

1. What is Database?

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

2. What is DBMS?

DBMS stands for Database Management System. DBMS is a system software responsible for the creation, retrieval, updation, and management of the database. It ensures that our data is consistent, organized, and is easily accessible by serving as an interface between the database and its end-users or application software.

3. What is RDBMS? How is it different from DBMS?

RDBMS stands for Relational Database Management System. The key difference [here](#), compared to DBMS, is that RDBMS stores data in the form of a collection of tables, and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2, and Amazon Redshift are based on RDBMS.

4. What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

5. What is the difference between SQL and MySQL?

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

6. What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

7. What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during the creation of the table or after creating using the ALTER TABLE command. The constraints are:

- **NOT NULL** - Restricts NULL value from being inserted into a column.
- **CHECK** - Verifies that all values in a field satisfy a condition.
- **DEFAULT** - Automatically assigns a default value if no value has been specified for the field.
- **UNIQUE** - Ensures unique values to be inserted into the field.
- **INDEX** - Indexes a field providing faster retrieval of records.
- **PRIMARY KEY** - Uniquely identifies each record in a table.
- **FOREIGN KEY** - Ensures referential integrity for a record in another table.

8. What is a Primary Key?

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.

A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

```

CREATE TABLE Students ( /* Create table with a single field as primary key
*/
    ID INT NOT NULL
    Name VARCHAR(255)
    PRIMARY KEY (ID)
);

CREATE TABLE Students ( /* Create table with multiple fields as primary key
*/
    ID INT NOT NULL
    LastName VARCHAR(255)
    FirstName VARCHAR(255) NOT NULL,
    CONSTRAINT PK_Student
    PRIMARY KEY (ID, FirstName)
);

ALTER TABLE Students /* Set a column as primary key */
ADD PRIMARY KEY (ID);
ALTER TABLE Students /* Set multiple columns as primary key */
ADD CONSTRAINT PK_Student /*Naming a Primary Key*/
PRIMARY KEY (ID, FirstName);

```

9. What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, there can be multiple unique constraints defined per table. The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

```

CREATE TABLE Students ( /* Create table with a single field as unique */
    ID INT NOT NULL UNIQUE
    Name VARCHAR(255)
);

CREATE TABLE Students ( /* Create table with multiple fields as unique */
    ID INT NOT NULL
    LastName VARCHAR(255)
    FirstName VARCHAR(255) NOT NULL
    CONSTRAINT PK_Student
    UNIQUE (ID, FirstName)
);

ALTER TABLE Students /* Set a column as unique */
ADD UNIQUE (ID);
ALTER TABLE Students /* Set multiple columns as unique */
ADD CONSTRAINT PK_Student /* Naming a unique constraint */
UNIQUE (ID, FirstName);

```

10. What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refers to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables. The table with the foreign key constraint is labeled as the child table, and the table containing the candidate key is labeled as the referenced or parent table.

```

CREATE TABLE Students ( /* Create table with foreign key - Way 1 */

```

```

ID INT NOT NULL
Name VARCHAR(255)
LibraryID INT
PRIMARY KEY (ID)
FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

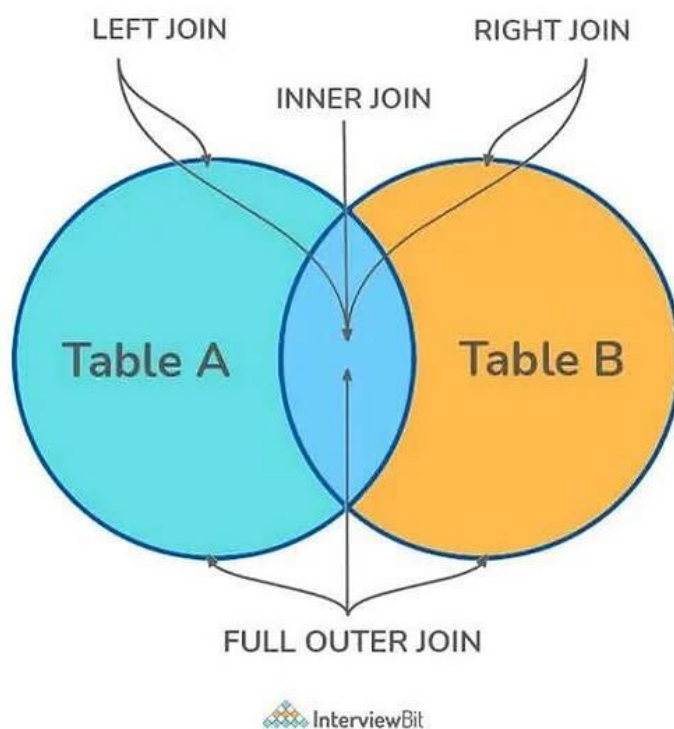
CREATE TABLE Students ( /* Create table with foreign key - Way 2 */
ID INT NOT NULL PRIMARY KEY
Name VARCHAR(255)
LibraryID INT FOREIGN KEY (Library_ID) REFERENCES Library(LibraryID)
);

ALTER TABLE Students /* Add a new foreign key */
ADD FOREIGN KEY (LibraryID)
REFERENCES Library (LibraryID);

```

11. What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.



There are four different types of JOINS in SQL:

- **(INNER) JOIN:** Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.

```
SELECT *
```

```
FROM Table_A
JOIN Table_B;
SELECT *
FROM Table_A
INNER JOIN Table_B;
```

- **LEFT (OUTER) JOIN:** Retrieves all the records/rows from the left and the matched records/rows from the right table.

```
SELECT *
FROM Table_A A
LEFT JOIN Table_B B
ON A.col = B.col;
```

- **RIGHT (OUTER) JOIN:** Retrieves all the records/rows from the right and the matched records/rows from the left table.

```
SELECT *
FROM Table_A A
RIGHT JOIN Table_B B
ON A.col = B.col;
```

- **FULL (OUTER) JOIN:** Retrieves all the records where there is a match in either the left or right table.

```
SELECT *
FROM Table_A A
FULL JOIN Table_B B
ON A.col = B.col;
```

12. What is a Self-Join?

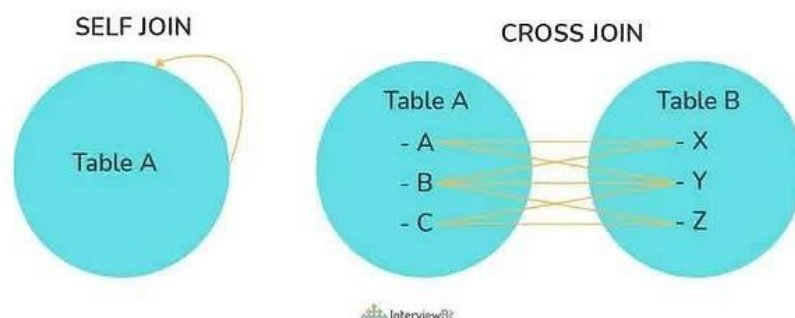
A **self JOIN** is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

```
SELECT A.emp_id AS "Emp_ID", A.emp_name AS "Employee",
B.emp_id AS "Sup_ID", B.emp_name AS "Supervisor"
FROM employee A, employee B
WHERE A.emp_sup = B.emp_id;
```

13. What is a Cross-Join?

Cross join can be defined as a cartesian product of the two tables included in the join. The table after join contains the same number of rows as in the cross-product of the number of rows in the two tables. If a WHERE clause is used in cross join then the query will work like an INNER JOIN.

```
SELECT stu.name, sub.subject
FROM students AS stu
CROSS JOIN subjects AS sub;
```

14. What is an Index? Explain its different types.

A database index is a data structure that provides a quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

```
CREATE INDEX index_name /* Create Index */
ON table_name (column_1, column_2);
DROP INDEX index_name; /* Drop Index */
```

There are different types of indexes that can be created for different purposes:

- **Unique and Non-Unique Index:**

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index.

```
CREATE UNIQUE INDEX myIndex
ON students (enroll_no);
```

Non-unique indexes, on the other hand, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

- **Clustered and Non-Clustered Index:**

Clustered indexes are indexes whose order of the rows in the database corresponds to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas, multiple non-clustered indexes can exist in the table.

The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the database in the same order as the corresponding keys appear in the clustered index.

Clustering indexes can improve the performance of most query operations because they provide a linear-access path to data stored in the database.

15. What is the difference between Clustered and Non-clustered index?

As explained above, the differences can be broken down into three small factors -

- Clustered index modifies the way records are stored in a database based on the indexed column. A non-clustered index creates a separate entity within the table which references the original table.

- Clustered index is used for easy and speedy retrieval of data from the database, whereas, fetching records from the non-clustered index is relatively slower.
- In SQL, a table can have a single clustered index whereas it can have multiple non-clustered indexes.

16. What is Data Integrity?

Data Integrity is the assurance of accuracy and consistency of data over its entire life-cycle and is a critical aspect of the design, implementation, and usage of any system which stores, processes, or retrieves data. It also defines integrity constraints to enforce business rules on the data when it is entered into an application or a database.

17. What is a Query?

A query is a request for data or information from a database table or combination of tables. A database query can be either a select query or an action query.

```
SELECT fname, lname      /* select query */
FROM myDb.students
WHERE student_id = 1;
UPDATE myDb.students     /* action query */
SET fname = 'Captain', lname = 'America'
WHERE student_id = 1;
```

18. What is a Subquery? What are its types?

A subquery is a query within another query, also known as a **nested query** or **inner query**. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. For example, here we fetch the contact information for students who have enrolled for the maths subject:

```
SELECT name, email, mob, address
FROM myDb.contacts
WHERE roll_no IN (
    SELECT roll_no
    FROM myDb.students
    WHERE subject = 'Maths');
```

There are two types of subquery - **Correlated** and **Non-Correlated**.

- A **correlated** subquery cannot be considered as an independent query, but it can refer to the column in a table listed in the FROM of the main query.
- A **non-correlated** subquery can be considered as an independent query and the output of the subquery is substituted in the main query.

19. 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;
```

20. What are some common clauses used with SELECT query in SQL?

Some common SQL clauses used in conjunction with a SELECT query are as follows:

- **WHERE** clause in SQL is used to filter records that are necessary, based on specific conditions.
- **ORDER BY** clause in SQL is used to sort the records based on some field(s) in ascending (**ASC**) or descending order (**DESC**).

```
SELECT *
FROM myDB.students
WHERE graduation_year = 2019
ORDER BY studentID DESC;
```

- **GROUP BY** clause in SQL is used to group records with identical data and can be used in conjunction with some aggregation functions to produce summarized results from the database.
- **HAVING** clause in SQL is used to filter records in combination with the **GROUP BY** clause. It is different from **WHERE**, since the **WHERE** clause cannot filter aggregated records.

```
SELECT COUNT(studentId), country
FROM myDB.students
WHERE country != "INDIA"
GROUP BY country
HAVING COUNT(studentID) > 5;
```

21. What are UNION, MINUS and INTERSECT commands?

The **UNION** operator combines and returns the result-set retrieved by two or more **SELECT** statements.

The **MINUS** operator in SQL is used to remove duplicates from the result-set obtained by the second **SELECT** query from the result-set obtained by the first **SELECT** query and then return the filtered results from the first.

The **INTERSECT** clause in SQL combines the result-set fetched by the two **SELECT** statements where records from one match the other and then returns this intersection of result-sets.

Certain conditions need to be met before executing either of the above statements in SQL -

- Each **SELECT** statement within the clause must have the same number of columns
- The columns must also have similar data types
- The columns in each **SELECT** statement should necessarily have the same order

```
SELECT name FROM Students /* Fetch the union of queries */
UNION
SELECT name FROM Contacts;
SELECT name FROM Students /* Fetch the union of queries with duplicates*/
UNION ALL
SELECT name FROM Contacts;
SELECT name FROM Students /* Fetch names from students */
MINUS /* that aren't present in contacts */
SELECT name FROM Contacts;
SELECT name FROM Students /* Fetch names from students */
INTERSECT /* that are present in contacts as well */
SELECT name FROM Contacts;
```

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

A database cursor is a control structure that allows for the traversal of records in a database. Cursors, in addition, facilitates processing after traversal, such as retrieval, addition, and deletion of database records. They can be viewed as a pointer to one row in a set of rows.

Working with SQL Cursor:

1. **DECLARE** a cursor after any variable declaration. The cursor declaration must always be associated with a **SELECT** Statement.

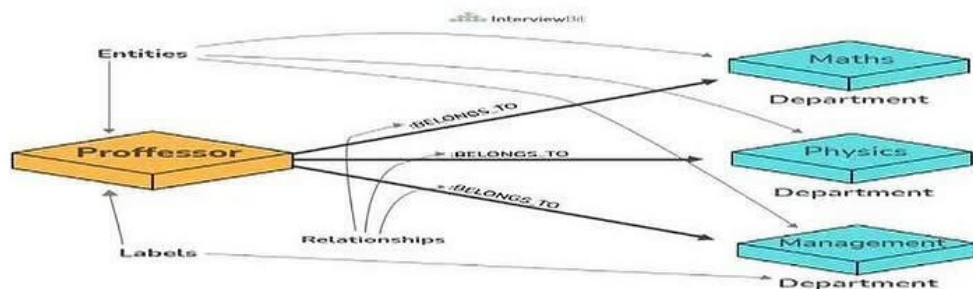
2. Open cursor to initialize the result set. The **OPEN** statement must be called before fetching rows from the result set.
3. **FETCH** statement to retrieve and move to the next row in the result set.
4. Call the **CLOSE** statement to deactivate the cursor.
5. Finally use the **DEALLOCATE** statement to delete the cursor definition and release the associated resources.

```
DECLARE @name VARCHAR(50)    /* Declare All Required Variables */
DECLARE db_cursor CURSOR FOR /* Declare Cursor Name*/
SELECT name
FROM myDB.students
WHERE parent_name IN ('Sara', 'Ansh')
OPEN db_cursor /* Open cursor and Fetch data into @name */
FETCH next
FROM db_cursor
INTO @name
CLOSE db_cursor /* Close the cursor and deallocate the resources */
DEALLOCATE db_cursor
```

23. What are Entities and Relationships?

Entity: An entity can be a real-world object, either tangible or intangible, that can be easily identifiable. For example, in a college database, students, professors, workers, departments, and projects can be referred to as entities. Each entity has some associated properties that provide it an identity.

Relationships: Relations or links between entities that have something to do with each other. For example - The employee's table in a company's database can be associated with the salary table in the same database.



24. List the different types of relationships in SQL.

- **One-to-One** - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.
- **One-to-Many & Many-to-One** - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.
- **Many-to-Many** - This is used in cases when multiple instances on both sides are needed for defining a relationship.
- **Self-Referencing Relationships** - This is used when a table needs to define a relationship with itself.

25. What is an Alias in SQL?

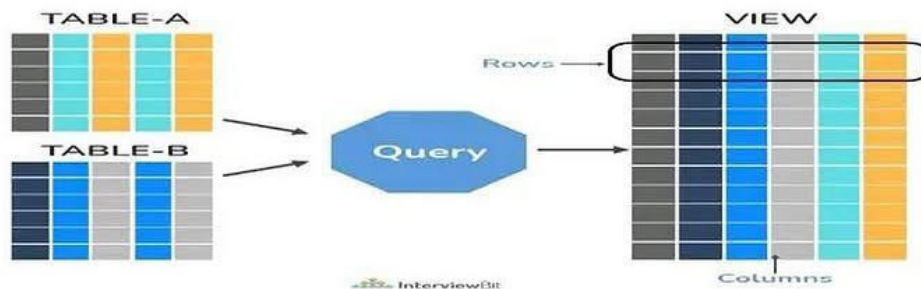
An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias 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;
```

26. What is a View?

A view in SQL 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 fields from one or more real tables in the database.



27. What is Normalization?

Normalization represents the way of organizing structured data in the database efficiently. It includes the creation of tables, establishing relationships between them, and defining rules for those relationships. Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

28. What is Denormalization?

Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema that 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 the query processor by an over-normalized structure.

29. What are the various forms of Normalization?

Normal Forms are used to eliminate or reduce redundancy in database tables. The different forms are as follows:

- **First Normal Form:**

A relation is in first normal form if every attribute in that relation is a **single-valued attribute**. If a relation contains a composite or multi-valued attribute, it violates the first normal form. Let's consider the following **students** table. Each student in the table, has a name, his/her address, and the books they issued from the public library -

Students Table

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter), Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho), Inferno (Dan Brown)	Mr.

Student	Address	Books Issued	Salutation
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward), Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

As we can observe, the Books Issued field has more than one value per record, and to convert it into 1NF, this has to be resolved into separate individual records for each book issued. Check the following table in 1NF form -

Students Table (1st Normal Form)

Student	Address	Books Issued	Salutation
Sara	Amanora Park Town 94	Until the Day I Die (Emily Carpenter)	Ms.
Sara	Amanora Park Town 94	Inception (Christopher Nolan)	Ms.
Ansh	62nd Sector A-10	The Alchemist (Paulo Coelho)	Mr.
Ansh	62nd Sector A-10	Inferno (Dan Brown)	Mr.
Sara	24th Street Park Avenue	Beautiful Bad (Annie Ward)	Mrs.
Sara	24th Street Park Avenue	Woman 99 (Greer Macallister)	Mrs.
Ansh	Windsor Street 777	Dracula (Bram Stoker)	Mr.

- **Second Normal Form:**

A relation is in second normal form if it satisfies the conditions for the first normal form and does not contain any partial dependency. A relation in 2NF has **no partial dependency**, i.e., it has no non-prime attribute that depends on any proper subset of any candidate key of the table. Often, specifying a single column Primary Key is the solution to the problem. Examples -

Example 1 - Consider the above example. As we can observe, the Students Table in the 1NF form has a candidate key in the form of [Student, Address] that can uniquely identify all records in the table. The field Books Issued (non-prime attribute) depends partially on the Student field. Hence, the table is not in 2NF. To convert it into the 2nd Normal Form, we will partition the tables into two while specifying a new **Primary Key** attribute to identify the individual records in the Students table. The **Foreign Key** constraint will be set on the other table to ensure referential integrity.

Students Table (2nd Normal Form)

Student_ID	Student	Address	Salutation
1	Sara	Amanora Park Town 94	Ms.
2	Ansh	62nd Sector A-10	Mr.
3	Sara	24th Street Park Avenue	Mrs.
4	Ansh	Windsor Street 777	Mr.

Books Table (2nd Normal Form)

Student_ID	Book Issued
------------	-------------