1. **What is the primary difference between SQL and PL/SQL?**
2. **What is the main difference between Primary Key, Unique Key, and Foreign Key?**
3. **What is the purpose of a JOIN statement? Also, explain the different types of JOIN clauses supported in SQL.**
4. **What is Normalization and how does it work?**
5. **What are the various forms of Normalization?**
6. **What is the difference between a superkey and the candidate key?**
7. **What are the different types of statements available in SQL?**
8. **What is the role of COMMIT in an SQL transaction?**
9. **What are the common properties of a database transaction?**
10. **What are the main points that differentiate between the "delete", "truncate" and "drop" commands?**
11. **What is an Index? Explain the different types of index.**
12. **What is the purpose of a Subquery?**
13. **What are Constraints? Explain the different Constraints available in SQL?**
14. **What is the difference between Union and Union ALL?**
15. **What is a stored procedure? Discuss its advantages and disadvantages?**
16. **What is a View? What are its advantages and disadvantages?**
17. **List the built-in functions, available in SQL?**
18. **What are Triggers? What are its benefits? Can we invoke a trigger explicitly?**
19. **What are the Clauses available in SQL?**
20. **What is the purpose of isolation levels in SQL?**

Q-1. What is the primary difference between SQL and PL/SQL?

Ans.

PL/SQL is an advanced form of SQL developed by Oracle in early 90's as a superset of SQL. It inculcates many additional programming features to enable application development at the database level. Please refer the below list.

- Modular structure.
- Control-Flow statements and loops.
- Types, constants, and variables.
- User-defined data types.
- Exceptional handling.

Q-2. What is the main difference between Primary Key, Unique Key, and Foreign Key?

Ans. Following are the key differences between Primary Key, Unique Key, and Foreign Key.

Primary Key:

- The primary key cannot have a NULL value.
- Every table can have only one primary key.
- By default, Primary key supports clustered index. Thus data in the database table are physically organized in the sequence of clustered index.
- It can be related to another table as a Foreign Key.
- It supports the generation of ID automatically with the help of Auto Increment field.

Unique Key:

- Unique Constraint may have a NULL value.
- Each table can have more than one Unique Constraint.
- By default, Unique key is a unique non-clustered index.
- It is not related to another table as a Foreign Key.
- Unique Constraint doesn't support Auto Increment value.

Foreign Key:

- A Foreign key is a field in a table whereas, it is the primary key in another table.
- It can accept multiple null values.
- A Foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index on the foreign key.
- We can have more than one foreign key in a table.
- There are advantages of having a foreign key supported with a clustered index, but you get only one per table. The advantage using a clustered index is that, on selecting the parent plus all child records, it can bring all child records next to each other.
- The Foreign key shouldn't have a null value. Else, the system will consider it as an orphan record.

Q-3. What is the purpose of a JOIN statement? Also, explain the different types of JOIN clauses supported in SQL.

Ans.

JOIN keyword is used to fetch data from two or more related tables. It returns rows where there is at least one match in both the tables included in the join.

SQL specifies five types of JOIN clauses as follows:

1. INNER JOIN (also called as "simple join"):

It returns all the rows for which there is at least one match in BOTH the tables. If join type is not specifically mentioned then "INNER JOIN" works as the default join.

SQL Syntax for INNER JOIN:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

2. LEFT JOIN (or LEFT OUTER JOIN):

Returns all rows from the left table, and the matching rows from the right table. Thus the result will contain all records from the left table, even if the JOIN condition doesn't find any matching records in the right table. This means, that if the "ON" clause does not match to any records in the right table, the JOIN will return a row in the result for that record in the left table, but with NULL in each column from the right table.

SQL Syntax for LEFT JOIN:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

3. RIGHT JOIN (or RIGHT OUTER JOIN):

It returns all rows from the right table and the corresponding matching rows from the left table. It is an exact opposite of the LEFT JOIN. Thus the result will contain all the records from the right table, even if the JOIN condition doesn't find any matching records in the left table. This means, that if the "ON" clause does not match to any records in the left table, the JOIN will return a row in the result for that record in the right table, but with NULL in each column from the left table.

SQL Syntax for RIGHT JOIN:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name;
```

4. FULL JOIN (or FULL OUTER JOIN):

It returns all the rows for which there is a match in either of the tables. Fundamentally, a FULL JOIN is a combination of the effect produced by both a LEFT JOIN and a RIGHT JOIN. Thus we can say that its result set is equivalent to performing a UNION of the results of left and right outer queries.

SQL Syntax for FULL OUTER JOIN:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

5. CROSS JOIN:

It returns a result set which is the multiplication of the number of rows in the first and the second table. If we do not apply WHERE clause along with CROSS JOIN, it returns the Cartesian Product. However, if we use WHERE clause along with CROSS JOIN, it functions like an INNER JOIN.
An alternative way of achieving the same result is to use column names separated by commas after SELECT and mentioning the table names involved, after a FROM clause.

SQL Syntax for CROSS JOIN:

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

6. SELF JOIN:

It is used to join a table to itself as if the table were two tables. To achieve this, we temporarily rename one of the tables in the SQL statement.

Syntax:

```
SELECT column_name(s)
FROM table1, table2
WHERE table1.common_field = table2.common_field;
```

Q-4. What is Normalization and how does it work?

Ans. The process of designing database tables to minimize the data redundancy is called normalization. We need to divide a database into two or more tables and define relationships between them.

Q-5. What are the various forms of Normalization?

Ans. Database normalization process provides following forms:

1. First normal form (1NF):

As per the rule of 1NF, an attribute(column) of a table can not hold multiple values. It should contain only atomic values.
For Example suppose a company stores details of its employees including the name, address, and the contact number. It is possible that some employees have more than one contacts. In that case, employee table is like this:

emp_id	emp_name	emp_address	emp_contact
001	Nidhi	Gurgaon	9873456789, 9990022334
002	Prakash	New Delhi	7838777343
003	Mallika	New Delhi	7838005674, 8876453212

Since multiple contact numbers are for the same employees so it gets stored in the same field.
As per the 1NF rule, the above table is not in 1NF. To make the table as 1NF compliant, we have to store only a single contact in one row as:

emp_id	emp_name	emp_address	emp_contact
001	Nidhi	Gurgaon	9873456789
001	Nidhi	Gurgaon	9990022334
002	Prakash	New Delhi	7838777343

003	Mallika	New Delhi	7838005674
003	Mallika	New Delhi	8876453212

2. Second normal form (2NF):

A table is in 2NF if the following conditions hold true:

- The table is in 1NF.
- No non-prime attribute is dependent on the proper subset of any candidate key of the table.

An attribute that is not part of any candidate key is known as a non-prime attribute.

For Example, a school stores data about the teachers and the subject they teach. A teacher can teach more than one subject. Thus the table will look like:

teacher_id	subject	teacher_age
01A	Maths	40
01A	Physics	40
01B	English	42
01C	Chemistry	40
01C	EVS	40

Candidate Keys:

{teacher_id, subject}

Nonprime attribute:

teacher_age

The table is in 1 NF. However, it is not in 2NF because nonprime attribute teacher_age is dependent on teacher_id alone which is a proper subset of the candidate key.

Now to make the table as 2NF compliant, we break the table as follows:

Teacher_details table:

teacher_id	teacher_age
01A	40
01B	42
01C	40

Teacher_subject table:

teacher_id	subject
01A	Maths
01A	Physics
01B	English
01C	Chemistry
01C	EVS

3. Third normal form (3NF):

A table is in 3NF if both the given conditions hold true:

- The table is in 2NF.
- For every functional dependency (X-> Y), at least one of the following conditions hold:
  - X is a super key of the table.
  - Y is a prime attribute of the table.

A prime attribute is an attribute that is part of the candidate key.
Suppose we have a table that stores information about employee address as:

emp_id	emp_name	emp_city	emp_state	emp_zip
001	Mayank	Ghaziabad	UP	201001
002	Saksham	Gwalior	MP	222999
003	Mallika	Gurgaon	Haryana	122001

In the above table following are the:

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes:

all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here the non-prime attributes emp_state, emp_city is dependent on emp_zip which is dependent on emp_id. Thus this created a transitive dependency on the super key emp_id which is a violation of the 3NF rule. To make it 3NF compliant, break the table as follows:

emp_id	emp_name	emp_zip
001	Mayank	201001
002	Saksham	222999
003	Mallika	122001

and

emp_id	emp_city	emp_state	emp_zip
001	Ghaziabad	UP	201001
002	Gwalior	MP	222999
003	Gurgaon	Haryana	1220001

4. Boyce-Codd normal form (BCNF):

It is an advanced version of 3NF also called as 3.5NF. BCNF is stricter than 3NF.

A table is BCNF compliant if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Q-6. What is the difference between a superkey and the candidate key?

Ans. A superkey is a combination of columns that uniquely identifies any row within a relational database management system (RDBMS) table.

Whereas, a candidate key is a superkey containing a minimum number of columns that can uniquely identify each row.

Q-7. What are the different types of statements available in SQL?

Ans.

1. DML (Data Manipulation Language):

These are used to manage records in the table. It includes the basic operations carried out on the tabular data like selecting few records, inserting new ones, deleting the unnecessary ones, and updating/modifying the existing ones. Following are different DML statements available in SQL:

- <SELECT> – retrieve data from the database
- <INSERT> – to insert data into a table
- <UPDATE> – it updates existing data within a table
- <DELETE> – to delete all records from a table
- <MERGE> – UPSERT operation (insert or update)
- <CALL> – to call a PL/SQL or Java subprogram

- <EXPLAIN PLAN> – define access path to data
- <LOCK TABLE> – control concurrency

2. DDL (Data Definition Language):

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects. Following are different DDL statements available in SQL:

- <CREATE> – to create objects in the database
- <ALTER> – alters the structure of the database
- <DROP> – to delete objects from the database
- <TRUNCATE> – remove all records from a table. It also frees all the space allocated to them.
- <COMMENT> – add comments to the data dictionary
- <RENAME> – to rename an object.

3. DCL (Data Control Language):

DCL statements control the level of access that users have to the database objects. Following are different DCL statements available in SQL:

- <GRANT> – it gives access privileges to the user for the database
- <REVOKE> – to withdraw the access privileges given by GRANT command.

4. TCL (Transaction Control Language):

It allows you to control and manage transactions to maintain the integrity of data within SQL statements. Following are different TCL statements:

- <COMMIT> – to save the work
- <SAVEPOINT> – identify a point in a transaction to which you can rollback at a later point in time when required
- <ROLLBACK> – restore the database to original since the last COMMIT
- <SET TRANSACTION> – Change transaction options like isolation level and what rollback segment to use.

Q-8. What is the role of COMMIT in an SQL transaction?

Ans.

COMMIT finalizes the changes, introduced by all SQL statements included in the transaction as permanent in the database.

Thus the changes made by the SQL statements of a transaction become visible to other user's session transactions that start only after the transaction gets committed.

Q-9. What are the common properties of a database transaction?

Ans.

Following are the properties on which every database depends to perform reliable transactions. We usually call them as ACID properties.

- **Atomicity:** A transaction may contain two or more discrete pieces of information. Atomicity means either commit all the data or nothing.
- **Consistency:** A transaction creates a new and valid state of data. However, if any failure occurs, it reverts the data to its original state before the start of the transaction.
- **Isolation:** A transaction under execution and not yet committed must remain isolated from any other transaction.
- **Durability:** System stores the committed data so that the data is available in its correct state, in case a failure or system restart happens.

Q-10. What are the main points that differentiate between the "delete", "truncate" and "drop" commands?

Ans.

A. DELETE:

- It is a DML statement.
- It applies a filter based on an optional WHERE clause to identify the rows that will get deleted.
- It is possible to roll back a transaction that got deleted.
- It does not reset the identity of the table.
- Triggers will get fired.
- On initiating a DELETE operation, all the data first gets copied into Rollback Tablespace and then delete operation gets performed. Thus we can get back the data by ROLLBACK command.
- Use this, only when you want to delete specific records. For example, <DELETE FROM table_name WHERE username = 'Aditya';>

SYNTAX:

To delete a particular row.

```
DELETE FROM table_name
WHERE column_name = column_value
```

To delete all rows.

```
DELETE FROM table_name
#Or
DELETE * FROM table_name
```

B. TRUNCATE:

- It is a DDL Statement.
- Removes all rows from a table and it becomes empty. But, the table structures, its columns, constraints, and indexes remain intact.

- It is not possible to roll back the TRUNCATE transaction.
- It resets the identity of table i.e. the auto-incrementing keys are reset to 1. It's just like having a brand new table.
- It is faster than DELETE and uses a lesser amount of system and transaction logs.
- TRUNCATE cannot be used on a table referenced by a FOREIGN KEY constraint.
- No Triggers will get fired.
- Cannot use WHERE conditions.
- Use this when you just want an empty table.

SYNTAX:

```
TRUNCATE TABLE table_name
```

C. DROP:

- It is a DDL statement.
- It not only removes the table from the database. Its structures, indexes, privileges, and constraints also get removed.
- It is not possible to roll back the DROP transaction.
- No Triggers will get fired.
- Use this when you don't need that table anymore.

SYNTAX:

```
DROP TABLE table_name
```

Q-11. What is an Index? Explain the different types of index.

Ans.

An index is a performance enhancement method that allows faster retrieval of records from the table. An index creates an entry for each value thus making data retrieval faster.

While creating an index, we should remember the columns which will be used to make SQL queries and create one or more indexes on those columns.

Following are the available indexes.

1. Clustered index:

It sorts and stores the rows of data in the table or view, based on its keys. These are the columns included in the index definition. There can be only one clustered index per table because sorting of data rows can be done only in one order.

2. Nonclustered index:

It contains the nonclustered index key value and each key value entry, in turn, has a pointer to the data row. Thus a nonclustered index contains a pointer to the physical location of the record. Each table can have 999 nonclustered indexes.

3. Unique Index:

This indexing does not allow the field to have duplicate values if the column is unique indexed. It can be applied automatically when a primary key is defined.

Q-12. What is the purpose of a Subquery?

Ans.

A Subquery also called as Nested query is a query within another SQL query and embedded within the WHERE clause. A Subquery is always executed first and passes its result to the main query. This data acts as a filter condition in the main query to further restrict the data to be retrieved. Subqueries work with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, and BETWEEN.

Following are some important properties of a Subquery that we must know:

- Always write a Subquery within a parenthesis.
- It can contain more than one column in SELECT clause only if the main query has multiple columns.
- We cannot use ORDER BY in a Subquery. Instead, use GROUP BY which performs the same function as ORDER BY.
- We cannot use BETWEEN operator with a subquery but, can use it within a Subquery.
- You can nest Subqueries up to 32 levels.

Q-13. What are Constraints? Explain the different Constraints available in SQL?

Ans.

These are the set of rules that determine or restrict the type of data that can go into a table, to maintain the accuracy and integrity of the data inside the table.

Following are the most frequent used constraints, applicable to a table:

- <NOT NULL> It restricts a column from holding a NULL value. It does not work on a table.
- <UNIQUE> It ensures that a field or column will only have unique values. It is applicable to both column and table.
- <PRIMARY KEY> uniquely identifies each record in a database table and it cannot contain NULL values.
- <FOREIGN KEY> It is used to relate two tables. The FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables.
- <CHECK CONSTRAINT> It is used to restrict the value of a column between a range. It performs a check on the values, before storing them into the database. It’s like condition checking before saving data into a column.
- <DEFAULT> It is used to insert a default value into a column.

Q-14. What is the difference between Union and Union ALL?

Ans.

UNION and UNION ALL merges the contents of two structurally-compatible tables into a single combined table.

- The difference between UNION andUNION ALL is that UNION will remove duplicate records whereas UNION ALL will include duplicate records.
- The performance of UNION ALL is better than UNION as UNION requires the server to do additional work of removing duplicates.

For performance reasons, it is recommended to use UNION ALL is the scenarios when it is certain that there will be no duplicates or cases where having duplicates is not a problem.

Q-15. What is a stored procedure? Discuss its advantages and disadvantages?

Ans.

A stored procedure is a group of SQL statements that has been created and stored in the database. Suppose there is a query that we execute very frequently. In that case, instead of writing that query, again and again. We can save it as a stored procedure and then just call the stored procedure to execute the SQL code. It also allows passing parameters to the stored procedure.

An Example of Stored Procedure.

```
USE testdb;
GO
    CREATE PROCEDURE test_procedure
    AS
   SELECT FirstName, LastName FROM testdb;
  GO
    EXEC test_procedure;

   GO
  DROP PROCEDURE test_procedure;
        GO
```

Following are the advantages of using stored procedure:

- A stored procedure allows modular programming. It means once we create a stored procedure and store it in the database, then we can call it any number of times as per our requirement.
- It allows faster execution in the case when the operation executes same SQL code repetitively. It gets parsed and optimized during its first execution. A compiled version of the stored procedure remains in memory cache for later use resulting in much faster execution time.
- Stored Procedure can reduce network traffic. Suppose there is an operation that requires executing large SQL code, creates a stored procedure. Thus rather than sending hundreds of lines of code over the network, it needs to send a single statement that executes the code.
- Stored procedures provide better security to your data. A user might be allowed to trigger a stored procedure even if he doesn't have the permission to execute the procedure's statements directly.

Its disadvantage is that the execution requires a database and thus utilizes more memory in the database server.

Q-16. What is a View? What are its advantages and disadvantages?

Ans.

A View is a virtual table which contains data from one or more tables. It selects only required values thus restricting the access to table data. And it also makes complex queries a bit easier.

Following are the advantages of using Views:

- It enables viewing data without storing the data in an object.
- Restrict the view of a table by hiding some of its columns.
- Join two or more tables and display it as a single object.
- Restrict the access of a table so that nobody can insert rows in the table without permission.

Disadvantages of Views:

- Cannot apply DML statements on it.
- A View becomes inactive if a table that is a part of View gets dropped.
- It is an object and hence, it consumes memory.

Q-17. List the built-in functions, available in SQL?

Ans.

Following are the important built-in functions available in SQL.

- AVG(): Returns the average value.
- COUNT(): Returns the number of rows.
- FIRST(): Returns the first value.
- LAST(): Returns the last value.
- MAX(): It gives the largest value as output.
- MIN(): It gives the smallest value as output.
- SUM(): Outputs the Sum.
- UCASE(): Converts a value to upper case.
- LCASE(): Converts a value to lower case.
- MID(): Extract the middle character from a string or number.
- LEN(): Returns the length of a text field.
- ROUND(): Round Off a numeric field to the number of decimals specified.
- NOW(): Returns the current system date and time.
- FORMAT(): defines how a field is to be displayed.

Q-18. What are Triggers? What are its benefits? Can we invoke a trigger explicitly?

Ans.

The trigger is a type of stored program, which gets fired automatically when some event occurs. We write a Trigger as a response to either of the following event:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
- SQL allows defining Trigger on the table, view, schema, or database associated with the event.

Following are its benefits:

- Generating some derived column values automatically.
- Enforcing referential integrity.
- Event logging and storing information on table access.
- Auditing.
- Synchronous replication of tables.
- Imposing security authorizations.
- Preventing invalid transactions.

It is not possible to invoke a trigger explicitly. It gets invoked automatically if an event gets executed on the table having an association with the trigger.

Q-19. What are the Clauses available in SQL?

Ans. Following are some of the frequently used Clauses in SQL:

1. WHERE:

Using WHERE clause, we can specify selection criteria to select required records from a table.

Syntax:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE condition1 [AND [OR]] condition2.....;
```

2. ORDER BY:

It is used to sort the records in your result set.

Syntax:

```
SELECT expressions
FROM tables
[WHERE conditions]
ORDER BY expression [ ASC | DESC ];
```

3. TOP:

It specifies the number of records to return. This clause is valuable for large tables with thousands of records.

Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name;
```

4. GROUP BY:

It is used to group values from a column, and, if required, perform calculations on that column. It applies to aggregate functions such as SUM, AVG, MAX, MIN, and COUNT.

Syntax:

```
SELECT
c1, c2,..., cn, aggregate_function(ci)
FROM
table
WHERE
where_conditions
GROUP BY c1 , c2,...,cn;
```

5. HAVING:

It is used in the SELECT statement to specify filter conditions for a group of rows or aggregates.

The MySQL HAVING clause is frequently used with the GROUP BY clause to apply a filter condition to the columns that appear in GROUP BY clause. The HAVING clause behaves like the WHERE clause in case the GROUP BY clause gets excluded.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

6. UNION:

It is used to combine the result-set of two or more SELECT statements. It is important to take care that each SELECT statement within the UNION has the same number of columns. Also, these columns have same data types or convertible data type and their order in the SELECT statement is also same.

Syntax:

```
SELECT column_name(s) FROM table1
UNION [DISTINCT | ALL]
SELECT column_name(s) FROM table2;
```

7. BETWEEN:

It is used to fetch values within a range in a SELECT, INSERT, UPDATE, or DELETE statement. These values can be numbers, text, or dates.

Syntax:

expression BETWEEN value1 AND value2;

Thus BETWEEN clause returns the records where expression lies within the range of value1 and value2.

Q-20. What is the purpose of isolation levels in SQL?

Ans.

Transactions use an isolation level that specifies the extent to which a transaction must be isolated from any data modifications caused by other transactions. These also help in identifying which concurrency side-effects are permissible.

Please refer the below list for more clarity on the different type of levels.

i. Read Committed.

It ensures that SELECT query will use committed values of the table only. If there is any active transaction on the table in some other session, then the SELECT query will wait for any such transactions to complete. Read Committed is the default transaction isolation level.

ii Read Uncommitted.

There is a transaction to update a table. But, it is not able to reach to any of these states like complete, commit or rollback. Then these values get displayed (as Dirty Read) in SELECT query of “Read Uncommitted” isolation transaction.

iii. Repeatable Read.

This level doesn’t guarantee that reads are repeatable. But it does ensure that data won’t change for the life of the transaction once.

iv. Serializable.

It is similar to Repeatable Read level. The only difference is that it stops Phantom Read and utilizes the range lock. If the table has an index, then it secures the records based on the range defined in the WHERE clause (like where ID between 1 and 3). If a table does not have an index, then it locks complete table.

v. Snapshot.

It is similar to Serializable isolation. The difference is that Snapshot does not hold a lock on a table during the transaction. Thus allowing the table to get modified in other sessions. Snapshot isolation maintains versioning in Tempdb for old data. In case any data modification happens in other sessions then existing transaction displays the old data from Tempdb.