

Ex:

Select * from Working_hours LIMIT 5;

Who are all the Top 5 highest paid employees?

Select * from employees Order by Salary Desc Limit 5

7.MySQL Group_Concat():

Group_Concat function will perform aggregate function on string columns like First_Name, Last_Name etc.

SYNTAX:

Select Columnname, Group_Concat(Expression or Columnname) from tablename GROUP BY Columnname;

Ex:
Select Employee_ID, GROUP_CONCAT(First_Name," ", Last_Name) from Employees Group By Employee_ID;

MySQL Operator:

1.AND:

Whenever we are using AND conditions in the query all the conditions should get satisfied and then only the result will be displayed.

Ex:
Who are all the employees working from office_id '2' as a 'Assistant Professor'
Select * from employees where Office_id = '2' AND Job_Title = 'Assistant Professor'

2.OR:
Whenever we are using OR conditions in the query atleast one of the conditions should get satisfied and then only the result will be displayed.

Ex:
Who are all the employees working from office_id '2' or '11'
Select * from employees where office_id = '2' or office_id = '11'

3.AND OR:

Who are all the employees working from office_id '3' and earning Salary Greater than or equal to 100000

```
Select * from employees where Office_id = '3' and Salary > '99999' or Salary = '100000'
```

4.IN:

Who are all the employees working from office_id '1000', '3','4'

```
Select * from employees where Office_id in ('1000', '3', '4')
```

5.NOT IN:

Who are all the employees not working from office_id '2', '3','4'

```
Select * from employees where Office_id not in ('2', '3','4')
```

6.NOT EQUAL:

Who are all the the employees who is not working as account executive?
Who are all the employees not working as 'Account Executive'?

```
Select * from employees where Job_Title !=  'Account Executive'
Select * from employees where not Job_Title =  'Account Executive'
```

7. MySQL NULL:

Who are all the employees that does not have surname?

```
Select * from employees where last_name is null
```

Who are all the employees that does not have manager?

```
Select * from employees where reports_to is null
```

8.MySQL NOT NULL:

Who are all the employees that does have manager?

```
Select * from employees where reports_to is not null
```

9.MYSQL BETWEEN:

Who are all the employees have the salary range between 60000 to 100000?

```
Select * from employees where Salary between '60000' and '100000'
```

10.MySQL LIKE:

LIKE operator can be used along with where clause in order to get specified pattern of result.

% ---> will allow one or more characters
_ ---> will allow only one single character at a time

SYNTAX:

```
Select * from tablename where columnname LIKE 'Pattern';
```

Ex:

```
Select First_name from employees where First_Name LIKE 'N%'
Select First_name from employees where First_Name LIKE 'N_'
Select First_name from employees where First_Name LIKE 'N____'
Select First_name from employees where First_Name LIKE '%N%'
Select First_name from employees where First_Name LIKE '_E%'
Select First_name from employees where First_Name LIKE '%E';
```

MySQL Alias:

Alias is nothing but the temporary name that we can give to column or table.
Generic ex:Nickname of a person.
Alias name will exist for the duration of that query only.

SYNTAX:

```
Select Column_name As Alias_Name from Tablename;

Select * from tablename As Alias_Name;
```

Ex:

```
Select employee_id As Unique_identifier, First_Name,Last_Name from employees
```

MySQL JOINS:

1.INNER JOIN:

Inner Join retrieves the data/records that have matching value in both the tables.

SYNTAX:

```
Select * from tableA Inner JOin TableB on TableA.colname = TableB.Colname
```

Ex:

```
Select * from Clients Join Invoices on Clients.Client_id = Invoices.Client_id

Select C.Client_id, C.Name,
I.Invoice_id, Invoice_total, Payment_total, payment_id, Amount, Payment_method
from Clients C Inner Join Invoices I on C.Client_id = I.Client_id
             Inner Join Payments P on I.Invoice_id = P.Invoice_id

Select * from Invoices I Join Payments P ON
(I.Client_id = P.Client_id and I.Invoice_id = P.Invoice_id)
```

2.Right Join or Right Outer Join:

It will fetch all the records from right table and only matching records from left table.

SYNTAX:

```
Select * from tableA Right Join Table B on TableA.Columnname = TableB.Colname
```

Ex:

```
Select * from Clients C  Right Join Invoices I on C.Client_id = I.Client_id
```

3.Left Join or Left Outer Join:

It will fetch all the records from left table and only matching records from right table.

SYNTAX:

```
Select * from tableA Left Join Table B on TableA.Columnname = TableB.Colname
```

Ex:

```
Select * from Clients Left Join Invoices on Clients.Client_id = Invoices.Client_id
```

4.CROSS JOIN:

It will return all possible combinations of records from both the tables.

SYNTAX:

Select * from tableA CROSS JOIN tableB

or

Select * from tableA,tableB

Ex:

Select * from Clients CROSS Join Invoices;

5.NATURAL JOIN:

It will be helpful to join two or more tables without giving ON condition.

SYNTAX:

Select * from tablename A NATURAL JOIN TABLE B;

Ex:

Select * from Clients NATURAL JOIn Invoices

Select * from Invoices NATURAL JOIn payments

6.SELF JOIN:

It will be useful to join the itself.

SYNTAX:

Select * from tablename A, tablename B where condition;

Ex:
select * from employees A, employees B
where A.Reports_to = B.Employee_id

7.UNION:
The UNION Operator is used to combine the result set of two or more SELECT statements.

---> Every SELECT Statement whithin UNION must have the same number of columns.
---> The columns must also have similar data types.
---> The columns in every SELECT statement must also be in the same order.

a)UNION: The UNION operator selects only distinct values.

SYNTAX:

SELECT column_name(S) from table1
UNION
SELECT column_name(S) from table2;

Ex:
```
Select Customer_id from customers
UNION
Select Customer_id from orders
```

Ex:2
```
Select Customer_id, Order_id, Order_Date, 'Delivered' As OrderStatus from Orders
where Order_Date < '2018-06-08'
UNION
Select Customer_id, Order_id, Order_Date, 'UnDelivered' As OrderStatus from Orders
where Order_Date > '2018-06-08'
```

Ex without UNION:
```
Select Customer_id, Order_id, Order_Date,
(case when Order_Date < '2018-06-08' then 'Delivered' else 'Undelivered' end) As
OrderStatus
from Orders
```

b)UNION ALL:The UNION ALL operator selects duplicate values also.

SYNTAX:

```
SELECT column_name(S) from table1
UNION ALL
SELECT column_name(S) from table2;
```

Ex:
```
Select Customer_id from customers
UNION ALL
Select Customer_id from orders;
```

How to take backup of a table or copy the table or Clone the two tables?
SYNTAX:
```
CREATE Table NewTableName
Select * from OldTableName;
```

Ex:
```
CREATE TABLE Customers_New
Select * from Customers;
```

VIEWS:
--->In SQL, a view is a virtual table based on the result set of an SQL statement.
--->A view contains rows and columns just like a real table. The fields in a view are from one or more real tables in the database.

For CREATING VIEW - SYNTAX:
```
CREATE VIEW View_name As
Select Column1,Column2....
from tablenamewhere condition;
```

Ex:
```
CREATE VIEW DueAmountbyClient As
Select C.Client_id, C.Name, Sum(I.Invoice_total - I.Payment_total) As Balance from
Clients C
Right Join INVOICES I ON
C.Client_id = I.Client_id
GROUP BY C.Client_id, C.Name
```

And then use Select * from DueAmountbyClient to check whether the view is created or not.

For Modifying View - SYNTAX:

```
ALTER VIEW DueAmountbyClient As
Select C.Client_id, C.Name, I.Invoice_id, Sum(I.Invoice_total - I.Payment_total) As
Balance from Clients C
Right Join INVOICES I ON
C.Client_id = I.Client_id
GROUP BY C.Client_id, C.Name, I.Invoice_id
```

How to see View definition:

```
SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA. VIEWS WHERE TABLE_SCHEMA =
'DATABASENAME' AND
 TABLE_NAME = 'VIEWNAME'

 SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA. VIEWS WHERE TABLE_SCHEMA =
'SQL_INVOICING' AND
 TABLE_NAME = 'DueAmountbyClient'
```

SUBQUERIES:

A subquery is a query which is nested into another SQL query and embedded with SELECT, INSERT,UPDATE or DELETE statement along with various operators.
We can also nest the subquery with another subquery. A subquery is also known as Inner query and the query that contains subquery is known as outer query.
The inner query will be executed first and it gives the result to the outer query and then outer query execution will be performed.
Any SQL database allows us to use sub query qnywhere, but it must be closed within parenthesis.

SYNTAX:

```
Select Columnames from tablename where Columnname OPERATOR (Select Columnname from
Tablename where condition);
```

Ex:

Find out the employees who are earning more than avg salary of all employees?

```
 Select Employee_id,First_Name, Last_Name,Salary
 from employees where Salary > (Select Avg(Salary) from employees)
```

Find out the employees who are earning Less salary than max salary among all employees?

```
   Select Employee_id,First_Name, Last_Name,Salary
  from employees where Salary < (Select Max(Salary) from employees)
```

PROCEDURES:
A procedure (Often called as Stored Procedures) is a collection of pre-compiled SQL statements/queries stored inside the database.
A procedure always contain a name, parameter lists and SQL statements.

SYNTAX:

```
DELIMITER &&
CREATE PROCEDURE procedure_name [[IN|OUT\INOUT]] paramater_name datatype,
[parameter datatype]]
BEGIN
Declaration_section
Executable_section
END &&;

Ex:PROCEDURE WITHOUT PARAMATER:
DELIMITER &&
CREATE PROCEDURE get_clients()
BEGIN
Select * from Clients;
END &&;

CALL `sql_invoicing`.`get_clients`();

CALL get_clients()

Ex:PROCEDURE WITH PARAMETER:
DELIMITER &&
CREATE PROCEDURE get_clients_by_state(State_Input CHAR(4))
BEGIN
Select * from Clients C where C.state = State_Input;
END &&;

CALL `sql_invoicing`.`get_clients_by_state`('NY');

Ex:DELIMITER &&
CREATE PROCEDURE get_employees_by_Officeid_reportsto(reports_to INT, office_id INT)
BEGIN
Select * from employees E where E.reports_to = reports_to and E.office_id =
office_id;
END &&;

Ex:
DELIMITER &&
CREATE PROCEDURE get_employees_by_Officeid_empid(employee_id INT)
BEGIN
Select * from employees E where E.employee_id = employee_id;
END &&;


WINDOWS FUNCTIONS:

Windows functions are nothing but aggregate functions and ranking functions that
can be applied over a set of rows. OVER clause is used with windows functions to
define that particular window.

--->Partitions will divide the data into set of rows.
---> Order by will be used to arrange that set of rows coming after partition in a
particular order.

SYNTAX:

Select Column1,
Window_function(Column2)
OVER({Partition by Col1} {Order by Col3}) As Alias_name from tablename;
```

Types of Windows Functions:
1. Aggregate Functions:WF

Various aggregate functions such as SUM(),COUNT(),AVG(), MAX(),MIN() applied over a particular window(set of rows) are called aggregate window functions.

Ex:Find out the avg sal of employees for each department and order employees within a department by age?

```
Select Name, Age, Department,Salary,
Avg(Salary)
OVER(Partition by Department) As Avg_Salary from windowfunc
```

Ex:
```
Select Name, Age, Department,Salary,
Avg(Salary)
OVER(Partition by Department) As Avg_Salary from windowfunc order by Age
```

Ex:
```
Select Emp_ID, Emp_Name, Working_Date, Working_hours,
Avg(Working_hours)
OVER(Partition by Emp_ID Order by Emp_ID ASC) As Avg_WH from working_hours
```

2. Ranking Functions: RANK(), DENSE_RANK(), ROW_NUMBER()

RANK() –
As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.

DENSE_RANK() –
It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

ROW_NUMBER() –
It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

Ex:Calculate row no, rank, dense rank of employees in employee table according to salary within each department.

```
Select row_number() OVER(Partition by Department Order by Salary Desc) As Sno,
Name,Department, Salary, Age,
Rank() Over(Partition by Department Order by Salary Desc) As Emp_Rank,
Dense_Rank() Over(Partition by Department Order by Salary Desc) As Emp_DenseRank
from windowfunc
```

3.Analytical Windows Functions:

Lead and Lag are the most important and used analytical windows functions. Other than this NTILE, FIRST_VALUE,LAST_VALUE etc.

Lag():
The LAG() function allows access to a value stored in a different row above the current row. The row above may be adjacent or some number of rows above, as sorted by a specified column or set of columns.

Let’s look its syntax:

```
LAG(expression [,offset[,default_value]]) OVER(ORDER BY columns)
```

Ex:

```
SELECT Quarter, sale_value,
LAG(sale_value) OVER(ORDER BY Quarter) as previous_sale_value,
LAG(sale_value) OVER(ORDER BY Quarter) - Sale_value As diff
FROM Annual_sale;
```

Ex:2

```
SELECT
      train_id,
      station,
      Train_time as station_time,
      Lag(Train_time) OVER (PARTITION BY train_id ORDER BY Train_time)
            AS time_to_next_station,
        Time(Lag(Train_time) OVER (PARTITION BY train_id ORDER BY Train_time) -
Train_time) As Diff
FROM train_time;
```

LEAD():
LEAD() is similar to LAG(). Whereas LAG() accesses a value stored in a row above, LEAD() accesses a value stored in a row below.

The syntax of LEAD() is just like that of LAG():

```
LEAD(expression [,offset[,default_value]]) OVER(ORDER BY columns)
```

Ex:

```
SELECT
      train_id,
      station,
      Train_time as station_time,
      lead(Train_time) OVER (PARTITION BY train_id ORDER BY Train_time)
            AS time_to_next_station,
        Time(lead(Train_time) OVER (PARTITION BY train_id ORDER BY Train_time) -
Train_time) As Diff
FROM train_time;
```

How to check the keywords available in Database library?

```
SELECT * FROM mysql.help_keyword;
```

MOST Used MySQL Functions or Keywords:

MySQL String Functions
Function Description
ASCII - Returns the ASCII value for the specific character
CHAR_LENGTH Returns - the length of a string (in characters)
CHARACTER_LENGTH - Returns the length of a string (in characters)
CONCAT - Adds two or more expressions together
CONCAT_WS - Adds two or more expressions together with a separator

FIELD Returns the index position of a value in a list of values
FIND_IN_SET Returns the position of a string within a list of strings
FORMAT Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
INSERT Inserts a string within a string at the specified position and for a certain number of characters
INSTR Returns the position of the first occurrence of a string in another string
LCASE Converts a string to lower-case
LEFT Extracts a number of characters from a string (starting from left)
LENGTH Returns the length of a string (in bytes)
LOCATE Returns the position of the first occurrence of a substring in a string
LOWER Converts a string to lower-case
LPAD Left-pads a string with another string, to a certain length
LTRIM Removes leading spaces from a string
MID Extracts a substring from a string (starting at any position)
POSITION Returns the position of the first occurrence of a substring in a string
REPEAT Repeats a string as many times as specified
REPLACE Replaces all occurrences of a substring within a string, with a new substring
REVERSE Reverses a string and returns the result
RIGHT Extracts a number of characters from a string (starting from right)
RPAD Right-pads a string with another string, to a certain length
RTRIM Removes trailing spaces from a string
SPACE Returns a string of the specified number of space characters
STRCMP Compares two strings
SUBSTR Extracts a substring from a string (starting at any position)
SUBSTRING Extracts a substring from a string (starting at any position)
SUBSTRING_INDEX Returns a substring of a string before a specified number of delimiter occurs
TRIM Removes leading and trailing spaces from a string
UCASE Converts a string to upper-case
UPPER Converts a string to upper-case

MySQL Numeric Functions
Function Description
ABS Returns the absolute value of a number
ACOS Returns the arc cosine of a number
ASIN Returns the arc sine of a number
ATAN Returns the arc tangent of one or two numbers
ATAN2 Returns the arc tangent of two numbers
AVG Returns the average value of an expression
CEIL Returns the smallest integer value that is >= to a number
CEILING Returns the smallest integer value that is >= to a number
COS Returns the cosine of a number
COT Returns the cotangent of a number
COUNT Returns the number of records returned by a select query
DEGREES Converts a value in radians to degrees
DIV Used for integer division
EXP Returns e raised to the power of a specified number
FLOOR Returns the largest integer value that is <= to a number
GREATEST Returns the greatest value of the list of arguments
LEAST Returns the smallest value of the list of arguments
LN Returns the natural logarithm of a number
LOG Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10 Returns the natural logarithm of a number to base 10
LOG2 Returns the natural logarithm of a number to base 2
MAX Returns the maximum value in a set of values
MIN Returns the minimum value in a set of values

MOD Returns the remainder of a number divided by another number
PI Returns the value of PI
POW Returns the value of a number raised to the power of another number
POWER Returns the value of a number raised to the power of another number
RADIANS Converts a degree value into radians
RAND Returns a random number
ROUND Rounds a number to a specified number of decimal places
SIGN Returns the sign of a number
SIN Returns the sine of a number
SQRT Returns the square root of a number
SUM Calculates the sum of a set of values
TAN Returns the tangent of a number
TRUNCATE Truncates a number to the specified number of decimal places

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the date
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format

TIME_TO_SEC Converts a time value into seconds
TIMEDIFF Returns the difference between two time/datetime expressions
TIMESTAMP Returns a datetime value based on a date or datetime value
TO_DAYS Returns the number of days between a date and date "0000-00-00"
WEEK Returns the week number for a given date
WEEKDAY Returns the weekday number for a given date
WEEKOFYEAR Returns the week number for a given date
YEAR Returns the year part for a given date
YEARWEEK Returns the year and week number for a given date

MySQL Advanced Functions
Function Description
BIN Returns a binary representation of a number
BINARY Converts a value to a binary string
CASE Goes through conditions and return a value when the first condition is met
CAST Converts a value (of any type) into a specified datatype
COALESCE Returns the first non-null value in a list
CONNECTION_ID Returns the unique connection ID for the current connection
CONV Converts a number from one numeric base system to another
CONVERT Converts a value into the specified datatype or character set
CURRENT_USER Returns the user name and host name for the MySQL account that the server used to authenticate the current client
DATABASE Returns the name of the current database
IF Returns a value if a condition is TRUE, or another value if a condition is FALSE
IFNULL Return a specified value if the expression is NULL, otherwise return the expression
ISNULL Returns 1 or 0 depending on whether an expression is NULL
LAST_INSERT_ID Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
NULLIF Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned
SESSION_USER Returns the current MySQL user name and host name
SYSTEM_USER Returns the current MySQL user name and host name
USER Returns the current MySQL user name and host name
VERSION Returns the current version of the MySQL database