**SQL INTERVIEW QUESTIONS:**

**Q. What is SQL**

SQL, or Structured Query Language, is a language designed to allow both technical and non-technical users to query, manipulate, and transform data from a relational database. And due to its simplicity, SQL databases provide safe and scalable storage for millions of websites and mobile applications.

[SQL](https://www.geeksforgeeks.org/sql-tutorial/) stands for Structured Query Language, which is a standard programming language used for managing relational databases.

**Q. What is a database?**

A database is a structured form of data storage in a computer or an organized collection of data that can be accessed in various ways. The Database Management System allows a user to interact with the database.

**Q. Does SQL support programming language features?**

No, SQL does not support programming language features as it is a command language. It does not have conditional statements like if-else or loops. It only has commands that can be use to query, update, delete etc data in a database.

**Q. 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.

 A query in itself is just a statement which declares what data we are looking for, where to find it in the database, and optionally, how to transform it before it is returned.

**Q. What is a sub-query? 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.

There are two types of sub-query - **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.

**Q. 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).

**Q. What do you mean by 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 labelled as the child table, and the table containing the candidate key is labelled as the referenced or parent table.

**Q. What do you mean by Data Definition Language?**

DDL allows the execution of queries that define the data like Create, Drop, Alter

**Q. What do you mean by Data Manipulation Language?**

DML is used to access or manipulate the data in the database. It allows us to perform the following functions:

* Insert data or rows in a DB
* Delete data from the DB
* Retrieve or fetch data
* Update data

**Q. 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.

**Q. 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.

Did you know?

There are many popular SQL databases including SQLite, MySQL, Postgres, Oracle and Microsoft SQL Server. All of them support the common SQL language standard but each implementation can differ in the additional features and storage types it supports.

**Q. What is Relational DB (RDBMS)?**

A relational database represents a collection of related (two-dimensional) tables. Each of the tables are similar to an Excel spreadsheet, with a fixed number of named columns (the attributes or properties of the table) and any number of rows of data.

**Q. 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.

**Q. 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.

**Q. 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;

**Q. 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.

**Q. 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

**Q. Use of Aliases**

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 \*/

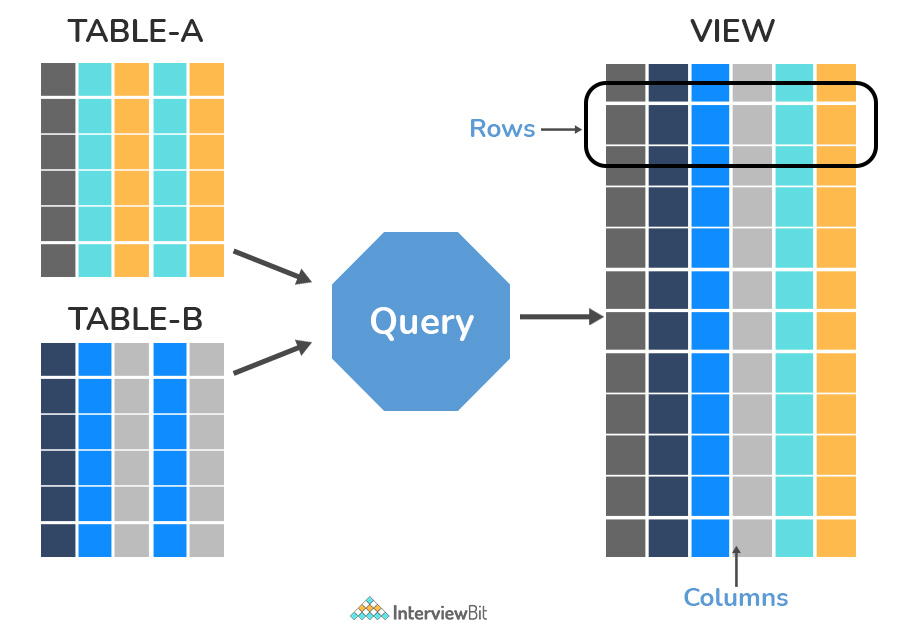
WHERE A.emp\_sup = B.emp\_id;

Aliases can be useful when:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

**Q. 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.



**Q. 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.

**Q. 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.

**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. |
| 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** |
| --- | --- |
| 1 | Until the Day I Die (Emily Carpenter) |
| 1 | Inception (Christopher Nolan) |
| 2 | The Alchemist (Paulo Coelho) |
| 2 | Inferno (Dan Brown) |
| 3 | Beautiful Bad (Annie Ward) |
| 3 | Woman 99 (Greer Macallister) |
| 4 | Dracula (Bram Stoker) |

**Example 2** - Consider the following dependencies in relation to R(W,X,Y,Z)

WX -> Y [W **and** X together determine Y]

XY -> Z [X **and** Y together determine Z]

Here, WX is the only candidate key and there is no partial dependency, i.e., any proper subset of WX doesn’t determine any non-prime attribute in the relation.

* **Third Normal Form**

A relation is said to be in the third normal form, if it satisfies the conditions for the second normal form and there is **no transitive dependency** between the non-prime attributes, i.e., all non-prime attributes are determined only by the candidate keys of the relation and not by any other non-prime attribute.

**Example 1** - Consider the Students Table in the above example. As we can observe, the Students Table in the 2NF form has a single candidate key Student\_ID (primary key) that can uniquely identify all records in the table. The field Salutation (non-prime attribute), however, depends on the Student Field rather than the candidate key. Hence, the table is not in 3NF. To convert it into the 3rd Normal Form, we will once again partition the tables into two while specifying a new ***Foreign Key*** constraint to identify the salutations for individual records in the Students table. The ***Primary Key*** constraint for the same will be set on the Salutations table to identify each record uniquely.

**Students Table (3rd Normal Form)**

| **Student\_ID** | **Student** | **Address** | **Salutation\_ID** |
| --- | --- | --- | --- |
| 1 | Sara | Amanora Park Town 94 | 1 |
| 2 | Ansh | 62nd Sector A-10 | 2 |
| 3 | Sara | 24th Street Park Avenue | 3 |
| 4 | Ansh | Windsor Street 777 | 1 |

**Books Table (3rd Normal Form)**

| **Student\_ID** | **Book Issued** |
| --- | --- |
| 1 | Until the Day I Die (Emily Carpenter) |
| 1 | Inception (Christopher Nolan) |
| 2 | The Alchemist (Paulo Coelho) |
| 2 | Inferno (Dan Brown) |
| 3 | Beautiful Bad (Annie Ward) |
| 3 | Woman 99 (Greer Macallister) |
| 4 | Dracula (Bram Stoker) |

**Salutations Table (3rd Normal Form)**

| **Salutation\_ID** | **Salutation** |
| --- | --- |
| 1 | Ms. |
| 2 | Mr. |
| 3 | Mrs. |

**Example 2** - Consider the following dependencies in relation to R(P,Q,R,S,T)

P -> QR [P together determine C]

RS -> T [B **and** C together determine D]

Q -> S

T -> P

For the above relation to exist in 3NF, all possible candidate keys in the above relation should be {P, RS, QR, T}.

* **Boyce-Codd Normal Form**

A relation is in Boyce-Codd Normal Form if satisfies the conditions for third normal form and for every functional dependency, Left-Hand-Side is super key. In other words, a relation in BCNF has non-trivial functional dependencies in form X –> Y, such that X is always a super key. For example - In the above example, Student\_ID serves as the sole unique identifier for the Students Table and Salutation\_ID for the Salutations Table, thus these tables exist in BCNF. The same cannot be said for the Books Table and there can be several books with common Book Names and the same Student\_ID

**Q. Difference between having and where.**

The **WHERE** and **HAVING** clauses in SQL are both used to filter data in queries, but they are applied at different stages of query processing and have distinct purposes.

**Key Differences:**

| **Feature** | **WHERE Clause** | **HAVING Clause** |
| --- | --- | --- |
| **Purpose** | Filters rows before any grouping or aggregation occurs. | Filters groups or aggregated data after grouping. |
| **Applies To** | Individual rows in a table. | Grouped rows or aggregate functions (e.g., SUM, COUNT). |
| **Use with Aggregates** | Cannot use aggregate functions (e.g., SUM, COUNT) directly. | Can use aggregate functions directly in the condition. |
| **Order of Execution** | Executes before the GROUP BY clause. | Executes after the GROUP BY clause. |

**Examples**

**1. Using WHERE Clause**

The WHERE clause filters rows based on a condition before any grouping or aggregation.

sql

CopyEdit

SELECT EmployeeID, Salary

FROM Employees

WHERE Salary > 50000;

* This query filters employees with a Salary greater than 50,000.

**2. Using HAVING Clause**

The HAVING clause filters aggregated data (results of a GROUP BY operation).

sql

CopyEdit

SELECT DepartmentID, AVG(Salary) AS AverageSalary

FROM Employees

GROUP BY DepartmentID

HAVING AVG(Salary) > 60000;

* This query calculates the average salary for each department and filters only those departments where the average salary exceeds 60,000.

**3. Combining WHERE and HAVING**

You can use both clauses in the same query for different purposes:

sql

CopyEdit

SELECT DepartmentID, COUNT(\*) AS EmployeeCount

FROM Employees

WHERE Status = 'Active' -- Filters individual rows before grouping

GROUP BY DepartmentID

HAVING COUNT(\*) > 5; -- Filters groups after grouping

* The WHERE clause filters out inactive employees.
* The HAVING clause filters departments with more than 5 active employees.

**Summary:**

* **WHERE**: Use for filtering raw data before grouping or aggregation.
* **HAVING**: Use for filtering aggregated results after grouping.

**Q. difference between drop, delete and truncate,**

The **DROP**, **DELETE**, and **TRUNCATE** commands in SQL are used to remove data or database objects, but they serve different purposes and work in distinct ways. Below is a detailed comparison:

**1. DROP**

The **DROP** command is used to remove entire database objects (e.g., tables, views, databases) permanently from the database.

* **Effect**: Removes the object and all its data permanently.
* **Use**: To delete the structure of a table, database, or other database object.
* **Rollback**: Cannot be rolled back (irreversible).
* **Example**:

sql

CopyEdit

DROP TABLE Employees; -- Deletes the entire Employees table including its data

DROP DATABASE CompanyDB; -- Deletes the entire database

**2. DELETE**

The **DELETE** command is used to remove specific rows from a table.

* **Effect**: Removes rows based on a condition (or all rows if no condition is specified).
* **Use**: To delete specific data while retaining the table structure.
* **Rollback**: Can be rolled back if used within a transaction (BEGIN TRANSACTION and COMMIT).
* **Can Use WHERE**: Yes, to filter rows to delete.
* **Example**:

sql

CopyEdit

DELETE FROM Employees WHERE EmployeeID = 101; -- Deletes a specific row

DELETE FROM Employees; -- Deletes all rows, but keeps the table structure

**3. TRUNCATE**

The **TRUNCATE** command is used to quickly and efficiently delete all rows from a table.

* **Effect**: Removes all rows but retains the table structure (like DELETE without a WHERE clause).
* **Use**: To delete all rows from a table when you don't need to track individual deletions.
* **Rollback**: Cannot be rolled back in some databases (depends on database engine).
* **Performance**: Faster than DELETE because it does not log individual row deletions.
* **Can Use WHERE**: No, it always deletes all rows.
* **Example**:

sql

CopyEdit

TRUNCATE TABLE Employees; -- Deletes all rows and resets any identity columns

**Comparison Table:**

| **Feature** | **DROP** | **DELETE** | **TRUNCATE** |
| --- | --- | --- | --- |
| **Scope** | Entire table or database object | Specific rows | All rows in a table |
| **Retains Table Structure** | No | Yes | Yes |
| **Can Use WHERE Clause** | No | Yes | No |
| **Logs Deletions** | No | Yes | Minimal or no logging |
| **Rollback Possible** | No | Yes | Depends on database engine |
| **Resets Identity Columns** | Yes | No | Yes |
| **Use Case** | Dropping objects like tables | Removing specific rows of data | Clearing all rows from a table |

**Summary:**

* Use **DROP** to completely remove a table, database, or other objects.
* Use **DELETE** to remove specific rows while retaining the table and its structure.
* Use **TRUNCATE** to quickly remove all rows from a table when you don’t need to retain data or log individual row deletions.

**Q. Types of SQL commands**

**Q. 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 a given length.
* ROUND() - Calculates the round-off integer value for a numeric field (or decimal point values).
* NOW() - Returns the current date & time.
* FORMAT() - Sets the format to display a collection of values.

**Q. Subquery in SQL**

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.

**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.

**What is User-defined function? What are its various types?**

The user-defined functions in SQL are like functions in any other programming language that accept parameters, perform complex calculations, and return a value. They are written to use the logic repetitively whenever required. There are two types of SQL user-defined functions:

* Scalar Function: As explained earlier, user-defined scalar functions return a single scalar value.
* Table-Valued Functions: User-defined table-valued functions return a table as output.
  + **Inline:** returns a table data type based on a single SELECT statement.
  + **Multi-statement:** returns a tabular result-set but, unlike inline, multiple SELECT statements can be used inside the function body.

**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.