**Q1: 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 creation of table or after creation 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.

**Q2: 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).

**Q3: 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.

**Q4: What is a Foreign Key?**

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refer 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 labelled as the child table,

and the table containing the candidate key is labelled 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)

);

**Q5: What are the types of join and explain each?**

There are various types of join which can be used to retrieve data and it depends on the relationship between tables.

**Inner Join (Join)**: return rows when there is at least one match of rows between the tables.

**Right Join**: return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

**Left Join**: return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

**Full Join (full outer join)**: return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

**Q6: 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.

**Q7: 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 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;**

**Q8: What is an Index? Explain its different types.**

A database index is a data structure that provides 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.

**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.

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

**Q9: 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. 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.

**Q10: 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 conjuction 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 WHERE clause cannot filter aggregated records.

SELECT COUNT(studentId), country FROM myDB.students

WHERE country != "INDIA"

GROUP BY country

HAVING COUNT(studentID) > 5;

**Q11: 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

**Q12: What is Cursor? How to use a Cursor?**

A database cursor is a control structure that allows for 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

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

**Open** cursor to initialize the result set. The OPEN statement must be called before fetching rows from the result set.

**FETCH** statement to retrieve and move to the next row in the result set.

Call the CLOSE statement to deactivate the cursor.

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

**Q13: 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.

**Q14: What is Normalization? What is Denormalization?**

Normalization represents the way of organizing structured data in the database efficiently.

It includes 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.

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 the query processor by an over-normalized structure.

**Q15: What are the TRUNCATE, DELETE and DROP statements?**

**DELETE** statement is used to delete rows from a table.

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

**DROP** command is used to remove an object from the database. If you drop a table,

all the rows in the table are deleted and the table structure is removed from the database.

**Q16: 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.

**Q17: 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.

**Q18: What are Aggregate and Scalar functions?**

An aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement. Following are the widely used SQL aggregate functions:

**AVG()** - Calculates the mean of a collection of values.

**COUNT()** - Counts the total number of records in a specific table or view.

**MIN()** - Calculates the minimum of a collection of values.

**MAX()** - Calculates the maximum of a collection of values.

**SUM()** - Calculates the sum of a collection of values.

**FIRST()** - Fetches the first element in a collection of values.

**LAST()** - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT function.

A scalar function returns a single value based on the input value. Following are the widely used SQL scalar functions:

**LEN()** - Calculates the total length of the given field (column).

**UCASE()** - Converts a collection of string values to uppercase characters.

**LCASE()** - Converts a collection of string values to lowercase characters.

**MID()** - Extracts substrings from a collection of string values in a table.

**CONCAT()** - Concatenates two or more strings.

**RAND()** - Generates a random collection of numbers of given length.

**ROUND()** - Calculates the round off integer value for a numeric field (or decimal point values).

**NOW()** - Returns the current data & time.

**FORMAT()** - Sets the format to display a collection of values.

**Q19: What are the properties of the transaction?**

Properties of the transaction are known as ACID properties, such as

**Atomicity**: Ensures the completeness of all transactions performed. Checks whether every transaction is completed successfully if not then transaction is aborted at the failure point and the previous transaction is rolled back to its initial state as changes undone

**Consistency**: Ensures that all changes made through successful transaction are reflected properly on database

**Isolation**: Ensures that all transactions are performed independently and changes made by one transaction are not reflected on other

**Durability**: Ensures that the changes made in the database with committed transactions persist as it is even after a system failure

**Q20: What is a Stored Procedure?**

A stored procedure is a subroutine available to applications that access a relational database management system (RDBMS). Such procedures are stored in the database data dictionary. The sole disadvantage of stored procedure is that it can be executed nowhere except in the database and occupies more memory in the database server. It also provides a sense of security and functionality as users who can't access the data directly can be granted access via stored procedures.

DELIMITER $$

CREATE PROCEDURE FetchAllStudents()

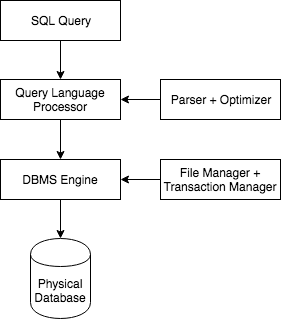
BEGIN

SELECT \* FROM myDB.students;

END $$

DELIMITER ;

**Q21: SQL process:**



**Q22: sql command**

### **DDL - Data Definition Language**

DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

* CREATE - create a new table, view for a table or other object in the database
* ALTER - modifies an existing database object, such as a table
* DROP - deletes an entire table, a view of a table or other objects in the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

### **DML - Data Manipulation Language**

DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in a database.

* SELECT - retrieves records from one or more tables
* INSERT - creates a record
* UPDATE - modifies records
* DELETE - deletes records
* MERGE - UPSERT operation (insert or update)
* CALL - call a PL/SQL or Java subprogram
* EXPLAIN PLAN - interpretation of the data access path
* LOCK TABLE - concurrency control

### **DCL - Data Control Language**

DCL is short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

* GRANT (Grant privilige(s) to user)
* REVOKE (Remove granted privilige(s) from a user)

### **TCL - Transaction Control Language**

TCL is short name of Transaction Control Language which deals with a transaction within a database.

* COMMIT - commits a transaction
* ROLLBACK - rollback a transaction in case of any error occurs
* SAVEPOINT - to rollback the transaction making points within groups
* SET TRANSACTION - specify characteristics of the transaction