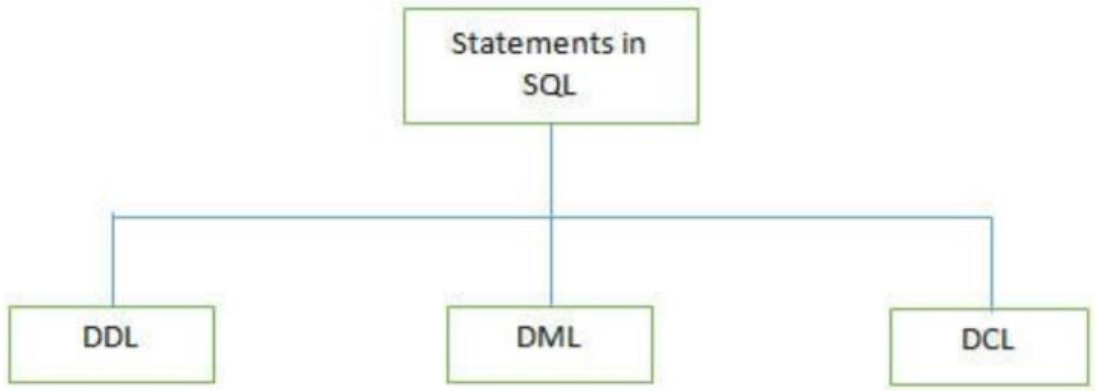
SQL Interview Questions

1. What is SQL?

Structured Query Language is a database tool which is used to create and access database to support software application.

2. What are different types of statements supported by SQL?



There are 3 types of SQL statements

1) DDL (Data Definition Language): It is used to define the database structure such as tables. It includes three statements such as Create, Alter, and Drop.

Some of the DDL Commands are listed below

- **CREATE**: It is used for creating the table.

```
CREATE TABLE&amp;amp;amp;nbsp;table_name
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
```

- **ALTER:** The ALTER table is used for modifying the existing table object in the database.

ALTER TABLE table_name
ADD column_name datatype

OR

ALTER TABLE table_name

DROP COLUMN column_name

2) DML (Data Manipulation Language): These statements are used to manipulate the data in records. Commonly used DML statements are Insert, Update, and Delete.

The Select statement is used as partial DML statement that is used to select all or relevant records in the table.

3) DCL (Data Control Language): These statements are used to set privileges such as Grant and Revoke database access permission to the specific user.

3. What is DBMS?

A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database.

DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

4. What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

Example: SQL Server.

5. Why do we use SQL constraints? Which constraints we can use while creating database in SQL?

Constraints are used to set the rules for all records in the table. If any constraints get violated then it can abort the action that caused it.

Constraints are defined while creating the database itself with CREATE TABLE statement or even after the table is created once with ALTER

TABLE statement.

There are 5 major constraints are used in SQL, such as

NOT NULL: That indicates that the column must have some value and cannot be left null

UNIQUE: This constraint is used to ensure that each row and column has unique value and no value is being repeated in any other row or

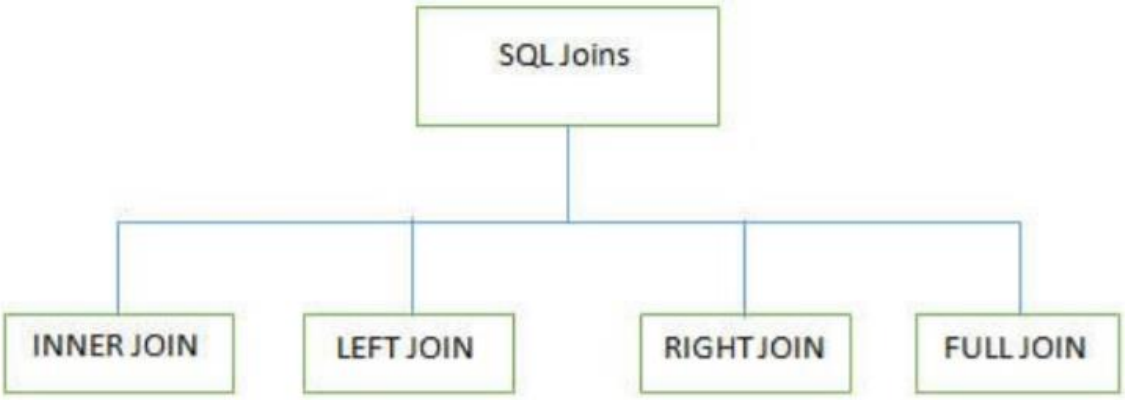
column

PRIMARY KEY: This constraint is used in association with NOT NULL and UNIQUE constraints such as on one or the combination of more than one column to identify the particular record with a unique identity.

FOREIGN KEY: It is used to ensure the referential integrity of data in the table and also matches the value in one table with another using Primary Key

CHECK: It is used to ensure whether the value in columns fulfills the specified condition

6. What are different JOINS used in SQL?



There are 4 major types of joins made to use while working on multiple tables in SQL databases

- **INNER JOIN:** It is also known as SIMPLE JOIN which returns all rows from BOTH tables when it has at least one column matched

Syntax: *SELECT column_name(s)*
FROM table_name1
INNER JOIN table_name2
ON column_name1=column_name2;

Example

In this example, we have a table *Employee* with the following data

Emp_Id	Last_Name	First_Name	Job_Role
E0011	Verma	Akhil	Administration
E0012	Samson	Nikita	Asst. Manager
E0013	Jordan	Nil	In charge
E0014	Smith	Joe	Technician

The second Table is *joining*

Emp_Id	Last_Name	First_Name	Joining_Date
E0012	Verma	Akhil	2016/04/18
E0013	Samson	Nikita	2016/04/19
E0014	Jordan	Nil	2016/05/01

Enter the following SQL statement

```
SELECT Employee.Emp_id, Joining.Joining_Date
  FROM Employee
  INNER JOIN Joining
  ON Employee.Emp_id = Joining.Emp_id
  ORDER BY Employee.Emp_id;
```

There will be 4 records selected. These are the results that you should see

Emp_Id	Joining_Date
E0012	2016/04/18
E0013	2016/04/19
E0014	2016/05/01

Employee and *orders* tables where there is a matching *customer_id* value in both the *Employee* and *orders* tables

- **LEFT JOIN (LEFT OUTER JOIN):** This join returns all rows from a LEFT table and its matched rows from a RIGHT table.

Syntax: *SELECT column_name(s)*
FROM table_name1
LEFT JOIN table_name2
ON column_name1=column_name2;

Example

In this example, we have a table *Employee* with the following data:

Emp_Id	Last_Name	First_Name	Job_Role
E0011	Verma	Akhil	Administration
E0012	Samson	Nikita	Asst. Manager
E0013	Jordan	Nil	In charge
E0014	Smith	Joe	Technician

Second Table is *joining*

Emp_Id	Last_Name	First_Name	Joining_Date
E0012	Verma	Akhil	2016/04/18
E0013	Samson	Nikita	2016/04/19
E0014	Jordan	Nil	2016/05/01
NULL	NULL	NULL	2016/03/01

Enter the following SQL statement

```
SELECT Employee.Emp_id, Joining.Joining_Date
FROM Employee
LEFT OUTER JOIN Joining
ON Employee.Emp_id = Joining.Emp_id
ORDER BY Employee.Emp_id;
```

There will be 4 records selected. These are the results that you should see:

Emp_Id	Joining_Date
NULL	*NULL*
E0012	2016/04/18
E0013	2016/04/19
E0014	2016/05/01

- **RIGHT JOIN (RIGHT OUTER JOIN):** This joins returns all rows from the RIGHT table and its matched rows from a LEFT table.

Syntax: *SELECT column_name(s)*
FROM table_name1
RIGHT JOIN table_name2
ON column_name1=column_name2;

Example

In this example, we have a table *Employee* with the following data

Emp_Id	Last_Name	First_Name	Job_Role
E0011	Verma	Akhil	Administration
E0012	Samson	Nikita	Asst. Manager
E0013	Jordan	Nil	In charge
E0014	Smith	Joe	Technician

The second Table is *joining*

Emp_Id	Last_Name	First_Name	Joining_Date
E0012	Verma	Akhil	2016/04/18
E0013	Samson	Nikita	2016/04/19
E0014	Jordan	Nil	2016/05/01
NULL	NULL	NULL	2016/03/01

Enter the following SQL statement

```
SELECT Employee.Emp_id, Joining.Joining_Date
FROM Employee
LEFT OUTER JOIN Joining
ON Employee.Emp_id = Joining.Emp_id
ORDER BY Employee.Emp_id;
```

There will be 4 records selected. These are the results that you should see

Emp_Id	Joining_Date
NULL	2016/03/01
E0012	2016/04/18
E0013	2016/04/19
E0014	2016/05/01

- **FULL JOIN (FULL OUTER JOIN):** This joins returns all when there is a match either in the RIGHT table or in the LEFT table.

Syntax: *SELECT column_name(s)*
FROM table_name1
FULL OUTER JOIN table_name2
ON column_name1=column_name2;

Example

In this example, we have a table *Employee* with the following data:

Emp_Id	Last_Name	First_Name	Job_Role
E0011	Verma	Akhil	Administration
E0012	Samson	Nikita	Asst. Manager
E0013	Jordan	Nil	In charge
E0014	Smith	Joe	Technician

Second Table is *joining*

Emp_Id	Last_Name	First_Name	Joining_Date
E0012	Verma	Akhil	2016/04/18
E0013	Samson	Nikita	2016/04/19
E0014	Jordan	Nil	2016/05/01
NULL	NULL	NULL	2016/03/01

Enter the following SQL statement:

```
1  SELECT Employee.Emp_id, Joining.Joining_Date
2  FROM Employee
3  FULL OUTER JOIN Joining
4  ON Employee.Emp_id = Joining.Emp_id
5  ORDER BY Employee.Emp_id;
```

There will be 8 records selected. These are the results that you should see

Emp_Id	Joining_Date
NULL	*NULL*
E0012	2016/04/18
E0013	2016/04/19
E0014	2016/05/01
NULL	2016/03/01
E0012	2016/04/18
E0013	2016/04/19
E0014	2016/05/01

7. What is normalization?

Normalization is the process of minimizing redundancy and dependency by organizing fields and table of a database. The main aim of Normalization is to add, delete or modify field that can be made in a single table.

8. What are all the different normalizations?

The normal forms can be divided into 4 forms, and they are explained below -.

1. First Normal Form (1NF): This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

2. Second Normal Form (2NF): Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

3.Third Normal Form (3NF): This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.

4. Fourth Normal Form (4NF): Meeting all the requirements of third normal form and it should not have multivalued dependencies.

9. How many Aggregate Functions are available there in SQL?

SQL Aggregate Functions calculates values from multiple columns in a table and returns a single value.

There are 7 aggregate functions we use in SQL

AVG(): Returns the average value from specified columns

COUNT(): Returns number of table rows

MAX(): Returns largest value among the records

MIN(): Returns smallest value among the records

SUM(): Returns the sum of specified column values

FIRST(): Returns the first value

LAST(): Returns Last value

10. What is an Index? What are all the different types of indexes?

An index is performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and it will be faster to retrieve data. This indexing does not allow the field to have duplicate values if the column is unique indexed. Unique index can be applied automatically when primary key is defined.

1. Clustered Index: This type of index reorders the physical order of the table and search based on the key values. Each table can have only one clustered index.

2. Non-Clustered Index: Non-Clustered Index does not alter the physical order of the table and maintains logical order of data. Each table can have 999 non clustered indexes.

11. What is SQL Injection?

SQL Injection is a type of database attack technique where malicious SQL statements are inserted into an entry field of database such that once it is executed the database is opened for an attacker. This technique is usually used for attacking Data-Driven Applications to have an access to sensitive data and perform administrative tasks on databases.

For Example: SELECT column_name(s) FROM table_name WHERE condition;

12. What is the difference between “Primary Key” and “Unique Key”?

1.We can have only one Primary Key in a table whereas we can have more than one Unique Key in a table.

2. The Primary Key cannot have a NULL value whereas a Unique Key may have only one null value.

3. By default, a Primary Key is a Clustered Index whereas by default, a Unique Key is a unique non-clustered index.

4. A Primary Key supports an Auto Increment value whereas a Unique Key doesn't support an Auto Increment value.

13. What is ISNULL() operator?

ISNULL function is used to check whether value given is NULL or not NULL in sql server. This function also provides to replace a value with the NULL.

14. What are Magic Tables in SQL Server?

Insert and Delete tables are created when the trigger is fired for any DML command. Those tables are called Magic Tables in SQL Server. These magic tables are used inside the triggers for data transaction.

15. What is a Cursor?

A database Cursor is a control which enables traversal over the rows or records in the table. This can be viewed as a pointer to one row in a set of rows. Cursor is very much useful for traversing such as retrieval, addition and removal of database records.

16. How to change Database name in SQL server?

Use the following code:

Supported in SQL server 2000 and 2005

Exec sp_renamedb “test”, “test1”

Supported in SQL Server 2005 and later version

ALTER Database “test1” Modify Name = “test”

17. What is referential integrity?

Referential integrity refers to the consistency that must be maintained between primary and foreign keys, i.e. every foreign key value must have a corresponding primary key value.

18. How exceptions can be handled in SQL Server Programming?

Exceptions are handled using TRY—-CATCH constructs and it is handles by writing scripts inside the TRY block and error handling in the CATCH block.

19. What is an execution plan? When would you use it? How would you view the execution plan?

An execution plan is basically a road map that graphically or textually shows the data retrieval methods chosen by the SQL Server query optimizer for a stored procedure or ad- hoc query and is a very useful

tool for a developer to understand the performance characteristics of a query or stored procedure since the plan is the one that SQL Server will place in its cache and use to execute the stored procedure or query. From within Query Analyzer is an option called "Show Execution Plan" (located on the Query drop-down menu). If this option is turned on it will display query execution plan in separate window when query is ran again.

20. How to implement one-to-one, one-to-many and many-to-many relationships while designing tables?

One-to-One relationship can be implemented as a single table and rarely as two tables with primary and foreign key relationships. One-to-Many relationships are implemented by splitting the data into two tables with primary key and foreign key relationships. Many-to-Many relationships are implemented using a junction table with the keys from both the tables forming the composite primary key of the junction table.

21. How to select UNIQUE records from a table using a SQL Query?

Consider below EMPLOYEE table as the source data

```
CREATE TABLE EMPLOYEE (
        EMPLOYEE_ID NUMBER(6,0),
        NAME VARCHAR2(20),
        SALARY NUMBER(8,2)
);

INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den',11000);


SELECT * FROM EMPLOYEE;
```

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
100	Jennifer	4400
101	Michael	13000
101	Michael	13000
101	Michael	13000
102	Pat	6000
102	Pat	6000
103	Den	11000

Using GROUP BY Function

Query:

```
SELECT EMPLOYEE_ID,
       NAME,
       SALARY
FROM EMPLOYEE
GROUP BY EMPLOYEE_ID, NAME, SALARY;
```

Result:

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
101	Michael	13000
102	Pat	6000
103	Den	11000

22. How to delete DUPLICATE records from a table using a SQL Query?

Consider the same EMPLOYEE table as source discussed in previous question

Using ROWID and ROW_NUMBER Analytic Function

STEP-1: Using ROW_NUMBER Analytic function, assign row numbers to each unique set of records. Select ROWID of the rows along with the source columns

Query:

```
SELECT ROWID,
       EMPLOYEE_ID,
       NAME,SALARY,
       ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS ROW_NUMBER
FROM EMPLOYEE;
```

Result:

ROWID	EMPLOYEE_ID	NAME	SALARY	ROW_NUMBER
AAASnBAAEAAACrWAAA	100	Jennifer	4400	1
AAASnBAAEAAACrWAAB	100	Jennifer	4400	2
AAASnBAAEAAACrWAAC	101	Michael	13000	1
AAASnBAAEAAACrWAAD	101	Michael	13000	2
AAASnBAAEAAACrWAAE	101	Michael	13000	3
AAASnBAAEAAACrWAAF	102	Pat	6000	1
AAASnBAAEAAACrWAAG	102	Pat	6000	2
AAASnBAAEAAACrWAAH	103	Den	11000	1

STEP-2: Select ROWID of records with ROW_NUMBER > 1

Query:

```
SELECT ROWID FROM(
      SELECT ROWID,
             EMPLOYEE_ID,
             NAME,
             SALARY,
             ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
      FROM EMPLOYEE)
WHERE ROW_NUMBER > 1;
```

Result:

ROWID
AAASnBAAEAAACrWAAB
AAASnBAAEAAACrWAAD
AAASnBAAEAAACrWAAE
AAASnBAAEAAACrWAAG

STEP-3: Delete the records from the source table using the ROWID values fetched in previous step

Query:

```
DELETE FROM EMP WHERE ROWID IN (
    SELECT ROWID FROM(
        SELECT ROWID,
        ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
    FROM EMPLOYEE)
WHERE ROW_NUMBER > 1);
```

Result:

The table EMPLOYEE will have below records after deleting the duplicates

ROWID	EMPLOYEE_ID	NAME	SALARY
AAASnBAAEAAACrWAAA	100	Jennifer	4400
AAASnBAAEAAACrWAAC	101	Michael	13000
AAASnBAAEAAACrWAAF	102	Pat	6000
AAASnBAAEAAACrWAAH	103	Den	11000

23.How to read TOP 5 records from a table using a SQL query?

Consider below table DEPARTMENTS as the source data

```
CREATE TABLE Departments(
   Department_ID number,
   Department_Name varchar(50)
);

INSERT INTO DEPARTMENTS VALUES('10','Administration');
INSERT INTO DEPARTMENTS VALUES('20','Marketing');
INSERT INTO DEPARTMENTS VALUES('30','Purchasing');
INSERT INTO DEPARTMENTS VALUES('40','Human Resources');
INSERT INTO DEPARTMENTS VALUES('50','Shipping');
INSERT INTO DEPARTMENTS VALUES('60','IT');
INSERT INTO DEPARTMENTS VALUES('70','Public Relations');
INSERT INTO DEPARTMENTS VALUES('80','Sales');

SELECT * FROM Departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales

ROWNUM is a “Pseudocolumn” that assigns a number to each row returned by a query indicating the order in which Oracle selects the row from a table. The first row selected has a ROWNUM of 1, the second has 2, and so on.

Query:

```
SELECT * FROM Departments WHERE ROWNUM <= 5;
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping

24. How to read LAST 5 records from a table using a SQL query?

Consider the same DEPARTMENTS table as source discussed in previous question.

In order to select the last 5 records we need to find (**count of total number of records – 5**) which gives the count of records from first to last but 5 records.Using the MINUS function we can compare all records from DEPARTMENTS table with records from first to last but 5 from DEPARTMENTS table which give the last 5 records of the table as result.

MINUS operator is used to return all rows in the first SELECT statement that are not present in the second SELECT statement.

Query:

```
SELECT * FROM Departments

MINUS

SELECT * FROM Departments WHERE ROWNUM <= (SELECT COUNT(*)-5 FROM Departments);
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales

25. How to find the employee with second MAX Salary using a SQL query?

Consider below EMPLOYEES table as the source data

```
CREATE TABLE Employees(
        EMPLOYEE_ID NUMBER(6,0),
        NAME VARCHAR2(20 BYTE),
        SALARY NUMBER(8,2)
        );

INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den', 11000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(104,'Alexander',3100);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(105,'Shelli',2900);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(106,'Sigal',2800);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(107,'Guy',2600);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(108,'Karen',2500);

SELECT * FROM Employees;
```

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
101	Michael	13000
102	Pat	6000
103	Den	11000
104	Alexander	3100
105	Shelli	2900
106	Sigel	2800
107	Guy	2600
108	Karen	2500

Without using SQL Analytic Functions

In order to find the second MAX salary, employee record with MAX salary needs to be eliminated. It can be achieved by using below SQL query.

Query:

```
SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
SELECT MAX(salary) AS salary FROM Employees);
```

Result:

SALARY
11000

The above query only gives the second MAX salary value. In order to fetch the entire employee record with second MAX salary we need to do a self-join on Employee table based on Salary value.

Query:

```
WITH
TEMP AS(
  SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
  SELECT MAX(salary) AS salary FROM Employees)
)
SELECT a.* FROM Employees a JOIN TEMP b on a.salary = b.salary
```

Result:

EMPLOYEE_ID	NAME	SALARY
103	Den	11000

26. How to find the employee with third MAX Salary using a SQL query without using Analytic Functions?

Consider the same EMPLOYEES table as source discussed in previous question

In order to find the third MAX salary, we need to eliminate the top 2 salary records. But we cannot use the same method we used for finding second MAX salary (not a best practice). Imagine if we have to find the fifth MAX salary. We should not be writing a query with four nested sub queries.

STEP-1:

The approach here is to first list all the records based on Salary in the descending order with MAX salary on top and MIN salary at bottom. Next, using ROWNUM select the top 2 records.

Query:

```
SELECT salary FROM(
SELECT salary FROM Employees ORDER BY salary DESC)
WHERE ROWNUM < 3;
```

Result:

Salary
13000
11000

STEP-2:

Next find the MAX salary from EMPLOYEE table which is not one of top two salary values fetched in the earlier step.

Query:

```
SELECT MAX(salary) as salary FROM Employees WHERE salary NOT IN (
    SELECT salary FROM(
    SELECT salary FROM Employees ORDER BY salary DESC)
    WHERE ROWNUM < 3
);
```

Result:

SALARY
6000

STEP-3:

In order to fetch the entire employee record with third MAX salary we need to do a self-join on Employee table based on Salary value.

Query:

```
WITH
TEMP AS(
  SELECT MAX(salary) as salary FROM Employees WHERE salary NOT IN (
      SELECT salary FROM(
      SELECT salary FROM Employees ORDER BY salary DESC)
      WHERE ROWNUM < 3)
)
SELECT a.* FROM Employees a join TEMP b on a.salary = b.salary
```

Result:

EMPLOYEE_ID	NAME	SALARY
102	Pat	6000

27. What is Trigger?

Trigger allows us to execute a batch of SQL code when a table event occurs (Insert, update or delete command executed against a specific table)

28. What is CHECK Constraint?

A CHECK constraint is used to limit the values or type of data that can be stored in a column. They are used to enforce domain integrity.

29. What is Database White Box Testing?

Database White Box Testing involves Database Consistency and ACID properties Database triggers and logical views Decision Coverage, Condition Coverage, and Statement Coverage Database Tables, Data Model, and Database Schema Referential integrity rules

30. What are Nested Triggers?

Triggers may implement data modification logic by using INSERT, UPDATE, and DELETE statement. These triggers that contain data

modification logic and find other triggers for data modification are called Nested Triggers.

31. Assume you have the below tables on sessions that users have, and a users table. Write a query to get the active user count of daily cohorts.

sessions

column_name	type
user_id	integer
session_id	integer
date	datetime

users + Add a view

column_name	type
user_id	integer
email	string
date	datetime

By definition, daily cohorts are active users from a particular day. First, we can use a subquery to get the sessions of new users by day using an inner join with users. This is to filter for only active users by a particular join date for the cohort. Then we can get a distinct count to return the active user count:

```
WITH new_users_by_date AS (
    SELECT sessions.*
    FROM sessions
  JOIN users on
    sessions.user_id = users.user_id
    sessions.date = users.date
)
SELECT date, COUNT(DISTINCT user_id) as active_user_count
    FROM new_users_by_date
    GROUP BY 1 ORDER BY 1 ASC
```

32. Assume you are given the below table on transactions from users for purchases. Write a query to get the list of customers where their earliest purchase was at least $50.

user_transactions

Aa column_name	type
transaction_id	integer
product_id	integer
user_id	integer
spend	float
transaction_date	datetime

Although we could use a self join on transaction_date = MIN(transaction_date) for each user, we can also use the RANK() window function to get the ordering of purchase by customer, and then use that subquery to filter on customers where the first purchase (rank one) is at least 50 dollars. Note that this requires the subquery to include spend as well

```
WITH purchase_rank AS (
    SELECT user_id, spend,
        RANK() OVER
            (PARTITION BY user_id ORDER BY transaction_date ASC) as rank
        FROM user_transactions u
)

SELECT
    user_id
FROM
    purchase_rank
WHERE rank = 1 AND spend >= 50.00
```

33. Assume you are given the below table on transactions from users. Write a query to get the number of users and total products bought per latest transaction date where each user is bucketed into their latest transaction date.

First, we need to get the latest transaction date for each user, along with the number of products they have purchased. This can be done in a subquery where we GROUP BY user_id and take a COUNT(DISTINCT product_id) to get the number of products they have purchased, and a MAX(transaction_date) to get the latest transaction date (while casting to a date). Then, using this subquery, we can simply do an aggregation by the transaction date column in the previous subquery, while doing a COUNT() on the number of users, and a SUM() on the number of products:

```
WITH latest_date AS (
    SELECT user_id,
        COUNT(DISTINCT product_id) AS num_products,
        MAX(transaction_date::DATE) AS curr_date
    FROM user_transactions
    GROUP BY )

SELECT curr_date,
    COUNT(user_id) AS num_users,
    SUM(num_products) AS total_products
FROM
latest_date
GROUP BY 1
```

34. Assume you are given the below tables on users and their time spent on sending and opening Snaps. Write a query to get the breakdown for each age breakdown of the percentage of time spent on sending versus opening snaps.

activities

Aa column_name	type
activity_id	integer
user_id	integer
type	string ('send', 'open')
time_spent	float
activity_date	datetime

age_breakdown

Aa column_name	type
user_id	integer
age_bucket	string

We can get the breakdown of total time spent on each activity by each user by filtering out for the activity_type and taking the sum of time spent. In doing this, we want to do an outer join with the age bucket to get the total time by age bucket for both activity types. This results in the below two subqueries. Then, we can use these two subqueries to sum them by joining on the appropriate age bucket and take the proportion for send time and the proportion for open time per age bucket:

```
WITH send_timespent AS (
    SELECT age_breakdown.age_bucket, SUM(activities.time_spent) AS send_timespent
    FROM age_breakdown
  LEFT JOIN activities on age_breakdown.user_id = activities.user_id
    WHERE activities.type = 'send'
    GROUP BY 1
),
open_timespent AS (
    SELECT age_breakdown.age_bucket, SUM(activities.time_spent) AS open_timespent
    FROM age_breakdown
  LEFT JOIN activities on age_breakdown.user_id = activities.user_id
    WHERE activities.type = 'open'
    GROUP BY 1
),

SELECT a.age_bucket,
    s.send_timespent / (s.send_timespent + o.open_timespent) AS pct_send,
    o.open_timespent / (s.send_timespent + o.open_timespent) AS pct_open,
FROM age_breakdown a
LEFT JOIN send_timespent s ON a.age_bucket = s.age_bucket
LEFT JOIN open_timespent o ON a.age_bucket = o.age_bucket
GROUP BY 1
```

35. Assume you are given the below table on reviews from users. Define a top-rated place as a business whose reviews only consist of 4 or 5 stars. Write a query to get the number and percentage of businesses that are top-rated places.

First, we need to get the places where the reviews are all 4 or 5 stars. We can do this using a HAVING clause, instead of a WHERE clause since the reviews need to all be 4 stars or above. For the HAVING condition, we can use a CASE statement that filters for 4 or 5 stars and then take a SUM over them. This can then be compared with the total row count of the particular business_id reviews to ensure that the count of top reviews matches with the total review count. With the relevant businesses, we can then do an outer join with the original table on business_id to get a COUNT of distinct business_id matches, and then the percentage by comparing the COUNT from the top places with the overall COUNT of business_id:

```
WITH top_places AS (
    SELECT business_id
    FROM user_transactions
    GROUP BY 1
    HAVING
        SUM(CASE WHEN rating >= 4 THEN 1 ELSE 0 END) = COUNT(*)
    )

SELECT
    COUNT(DISTINCT t.business_id) AS top_places,
    COUNT(DISTINCT t.business_id)/COUNT(DISTINCT r.business_id) AS top_places_pct
FROM reviews r
LEFT JOIN top_places t
    ON r.business_id = t.business_id
```

36. What is an SQL View?

A view is a virtual table whose contents are obtained from an existing table or tables, called base tables. The retrieval happens through an SQL statement, incorporated into the view. So, you can think of a view object as a view into the base table. The view itself does not contain any real data; the data is electronically stored in the base table. The view simply shows the data contained in the base table.

37.

Given the following tables:

```
sql> SELECT * FROM runners;
+----+--------------+
| id | name         |
+----+--------------+
|  1 | John Doe     |
|  2 | Jane Doe     |
|  3 | Alice Jones  |
|  4 | Bobby Louis  |
|  5 | Lisa Romero  |
+----+--------------+
```

```
sql> SELECT * FROM races;
+----+----------------+-----------+
| id | event          | winner_id |
+----+----------------+-----------+
|  1 | 100 meter dash |  2        |
|  2 | 500 meter dash |  3        |
|  3 | cross-country  |  2        |
|  4 | triathalon     |  NULL     |
+----+----------------+-----------+
```

What will be the result of the query below?

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)
```

Surprisingly, given the sample data provided, the result of this query will be an empty set. The reason for this is as follows: If the set being evaluated by the `SQL NOT IN` condition contains *any* values that are null, then the outer query here will return an empty set, even if there are many runner ids that match winner_ids in the `races` table.

Knowing this, a query that avoids this issue would be as follows:

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races WHERE winner_id IS NOT null)
```

Note, this is assuming the standard SQL behavior that you get without modifying the default `ANSI_NULLS` setting.

38.

Given two tables created and populated as follows:

```
CREATE TABLE dbo.envelope(id int, user_id int);
CREATE TABLE dbo.docs(idnum int, pageseq int, doctext varchar(100));

INSERT INTO dbo.envelope VALUES
  (1,1),
  (2,2),
  (3,3);

INSERT INTO dbo.docs(idnum,pageseq) VALUES
  (1,5),
  (2,6),
  (null,0);
```

What will the result be from the following query:

```
UPDATE docs SET doctext=pageseq FROM docs INNER JOIN envelope ON envelope.id=docs.idnum
WHERE EXISTS (
  SELECT 1 FROM dbo.docs
  WHERE id=envelope.id
);
```

The result of the query will be as follows:

```
idnum   pageseq   doctext
1       5         5
2       6         6
NULL    0         NULL
```

The `EXISTS` clause in the above query is a red herring. It will *always* be true since `ID` is *not* a member of `dbo.docs`. As such, it will refer to the `envelope` table comparing itself to itself!

The `idnum` value of `NULL` will not be set since the join of `NULL` will not return a result when attempting a match with any value of `envelope`.

39.

Assume a schema of `Emp ( Id, Name, DeptId ) , Dept ( Id, Name)`.
If there are 10 records in the `Emp` table and 5 records in the `Dept` table, how many rows will be displayed in the result of the following SQL query:

```
Select * From Emp, Dept
```

Explain your answer.

The query will result in 50 rows as a “cartesian product” or “cross join”, which is the default whenever the ‘where’ clause is omitted.

40.

STUDENT

Sid	name	login	age	gpa
53666	Kayne	A@cs	28	4.0
53655	Tupac	B@cs	26	3.5
53688	Bieber	C@cs	22	3.9

ENROLLED

Sid	cid	grade
53666	15-415	C
53688	15-721	A
53688	15-826	B
53655	15-415	C
53666	15-721	C

Which of the following is the correct outcome of the SQL query below?

```
Query: SELECT cid FROM ENROLLED WHERE grade = 'C'
```

A. Extract the course ids(cid) where student receive the grade C in the course

B. Extract the unique course ids(cid) where student receive the grade C in the course

C. Error

D. None of these

Solution: A

The query will extract the course ids where student receive the grade “C” in the course.

41. What is the correct outcome of the SQL query below?

```
Query: SELECT student.name, enrolled.grade FROM student, enrolled WHERE student.sid = enrolled.sid AN
D enrolled.cid = '15-415' AND enrolled.grade IN ('A', 'B')
```

The above query first joined the ENROLLED and STUDENT tables then it will evaluate the where condition and then it will return the name, grade of the students, those took 15-415 and got a grade ‘A’ or ‘B’ in the course. But for the given two tables it will give zero records in output.

42. Which of the following query will find all the unique students who have taken more than one course?

```
A. SELECT DISTINCT e1.sid FROM enrolled AS e1, enrolled AS e2 WHERE e1.sid != e2.sid AND e1.cid != e2.cid
B. SELECT DISTINCT e1.sid FROM enrolled AS e1, enrolled AS e2 WHERE e1.sid = e2.sid AND e1.cid = e2.cid
C. SELECT DISTINCT e1.sid FROM enrolled AS e1, enrolled AS e2 WHERE e1.sid != e2.sid AND e1.cid != e2.cid
D. SELECT DISTINCT e1.sid FROM enrolled AS e1, enrolled AS e2 WHERE e1.sid = e2.sid AND e1.cid != e2.cid
```

Solution: D

Option D would be a right option. This query will first apply self join on enrolled table and then it evaluate the condition e1.sid = e2.sid AND e1.cid != e2.cid.

43. What are the tuples additionally deleted to preserve reference integrity when the rows (2,4) are deleted from the below table. Suppose you are using ‘ON DELETE CASCADE’.

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

When (2,4) is deleted. Since C is a foreign key referring A with delete on cascade, all entries with value 2 in C must be deleted. So (5, 2) and (7, 2) are deleted. As a result of this 5 and 7 are deleted from A which causes (9, 5) to be deleted.

44. Suppose you have a table “Loan_Records”.

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

```
SELECT Count(*)  FROM  ( (SELECT Borrower, Bank_Manager FROM   Loan_Records) AS S
NATURAL JOIN (SELECT Bank_Manager, Loan_Amount FROM   Loan_Records) AS T );
```

What is the output of the following SQL query?

Temporary table S is given below

Borrower	Bank_Manager
Ramesh	Sunderajan
Suresh	Ramgopal
Mahesh	Sunderjan

Temporary table T is given below

Bank_Manager	Loan_Amount
Sunderajan	10000.00
Ramgopal	5000.00
Sunderjan	7000.00

If you apply natural join on both tables (S and T) and evaluate the condition on ‘Bank_Manager’. You will get the following intermediate table after apply

natural join

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000
Ramesh	Sunderajan	7000
Suresh	Ramgopal	5000
Mahesh	Sunderajan	10000
Mahesh	Sunderajan	7000

"Sunderjan" appears two times in Bank_Manager column, so their will be four entries with Bank_Manager as "Sunderjan". So count(*) will give the 5 output in outer query.

45. What will be the output of the below query?

```
Query: SELECT Company, AVG(Salary) FROM AV1 HAVING AVG(Salary) > 1200 GROUP BY Company WHERE Salary > 1000 ;
```

This query will give the error because 'WHERE' is always evaluated before 'GROUP BY' and 'Having' is always evaluated after 'GROUP BY'.

46. SQL Query to find the second highest salary of Employee

There are many ways to find the second highest salary of an Employee in SQL, you can either use SQL Join or Subquery to solve this problem. Here is an SQL query using Subquery:

```
SELECT MAX(Salary)
FROM Employee
WHERE Salary NOT IN (select MAX(Salary) from Employee );
```

47. SQL Query to find Max Salary from each department.

You can find the maximum salary for each department by grouping all records by DeptId and then using MAX() function to calculate the maximum salary in each group or each department.

```
SELECT DeptID, MAX(Salary)
FROM Employee
GROUP BY DeptID.
```

These questions become more interesting if the Interviewer will ask you to print the department name instead of the department id, in that case, you need to join the Employee table with Department using the foreign key DeptID, make sure you do LEFT or RIGHT OUTER JOIN to include departments without any employee as well.

```
SELECT DeptName, MAX(Salary)
FROM Employee e RIGHT JOIN Department d
ON e.DeptId = d.DeptID
GROUP BY DeptName;
```

In this query, we have used RIGHT OUTER JOIN because we need the name of the department from the Department table which is on the right side of the JOIN clause, even if there is no reference of dept_id on the Employee table.

48. Write SQL Query to display the current date?

SQL has built-in function called GetDate() which returns the current timestamp.

```
SELECT GetDate();
```

49. Write an SQL Query to print the name of the distinct employee whose DOB is between 01/01/1960 to 31/12/1975.

```
SELECT DISTINCT EmpName
FROM Employees
WHERE DOB BETWEEN '01/01/1960' AND '31/12/1975';
```

50. Write an SQL Query to find an employee whose salary is equal to or greater than 10000.

```
SELECT EmpName FROM  Employees WHERE  Salary>=10000;
```

51. Write SQL Query to find duplicate rows in a database? and then write SQL query to delete them?

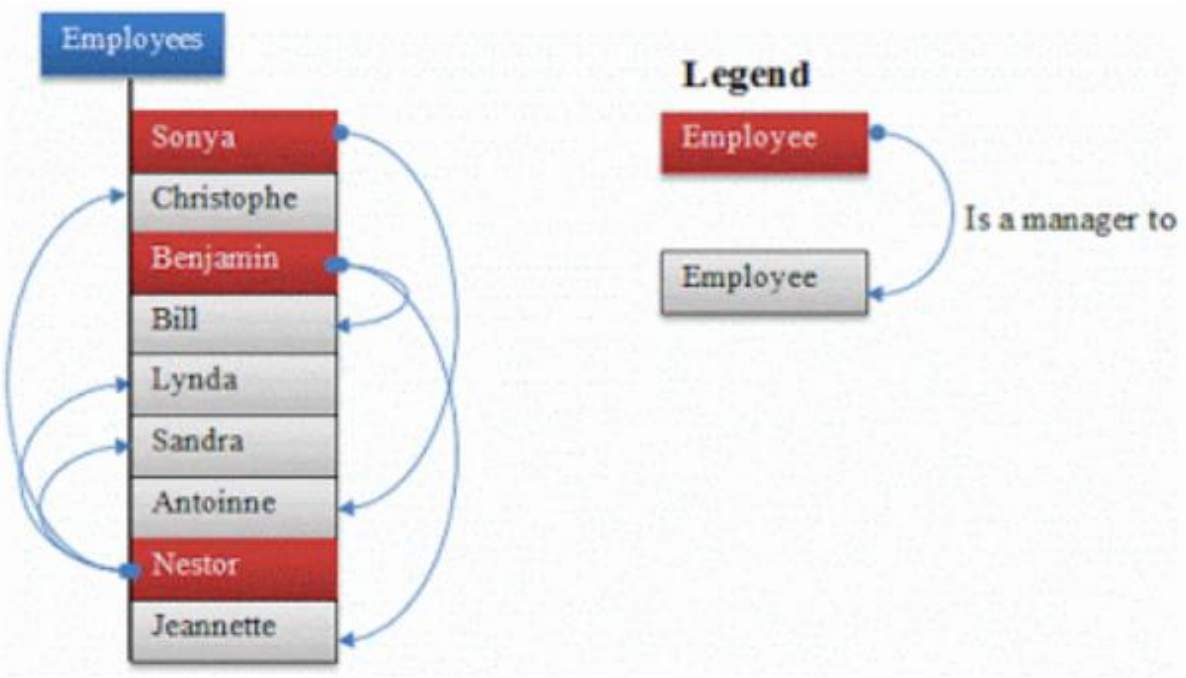
```
SELECT * FROM emp a
WHERE rowid = (SELECT MAX(rowid)
FROM EMP b
WHERE a.empno=b.empno)
```

to Delete:

```
DELETE FROM emp a
WHERE rowid != (SELECT MAX(rowid) FROM emp b WHERE a.empno=b.empno);
```

52. How do you find all employees who are also managers?

You have given a standard employee table with an additional column mgr_id, which contains the employee id of the manager.



You need to know about self-join to solve this problem. In Self Join, you can join two instances of the same table to find out additional details as shown below

```
SELECT e.name, m.name
FROM Employee e, Employee m
WHERE e.mgr_id = m.emp_id;
```

this will show employee name and manager name in two columns like

```
name  manager_name
John   David
```

One follow-up is to modify this query to include employees which don't have a manager. To solve that, instead of using the inner join, just use the left outer join, this will also include employees without managers.

53. The Trips table holds all taxi trips. Each trip has a unique Id, while Client_Id and Driver_Id are both foreign keys to the Users_Id at the Users table. Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').

Id	Client_Id	Driver_Id	City_Id	Status	Request_at
1	1	10	1	completed	2013-10-01
2	2	11	1	cancelled_by_driver	2013-10-01
3	3	12	6	completed	2013-10-01
4	4	13	6	cancelled_by_client	2013-10-01
5	1	10	1	completed	2013-10-02
6	2	11	6	completed	2013-10-02
7	3	12	6	completed	2013-10-02
8	2	12	12	completed	2013-10-03
9	3	10	12	completed	2013-10-03
10	4	13	12	cancelled_by_driver	2013-10-03

The `Users` table holds all users. Each user has an unique Users_Id, and Role is an ENUM type of ('client', 'driver', 'partner').

Users_Id	Banned	Role
1	No	client
2	Yes	client
3	No	client
4	No	client
10	No	driver
11	No	driver
12	No	driver
13	No	driver

Write a SQL query to find the cancellation rate of requests made by unbanned users between Oct 1, 2013 and Oct 3, 2013. For the above

tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.

```
+------------+-------------------+
|    Day     | Cancellation Rate |
+------------+-------------------+
| 2013-10-01 |       0.33        |
| 2013-10-02 |       0.00        |
| 2013-10-03 |       0.50        |
+------------+-------------------+
```

The solution looks like that:

```
select
        result.Request_at as "Day",
        round(sum(case when result.Status = 'completed' then 0 else 1 end)
/ count(*), 2) as "Cancellation Rate"
from (
    select
                Driver_Id,
                Status,
                Request_at
    from trips left join users on trips.client_id=users.users_id
    where users.banned = 'NO'
     ) result
        left join users on result.driver_id=users.users_id
where
        users.Banned ='NO'
        and result.Request_at between '2013-10-01' and '2013-10-03'
group by
        result.Request_at
```

54. Write a SQL query to find all duplicate emails in a table named Person.

Table: `Customers`.

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

For example, your query should return the following for the above table:

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

Solution:

Since all email are in lowercase we can simply groupby email and print those that have a count >1.

```
SELECT EMAIL
FROM PERSON
GROUP BY EMAIL
HAVING COUNT(*)>1
```

55. Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

```
+---------+------------------+------------------+
| Id(INT) | RecordDate(DATE) | Temperature(INT) |
+---------+------------------+------------------+
|       1 |       2015-01-01 |               10 |
|       2 |       2015-01-02 |               25 |
|       3 |       2015-01-03 |               20 |
|       4 |       2015-01-04 |               30 |
+---------+------------------+------------------+
```

For example, return the following Ids for the above `Weather` table:

```
+----+
| Id |
+----+
|  2 |
|  4 |
+----+
```

The solution is to join the table to itself when the dates differ by one day (`DATEDIFF()` function) and make sure that the temperature is higher than the previous date.

```
SELECT W1.ID
FROM WEATHER W1 INNER JOIN WEATHER W2 ON DATEDIFF(W1.RecordDate, W2.RecordDate) = 1
WHERE W1.Temperature > W2.Temperature
```

56. The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

The solution is to join again the table to itself as shown below:

```
SELECT E1.NAME AS EMPLOYEE
FROM EMPLOYEE E1 INNER JOIN EMPLOYEE E2 ON E1.MANAGERID=E2.ID
WHERE E1.SALARY>E2.SALARY
```

57. The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Jim   | 90000  | 1            |
| 3  | Henry | 80000  | 2            |
| 4  | Sam   | 60000  | 2            |
| 5  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| IT         | Jim      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

The `Department` table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter).

The solution looks like that:

```
select
    department.name as Department,
        result.Name as Employee,
        result.Salary as Salary
from
    (select
                e1.DepartmentId,
                e1.Name,e1.Salary
    from
                Employee e1 left join Employee e2 on e1.Salary <= e2.Salary
        and e1.DepartmentId = e2.DepartmentId
    group by
        e1.DepartmentId,
        e1.Salary,
        e1.Name
    having count(distinct e2.Salary) = 1)
   result join department on result.DepartmentId=department.id
```

58. X city opened a new cinema, many people would like to go to this cinema. The cinema also gives out a poster indicating the movies' ratings and descriptions.

Please write a SQL query to output movies with an odd numbered ID and a description that is not 'boring'. Order the result by rating.

For example, table `cinema`:

```
+---------+-----------+-------------+-----------+
|   id    | movie     |  description |  rating   |
+---------+-----------+-------------+-----------+
|   1     | War       |   great 3D   |   8.9     |
|   2     | Science   |   fiction    |   8.5     |
|   3     | irish     |   boring     |   6.2     |
|   4     | Ice song  |   Fantacy    |   8.6     |
|   5     | House card|   Interesting|   9.1     |
+---------+-----------+-------------+-----------+
```

The solution looks like that:

```
SELECT *
FROM CINEMA
WHERE ID%2=1 AND DESCRIPTION != 'BORING'
ORDER BY RATING DESC
```

59.

Write a SQL query to get the n^{th} highest salary from the `Employee` table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the n^{th} highest salary where $n = 2$ is 200. If there is no n^{th} highest salary, then the query should return null.

```
+------------------------+
| getNthHighestSalary(2) |
+------------------------+
| 200                    |
+------------------------+
```

The solution looks pretty similar with the one presented above for the Department Highest Salary problem:

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
      SELECT E1.SALARY
      FROM EMPLOYEE E1 LEFT JOIN EMPLOYEE E2 ON E1.SALARY <= E2.SALARY
      GROUP BY E1.SALARY
      HAVING COUNT(DISTINCT E2.SALARY) = N

  );
END
```

60. What is Identity?

Identity (or AutoNumber) is a column that automatically generates numeric values. A start and increment value can be set, but most DBA leave these at 1. A GUID column also generates numbers; the value of this cannot be controlled. Identity/GUID columns do not need to be indexed.

61. What are indexes?

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and the use of more storage space to maintain the extra copy of data. Data can be stored only in one order on disk. To support faster access according to different values, faster search like binary search for different values is desired, For this purpose, indexes are created on tables. These indexes need extra space on disk, but they allow faster search according to different frequently searched values.

62.What is a Composite Primary Key ?

A Composite primary key is a set of columns whose values uniquely identify every row in a table. What it means is that, table which contains composite primary key will be indexed based on columns specified in the primary key. This key will be referred in Foreign Key tables.

63.What are user defined datatypes and when you should go for them?

User defined datatypes let you extend the base SQL Server datatypes by providing a descriptive name, and format to the database. Take for example, in your database, there is a column called Flight_Num which appears in many tables. In all these tables it should be varchar(8). In this case you could create a user defined datatype called Flight_num_type of varchar(8) and use it across all your tables.

64.Explain different isolation levels

An isolation level determines the degree of isolation of data between concurrent transactions. The default SQL Server isolation level is Read Committed. Here are the other isolation levels (in the ascending order of isolation): Read Uncommitted, Read Committed, Repeatable Read, Serializable. See SQL Server books online for an explanation of the isolation levels. Be sure to read about SET TRANSACTION ISOLATION LEVEL, which lets you customize the isolation level at the connection level.

65. Explain Active/Active and Active/Passive cluster configurations

Hopefully you have experience setting up cluster servers. But if you don't, at least be familiar with the way clustering works and the two clustering configurations Active/Active and Active/Passive. SQL Server books online has enough information on this topic and there is a good white paper available on Microsoft site.

66. What is lock escalation?

Lock escalation is the process of converting a lot of low level locks (like row locks, page locks) into higher level locks (like table locks). Every lock is a memory structure too many locks would mean, more memory being occupied by locks. To prevent this from happening, SQL Server escalates the many fine-grain locks to fewer coarse-grain locks. Lock escalation threshold was definable in SQL Server 6.5, but from SQL Server 7.0 onwards it's dynamically managed by SQL Server.

67. What is a table called, if it has neither Cluster nor Non-cluster Index? What is it used for?

Unindexed table or Heap. Microsoft Press Books and Book on Line (BOL) refers it as Heap. A heap is a table that does not have a clustered index and, therefore, the pages are not linked by pointers. The IAM pages are the only structures that link the pages in a table together.

Unindexed tables are good for fast storing of data. Many times it is better to drop all indexes from table and then do bulk of inserts and to restore those indexes after that

68. What is a Scheduled Jobs or What is a Scheduled Tasks?

Scheduled tasks let user automate processes that run on regular or predictable cycles. User can schedule administrative tasks, such as cube processing, to run during times of slow business activity. User can also determine the order in which tasks run by creating job steps within a SQL Server Agent job. E.g. back up database, Update Stats of Tables. Job steps give user control over flow of execution. If one job fails, user can configure SQL Server Agent to continue to run the remaining tasks or to stop execution.

69. How to get @@ERROR and @@ROWCOUNT at the same time?

If @@Rowcount is checked after Error checking statement then it will have 0 as the value of @@Recordcount as it would have been reset. And if @@Recordcount is checked before the error-checking statement then @@Error would get reset. To get @@error and @@rowcount at the same time do both in same statement and store them in local variable. SELECT @RC = @@ROWCOUNT, @ER = @@ERROR

70. What is CHECK Constraint?

A CHECK constraint is used to limit the values that can be placed in a column. The check constraints are used to enforce domain integrity.

71. From the following table of user IDs, actions, and dates, write a query to return the publication and cancellation rate for each user.

users

user_id	action	date
1	start	1-1-20
1	cancel	1-2-20
2	start	1-3-20
2	publish	1-4-20
3	start	1-5-20
3	cancel	1-6-20
4	start	1-7-20

Desired output

user_id	publish_rate	cancel_rate
1	0.5	0.5
2	1.0	0.0
3	0.0	1.0

```
WITH users (user_id, action, date)
AS (VALUES
(1,'start', CAST('01-01-20' AS date)),
(1,'cancel', CAST('01-02-20' AS date)),
(2,'start', CAST('01-03-20' AS date)),
(2,'publish', CAST('01-04-20' AS date)),
(3,'start', CAST('01-05-20' AS date)),
(3,'cancel', CAST('01-06-20' AS date)),
(1,'start', CAST('01-07-20' AS date)),
(1,'publish', CAST('01-08-20' AS date))),

-- retrieve count of starts, cancels, and publishes for each user

t1 AS (
SELECT user_id,
sum(CASE WHEN action = 'start' THEN 1 ELSE 0 END) AS starts,
sum(CASE WHEN action = 'cancel' THEN 1 ELSE 0 END) AS cancels,
sum(CASE WHEN action = 'publish' THEN 1 ELSE 0 END) AS publishes
FROM users
GROUP BY 1
ORDER BY 1)
```

```
-- calculate publication, cancelation rate for each user by dividing
by number of starts, casting as float by multiplying by 1.0

SELECT user_id, 1.0*publishes/starts AS publish_rate,
1.0*cancels/starts AS cancel_rate
FROM t1
```

72. From the following table of transactions between two users, write a query to return the change in net worth for each user, ordered by decreasing net change.

transactions

sender	receiver	amount	transaction_date
5	2	10	2-12-20
1	3	15	2-13-20
2	1	20	2-13-20
2	3	25	2-14-20
3	1	20	2-15-20
3	2	15	2-15-20
1	4	5	2-16-20

Desired output

user	net_change
1	20
3	5
4	5
5	-10
2	-20

```
WITH transactions (sender, receiver, amount, transaction_date)
AS (VALUES
(5, 2, 10, CAST('2-12-20' AS date)),
(1, 3, 15, CAST('2-13-20' AS date)),
(2, 1, 20, CAST('2-13-20' AS date)),
(2, 3, 25, CAST('2-14-20' AS date)),
(3, 1, 20, CAST('2-15-20' AS date)),
(3, 2, 15, CAST('2-15-20' AS date)),
(1, 4, 5, CAST('2-16-20' AS date))),

-- sum amounts for each sender (debits) and receiver (credits)

debits AS (
SELECT sender, sum(amount) AS debited
FROM transactions
GROUP BY sender ),

credits AS (
SELECT receiver, sum(amount) AS credited
FROM transactions
GROUP BY receiver )
```

```
-- full (outer) join debits and credits tables on user id, taking net
change as difference between credits and debits, coercing nulls to
zeros with coalesce()


SELECT coalesce(sender, receiver) AS user,
coalesce(credited, 0) - coalesce(debited, 0) AS net_change
FROM debits d
FULL JOIN credits c
ON d.sender = c.receiver
ORDER BY 2 DESC
```

73.

From the following table containing a list of dates and items ordered, write a query to return the most frequent item ordered on each date. Return multiple items in the case of a tie.

items

date	item
1-1-20	apple
1-1-20	apple
1-1-20	pear
1-1-20	pear
1-2-20	pear
1-2-20	pear
1-2-20	pear
1-2-20	orange

Desired output

date	item
1-1-20	apple
1-1-20	pear
1-2-20	pear

```
WITH items (date, item)
AS (VALUES
(CAST('01-01-20' AS date),'apple'),
(CAST('01-01-20' AS date),'apple'),
(CAST('01-01-20' AS date),'pear'),
(CAST('01-01-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS date),'pear'),
(CAST('01-02-20' AS date),'orange')),

-- add an item count column to existing table, grouping by date and
item columns


t1 AS (
SELECT date, item, count(*) AS item_count
FROM items
GROUP BY 1, 2
ORDER BY 1),

-- add a rank column in descending order, partitioning by date

t2 AS (
SELECT *, rank() OVER (PARTITION by date ORDER BY item_count DESC) AS
date_rank
FROM t1)
```

```
-- return all dates and items where rank = 1


SELECT date, item
FROM t2
WHERE date_rank = 1
```

74.

From the following table of user actions, write a query to return for each user the time elapsed between the last action and the second-to-last action, in ascending order by user ID.

users

user_id	action	action_date
1	Start	2-12-20
1	Cancel	2-13-20
2	Start	2-11-20
2	Publish	2-14-20
3	Start	2-15-20
3	Cancel	2-15-20
4	Start	2-18-20
1	Publish	2-19-20

Desired output

user_id	days_elapsed
1	6
2	3
3	0
4	NULL

```
WITH users (user_id, action, action_date)
AS (VALUES
(1, 'start', CAST('2-12-20' AS date)),
(1, 'cancel', CAST('2-13-20' AS date)),
(2, 'start', CAST('2-11-20' AS date)),
(2, 'publish', CAST('2-14-20' AS date)),
(3, 'start', CAST('2-15-20' AS date)),
(3, 'cancel', CAST('2-15-20' AS date)),
(4, 'start', CAST('2-18-20' AS date)),
(1, 'publish', CAST('2-19-20' AS date))),

-- create a date rank column, partitioned by user ID, using the
row_number() window function

t1 AS (
SELECT *, row_number() OVER (PARTITION by user_id ORDER BY
action_date DESC) AS date_rank
FROM users ),

-- filter on date rank column to pull latest and next latest actions
from this table
```

```
latest AS (
SELECT *
FROM t1
WHERE date_rank = 1 ),

next_latest AS (
SELECT *
FROM t1
WHERE date_rank = 2 )

-- left join these two tables (everyone will have a latest action,
not everyone will have a second latest action), subtracting latest
from second latest to get time elapsed

SELECT l1.user_id,
       l1.action_date - l2.action_date AS days_elapsed
FROM latest l1
LEFT JOIN next_latest l2
ON l1.user_id = l2.user_id
ORDER BY 1
```

75.

A company defines its super users as those who have made at least two transactions. From the following table, write a query to return, for each user, the date when they become a super user, ordered by oldest super users first. Users who are not super users should also be present in the table.

users

user_id	product_id	transaction_date
1	101	2-12-20
2	105	2-13-20
1	111	2-14-20
3	121	2-15-20
1	101	2-16-20
2	105	2-17-20
4	101	2-16-20
3	105	2-15-20

Desired output

user_id	superuser_date
1	2-14-20
3	2-15-20
2	2-17-20
4	NULL

```
WITH users (user_id, product_id, transaction_date)
AS (VALUES
(1, 101, CAST('2-12-20' AS date)),
(2, 105, CAST('2-13-20' AS date)),
(1, 111, CAST('2-14-20' AS date)),
(3, 121, CAST('2-15-20' AS date)),
(1, 101, CAST('2-16-20' AS date)),
(2, 105, CAST('2-17-20' AS date)),
(4, 101, CAST('2-16-20' AS date)),
(3, 105, CAST('2-15-20' AS date))),


-- create a transaction number column using row_number() function,
partitioning by user ID


t1 AS (
SELECT *, row_number() OVER (PARTITION by user_id ORDER BY
transaction_date ASC) AS transaction_number
FROM users),


-- filter resulting table on transaction_number = 2


t2 AS (
SELECT user_id, transaction_date
FROM t1
WHERE transaction_number = 2 ),
```

```
-- left join super users onto full user table, order by date


t3 AS (
SELECT DISTINCT user_id
FROM users )


SELECT t3.user_id, transaction_date AS superuser_date
FROM t3
LEFT JOIN t2
ON t3.user_id = t2.user_id
ORDER BY 2
```

76. What is the SELECT statement?

SELECT operator in SQL is used to select data from a database. The data returned is stored in a result table, called the result-set.

```
SELECT * FROM myDB.students;
```

77. What is an Alias in SQL?

An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a **correlation name** .
An alias is represented explicitly by the **AS** keyword but in some cases the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

```
SELECT A.emp_name AS "Employee"      /* Alias using AS keyword */
B.emp_name AS "Supervisor"
FROM employee A, employee B          /* Alias without AS keyword */
WHERE A.emp_sup = B.emp_id;
```

78. What is Denormalization?

Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema which has redundant information. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in query processor by an over-normalized structure.

79. What is the difference between DROP and TRUNCATE statements?

If a table is dropped, all things associated with the tables are dropped as well. This includes - the relationships defined on the table with other tables, the integrity checks and constraints, access privileges and other grants that the table has. To create and use the table again in its original form, all these relations, checks, constraints, privileges and relationships need to be redefined. However, if a table is truncated, none of the above problems exist and the table retains its original structure.

80. What is the difference between DELETE and TRUNCATE statements?

The TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

The DELETE command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

81. What do you mean by data integrity?

Data Integrity defines the accuracy as well as the consistency of the data stored in a database. It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database.

82. How can you fetch first 5 characters of the string?

There are a lot of ways to fetch characters from a string. For example:

Select SUBSTRING(StudentName,1,5) as studentname from student

83. What is a Stored Procedure?

A Stored Procedure is a function which consists of many SQL statements to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required which saves time and avoid writing code again and again.

84. What do you mean by Collation?

Collation is defined as a set of rules that determine how data can be sorted as well as compared. Character data is sorted using the rules that define the correct character sequence along with options for specifying case-sensitivity, character width etc.

85. What is a Datawarehouse?

Datawarehouse refers to a central repository of data where the data is assembled from multiple sources of information. Those data are consolidated, transformed and made available for the mining as well as online processing. Warehouse data also have a subset of data called Data Marts.

86. What is a primary key?

A primary key is used to uniquely identify all table records. It cannot have NULL values, and it must contain unique values. A table can have only one primary key that consists of single or multiple fields.

87. What is a Unique Key?

The key which can accept only the null value and cannot accept the duplicate values is called Unique Key. The role of the unique key is to make sure that each column and row are unique.

88. What is the difference between Primary key and Unique Key?

Both Primary and Unique key carry unique values but the primary key can not have a null value where the Unique key can. And in a table, there cannot be more than one Primary key but unique keys can be multiple.

89. What is a foreign key?

A foreign key is an attribute or a set of attributes that references to the primary key of some other table. Basically, it is used to link together two tables.

90. What are Entities and Relationships?

Entities: Entity can be a person, place, thing, or any identifiable object for which data can be stored in a database.

Relationships: Relationships between entities can be referred to as the connection between two tables or entities.

91. What is the ACID property in a database?

The full form of ACID is Atomicity, Consistency, Isolation, and Durability. To check the reliability of the transactions, ACID properties are used.

- Atomicity refers to completed or failed transactions, where transaction refers to a single logical operation on data. This implies that if any aspect of a transaction fails, the whole transaction fails and the database state remains unchanged.

- Consistency means that the data meets all of the validity guidelines. The transaction never leaves the database without finishing its state.
- Concurrency management is the primary objective of isolation.
- Durability ensures that once a transaction is committed, it will occur regardless of what happens in between, such as a power outage, a fire, or some other kind of disturbance.

92. What do you know about the stuff() function?

The stuff function deletes a part of the string and then inserts another part into the string starting at a specified position.

Syntax:

```
STUFF(String1, Position, Length, String2)
```

93. What is a stored procedure? Give an example.

A stored procedure is a prepared SQL code that can be saved and reused. In other words, we can consider a stored procedure to be a function consisting of many SQL statements to access the database system. We can consolidate several SQL statements into a stored procedure and execute them whenever and wherever required.

A stored procedure can be used as a means of modular programming, i.e., we can create a stored procedure once, store it, and call it multiple times as required. This also supports faster execution when compared to executing multiple queries.

94. From the following user activity table, write a query to return the fraction of users who are retained (show some activity) a given number of days after joining. By convention, users are considered active on their join day (day 0).

users

user_id	action_date	action
1	01-01-20	Join
1	01-02-20	Access
2	01-02-20	Join
3	01-02-20	Join
1	01-03-20	Access
3	01-03-20	Access
1	01-04-20	Access

Desired output:

day_no	n_total	n_active	retention
0	3	3	1.00
1	3	2	0.67
2	3	1	0.33
3	1	1	1.00

```
WITH users (user_id, action_date, action)
AS (VALUES
(1, CAST('01-01-20' AS date), 'Join'),
(1, CAST('01-02-20' AS date), 'Access'),
(2, CAST('01-02-20' AS date), 'Join'),
(3, CAST('01-02-20' AS date), 'Join'),
(1, CAST('01-03-20' AS date), 'Access'),
(3, CAST('01-03-20' AS date), 'Access'),
(1, CAST('01-04-20' AS date), 'Access')),

-- get join dates for each user

join_dates AS (
SELECT user_id, action_date AS join_date
FROM users
WHERE action = 'Join' ),

-- create vector containing all dates in date range

date_vector AS (
SELECT cast(generate_series(min(action_date), max(action_date),
            '1 day'::interval) AS date) AS dates
FROM users ),
```

```
-- cross join to get all possible user-date combinations


all_users_dates AS (
SELECT DISTINCT user_id, d.dates
FROM users
CROSS JOIN date_vector d ),


-- left join users table onto all user-date combinations on matching
user ID and date (null on days where user didn't engage), join onto
this each user's signup date, exclude user-date combinations falling
before user signup


t1 AS (
SELECT a.dates - c.join_date AS day_no, b.user_id
FROM all_users_dates a
LEFT JOIN users b
ON a.user_id = b.user_id
AND a.dates = b.action_date
JOIN join_dates c
ON a.user_id = c.user_id
WHERE a.dates - c.join_date >= 0 )
```

```
-- grouping by days since signup, count (non-null) user IDs as active
users, total users, and the quotient as retention rate


SELECT day_no, count(*) AS n_total,
       count(DISTINCT user_id) AS n_active,
       round(1.0*count(DISTINCT user_id)/count(*), 2) AS retention
FROM t1
GROUP BY 1
```

95. From the given trips and users tables for a taxi service, write a query to return the cancellation rate in the first two days in October, rounded to two decimal places, for trips not involving banned riders or drivers. From LeetCode.

trips

trip_id	rider_id	driver_id	status	request_date
1	1	10	completed	2020-10-01
2	2	11	cancelled_by_driver	2020-10-01
3	3	12	completed	2020-10-01
4	4	10	cancelled_by_rider	2020-10-02
5	1	11	completed	2020-10-02
6	2	12	completed	2020-10-02
7	3	11	completed	2020-10-03

users

user_id	banned	type
1	no	rider
2	yes	rider
3	no	rider
4	no	rider
10	no	driver
11	no	driver
12	no	driver

Desired output

request_date	cancel_rate
2020-10-01	0.50
2020-10-02	0.33

```
WITH trips (trip_id, rider_id, driver_id, status, request_date)
AS (VALUES
(1, 1, 10, 'completed', CAST('2020-10-01' AS date)),
(2, 2, 11, 'cancelled_by_driver', CAST('2020-10-01' AS date)),
(3, 3, 12, 'completed', CAST('2020-10-01' AS date)),
(4, 4, 10, 'cancelled_by_rider', CAST('2020-10-02' AS date)),
(5, 1, 11, 'completed', CAST('2020-10-02' AS date)),
(6, 2, 12, 'completed', CAST('2020-10-02' AS date)),
(7, 3, 11, 'completed', CAST('2020-10-03' AS date))),


users (user_id, banned, type)
AS (VALUES
(1, 'no', 'rider'),
(2, 'yes', 'rider'),
(3, 'no', 'rider'),
(4, 'no', 'rider'),
(10, 'no', 'driver'),
(11, 'no', 'driver'),
(12, 'no', 'driver'))
```

```
-- filter trips table to exclude banned riders and drivers, then
calculate cancellation rate as 1 - fraction of trips completed,
rounding as requested and filtering to first two days of the month


SELECT request_date, round(1 - 1.0*sum(CASE WHEN status = 'completed'
THEN 1 ELSE 0 END)/count(*), 2) AS cancel_rate
FROM trips
WHERE rider_id NOT IN (SELECT user_id
                       FROM users
                       WHERE banned = 'yes' )
AND driver_id NOT IN (SELECT user_id
                      FROM users
                      WHERE banned = 'yes' )
GROUP BY request_date
HAVING extract(DAY FROM request_date) <= 2
```

96. Explain the difference between OLTP and OLAP.

OLTP: It stands for Online Transaction Processing, and we can consider it to be a category of software applications that is efficient for supporting transaction-oriented programs. One of the important attributes of the OLTP system is its potential to keep up the consistency. The OLTP system often follows decentralized planning to keep away from single points of failure. This system is generally designed for a large audience of end-users to perform short transactions. Also, queries involved in such databases are generally simple, need fast response time, and in comparison, return only a few records. So, the number of transactions per second acts as an effective measure for those systems.

OLAP: OLAP stands for Online Analytical Processing, and it is a category of software programs that are identified by a comparatively lower frequency of online transactions. For OLAP systems, the efficiency of computing depends highly on the response time. Hence, such systems are generally used for data mining or maintaining aggregated historical data, and they are usually used in multi-dimensional schemas.

97. What do you understand by Self Join?

Self Join in SQL is used for joining a table with itself. Here, depending upon some conditions, each row of the table is joined with itself and with other rows of the table.

98. What is the difference between Union and Union All operators?

The **Union** operator is used to combine the result set of two or more select statements. For example, the first select statement returns the fish shown in Image A, and the second returns the fish shown in Image B. Then, the Union operator will return the result of the two select statements as shown in Image A U B. Also, if there is a record present in both tables, then we will get only one of them in the final result.

99. What is Cursor? How to use a Cursor?

A database Cursor is a control that allows you to navigate around the table's rows or documents. It can be referred to as a pointer for a row in

the set of rows. Cursors are extremely useful for database traversal operations like extraction, insertion, and elimination.

- After any variable declaration, DECLARE a cursor. A SELECT Statement must always be aligned with the cursor declaration.
- To initialize the result set, OPEN statements must be called before fetching the rows from the result table.
- To grab and switch to the next row in the result set, use the FETCH statement.
- To deactivate the cursor, use the CLOSE expression.
- Finally, use the DEALLOCATE clause to uninstall the cursor description and clear all the resources associated with it.

100. What is the use of the Intersect operator?

The Intersect operator helps combine two select statements and returns only those records that are common to both the select statements. So, after we get Table A and Table B over here and if we apply the Intersect operator on these two tables, then we will get only those records that are common to the result of the select statements of these two.