



InterviewBit

SQL Query Interview Questions



To view the live version of the page, [click here](#).

© Copyright by Interviewbit

Contents

SQL Query Interview Questions for Freshers

1. Let us consider the following schema:
2. Let us consider the following schema:
3. Given the following schema:
4. Given the following schema:
5. Consider the following schema:
6. Given the following schema:
7. Consider the following table schema:

SQL Query Interview Questions for Experienced

8. Consider the following Schema:
9. Consider the following schema:
10. Given the following schema:
11. Given the following schema:
12. Given the following schema:

Let's get Started

SQL stands for Structured Query Language. Initially, it was named Structured English Query Language (SEQUEL). It is used in relational database management systems to store and manage data (RDMS).

It is a relational database system with standard language. A user can create, read, update, and delete relational databases and tables with it. SQL is the standard database language used by every RDBMS, including MySQL, Informix, Oracle, MS Access, and SQL Server. SQL allows users to query the database using English-like phrases in a variety of ways.

SQL is also distinct from other computer languages because it specifies what the user wants the machine to perform rather than how it should accomplish it. (In more technical terms, SQL is a declarative rather than procedural language.) There are no IF statements in SQL for testing conditions, and no GOTO, DO, or FOR statements for controlling program flow. SQL statements, on the other hand, indicate how a collection of data should be organized, as well as what data should be accessed or added to the database. The DBMS determines the sequence of actions to complete such tasks.



What is Query in SQL?

A [query](#) is a request for information or data from a database table or set of tables. For example, let us assume that we have a database that stores details about books written by various authors. Now, if we want to know how many books have been written by a particular author, then this question can be referred to as a query that we want to do to the database.

Use cases of SQL:

- *SQL is an interactive question language.* Users write SQL commands into interactive SQL software to extract information and display them on the screen, making it a useful and simple tool for ad hoc database queries.
- *SQL is a database programming language.* To access the information in a database, programmers incorporate SQL instructions into their utility packages. This method of database access is used by both user-written packages and database software packages
- *SQL is a server/client language.* SQL allows personal computer programs to interface with database servers that store shared data through a network. Many well-known enterprise-class apps use this client/server design.
- *SQL is a distributed database language.* SQL is used in distributed database control systems to help spread data across numerous linked computer structures. Every device's DBMS software program uses SQL to communicate with other systems, issuing requests for information access.

SQL queries are one of the most frequently asked interview questions. You can expect questions ranging from basic SQL queries to advanced SQL queries. So, without any further ado, let us look at the most frequently asked **SQL Query Interview Questions and their Answers**.

SQL Query Interview Questions for Freshers

1. Let us consider the following schema:

Table: Person

Column Name	Type
id	int
email	varchar

Here, id is the primary key column for this table. Email represents the email id of the person. For the sake of simplicity, we assume that the emails will not contain uppercase letters. Write an SQL query to report all the duplicate emails. You can return the result table in any order.

Example:

Input: Person table:



id	email
1	a@gmail.com
2	c@yahoo.com
3	a@gmail.com

Output:

Email
a@gmail.com

Explanation: a@gmail.com is repeated two times.

- **Approach 1:**

We can first have all the distinct email ids and their respective counts in our result set. For this, we can use the GROUP BY operator to group the tuples by their email id. We will use the COUNT operator to have the total number of a particular email id in the given table. The query for obtaining this resultant set can be written as:

```
select email, count(email) as email_count
from Person
group by email;
```

Now, we query in the above resultant query set to find out all the tuples which have an email id count greater than 1. This can be achieved using the following query:

```
select email from
(
  select email, count(email) as email_count
  from Person
  group by email
)
where email_count > 1;
```

- **Approach 2:**

The HAVING clause, which is significantly simpler and more efficient, is a more popular technique to add a condition to a GROUP BY. So, we can first group the tuples by the email ids and then have a condition to check if their count is greater than 1, only then do we include it in our result set. So we may change the solution above to this one.

```
select email
from Person
group by email
having count(email) > 1;
```

- **Approach 3:**

We can use the concept of joins to solve this problem. We will self-join the Person table with the condition that their email ids should be the same and their ids should be different. Having done this, we just need to count the number of tuples in our resultant set with distinct email ids. For this, we use the DISTINCT operator. This can be achieved using the following query:

```
SELECT DISTINCT p1.email
FROM Person p1, Person p2
WHERE p1.email = p2.email and p1.id != p2.id;
```

2. Let us consider the following schema:

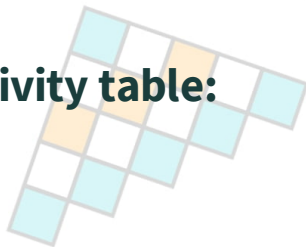
Table: Activity

Column Name	Type
playerId	int
deviceId	int
eventDate	date
gamesplayed	int

This table's primary key is (playerId, eventDate). The activities of numerous game participants are depicted in this table. Each row indicates a person that logged in and played a particular number of games (perhaps 0) before moving on to another device at a later date. Construct a SQL query to provide each player's first login date. You can return the result table in any order.

Example 1:

Input: Activity table:



PlayerId	deviceId	eventDate	gamesPlayed
1	2	2021-08-09	9
1	2	2021-04-07	3
2	3	2021-06-25	1
3	1	2021-03-02	1
3	4	2021-07-03	3

Output:

playerId	firstLogin
1	2021-04-07
2	2021-06-25
3	2021-07-03

Explanation:

The player with playerId 1 has two login event dates in the example above. However, because the first login event date is 2021-04-07, we display it. Similarly, the first login event date for the player with playerId 2 is 2021-06-25, and the first login event date for the player with playerId 3 is 2021-07-03.

- **Approach 1:**

We can first group the tuples by their player_id. Now, we want the most initial date when the player logged in to the game. For this, we can use the MIN operator and find the initial date on which the player logged in. The query can be written as follows:

```
select playerId, min(eventDate) as firstLogin from Activity group by playerId
```

- **Approach 2:**

We can partition the tuples by the player_id and order them by their event_id such that all the tuples having the same player_id are grouped together. We then number every tuple in each of the groups starting with the number 1. Now, we just have to display the event_date for the tuple having row number 1. For this, we use the ROW_NUMBER operator. The SQL query for it can be written as follows:

```
SELECT playerId, eventDate AS firstLogin
FROM
(
    SELECT playerId, eventDate, ROW_NUMBER() OVER (PARTITION BY playerId ORDER BY eventDate)
    FROM Activity
) AS t
WHERE seq = 1
```

- **Approach 3:**

We follow a similar kind of approach as used in Approach 2. But instead of using the ROW_NUMBER operator, we can use the FIRST_VALUE operator to find the first event_date. The SQL query for it can be written as follows:

```
select distinct(playerId),
FIRST_VALUE(eventDate) OVER(PARTITION BY playerId ORDER BY eventDate) as firstLogin
from Activity;
```

3. Given the following schema:

Table: Customers

Column Name	Type
id	int
name	varchar

The primary key column for this table is id. Each row in the table represents a customer's ID and name.


Table: Orders

Column Name	Type
id	int
customerId	int

The primary key column for this table is `id`. `customerId` is a foreign key of the ID from the Customers table. The ID of an order and the ID of the customer who placed it are listed in each row of this table. Write an SQL query to report all customers who never order anything. You can return the result table in any order.

Example:

Input: Customers table:



id	name
1	Ram
2	Sachin
3	Rajat
4	Ankit

Orders table:

id	customerid
1	2
2	1

Output

Customers
Rajat
Ankit

Explanation: Here, the customers Sachin and Ram have placed an order having order id 1 and 2 respectively. Thus, the customers Rajat and Ankit have never placed an order. So, we print their names in the result set.

- **Approach 1:**

In this approach, we first try to find the customers who have ordered at least once. After having found this, we find the customers whose customer Id is not present in the previously obtained result set. This gives us the customers who have not placed a single order yet. The SQL query for it can be written as follows

```
select customers.name as 'Customers'
from customers
where customers.id not in
(
    select customerid from orders
);
```

- **Approach 2:**

In this approach, we use the concept of JOIN. We will LEFT JOIN the customer table with the order table based on the condition that id of the customer table must be equal to that of the customer id of the order table. Now, in our joined resultant table, we just need to find those customers whose order id is null. The SQL query for this can be written as follows:

```
select c.name as 'Customers' from Customers c
left join Orders o ON (o.customerId = c.id)
where o.id is null
```

Here, we first create aliases of the tables Customers and Orders with the name 'c' and 'o' respectively. Having done so, we join them with the condition that o.customerId = c.id. At last, we check for the customers whose o.id is null.

4. Given the following schema:


Table: Cinema

Column Name	Type
id	int
movie	varchar
description	varchar
rating	float

The primary key for this table is id. Each row includes information about a movie's name, genre, and rating. rating is a float with two decimal digits in the range [0, 10]. Write an SQL query to report the movies with an odd-numbered ID and a description that is not "boring". Return the result table ordered by rating in descending order.

Example:

Input: Cinema table:



id	movie	description	rating
1	War	thriller	8.9
2	Dhakkad	action	2.1
3	Gippi	boring	1.2
4	Dangal	wrestling	8.6
5	P.K.	Sci-Fi	9.1

Output

id	movie	description	rating
5	P.K.	Sci-Fi	9.1
1	War	thriller	8.9

Explanation:

There are three odd-numbered ID movies: 1, 3, and 5. We don't include the movie with ID = 3 in the answer because it's boring. We put the movie with id 5 at the top since it has the highest rating of 9.1.

This question has a bit of ambiguity on purpose. You should ask the interviewer whether we need to check for the description to exactly match “boring” or we need to check if the word “boring” is present in the description. We have provided solutions for both cases.

- **Approach 1 (When the description should not be exactly “boring” but can include “boring” as a substring):**

In this approach, we use the MOD operator to check whether the id of a movie is odd or not. Now, for all the odd-numbered id movies, we check if its description is not boring. At last, we sort the resultant data according to the descending order of the movie rating. The SQL query for this can be written as follows:

```
select *  
from cinema  
where mod(id, 2) = 1 and description != 'boring'  
order by rating DESC;
```

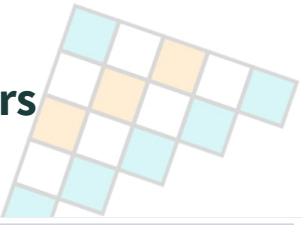
- **Approach 2 (When the description should not even contain “boring” as a substring in our resultant answer):**

In this approach, we use the LIKE operator to match the description having “boring” as a substring. We then use the NOT operator to eliminate all those results. For the odd-numbered id, we check it similarly as done in the previous approach. Finally, we order the result set according to the descending order of the movie rating. The SQL query for it can be written as follows:

```
SELECT *  
FROM Cinema  
WHERE id % 2 = 1 AND description NOT LIKE '%boring%'  
ORDER BY rating DESC;
```

5. Consider the following schema:

Table: Users



Column Name	Type
account_number	int
name	varchar

The account is the primary key for this table. Each row of this table contains the account number of each user in the bank. There will be no two users having the same name in the table.

Table: Transactions

Column Name	Type
trans_id	int
account_number	int
amount	int
transacted_on	date

trans_id is the primary key for this table. Each row of this table contains all changes made to all accounts. The amount is positive if the user received money and negative if they transferred money. All accounts start with a balance of 0.

Construct a SQL query to display the names and balances of people who have a balance greater than \$10,000. The balance of an account is equal to the sum of the amounts of all transactions involving that account. You can return the result table in any order.

Example:

Input: Users table:

Account_number	name
12300001	Ram
12300002	Tim
12300003	Shyam

Transactions table:

trans_id	account_number	amount	transacted_on
1	12300001	8000	2022-03-01
2	12300001	8000	2022-03-01
3	12300001	-3000	2022-03-02
4	12300002	4000	2022-03-12
5	12300003	7000	2022-02-07
6	12300003	7000	2022-03-07
7	12300003	-4000	2022-03-11

Output:

name	balance
Ram	13000

Explanation:

- Ram's balance is $(8000 + 8000 - 3000) = 11000$.
- Tim's balance is 4000.
- Shyam's balance is $(7000 + 7000 - 4000) = 10000$.
- **Approach 1:**

In this approach, we first create aliases of the given two tables' users and transactions. We can natural join the two tables and then group them by their account number. Next, we use the SUM operator to find the balance of each of the accounts after all the transactions have been processed. The SQL query for this can be written as follows:

```
SELECT u.name, SUM(t.amount) AS balance
FROM Users natural join Transactions t
GROUP BY t.account_number
HAVING balance > 10000;
```

6. Given the following schema:

Table: Employee

Column Name	Type
id	int
name	varchar
department	varchar
managerId	int

All employees, including their managers, are present at the Employee table. There is an Id for each employee, as well as a column for the manager's Id. Write a SQL query that detects managers with at least 5 direct reports from the Employee table.

Example:

Input:

Id	Name	Department	ManagerId
201	Ram	A	null
202	Naresh	A	201
203	Krishna	A	201
204	Vaibhav	A	201
205	Jainender	A	201
206	Sid	B	201



Output:

Name
Ram

- **Approach:**

In this problem, we first find all the manager ids who have more than 5 employees under them. Next, we find all the employees having the manager id present in the previously obtained manager id set.

The SQL query for this can be written as follows:

```
SELECT Name
FROM Employee
WHERE id IN
  (SELECT ManagerId
   FROM Employee
   GROUP BY ManagerId
   HAVING COUNT(DISTINCT Id) >= 5);
```

7. Consider the following table schema:

Construct an SQL query to retrieve duplicate records from the Employee table.

Table: Employee

Column Name	Type
id	int
fname	varchar
lname	varchar
department	varchar
projectId	varchar
address	varchar
dateofbirth	varchar
gender	varchar

Table: Salary

Column Name	Type
id	int
position	varchar
dateofJoining	varchar
salary	varchar

Now answer the following questions:

1. Construct an SQL query that retrieves the fname in upper case from the Employee table and uses the ALIAS name as the EmployeeName in the result.

```
SELECT UPPER(fname) AS EmployeeName FROM Employee;
```

2. Construct an SQL query to find out how many people work in the "HR" department

```
SELECT COUNT(*) FROM Employee WHERE department = 'HR';
```

3. Construct an SQL query to retrieve the first four characters of the 'lname' column from the Employee table.

```
SELECT SUBSTRING(lname, 1, 4) FROM Employee;
```

4. Construct a new table with data and structure that are copied from the existing table 'Employee' by writing a query. The name of the new table should be 'SampleTable'.

```
SELECT * INTO SampleTable FROM Employee WHERE 1 = 0
```

5. Construct an SQL query to find the names of employees whose first names start with "S".

```
SELECT * FROM Employee WHERE fname LIKE 'S%';
```

6. Construct an SQL query to count the number of employees grouped by gender whose dateOfBirth is between 01/03/1975 and 31/12/1976.

```
SELECT COUNT(*), gender FROM Employee WHERE dateOfBirth BETWEEN '01/03/1975 ' AND '31/12/1976 ';
```

7. Construct an SQL query to retrieve all employees who are also managers.

```
SELECT emp.fname, emp.lname, sal.position  
FROM Employee emp INNER JOIN Salary sal ON  
emp.id = sal.id AND sal.position IN ('Manager');
```

8. Construct an SQL query to retrieve the employee count broken down by department and ordered by department count in ascending manner.

```
SELECT department, COUNT(id) AS DepartmentCount  
FROM Employee GROUP BY department  
ORDER BY DepartmentCount ASC;
```

9. Construct an SQL query to retrieve duplicate records from the Employee table.

```
SELECT id, fname, department, COUNT(*) as Count  
FROM Employee GROUP BY id, fname, department  
HAVING COUNT(*) > 1;
```

SQL Query Interview Questions for Experienced

8. Consider the following Schema:

Table: Tree

Column Name	Type
id	int
parent_id	int

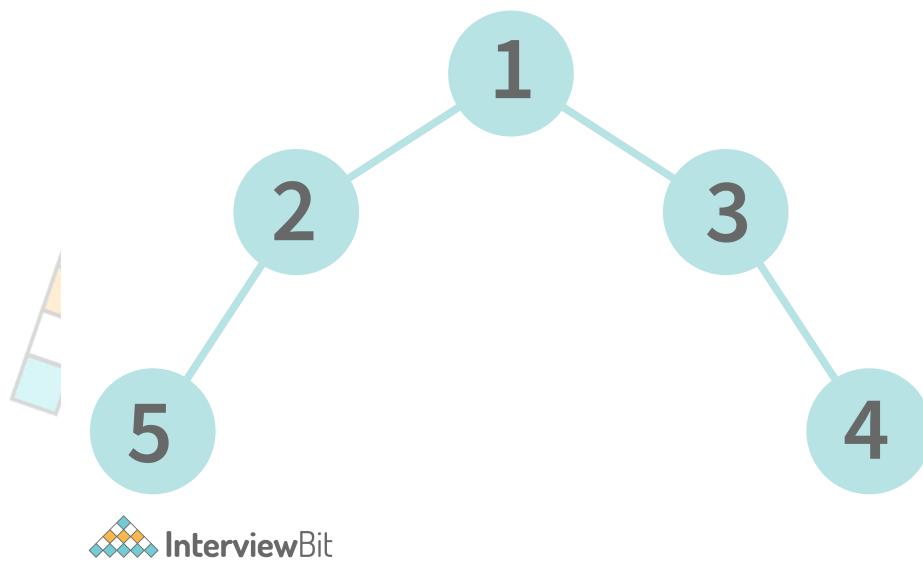
Here, **id** is the primary key column for this table. **id** represents the unique identity of a tree node and **parent_id** represents the unique identity of the parent of the current tree node. The **id** of a node and the **id** of its parent node in a tree are both listed in each row of this table. There is always a valid tree in the given structure.

Every node in the given tree can be categorized into one of the following types:

1. **"Leaf"**: When the tree node is a leaf node, we label it as **"Leaf"**
2. **"Root"**: When the tree node is a root node, we label it as **"Root"**
3. **"Inner"**: When the tree node is an inner node, we label it as **"Inner"**

Write a SQL query to find and return the type of each of the nodes in the given tree. You can return the result in any order.

Example:



Input: Tree Table

id	parent_id
1	null
2	1
3	1
4	3
5	2

Output:

id	type
1	Root
2	Inner
3	Inner
4	Leaf
5	Leaf

Explanation:

1. Because node 1's parent node is null, and it has child nodes 2 and 3, Node 1 is the root node.
2. Because node 2 and node 3 have parent node 1 and child nodes 5 and 4 respectively, Node 2 and node 3 are inner nodes.
3. Because nodes 4 and 5 have parent nodes but no child nodes, nodes 4, and 5 are leaf nodes.

Approach 1:

In this approach, we subdivide our problem of categorizing the type of each of the nodes in the tree. We first find all the root nodes and add them to our resultant set with the type "root". Then, we find all the leaf nodes and add them to our resultant set with the type "leaf". Similarly, we find all the inner nodes and add them to our resultant set with the type "inner". Now let us look at the query for finding each of the node types.

- **For root nodes:**

```
SELECT
    id, 'Root' AS Type
FROM
    tree
WHERE
    parent_id IS NULL
```

Here, we check if the parent_id of the node is null, then we assign the type of node as 'Root' and include it in our result set.

- **For leaf nodes:**

```
SELECT
    id, 'Leaf' AS Type
FROM
    tree
WHERE
    id NOT IN (SELECT DISTINCT
                parent_id
              FROM
                tree
              WHERE
                parent_id IS NOT NULL)
    AND parent_id IS NOT NULL
```

Here, we first find all the nodes that have a child node. Next, we check if the current node is present in the set of root nodes. If present, it cannot be a leaf node and we eliminate it from our answer set. We also check that the parent_id of the current node is not null. If both the conditions satisfy then we include it in our answer set.

- **For inner nodes:**

```
SELECT
    id, 'Inner' AS Type
FROM
    tree
WHERE
    id IN (SELECT DISTINCT
           parent_id
         FROM
           tree
         WHERE
           parent_id IS NOT NULL)
    AND parent_id IS NOT NULL
```

Here, we first find all the nodes that have a child node. Next, we check if the current node is present in the set of root nodes. If not present, it cannot be an inner node and we eliminate it from our answer set. We also check that the parent_id of the current node is not null. If both the conditions satisfy then we include it in our answer set.

At last, we combine all three resultant sets using the UNION operator. So, the final SQL query is as follows:


```
SELECT
    id, 'Root' AS Type
FROM
    tree
WHERE
    parent_id IS NULL

UNION

SELECT
    id, 'Leaf' AS Type
FROM
    tree
WHERE
    id NOT IN (SELECT DISTINCT
                parent_id
                FROM
                    tree
                WHERE
                    parent_id IS NOT NULL)
    AND parent_id IS NOT NULL

UNION

SELECT
    id, 'Inner' AS Type
FROM
    tree
WHERE
    id IN (SELECT DISTINCT
           parent_id
           FROM
               tree
           WHERE
               parent_id IS NOT NULL)
    AND parent_id IS NOT NULL

ORDER BY id;
```

Approach 2:

In this approach, we use the control statement CASE. This simplifies our query a lot from the previous approach. We first check if a node falls into the category of “Root”. If the node does not satisfy the conditions of a root node, it implies that the node will either be a “Leaf” node or an “Inner” node. Next, we check if the node falls into the category of “Inner” node. If it is not an “Inner” node, there is only one option left, which is the “Leaf” node.

The SQL query for this approach can be written as follows:

```
SELECT
    id AS `Id`,
    CASE
        WHEN tree.id = (SELECT aliastree.id FROM tree aliastree WHERE aliastree.parent_id = tree.id)
        THEN 'Root'
        WHEN tree.id IN (SELECT aliastree.parent_id FROM tree aliastree)
        THEN 'Inner'
        ELSE 'Leaf'
    END AS Type
FROM
    tree
ORDER BY `Id`;
```

Approach 3:

In this approach, we follow a similar logic as discussed in the previous approach. However, we will use the IF operator instead of the CASE operator. The SQL query for this approach can be written as follows:

```
SELECT
    aliastree.id,
    IF(ISNULL(aliastree.parent_id),
        'Root',
        IF(aliastree.id IN (SELECT parent_id FROM tree), 'Inner', 'Leaf')) Type
FROM
    tree aliastree
ORDER BY aliastree.id
```

9. Consider the following schema:

Table: Seat

Column Name	type
id	int
student	varchar

The table contains a list of students. Every tuple in the table consists of a seat id along with the name of the student. You can assume that the given table is sorted according to the seat id and that the seat ids are in continuous increments. Now, the class teacher wants to swap the seat id for alternate students in order to give them a last-minute surprise before the examination. You need to write a query that swaps alternate students' seat id and returns the result. If the number of students is odd, you can leave the seat id for the last student as it is.

Example:

id	student
1	Ram
2	Shyam
3	Vaibhav
4	Govind
5	Krishna

For the same input, the output is:

id	student
1	Shyam
2	Ram
3	Govind
4	Vaibhav
5	Krishna

- **Approach 1:**

In this approach, first we count the total number of students. Having done so, we consider the case when the seat id is odd but is not equal to the total number of students. In this case, we simply increment the seat id by 1. Next, we consider the case when the seat id is odd but is equal to the total number of students. In this case, the seat id remains the same. At last, we consider the case when the seat id is even. In this case, we decrement the seat id by 1.

The SQL query for this approach can be written as follows:

```
SELECT
  CASE WHEN MOD(id, 2) != 0 AND counts != id THEN id + 1 -- for odd ids
        WHEN MOD(id, 2) != 0 AND counts = id THEN id -- special case for last seat
        ELSE id - 1 -- For even ids
      END as id,
  student
FROM
  seat, (SELECT COUNT(*) as counts
        FROM seat) AS seat_count
ORDER by id;
```

• Approach 2:

In this approach, we use the ROW_NUMBER operator. We increment the id for the odd-numbered ids by 1 and decrement the even-numbered ids by 1. We then sort the tuples, according to the id values. Next, we assign the row number as the id for the sorted tuples. The SQL query for this approach can be written as follows:

```
select row_number()
  over (order by
        (if(id%2=1,id+1,id-1))
       ) as id, student
from seat;
```

10. Given the following schema:

Table: Employee

Column Name	type
id	int
name	varchar
salary	int
departmentId	int

id is the primary key column for this table. departmentId is a foreign key of the ID from the Department table. Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

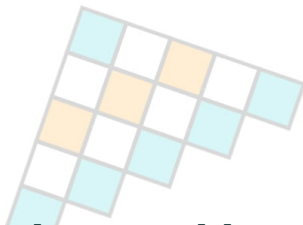
Table: Department

Column Name	type
id	int
name	varchar

id is the primary key column for this table. Each row of this table indicates the ID of a department and its name. The executives of an organization are interested in seeing who earns the most money in each department. A high earner in a department is someone who earns one of the department's top three unique salaries.

Construct a SQL query to identify the high-earning employees in each department. You can return the result table in any order.

Example:



Input: Employee table:

id	name	salary	departmentId
1	Ram	85000	1
2	Divya	80000	2
3	Tim	60000	2
4	Kim	90000	1
5	Priya	69000	1
6	Saket	85000	1
7	Will	70000	1

Department table:

id	name
1	Marketing
2	HR

Output:

Department	Employee	Salary
Marketing	Kim	90000
Marketing	Ram	85000
Marketing	Saket	85000
Marketing	Will	70000
HR	Divya	80000
HR	Tim	60000

Explanation:

- Kim has the greatest unique income in the Marketing department - Ram and Saket have the second-highest unique salary.
- Will has the third-highest unique compensation.

In the HR department:

- Divya has the greatest unique income.
- Tim earns the second-highest salary.
- Because there are only two employees, there is no third-highest compensation.

Approach 1:

In this approach, let us first assume that all the employees are from the same department. So let us first figure out how we can find the top 3 high-earner employees. This can be done by the following SQL query:

```
select emp1.Name as 'Employee', emp1.Salary
from Employee emp1
where 3 >
(
    select count(distinct emp2.Salary)
    from Employee emp2
    where emp2.Salary > emp1.Salary
);
```

Here, we have created two aliases for the Employee table. For every tuple of the emp1 alias, we compare it with all the distinct salaries to find out how many salaries are less than it. If the number is less than 3, it falls into our answer set.

Next, we need to join the Employee table with the Department table in order to obtain the high-earner employees department-wise. For this, we run the following SQL command:

```
SELECT
    d.Name AS 'Department', e1.Name AS 'Employee', e1.Salary
FROM
    Employee e1
    JOIN
    Department d ON e1.DepartmentId = d.Id
WHERE
    3 > (SELECT
        COUNT(DISTINCT e2.Salary)
        FROM
            Employee e2
        WHERE
            e2.Salary > e1.Salary
            AND e1.DepartmentId = e2.DepartmentId
    );
```

Here, we join the Employee table and the Department table based on the department ids in both tables. Also, while finding out the high-earner employees for a specific department, we compare the department ids of the employees as well to ensure that they belong to the same department.

Approach 2:

In this approach, we use the concept of the DENSE_RANK function in SQL. We use the DENSE_RANK function and not the RANK function since we do not want the ranking number to be skipped. The SQL query using this approach can be written as follows:

```
SELECT Final.Department, Final.Employee, Final.Salary FROM
  (SELECT D.name AS Department, E.name AS Employee, E.salary AS Salary,
    DENSE_RANK() OVER (PARTITION BY D.name ORDER BY E.salary DESC) Rank
  FROM Employee E, Department D
  WHERE E.departmentId = D.id) Final
WHERE Final.Rank < 4;
```

Here, we first run a subquery where we partition the tuples by their department and rank them according to the decreasing order of the salaries of the employees. Next, we select those tuples from this set, whose rank is less than 4.

11. Given the following schema:

Table: Stadium

Column Name	type
id	int
date_visited	date
count_people	int

date_visited is the primary key for this table. The visit date, the stadium visit ID, and the total number of visitors are listed in each row of this table. No two rows will share the same visit date, and the dates get older as the id gets bigger. Construct a SQL query to display records that have three or more rows of consecutive ids and a total number of people higher than or equal to 100. Return the result table in ascending order by visit date.

Example:

Input: Stadium table:



id	date_visited	count_people
1	2022-03-01	6
2	2022-03-02	102
3	2022-03-03	135
4	2022-03-04	90
5	2022-03-05	123
6	2022-03-06	115
7	2022-03-07	101
8	2022-03-09	235

Output:

id	date_visited	count_people
5	2022-03-05	123
6	2022-03-06	115
7	2022-03-07	101
8	2022-03-09	235

Explanation:

The four rows with ids 5, 6, 7, and 8 have consecutive ids and each of them has ≥ 100 people attended. Note that row 8 was included even though the date_visited was not the next day after row 7.

The rows with ids 2 and 3 are not included because we need at least three consecutive ids.

- **Approach 1:**

In this approach, we first create three aliases of the given table and cross-join all of them. We filter the tuples such that the number of people in each of the alias' should be greater than or equal to 100.

The query for this would be

```
select distinct t1.*  
from stadium t1, stadium t2, stadium t3  
where t1.count_people >= 100 and t2.count_people >= 100 and t3.count_people >= 100;
```

Now, we have to check for the condition of consecutive 3 tuples. For this, we compare the ids of the three aliases to check if they form a possible triplet with consecutive ids. We do this by the following query:

```
select t1.*
from stadium t1, stadium t2, stadium t3
where t1.count_people >= 100 and t2.count_people >= 100 and t3.count_people >= 100
and
(
    (t1.id - t2.id = 1 and t1.id - t3.id = 2 and t2.id - t3.id = 1)
    or
    (t2.id - t1.id = 1 and t2.id - t3.id = 2 and t1.id - t3.id = 1)
    or
    (t3.id - t2.id = 1 and t2.id - t1.id = 1 and t3.id - t1.id = 2)
);
```

The above query may contain duplicate triplets. So we remove them by using the DISTINCT operator. The final query becomes as follows:

```
select distinct t1.*
from stadium t1, stadium t2, stadium t3
where t1.count_people >= 100 and t2.count_people >= 100 and t3.count_people >= 100
and
(
    (t1.id - t2.id = 1 and t1.id - t3.id = 2 and t2.id - t3.id = 1)
    or
    (t2.id - t1.id = 1 and t2.id - t3.id = 2 and t1.id - t3.id = 1)
    or
    (t3.id - t2.id = 1 and t2.id - t1.id = 1 and t3.id - t1.id = 2)
)
order by t1.id;
```

- **Approach 2:**

In this approach, we first filter out all the tuples where the number of people is greater than or equal to 100. Next, for every tuple, we check, if there exist 2 other tuples with ids such that the three ids when grouped together form a consecutive triplet. The SQL query for this approach can be written as follows:

```
with cte as
(select * from stadium
where count_people >= 100)

select cte.id, cte.date_visited, cte.count_people
from cte
where
((cte.id + 1) in (select id from cte)
and
(cte.id + 2) in (select id from cte))

or
((cte.id - 1) in (select id from cte)
and
(cte.id - 2) in (select id from cte))

or
((cte.id + 1) in (select id from cte)
and
(cte.id - 1) in (select id from cte))
```

12. Given the following schema:

Table: Employee

Column Name	Type
id	int
company	varchar
salary	int

Here, id is the id of the employee. company is the name of the company he/she is working in. salary is the salary of the employee
Construct a SQL query to determine each company's median salary. If you can solve it without utilising any built-in SQL functions, you'll get bonus points.

Example:

Input:



Id	Company	Salary
1	Amazon	1100
2	Amazon	312
3	Amazon	150
4	Amazon	1300
5	Amazon	414
6	Amazon	700
7	Microsoft	110
8	Microsoft	105
9	Microsoft	470
10	Microsoft	1500
11	Microsoft	1100
12	Microsoft	290
13	Google	2000
14	Google	2200
15	Google	2200
16	Google	2400
17	Google	1000

Output:

Id	Company	Salary
5	Amazon	414
6	Amazon	700
12	Microsoft	290
9	Microsoft	470
14	Google	2200

- **Approach 1:**

In this approach, we have a subquery where we partition the tuples according to the company name and rank the tuples in the increasing order of salary and id. We also find the count of the total number of tuples in each company and then divide it by 2 in order to find the median tuple. After we have this result, we run an outer query to fetch the median salary and the employee id for each of the companies.

The SQL query for this can be written as follows:

```
select table.id, table.company, table.salary
from (select id, company, salary,
      dense_rank() over (partition by company order by salary, id) as Ranking,
      count(1) over (partition by company) / 2.0 as EmployeeCount
from Employee ) table
where Ranking between EmployeeCount and EmployeeCount + 1;
```

Conclusion:

In this article, we have covered the most frequently asked interview questions on SQL queries. To go through the most frequently asked theoretical interview questions on SQL, you can visit this [link](#). While appearing for an SQL interview, you can also expect questions on Database Management Systems (DBMS). To go through the most frequently asked interview questions on DBMS, you can visit this [link](#).

Additional Resources

- [SQL Programming](#)
- [SQL Cheat Sheet](#)
- [SQL Commands](#)
- [SQL Server Interview Questions](#)

Links to More Interview Questions

[C Interview Questions](#)

[Php Interview Questions](#)

[C Sharp Interview Questions](#)

[Web Api Interview Questions](#)

[Hibernate Interview Questions](#)

[Node Js Interview Questions](#)

[Cpp Interview Questions](#)

[Oops Interview Questions](#)

[Devops Interview Questions](#)

[Machine Learning Interview Questions](#)

[Docker Interview Questions](#)

[Mysql Interview Questions](#)

[Css Interview Questions](#)

[Laravel Interview Questions](#)

[Asp Net Interview Questions](#)

[Django Interview Questions](#)

[Dot Net Interview Questions](#)

[Kubernetes Interview Questions](#)

[Operating System Interview Questions](#)

[React Native Interview Questions](#)

[Aws Interview Questions](#)

[Git Interview Questions](#)

[Java 8 Interview Questions](#)

[Mongodb Interview Questions](#)

[Dbms Interview Questions](#)

[Spring Boot Interview Questions](#)

[Power Bi Interview Questions](#)

[Pl Sql Interview Questions](#)

[Tableau Interview Questions](#)

[Linux Interview Questions](#)

[Ansible Interview Questions](#)

[Java Interview Questions](#)

[Jenkins Interview Questions](#)