**SQL Questions & Answers:**

**1)What is SQL?**

SQL stands for Structured Query Language , and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.

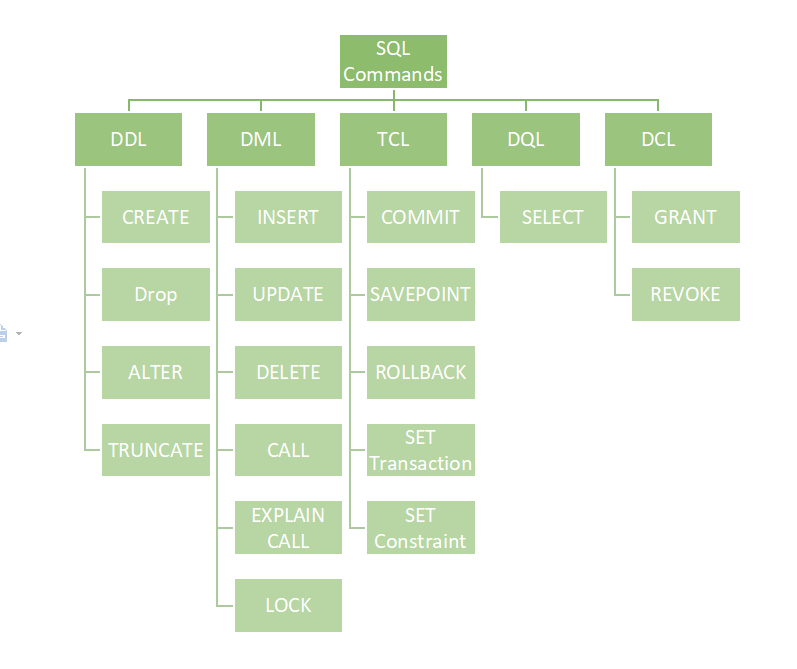
### What are the usages of SQL?

SQL is responsible for maintaining the relational data and the data structures present in the database. Some of the common usages are given below:

* To execute queries against a database
* To retrieve data from a database
* To inserts records in a database
* To updates records in a database
* To delete records from a database
* To create new databases
* To create new tables in a database
* To create views in a database
* To perform complex operations on the database.

### 2) What are the subsets of SQL?

The following are the four significant subsets of the SQL:



* **Data definition language (DDL):** It defines the data structure that consists of commands like CREATE, ALTER, DROP, etc.
* **Data manipulation language (DML):** It is used to manipulate existing data in the database. The commands in this category are SELECT, UPDATE, INSERT, etc.
* **Data control language (DCL):** It controls access to the data stored in the database. The commands in this category include GRANT and REVOKE.
* **Transaction Control Language (TCL):** It is used to deal with the transaction operations in the database. The commands in this category are COMMIT, ROLLBACK, SET TRANSACTION, SAVEPOINT, etc.

### 3) What is the purpose of DDL Language?

DDL stands for Data definition language. It is the subset of a database that defines the data structure of the database when the database is created. **For example,** we can use the DDL commands to add, remove, or modify tables. It consists of the following commands: CREATE, ALTER and DELETE database objects such as schema, tables, indexes, view, sequence, etc.

**Example:**

1. CREATE TABLE Students
2. (
3. Roll\_no INT,
4. Name VARCHAR(45),
5. Branch VARCHAR(30),
6. );

4) What is the purpose of DML Language?

Data manipulation language makes the user able to retrieve and manipulate data in a relational database. The DML commands can only perform read-only operations on data. We can perform the following operations using DDL language:

* Insert data into the database through the INSERT command.
* Retrieve data from the database through the SELECT command.
* Update data in the database through the UPDATE command.
* Delete data from the database through the DELETE command.

**Example**

1. INSERT INTO Student VALUES (111, 'George', 'Computer Science')

### 5) What is the purpose of DCL Language?

Data control language allows users to control access and permission management to the database. It is the subset of a database, which decides that what part of the database should be accessed by which user at what point of time. It includes two commands, GRANT and REVOKE.

**GRANT:** It enables system administrators to assign privileges and roles to the specific user accounts to perform specific tasks on the database.

**REVOKE:** It enables system administrators to revoke privileges and roles from the user accounts so that they cannot use the previously assigned permission on the database.

**Example**

1. GRANT \* ON mydb.Student TO javatpoint@localhsot;

### 6) What is the difference between a primary key and a unique key?

|  |  |
| --- | --- |
| **Primary Key** | **Unique Key** |
| The primary key act as a unique identifier for each record in the table. | The unique key is also a unique identifier for records when the primary key is not present in the table. |
| We cannot store NULL values in the primary key column. | We can store NULL value in the unique key column, but only one NULL is allowed. |
| We cannot change or delete the primary key column values. | We can modify the unique key column values. |

7) What is Normalization in a Database?

Normalization is used to minimize redundancy and dependency by organizing fields and table of a database.

There are some rules of database normalization, which is commonly known as Normal From, and they are:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce-Codd normal form(BCNF)

Using these steps, the redundancy, anomalies, inconsistency of the data in the database can be removed.

7.1) What is the primary use of Normalization?

Normalization is mainly used to add, delete or modify a field that can be made in a single table. The primary use of Normalization is to remove redundancy and remove the insert, delete and update distractions. Normalization breaks the table into small partitions and then links them using different relationships to avoid the chances of redundancy.

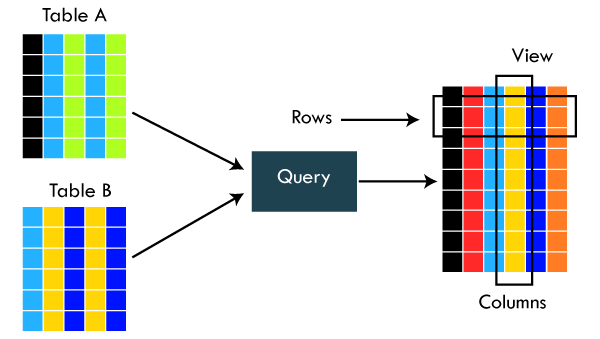
8) What are the different types of SQL operators?

Operators are the special keywords or special characters reserved for performing particular operations. They are also used in SQL queries. We can primarily use these operators within the WHERE clause of SQL commands. It's a part of the command to filters data based on the specified condition. The SQL operators can be categorized into the following types:

* **Arithmetic operators:** These operators are used to perform mathematical operations on numerical data. The categories of this operators are addition (+), subtraction (-), multiplication (\*), division (/), remainder/modulus (%), etc.
* **Logical operators:** These operators evaluate the expressions and return their results in True or False. This operator includes ALL, AND, ANY, ISNULL, EXISTS, BETWEEN, IN, LIKE, NOT, OR, UNIQUE.
* **Comparison operators:** These operators are used to perform comparisons of two values and check whether they are the same or not. It includes equal to (=), not equal to (!= or <>), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), not less than (!<), not greater than (!>), etc.
* **Bitwise operators:** It is used to do bit manipulations between two expressions of integer type. It first performs conversion of integers into binary bits and then applied operators such as AND (& symbol), OR (|, ^), NOT (~), etc.
* **Compound operators:** These operators perform operations on a variable before setting the variable's result to the operation's result. It includes Add equals (+=), subtract equals (-=), multiply equals (\*=), divide equals (/=), modulo equals (%=), etc.
* **String operators:** These operators are primarily used to perform concatenation and pattern matching of strings. It includes + (String concatenation), += (String concatenation assignment), % (Wildcard), [] (Character(s) matches), [^] (Character(s) not to match), \_ (Wildcard match one character), etc.

09) What is a view in SQL?

A view is a database object that has no values. It is a virtual table that contains a subset of data within a table. It looks like an actual table containing rows and columns, but it takes less space because it is not present physically. It is operated similarly to the base table but does not contain any data of its own. Its name is always unique. A view can have data from one or more tables. If any changes occur in the underlying table, the same changes reflected in the views also.



The primary use of a view is to implement the security mechanism. It is the searchable object where we can use a query to search the view as we use for the table. It only shows the data returned by the query that was declared when the view was created.

We can create a view by using the following syntax:

1. CREATE VIEW view\_name AS
2. SELECT column\_lists FROM table\_name
3. WHERE condition;

### 10) What is an Index in SQL?

An index is a disc structure associated with a table or view that speeds up row retrieval. It reduces the cost of the query because the query's high cost will lead to a fall in its performance. It is used to increase the performance and allow faster retrieval of records from the table. Indexing reduces the number of data pages we need to visit to find a particular data page. It also has a unique value meaning that the index cannot be duplicated. An index creates an entry for each value which makes it faster to retrieve data.

**For example:** Suppose we have a book which carries the details of the countries. If you want to find out information about India, why will you go through every page of that book? You could directly go to the index. Then from the index, you can go to that particular page where all the information about India is given.

### 10.1) What are the different types of indexes in SQL?

SQL indexes are nothing more than a technique of minimizing the query's cost. The higher the query's cost, the worse the query's performance. The following are the different types of Indexes supported in SQL:

* Unique Index
* Clustered Index
* Non-Clustered Index
* Bit-Map Index
* Normal Index
* Composite Index
* B-Tree Index
* Function-Based Index

11) What is the difference between clustered and non-clustered indexes in SQL?

Indexing is a method to get the requested data very fast. There are mainly two types of indexes in SQL, clustered index and non-clustered index. The differences between these two indexes are very important from an SQL performance perspective. The following comparison chart explains their main differences:

|  |  |
| --- | --- |
| **Clustered Index** | **Non-Clustered Index** |
| A clustered index is a table or view where the data for the rows are stored. In a relational database, if the table column contains a primary key, MySQL automatically creates a clustered index named PRIMARY. | The indexes other than PRIMARY indexes (clustered indexes) are called non-clustered indexes. It has a structure separate from the data row. The non-clustered indexes are also known as secondary indexes. |
| Clustered indexes store the data information and the data itself. | Non-clustered indexes stores only the information, and then it will refer you to the data stored in clustered data. |
| There can only be one clustered index per table. | There can be one or more non-clustered indexes in a table. |
| A clustered index determines how data is stored physically in the table. Therefore, reading from a clustered index is faster. | It creates a logical ordering of data rows and uses pointers for accessing the physical data files. Therefore, reading from a clustered index is slower. |
| A clustered index always contains an index id of 0. | A non-clustered index always contains an index id>0. |

### 12) Which are joins in SQL? Name the most commonly used SQL joins?

SQL joins are used to retrieve data from multiple tables into a meaningful result set. It is performed whenever you need to fetch records from two or more tables. They are used with SELECT statement and join conditions.

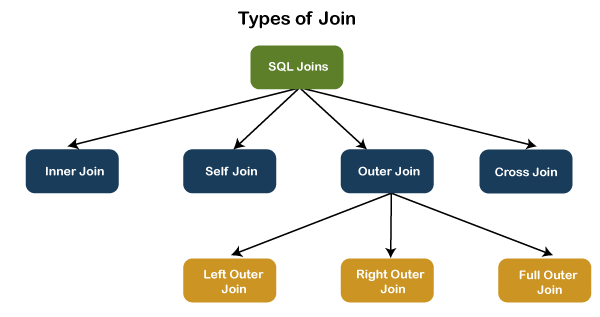
The following are the most commonly used joins in SQL:

* INNER JOIN
* LEFT OUTER JOIN
* RIGHT OUTER JOIN

### 12.1) What are the different types of joins in SQL?

Joins are used to merge two tables or retrieve data from tables. It depends on the relationship between tables. According to the ANSI standard, the following are the different types of joins used in SQL:

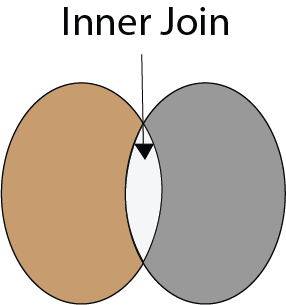
* INNER JOIN
* SELF JOIN
* LEFT OUTER JOIN
* RIGHT OUTER JOIN
* FULL OUTER JOIN
* CROSS JOIN



### 12.2) What is INNER JOIN in SQL?

Inner join returns only those records from the tables that match the specified condition and hides other rows and columns. In simple words, it fetches rows when there is at least one match of rows between the tables is found. INNER JOIN keyword joins the matching records from two tables. It is assumed as a default join, so it is optional to use the INNER keyword with the query.

The below visual representation explain this join more clearly:



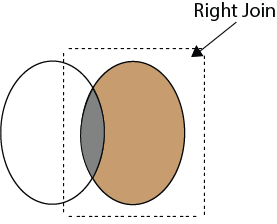
The following syntax illustrates the INNER JOIN:

1. SELECT column\_lists
2. FROM table1
3. INNER JOIN table2 ON join\_condition1
4. INNER JOIN table3 ON join\_condition2
5. ...;

### 12.3) What is the Right JOIN in SQL?

The Right join is used to retrieve all rows from the right-hand table and only those rows from the other table that fulfilled the join condition. It returns all the rows from the right-hand side table even though there are no matches in the left-hand side table. If it finds unmatched records from the left side table, it returns a Null value. This join is also known as Right Outer Join.

The below visual representation explain this join more clearly:



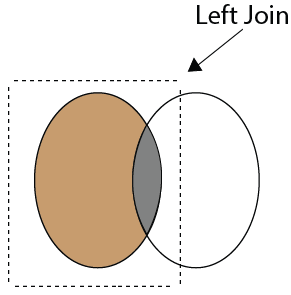
The following syntax illustrates the RIGHT JOIN:

1. SELECT colum\_lists
2. FROM table1
3. RIGHT JOIN table2
4. ON join\_condition;

### 12.4) What is Left Join in SQL?

The Left Join is used to fetch all rows from the left-hand table and common records between the specified tables. It returns all the rows from the left-hand side table even though there are no matches on the right-hand side table. If it will not find any matching record from the right side table, then it returns null. This join can also be called a Left Outer Join.

The following visual representation explains it more clearly:



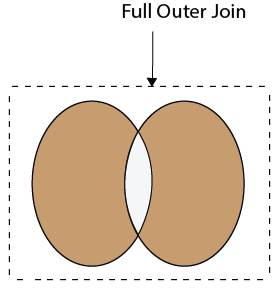
The following syntax illustrates the RIGHT JOIN:

1. SELECT colum\_lists
2. FROM table1
3. LEFT JOIN table2
4. ON join\_condition;

### 12.5) What is Full Join in SQL?

The Full Join results from a combination of both left and right join that contains all the records from both tables. It fetches 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 tables. If a match is not found, it puts NULL value. It is also known as FULL OUTER JOIN.

The following visual representation explains it more clearly:



The following syntax illustrates the FULL JOIN:

1. SELECT \* FROM table1
2. FULL OUTER JOIN table2
3. ON join\_condition;

### 13) What is a "TRIGGER" in SQL?

A trigger is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. It allows us to execute a batch of code when an insert, update or delete command is run against a specific table because the trigger is the set of activated actions whenever DML commands are given to the system.

SQL triggers have two main components one is action, and another is an event. When certain actions are taken, an event occurs as a result of those actions.

syntax:

1. CREATE TRIGGER trigger\_name
2. (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
3. ON table\_name FOR EACH ROW
4. BEGIN
5. --variable declarations
6. --trigger code
7. END;

### 14) How to write an SQL query to find students' names start with 'A'?

We can write the following query to get the student details whose name starts with A:

1. SELECT \* FROM student WHERE stud\_name like 'A%';

Here is the demo example where we have a table named student that contains two names starting with the 'A' character.

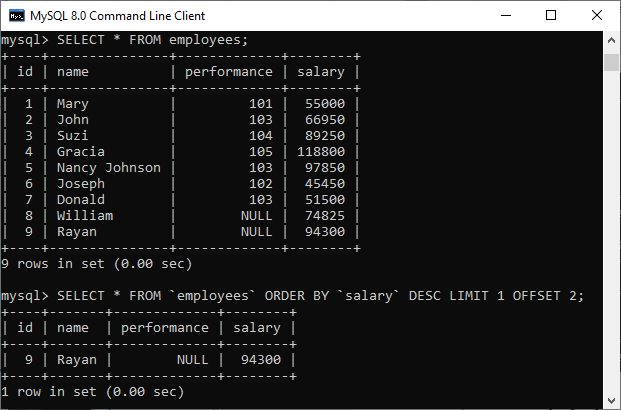
### SQL Interview Questions and Answers

### 15) Write the SQL query to get the third maximum salary of an employee from a table named employees.

The following query is the simplest way to get the third maximum salary of an employee:

1. SELECT \* FROM `employees` ORDER BY `salary` DESC LIMIT 1 OFFSET 2

Here is the demo example that shows how to get the third maximum salary of an employee.



The following are the alternative way to get the third-highest salary of an employee:

**A. Using LIMIT Keyword**

1. SELECT salary FROM employees
2. ORDER BY salary DESC
3. LIMIT 2, 1;

**B. Using Subquery**

1. SELECT salary
2. FROM
3. (SELECT salary
4. FROM employees
5. ORDER BY salary DESC
6. LIMIT 3) AS Temp
7. ORDER BY salary LIMIT 1;

**C. Using TOP Keyword**

1. SELECT TOP 1 salary
2. FROM
3. (SELECT DISTINCT TOP 3 salary
4. FROM employees
5. ORDER BY salary DESC) AS Temp
6. ORDER BY salary ASC

16) What is the difference between DELETE and TRUNCATE statements in SQL?

The main difference between them is that the delete statement deletes data without resetting a table's identity, whereas the truncate command resets a particular table's identity. The following comparison chart explains it more clearly:

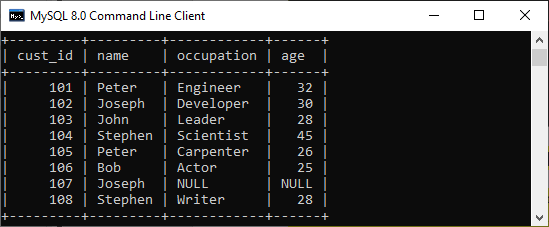
|  |  |  |
| --- | --- | --- |
| **No.** | **DELETE** | **TRUNCATE** |
| 1) | The delete statement removes single or multiple rows from an existing table depending on the specified condition. | The truncate command deletes the whole contents of an existing table without the table itself. It preserves the table structure or schema. |
| 2) | DELETE is a **DML command.** | TRUNCATE is a **DML command.** |
| 3) | We **can use the WHERE** clause in the DELETE command. | We **cannot use the WHERE** clause with TRUNCATE. |
| 4) | DELETE statement is used **to delete a row** from a table. | TRUNCATE statement is used **to remove all the rows** from a table. |
| 5) | DELETE is **slower** because it maintained the log. | TRUNCATE statement is **faster** than DELETE statement as it deletes entire data at a time without maintaining transaction logs. |
| 6) | You **can roll back** data after using the DELETE statement. | It is **not possible to roll back** after using the TRUNCATE statement. |
| 7) | DELETE query **takes more space.** | TRUNCATE query **occupies less space.** |

### 17) How do we use the DISTINCT statement? What is its use?

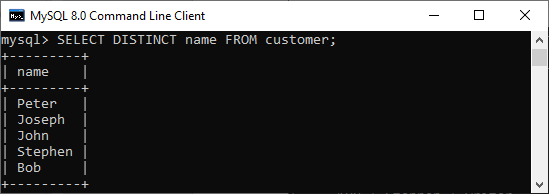
The DISTINCT keyword is used to ensure that the fetched value always has unique values. It does not allow to have duplicate values. The DISTINCT keyword is used with the SELECT statement and retrieves different values from the table's column. We can use it with the help of the following syntax:

1. SELECT DISTINCT column\_lists FROM table\_name WHERE [condition];

Suppose we have a table 'customer' containing eight records in which the name column has some duplicate values.



If we want to get the name column without any duplicate values, the DISTINCT keyword is required. Executing the below command will return a name column with unique values.



18) What is the difference between the WHERE and HAVING clauses?

The main difference is that the WHERE clause is used to filter records before any groupings are established, whereas the HAVING clause is used to filter values from a group. The below comparison chart explains the most common differences:

|  |  |
| --- | --- |
| **WHERE** | **HAVING** |
| This clause is implemented in row operations. | This clause is implemented in column operations. |
| It does not allow to work with aggregate functions. | It can work with aggregate functions. |
| This clause can be used with the SELECT, UPDATE, and DELETE statements. | This clause can only be used with the SELECT statement. |

### 19) How many Aggregate functions are available in SQL?

**SQL provides seven (7) aggregate functions, which are given below:**

* **AVG():** This function is used to returns the average value from specified columns.
* **COUNT():** This function is used to returns the number of table rows, including rows with null values.
* **MAX():** This function is used to returns the largest value among the group.
* **MIN():** This function is used to returns the smallest value among the group.
* **SUM():** This function is used to returns the total summed values(non-null) of the specified column.
* **FIRST()**: This function is used to returns the first value of an expression.
* **LAST()**: This function is used to returns the last value of an expression.

**20) What is AUTO\_INCREMENT?**

* AUTO\_INCREMENT is used in SQL to automatically generate a unique number whenever a new record is inserted into a table.
* Since the primary key is unique for each record, this primary field is added as the AUTO\_INCREMENT field so that it is incremented when a new record is inserted.
* The AUTO-INCREMENT value starts from 1 and is incremented by 1 whenever a new record is inserted.
* Syntax:

CREATE TABLE Employee(

Employee\_id int NOT NULL AUTO-INCREMENT,

Employee\_name varchar(255) NOT NULL,

Employee\_designation varchar(255)

Age int,

PRIMARY KEY (Employee\_id)

)

**21)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.

**Syntax:**

CREATE PROCEDURE procedure\_name

AS

Sql\_statement

GO;

To execute we will use this:

EXEC procedure\_name

**Example:**

We are going to create a stored procedure that will help us extract the age of the employees.

create procedure employee\_age

as

select e\_age from employee

go

Now, we will execute it.

exec employee\_age

**22) 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

**23) 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.

24) **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;

**25) 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.

**26) What are some common clauses used with SELECT query in SQL?**

Some common SQL clauses used in conjuction 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;

**27) 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

### ****28)What is the difference between Union and Union All command?****

This is one of the tricky SQL Interview Questions. Interviewer may ask you this question in another way as what are the advantages of Union All over Union.

Both Union and Union All concatenate the result of two tables but the way these two queries handle duplicates are different.

**Union:**It omits duplicate records and returns only distinct result set of two or more select statements.  
**Union All:** It returns all the rows including duplicates in the result set of different select statements.

Performance wise Union All is faster than Union, Since Union All doesn’t remove duplicates. Union query checks the duplicate values which consumes some time to remove the duplicate records.

Assume: Table1 has 10 records, Table2 has 10 records. Last record from both the tables are same.

If you run Union query.

SELECT \* FROM Table1

UNION

SELECT \* FROM Table2

**Output:** Total 19 records

If you run Union query.

SELECT \* FROM Table1

UNION ALL

SELECT \* FROM Table2

**Output:** Total 20 records

### ****29)What is ETL in SQL?****

ETL stands for Extract, Transform and Load. It is a three-step process, where we would have to start off by extracting the data from sources. Once we collate the data from different sources, what we have is raw data. This raw data has to be transformed into the tidy format, which will come in the second phase. Finally, we would have to load this tidy data into tools which would help us to find insights.

**30)What is a schema in SQL?**

Our database comprises of a lot of different entities such as tables, stored procedures, functions, database owners and so on. To make sense of how all these different entities interact, we would need the help of schema. So, you can consider schema to be the logical relationship between all the different entities which are present in the database.

Once we have a clear understanding of the schema, this helps in a lot of ways:

* We can decide which user has access to which tables in the database.
* We can modify or add new relationships between different entities in the database.

Overall, you can consider a schema to be a blueprint for the database, which will give you the complete picture of how different objects interact with each other and which users have access to different entities.

### ****31)Difference between NVL and NVL2 functions?****

Both the NVL(exp1, exp2) and NVL2(exp1, exp2, exp3) functions check the value exp1 to see if it is null. With the NVL(exp1, exp2) function, if exp1 is not null, then the value of exp1 is returned; otherwise, the value of exp2 is returned, but case to the same data type as that of exp1. With the NVL2(exp1, exp2, exp3) function, if exp1 is not null, then exp2 is returned; otherwise, the value of exp3 is returned.

# **SQL Scenario based Interview Questions**

## **1)How to select UNIQUE records from a table using a SQL Query?**

* 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 |

**METHOD-1: Using GROUP BY Function**

**GROUP BY clause is used with SELECT statement to collect data from multiple records and group the results by one or more columns. The GROUP BY clause returns one row per group. By applying GROUP BY function on all the source columns, unique records can be queried from the table.**

Below is the query to fetch the unique records using GROUP BY function.

**Query:**

SELECT EMPLOYEE\_ID,

NAME,

SALARY

FROM EMPLOYEE

GROUP BY EMPLOYEE\_ID, NAME, SALARY;

**METHOD-2: Using ROW\_NUMBER Analytic Function**

**The ROW\_NUMBER Analytic function is used to provide consecutive numbering of the rows in the result by the ORDER selected for each PARTITION specified in the OVER clause. It will assign the value 1 for the first row and increase the number of the subsequent rows.**

Using ROW\_NUMBER Analytic function, assign row numbers to each unique set of records.

**Query:**

SELECT EMPLOYEE\_ID,

NAME,

SALARY,

ROW\_NUMBER() OVER(PARTITION BY EMPLOYEE\_ID,NAME,SALARY ORDER BY EMPLOYEE\_ID) AS ROW\_NUMBER

FROM EMPLOYEE;

**Once row numbers are assigned, by querying the rows with row number 1 will give the unique records from the table.**

**Query:**

SELECT EMPLOYEE\_ID, NAME, SALARY

FROM( SELECT

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;

**2)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 |

**METHOD-1: 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);

**METHOD-2: Using SQL Analytic Functions**

**Query:**

**The DENSE\_RANK is an analytic function that calculates the rank of a row in an ordered set of rows starting from 1. Unlike the RANK function, the DENSE\_RANK function returns rank values as consecutive integers.**

SELECT Employee\_Id,

Name,

Salary

FROM(

SELECT Employees.\*,

DENSE\_RANK() OVER(ORDER BY Salary DESC) as SALARY\_RANK

FROM Employees)

WHERE SALARY\_RANK =2

### What is Query to find Second highest salary for employee?

This is most asked Real Time Scenarios in SQL in many industries. There are lot of real time situation where user needs to deal with this kind of situation. User will try multiple queries to find out the same result.

Query 1 :

**Select distinct Salary from Employee e1 where 2=Select count(distinct Salary) from Employee e2 where e1.salary<=e2.salary**;

Query 2:

**select min(salary)from(select distinct salary from emp order by salary desc)where rownum<=2;**

Query 3:

**select \* from(Select S.\*,DENSE\_RANK() OVER (PARTITION BY DNO ORDER BY SALARY DESC) DR from Source) S Where S.DR=2;**

### 3)Fetching Nth Record from the table.

There are some situations where user needs to find out the Nth records from the table. I will divide this scenario in to 3 parts for better understanding of people.

Query 1 :  Query to find First Record from the table.

**Select \* from Employee where [Rownum](http://www.complexsql.com/rowid-rownum/" \t "_blank)=1;**

Query 2: Query to find last record from the table.

**Select \* from Employee where Rowid= select max(Rowid) from Employee;**

Query 3 : Query to find Nth Record from the table.

**select \* from ( select a.\*, rownum rnum from ( YOUR\_QUERY\_GOES\_HERE — including the order by ) a where rownum <= N\_ROWS ) where rnum >= N\_ROWS;**

### ****4)Find and delete duplicate rows****

There are real world situations where user needs to find and delete duplicate rows from the table. These are most used SQL queries in real world to find the duplicate rows and delete it. When there is a situation where user needs to add unique constraint to column,user needs to delete duplicate rows.

Query 1 :  Query to find duplicate rows.

**select a.\* from Employee a where rowid !=**  
**(select max(rowid) from Employee b where  a.Employee\_num =b.Employee\_num;**

Query 2: Query to delete duplicate rows

**Delete from Employee a where rowid !=** **(select max(rowid) from Employee b where  a.Employee\_num =b.Employee\_num;**

### 5)How to create a table which has same structure  or how to create duplicate table.

There are so many situations where user needs to create duplicate tables for testing purpose. There are some needs where user needs to create the structure of the table. The following are 2 most important queries which are used in 90% of Real Time Scenarios in SQL.

Query 1: Create the duplicate table with data

**Create table Employee\_1 as Select \* from Employee;**

Query 2: Create the table structure duplicate to another table.

**Create table Employee\_1 as Select \* from Employee where 1=2;**

### ****6)Write a query to find out the data between ranges?****

In day-to-day activities, the user needs to find out the data between some range. To achieve this user needs to use Between..and operator or Greater than and less than the operator.

Query 1: Using Between..and operator

Select \* from Employee where salary between 25000 and 50000;

Query 2: Using operators (Greater than and less than)

Select \* from Employee where salary >= 25000 and salary <= 50000;

# SQL Query for Practice

**Table – EmployeeDetails**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmpId** | **FullName** | **ManagerId** | **DateOfJoining** | **City** |
| 121 | John Snow | 321 | 01/31/2014 | Toronto |
| 321 | Walter White | 986 | 01/30/2015 | California |
| 421 | Kuldeep Rana | 876 | 27/11/2016 | New Delhi |

**Table – EmployeeSalary**

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Project** | **Salary** | **Variable** |
| 121 | P1 | 8000 | 500 |
| 321 | P2 | 10000 | 1000 |
| 421 | P1 | 12000 | 0 |

**Ques.1. Write an SQL query to fetch the EmpId and FullName of all the employees working under Manager with id – ‘986’.**  
Ans. We can use the EmployeeDetails table to fetch the employee details with a where clause for the manager-

SELECT EmpId, FullName

FROM EmployeeDetails

WHERE ManagerId = 986;

**Ques.2. Write an SQL query to fetch the different projects available from the EmployeeSalary table.**  
Ans. While referring to the EmployeeSalary table, we can see that this table contains project values corresponding to each employee, or we can say that we will have duplicate project values while selecting Project values from this table.  
  
So, we will use the distinct clause to get the unique values of the Project.

SELECT DISTINCT(Project)

FROM EmployeeSalary;

**Ques.3. Write an SQL query to fetch the count of employees working in project ‘P1’.**  
Ans. Here, we would be using aggregate function count() with the SQL **where** clause-

SELECT COUNT(\*)

FROM EmployeeSalary

WHERE Project = 'P1';

**Ques.4. Write an SQL query to find the maximum, minimum, and average salary of the employees.**Ans. We can use the aggregate function of SQL to fetch the max, min, and average values-

SELECT Max(Salary),

Min(Salary),

AVG(Salary)

FROM EmployeeSalary;

**Ques.5. Write an SQL query to find the employee id whose salary lies in the range of 9000 and 15000.**  
Ans. Here, we can use the ‘Between’ operator with a where clause.

SELECT EmpId, Salary

FROM EmployeeSalary

WHERE Salary BETWEEN 9000 AND 15000;

**Ques.6. Write an SQL query to fetch those employees who live in Toronto and work under manager with ManagerId – 321.**  
Ans. Since we have to satisfy both the conditions – employees living in ‘Toronto’ and working in Project ‘P2’. So, we will use AND operator here-

SELECT EmpId, City, ManagerId

FROM EmployeeDetails

WHERE City='Toronto' AND ManagerId='321';

**Ques.7. Write an SQL query to** f**etch all the employees who either live in California or work under a manager with ManagerId – 321.**  
Ans. This interview question requires us to satisfy either of the conditions – employees living in ‘California’ and working under Manager with ManagerId ‘321’. So, we will use the OR operator here-

SELECT EmpId, City, ManagerId

FROM EmployeeDetails

WHERE City='California' OR ManagerId='321';

**Ques.8. Write an SQL query to fetch all those employees who work on Project other than P1.**  
Ans. Here, we can use the NOT operator to fetch the rows which are not satisfying the given condition.

SELECT EmpId

FROM EmployeeSalary

WHERE NOT Project='P1';

Or using the not equal to operator-

SELECT EmpId

FROM EmployeeSalary

WHERE Project <> 'P1';

For the difference between NOT and <> SQL operators, check this link – [Difference between the NOT and != operators](https://answers.splunk.com/answers/93873/difference-between-the-not-and-operators.html).

**Ques.9. Write an SQL query to display the total salary of each employee adding the Salary with Variable value.**  
Ans. Here, we can simply use the ‘+’ operator in SQL.

SELECT EmpId,

Salary+Variable as TotalSalary

FROM EmployeeSalary;

**Ques.10. Write an SQL query to fetch the employees whose name begins with any two characters, followed by a text “hn” and ending with any sequence of characters.**  
Ans. For this question, we can create an SQL query using like operator with ‘\_’ and ‘%’ wild card characters, where ‘\_’ matches a single character and ‘%’ matches ‘0 or multiple characters’.

SELECT FullName

FROM EmployeeDetails

WHERE FullName LIKE ‘\_\_hn%’;

**Ques.11. Write an SQL query to fetch all the EmpIds which are present in either of the tables – ‘EmployeeDetails’ and ‘EmployeeSalary’.**  
Ans. In order to get unique employee ids from both the tables, we can use Union clause which can combine the results of the two SQL queries and return unique rows.

SELECT EmpId FROM EmployeeDetails

UNION

SELECT EmpId FROM EmployeeSalary;

**Ques.12. Write an SQL query to fetch common records between two tables.**  
Ans. SQL Server – Using INTERSECT operator-

SELECT \* FROM EmployeeSalary

INTERSECT

SELECT \* FROM ManagerSalary;

MySQL – Since MySQL doesn’t have INTERSECT operator so we can use the sub query-

SELECT \*

FROM EmployeeSalary

WHERE EmpId IN

(SELECT EmpId from ManagerSalary);

**Ques.13. Write an SQL query to fetch records that are present in one table but not in another table.**  
Ans. SQL Server – Using MINUS- operator-

SELECT \* FROM EmployeeSalary

MINUS

SELECT \* FROM ManagerSalary;

MySQL – Since MySQL doesn’t have MINUS operator so we can use LEFT join-

SELECT EmployeeSalary.\*

FROM EmployeeSalary

LEFT JOIN

ManagerSalary USING (EmpId)

WHERE ManagerSalary.EmpId IS NULL;

**Ques.14. Write an SQL query to fetch the EmpIds that are present in both the tables –   ‘EmployeeDetails’ and ‘EmployeeSalary.**  
Ans. Using sub query-

SELECT EmpId FROM

EmployeeDetails

where EmpId IN

(SELECT EmpId FROM EmployeeSalary);

**Ques.15. Write an SQL query to fetch the EmpIds that are present in EmployeeDetails but not in EmployeeSalary.**  
Ans. Using sub query-

SELECT EmpId FROM

EmployeeDetails

where EmpId Not IN

(SELECT EmpId FROM EmployeeSalary);

**Ques.16. Write an SQL query to fetch the employee full names and replace the space with ‘-’.**  
Ans. Using ‘Replace’ function-

SELECT REPLACE(FullName, ' ', '-')

FROM EmployeeDetails;

**Ques.17. Write an SQL query to fetch the position of a given character(s) in a field.**  
Ans. Using ‘Instr’ function-

SELECT INSTR(FullName, 'Snow')

FROM EmployeeDetails;

**Ques.18. Write an SQL query to display both the EmpId and ManagerId together.**  
Ans. Here we can use the CONCAT command.

SELECT CONCAT(EmpId, ManagerId) as NewId

FROM EmployeeDetails;

**Ques.19. Write a query to fetch only the first name(string before space) from the FullName column of the EmployeeDetails table.**  
Ans. In this question, we are required to first fetch the location of the space character in the FullName field and then extract the first name out of the FullName field.  
  
For finding the location we will use the LOCATE method in MySQL and CHARINDEX in SQL SERVER and for fetching the string before space, we will use the SUBSTRING OR MID method.  
  
MySQL – using MID

SELECT MID(FullName, 1, LOCATE(' ',FullName))

FROM EmployeeDetails;

SQL Server – using SUBSTRING

SELECT SUBSTRING(FullName, 1, CHARINDEX(' ',FullName))

FROM EmployeeDetails;

**Ques.20. Write an SQL query to upper case the name of the employee and lower case the city values.**  
Ans. We can use SQL Upper and Lower functions to achieve the intended results.

SELECT UPPER(FullName), LOWER(City)

FROM EmployeeDetails;

**Ques.21. Write an SQL query to find the count of the total occurrences of a particular character – ‘n’ in the FullName field.**  
Ans. Here, we can use the ‘Length’ function. We can subtract the total length of the FullName field with a length of the FullName after replacing the character – ‘n’.

SELECT FullName,

LENGTH(FullName) - LENGTH(REPLACE(FullName, 'n', ''))

FROM EmployeeDetails;

**Ques.22. Write an SQL query to update the employee names by removing leading and trailing spaces.**  
Ans. Using the ‘Update’ command with the ‘LTRIM’ and ‘RTRIM’ function.

UPDATE EmployeeDetails

SET FullName = LTRIM(RTRIM(FullName));

**Ques.23. Fetch all the employees who are not working on any project.**  
Ans. This is one of the very basic interview questions in which the interviewer wants to see if the person knows about the commonly used – Is NULL operator.

SELECT EmpId

FROM EmployeeSalary

WHERE Project IS NULL;

**Ques.24. Write an SQL query to fetch employee names having a salary greater than or equal to 5000 and less than or equal to 10000.**  
Ans. Here, we will use BETWEEN in the ‘where’ clause to return the EmpId of the employees with salary satisfying the required criteria and then use it as subquery to find the fullName of the employee from EmployeeDetails table.

SELECT FullName

FROM EmployeeDetails

WHERE EmpId IN

(SELECT EmpId FROM EmployeeSalary

WHERE Salary BETWEEN 5000 AND 10000);

**Ques.25. Write an SQL query to find the current date-time.**  
Ans. MySQL-

SELECT NOW();

SQL Server-

SELECT getdate();

Oracle-

SELECT SYSDATE FROM DUAL;

**Ques.26. Write an SQL query to fetch all the Employees details from EmployeeDetails table who joined in the Year 2020.**  
Ans. Using BETWEEN for the date range ’01-01-2020′ AND ’31-12-2020′-

SELECT \* FROM EmployeeDetails

WHERE DateOfJoining BETWEEN '2020/01/01'

AND '2020/12/31';

Also, we can extract year part from the joining date (using YEAR in mySQL)-

SELECT \* FROM EmployeeDetails

WHERE YEAR(DateOfJoining) = '2020';

**Ques.27. Write an SQL query to fetch all employee records from EmployeeDetails table who have a salary record in EmployeeSalary table.**  
Ans. Using ‘Exists’-

SELECT \* FROM EmployeeDetails E

WHERE EXISTS

(SELECT \* FROM EmployeeSalary S

WHERE E.EmpId = S.EmpId);

**Ques.28. Write an SQL query to fetch project-wise count of employees sorted by project’s count in descending order.**  
Ans. The query has two requirements – first to fetch the project-wise count and then to sort the result by that count.  
  
For project-wise count, we will be using the GROUP BY clause and for sorting, we will use the ORDER BY clause on the alias of the project-count.

SELECT Project, count(EmpId) EmpProjectCount

FROM EmployeeSalary

GROUP BY Project

ORDER BY EmpProjectCount DESC;

**Ques.29. Write a query to fetch employee names and salary records. Display the employee details even if the salary record is not present for the employee.**  
Ans. This is again one of the very common interview questions in which the interviewer just wants to check the basic knowledge of SQL JOINS.  
  
Here, we can use left join with EmployeeDetail table on the left side of the EmployeeSalary table.

SELECT E.FullName, S.Salary

FROM EmployeeDetails E

LEFT JOIN

EmployeeSalary S

ON E.EmpId = S.EmpId;

**Ques.30. Write an SQL query to join 3 tables.**

Ans. Considering 3 tables TableA, TableB, and TableC, we can use 2 joins clauses like below-

SELECT column1, column2

FROM TableA

JOIN TableB ON TableA.Column3 = TableB.Column3

JOIN TableC ON TableA.Column4 = TableC.Column4;

## SQL Query for Experienced

**Ques. 31. Write an SQL query to fetch all the Employees who are also managers from the EmployeeDetails table.**  
Ans. Here, we have to use Self-Join as the requirement wants us to analyze the EmployeeDetails table as two tables. We will use different aliases ‘E’ and ‘M’ for the same EmployeeDetails table.

SELECT DISTINCT E.FullName

FROM EmployeeDetails E

INNER JOIN EmployeeDetails M

ON E.EmpID = M.ManagerID;

To learn more about Self Join along with some more queries, you can watch the below video that explains the self join concept in a very simple way.

Self Join and Its Demonstrati

**Ques.32. Write an SQL query to fetch duplicate records from EmployeeDetails (without considering the primary key – EmpId).**  
Ans. In order to find duplicate records from the table, we can use GROUP BY on all the fields and then use the HAVING clause to return only those fields whose count is greater than 1 i.e. the rows having duplicate records.

SELECT FullName, ManagerId, DateOfJoining, City, COUNT(\*)

FROM EmployeeDetails

GROUP BY FullName, ManagerId, DateOfJoining, City

HAVING COUNT(\*) > 1;

**Ques.33. Write an SQL query to remove duplicates from a table without using a temporary table.**  
Ans. Here, we can use delete with alias and inner join. We will check for the equality of all the matching records and them remove the row with higher EmpId.

DELETE E1 FROM EmployeeDetails E1

INNER JOIN EmployeeDetails E2

WHERE E1.EmpId > E2.EmpId

AND E1.FullName = E2.FullName

AND E1.ManagerId = E2.ManagerId

AND E1.DateOfJoining = E2.DateOfJoining

AND E1.City = E2.City;

**Ques.34. Write an SQL query to fetch only odd rows from the table.**  
Ans. In case we have an auto-increment field e.g. EmpId then we can simply use the below query-

SELECT \* FROM EmployeeDetails

WHERE MOD (EmpId, 2) <> 0;

In case we don’t have such a field then we can use the below queries.  
  
Using Row\_number in SQL server and checking that the remainder when divided by 2 is 1-

SELECT E.EmpId, E.Project, E.Salary

FROM (

SELECT \*, Row\_Number() OVER(ORDER BY EmpId) AS RowNumber

FROM EmployeeSalary

) E

WHERE E.RowNumber % 2 = 1;

Using a user defined variable in MySQL-

SELECT \*

FROM (

SELECT \*, @rowNumber := @rowNumber+ 1 rn

FROM EmployeeSalary

JOIN (SELECT @rowNumber:= 0) r

) t

WHERE rn % 2 = 1;

**Ques.35. Write an SQL query to fetch only even rows from the table.**  
Ans. In case we have an auto-increment field e.g. EmpId then we can simply use the below query-

SELECT \* FROM EmployeeDetails

WHERE MOD (EmpId, 2) = 0;

In case we don’t have such a field then we can use the below queries.  
  
Using Row\_number in SQL server and checking that the remainder when divided by 2 is 1-

SELECT E.EmpId, E.Project, E.Salary

FROM (

SELECT \*, Row\_Number() OVER(ORDER BY EmpId) AS RowNumber

FROM EmployeeSalary

) E

WHERE E.RowNumber % 2 = 0;

Using a user defined variable in MySQL-

SELECT \*

FROM (

SELECT \*, @rowNumber := @rowNumber+ 1 rn

FROM EmployeeSalary

JOIN (SELECT @rowNumber:= 0) r

) t

WHERE rn % 2 = 0;

**Ques.36. Write an SQL query to create a new table with data and structure copied from another table.**  
Ans.

CREATE TABLE NewTable

SELECT \* FROM EmployeeSalary;

**Ques.37. Write an SQL query to create an empty table with the same structure as some other table.**  
Ans. Here, we can use the same query as above with False ‘WHERE’ condition-

CREATE TABLE NewTable

SELECT \* FROM EmployeeSalary where 1=0;

**Ques.38. Write an SQL query to fetch top n records?**  
Ans. In MySQL using LIMIT-

SELECT \*

FROM EmployeeSalary

ORDER BY Salary DESC LIMIT N;

In SQL server using TOP command-

SELECT TOP N \*

FROM EmployeeSalary

ORDER BY Salary DESC;

**Ques.39. Write an SQL query to find the nth highest salary from table.**  
Ans, Using Top keyword (SQL Server)-

SELECT TOP 1 Salary

FROM (

SELECT DISTINCT TOP N Salary

FROM Employee

ORDER BY Salary DESC

)

ORDER BY Salary ASC;

Using limit clause(MySQL)-

SELECT Salary

FROM Employee

ORDER BY Salary DESC LIMIT N-1,1;

**Ques.40. Write SQL query to find the 3rd highest salary from a table without using the TOP/limit keyword.**  
Ans. This is one of the most commonly asked interview questions. For this, we will use a correlated subquery.  
  
In order to find the 3rd highest salary, we will find the salary value until the inner query returns a count of 2 rows having the salary greater than other distinct salaries.

SELECT Salary

FROM EmployeeSalary Emp1

WHERE 2 = (

SELECT COUNT( DISTINCT ( Emp2.Salary ) )

FROM EmployeeSalary Emp2

WHERE Emp2.Salary > Emp1.Salary

)

For nth highest salary-

SELECT Salary

FROM EmployeeSalary Emp1

WHERE N-1 = (

SELECT COUNT( DISTINCT ( Emp2.Salary ) )

FROM EmployeeSalary Emp2

WHERE Emp2.Salary > Emp1.Salary

)