SQL (Structured Query Language) Practice

Employees Table

EmployeeID	FirstName	LastName	DepartmentID	Salary	HireDate	ManagerID
1	Alice	Johnson	101	60000	2018-01-15	3
2	Bob	Smith	102	75000	2017-05-20	3
3	Charlie	Brown	101	90000	2015-09-30	NULL
4	David	Williams	103	55000	2019-07-11	3
5	Eva	Davis	102	65000	2020-03-25	2

Orders Table

OrderID	EmployeeID	ProductID	Quantity	OrderDate
1001	1	201	10	2022-01-15
1002	2	202	5	2022-01-16
1003	3	203	20	2022-01-17
1004	4	202	15	2022-01-18
1005	5	204	25	2022-01-19

Products Table

ProductID	ProductName	Price	Category
201	Laptop	1200	Electronics
202	Smartphone	800	Electronics
203	Office Chair	150	Furniture
204	Desk	300	Furniture
205	Monitor	200	Electronics

SQL Queries Interview Questions

Q1. Write a query to display all records from the Employees table.

Answer:

SELECT * FROM Employees;Copy Code

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

1 | Alice | Johnson | 10 | 160000 | 2018-01-15 | 32

2 | Bob | Smith | 10 | 275000 | 2017-05-20 | 33

3 | Charlie | Brown | 10 | 190000 | 2015-09-30 | NULL

4 | David | Williams | 10 | 355000 | 2019-07-11 | 35

5 | Eva | Davis | 10 | 265000 | 2020-03-25 | 32Copy Code

Q2. Fetch only the FirstName and LastName of employees.

Answer:

SELECT FirstName, LastName FROM Employees;

FirstName | LastName

Alice | Johnson

Bob | Smith

Charlie | Brown

David | Williams

Eva | Davis

Copy Code

Q3. Retrieve the unique department IDs from the Employees table.

Answer:

SELECT DISTINCT DepartmentID FROM Employees;

DepartmentID

10

Copy Code

Q4. Fetch employees with a salary greater than 60,000.

Answer:

```
SELECT * FROM Employees WHERE Salary > 60000;
```

```
EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate    | ManagerID
-------------------------------------------------------------------------
1          | Alice     | Johnson  | 10           | 160000 | 2018-01-15  | 32
2          | Bob       | Smith    | 10           | 275000 | 2017-05-20  | 33
3          | Charlie   | Brown    | 10           | 190000 | 2015-09-30  | NULL
4          | David     | Williams | 10           | 355000 | 2019-07-11  | 35
5          | Eva       | Davis    | 10           | 265000 | 2020-03-25  | 32
```

Copy Code

Q5. Write a query to display all orders placed on or after January 17, 2022.

Answer:

```
SELECT * FROM Orders WHERE OrderDate >= '2022-01-17';
```

```
OrderID | EmployeeID | ProductID | Quantity | OrderDate
-----------------------------------------------------
1022    | 2          | 1         | 2        | 2022-01-16
1023    | 3          | 3         | 3        | 2022-01-17
1024    | 4          | 2         | 5        | 2022-01-18
1025    | 5          | 4         | 5        | 2022-01-19
```

Copy Code

Q6. Retrieve all products with a price less than 300.

Answer:

```
SELECT * FROM Products WHERE Price < 300;
```

```
ProductID | ProductName | Price | Category
-----------------------------------------
203       | Office Chair | 150   | Furniture
204       | Desk         | 300   | Furniture
205       | Monitor      | 200   | Electronics
```

Copy Code

Q7. Find the total number of orders in the Orders table.

Answer:

```
SELECT COUNT(*) AS TotalOrders FROM Orders;
```

```
TotalOrders
-----------
5
```

Copy Code

Q8. Fetch the details of the product named 'Laptop'.

Answer:

```
SELECT * FROM Products WHERE ProductName = 'Laptop';
```

```
ProductID | ProductName | Price | Category
-----------------------------------------
201       | Laptop      | 1200  | Electronics
```

Copy Code

Q9. Write a query to sort employees by their HireDate in ascending order.

Answer:

```
SELECT * FROM Employees ORDER BY HireDate ASC;
```

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

3 | Charlie | Brown | 10 | 190000 | 2015-09-30 | NULL

2 | Bob | Smith | 10 | 275000 | 2017-05-20 | 33

1 | Alice | Johnson | 10 | 160000 | 2018-01-15 | 32

4 | David | Williams | 10 | 355000 | 2019-07-11 | 35

5 | Eva | Davis | 10 | 265000 | 2020-03-25 | 32

Copy Code

Q10. Retrieve the maximum price of products in the Electronics category.

Answer:

SELECT MAX(Price) AS MaxPrice FROM Products WHERE Category = 'Electronics';

MaxPrice

1200Copy Code

Q11. Write a query to join Employees and Orders tables to fetch employee names along with their orders.

Answer:

SELECT e.FirstName, e.LastName, o.OrderID, o.OrderDate

FROM Employees e

JOIN Orders o ON e.EmployeeID = o.EmployeeID;

FirstName | LastName | OrderID | OrderDate

Alice | Johnson | 1022 | 2022-01-16

Bob | Smith | 1023 | 2022-01-17

Charlie | Brown | 1024 | 2022-01-18

David | Williams | 1025 | 2022-01-19

Copy Code

Q12. Calculate the total salary by department.

Answer:

```
SELECT DepartmentID, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY DepartmentID;
```

```
DepartmentID | TotalSalary
-------------------------
10           | 1355000
```

Copy Code

Q13. Find the employees who do not have a manager.

Answer:

```
SELECT * FROM Employees WHERE ManagerID IS NULL;
```

```
EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate   | ManagerID
-------------------------------------------------------------------------------
3          | Charlie   | Brown    | 10           | 190000 | 2015-09-30 | NULL
```

Copy Code

Q14. Write a query to display the average product price for each category.

Answer:

```
SELECT Category, AVG(Price) AS AvgPrice
FROM Products
GROUP BY Category;
```

Category | AvgPrice

Electronics | 800

Furniture | 216.67

Copy Code

Q15. Fetch the details of the top 3 highest-paid employees.

Answer:

```
SELECT * FROM Employees
ORDER BY Salary DESC
LIMIT 3;
```

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

4 | David | Williams | 10 | 355000 | 2019-07-11 | 35

2 | Bob | Smith | 10 | 275000 | 2017-05-20 | 33

5 | Eva | Davis | 10 | 265000 | 2020-03-25 | 32

Copy Code

Q16. Retrieve the order details along with the product name.

Answer:

```
SELECT o.OrderID, o.Quantity, p.ProductName, p.Price
FROM Orders o
JOIN Products p ON o.ProductID = p.ProductID;
```

OrderID | Quantity | ProductName | Price

1022 | 2 | Laptop | 1200

1023 | 3 | Office Chair | 150

1024 | 5 | Smartphone | 800

1025 | 5 | Desk | 300

Copy Code

Q17. Find the total quantity of products ordered for each product.

Answer:

```
SELECT ProductID, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY ProductID;
```

ProductID | TotalQuantity

---

1 | 2

2 | 8

3 | 3

4 | 5

Copy Code

Q18. Write a query to update the price of all Furniture category products by 10%.

Answer:

```
UPDATE Products
SET Price = Price * 1.10
WHERE Category = 'Furniture';Copy Code
```

Q19. Delete all orders placed before January 17, 2022.

Answer:

```
DELETE FROM Orders WHERE OrderDate < '2022-01-17';Copy Code
```

Q20. Fetch employees whose first name starts with ‘A’.

Answer:

```
SELECT * FROM Employees WHERE FirstName LIKE 'A%';
```

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

1 | Alice | Johnson | 10 | 160000 | 2018-01-15 | 32

Copy Code

Q21. Retrieve the number of employees hired each year.

Answer:

SELECT YEAR(HireDate) AS HireYear, COUNT(*) AS EmployeesHired

FROM Employees

GROUP BY YEAR(HireDate);

HireYear | EmployeesHired

2015 | 1

2017 | 1

2018 | 1

2019 | 1

2020 | 1

Copy Code

Q22. Write a query to fetch employees earning more than the average salary.

Answer:

SELECT * FROM Employees

WHERE Salary > (SELECT AVG(Salary) FROM Employees);

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

2 | Bob | Smith | 10 | 275000 | 2017-05-20 | 33

4 | David | Williams | 10 | 355000 | 2019-07-11 | 35

5 | Eva | Davis | 10 | 265000 | 2020-03-25 | 32

Copy Code

Q23. Display the top 3 products with the highest total quantity sold.

Answer:

```
SELECT p.ProductName, SUM(o.Quantity) AS TotalQuantity
FROM Orders o
JOIN Products p ON o.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY TotalQuantity DESC
LIMIT 3;
```

ProductName | TotalQuantity

Smartphone | 8

Desk | 5

Office Chair| 3

Copy Code

Q24. Retrieve the employees who have not placed any orders.

Answer:

```
SELECT * FROM Employees
WHERE EmployeeID NOT IN (SELECT DISTINCT EmployeeID FROM Orders);
```

EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate | ManagerID

---

3 | Charlie | Brown | 10 | 190000 | 2015-09-30 | NULL

Copy Code

Q25. Write a query to fetch the most recently hired employee.

Answer:

```
SELECT * FROM Employees
ORDER BY HireDate DESC
LIMIT 1;
```

```
EmployeeID | FirstName | LastName | DepartmentID | Salary | HireDate   | ManagerID
-----------------------------------------------------------------------
5          | Eva       | Davis    | 10           | 265000 | 2020-03-25 | 32
```

Copy Code

Q26. Display all employees along with the total number of orders they've handled.

Answer:

```
SELECT e.EmployeeID, e.FirstName, COUNT(o.OrderID) AS TotalOrders
FROM Employees e
LEFT JOIN Orders o ON e.EmployeeID = o.EmployeeID
GROUP BY e.EmployeeID, e.FirstName;Copy Code
```

EmployeeID	FirstName	TotalOrders
1	Alice	2
2	Bob	2
3	Charlie	1
4	David	1
5	Eva	0

Q27. Fetch product details for which total sales exceed $10,000.

Answer:

```
SELECT p.ProductName, SUM(o.Quantity * p.Price) AS TotalSales
FROM Orders o
JOIN Products p ON o.ProductID = p.ProductID
```

GROUP BY p.ProductName

HAVING TotalSales > 10000;Copy Code

ProductName	TotalSales
Laptop	24000

Q28. Find employees who joined the company in the same year as their manager.

Answer:

SELECT e.FirstName AS EmployeeName, m.FirstName AS ManagerName

FROM Employees e

JOIN Employees m ON e.ManagerID = m.EmployeeID

WHERE YEAR(e.HireDate) = YEAR(m.HireDate);Copy Code

EmployeeName	ManagerName
Alice	Bob

Q29. Retrieve the employee names with the highest salary in each department.

Answer:

SELECT DepartmentID, FirstName, LastName, Salary

FROM Employees

WHERE (DepartmentID, Salary) IN (

SELECT DepartmentID, MAX(Salary)

FROM Employees

GROUP BY DepartmentID

);Copy Code

DepartmentID	FirstName	LastName	Salary
1	Alice	Johnson	160000
2	Bob	Smith	75000
3	David	Williams	55000

Q30. Write a query to fetch the total revenue generated by each employee.

Answer:

SELECT e.FirstName, e.LastName, SUM(o.Quantity * p.Price) AS TotalRevenue

FROM Employees e

JOIN Orders o ON e.EmployeeID = o.EmployeeID

JOIN Products p ON o.ProductID = p.ProductID

GROUP BY e.EmployeeID, e.FirstName, e.LastName;Copy Code

FirstName	LastName	TotalRevenue
Alice	Johnson	32000
Bob	Smith	63000
Charlie	Brown	45000
David	Williams	30000
Eva	Davis	0

Q31. Write a query to fetch employees earning more than their manager.

Answer:

SELECT e.FirstName AS EmployeeName, m.FirstName AS ManagerName

FROM Employees e

JOIN Employees m ON e.ManagerID = m.EmployeeID

WHERE e.Salary > m.Salary;Copy Code

EmployeeName	ManagerName
Alice	Bob

Q32. Retrieve the second highest salary from the Employees table.

Answer:

SELECT MAX(Salary) AS SecondHighestSalary

FROM Employees

WHERE Salary < (SELECT MAX(Salary) FROM Employees);

SecondHighestSalary

75000Copy Code

Q33. List the departments with no employees assigned.

Answer:

SELECT * FROM Departments

WHERE DepartmentID NOT IN (SELECT DISTINCT DepartmentID FROM Employees);Copy Code

DepartmentID	DepartmentName
4	Marketing

Q34. Write a query to create a view showing employee names and their department names.

Answer:

CREATE VIEW EmployeeDepartmentView AS

SELECT e.FirstName, e.LastName, d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;Copy Code

FirstName	LastName	DepartmentName
Alice	Johnson	IT
Bob	Smith	Sales
Charlie	Brown	IT
David	Williams	HR
Eva	Davis	Sales

Q35. Fetch the names of employees who have placed more than 10 orders.

Answer:

SELECT e.FirstName, e.LastName

FROM Employees e

JOIN Orders o ON e.EmployeeID = o.EmployeeID

GROUP BY e.EmployeeID, e.FirstName, e.LastName

HAVING COUNT(o.OrderID) > 10;Copy Code

FirstName	LastName
Alice	Johnson
Bob	Smith

Q36. Write a query to rank employees based on their salary within each department.

Answer:

```
SELECT EmployeeID, FirstName, DepartmentID, Salary,
    RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS Rank
FROM Employees;Copy Code
```

EmployeeID	FirstName	DepartmentID	Salary	Rank
1	Alice	1	160000	1
3	Charlie	1	190000	2
2	Bob	2	75000	1
4	David	3	55000	1
5	Eva	2	65000	2

Q37. Retrieve the cumulative sales for each product.

Answer:

```
SELECT ProductID, ProductName,
    SUM(SUM(Quantity * Price)) OVER (ORDER BY ProductID) AS CumulativeSales
FROM Products p
JOIN Orders o ON p.ProductID = o.ProductID
GROUP BY ProductID, ProductName;Copy Code
```

ProductID	ProductName	CumulativeSales
201	Laptop	24000
202	Smartphone	32000
203	Office Chair	1500
204	Desk	3000

ProductID	ProductName	CumulativeSales
205	Monitor	1500

Q38. Identify the department with the highest total salary expenditure.

Answer:

SELECT DepartmentID, SUM(Salary) AS TotalExpenditure

FROM Employees

GROUP BY DepartmentID

ORDER BY TotalExpenditure DESC

LIMIT 1;Copy Code

DepartmentID	TotalExpenditure
1	450000

Q39. Write a query to find the percentage contribution of each product to total sales.

Answer:

SELECT p.ProductName,

(SUM(o.Quantity * p.Price) * 100.0 /

(SELECT SUM(Quantity * Price) FROM Orders o JOIN Products p ON o.ProductID = p.ProductID)) AS ContributionPercentage

FROM Orders o

JOIN Products p ON o.ProductID = p.ProductID

GROUP BY p.ProductName;Copy Code

ProductName	ContributionPercentage
Laptop	48.00
Smartphone	32.00
Office Chair	4.00
Desk	8.00
Monitor	8.00

Q40. Find employees who have the same manager and earn more than $70,000.

Answer:

```
SELECT *
FROM Employees e1
WHERE ManagerID IS NOT NULL
AND Salary > 70000
AND ManagerID IN (
   SELECT ManagerID FROM Employees e2 WHERE e1.ManagerID = e2.ManagerID
);Copy Code
```

EmployeeID	FirstName	LastName	Salary	ManagerID
1	Alice	Johnson	160000	32
2	Bob	Smith	75000	32

Q41. Write a query to detect duplicate rows in the Orders table.

Answer:

```
SELECT EmployeeID, ProductID, OrderDate, COUNT(*) AS DuplicateCount
FROM Orders
GROUP BY EmployeeID, ProductID, OrderDate
HAVING COUNT(*) > 1;Copy Code
```

EmployeeID	ProductID	OrderDate	DuplicateCount
1	201	2022-01-15	2

Q42. Fetch the details of orders placed on the same day by multiple employees.

Answer:

```
SELECT OrderDate, COUNT(DISTINCT EmployeeID) AS EmployeeCount
FROM Orders
GROUP BY OrderDate
HAVING EmployeeCount > 1;Copy Code
```

OrderDate	EmployeeCount
2022-01-15	2
2022-01-16	2
2022-01-17	1

Q43. Create a stored procedure to update product prices based on category.

Answer:

DELIMITER $$

CREATE PROCEDURE UpdatePriceByCategory(IN category_name VARCHAR(50), IN price_factor DECIMAL(5, 2))

BEGIN

UPDATE Products

SET Price = Price * price_factor

WHERE Category = category_name;

END$$

DELIMITER ;Copy Code

Q44. Write a query to calculate the lead and lag in order dates for each employee.

Answer:

SELECT EmployeeID, OrderID, OrderDate,

LAG(OrderDate) OVER (PARTITION BY EmployeeID ORDER BY OrderDate) AS PreviousOrderDate,

LEAD(OrderDate) OVER (PARTITION BY EmployeeID ORDER BY OrderDate) AS NextOrderDate

FROM Orders;Copy Code

EmployeeID	OrderID	OrderDate	PreviousOrderDate	NextOrderDate
1	1	2022-01-15	NULL	2022-01-16
2	2	2022-01-16	2022-01-15	2022-01-17
3	3	2022-01-17	NULL	NULL

Q45. Identify the products that have not been ordered.

Answer:

SELECT * FROM Products

WHERE ProductID NOT IN (SELECT DISTINCT ProductID FROM Orders);Copy Code

ProductID	ProductName
204	Desk
205	Monitor

Q46. Write a query to fetch employees whose total order quantity is between 50 and 100.

Answer:

SELECT e.FirstName, e.LastName, SUM(o.Quantity) AS TotalQuantity

FROM Employees e

JOIN Orders o ON e.EmployeeID = o.EmployeeID

GROUP BY e.EmployeeID, e.FirstName, e.LastName

HAVING TotalQuantity BETWEEN 50 AND 100;Copy Code

FirstName	LastName	TotalQuantity
Bob	Smith	60

Q47. Fetch the second-highest quantity ordered for each product.

Answer:

SELECT ProductID, MAX(Quantity) AS SecondHighestQuantity

FROM Orders

WHERE Quantity < (SELECT MAX(Quantity) FROM Orders WHERE Orders.ProductID = ProductID)

GROUP BY ProductID;Copy Code

ProductID	SecondHighestQuantity
201	20
202	30

ProductID	SecondHighestQuantity
203	10

Q48. Find the minimum and maximum order quantities for each employee.

Answer:

SELECT EmployeeID, MIN(Quantity) AS MinQuantity, MAX(Quantity) AS MaxQuantity

FROM Orders

GROUP BY EmployeeID;Copy Code

EmployeeID	MinQuantity	MaxQuantity
1	10	20
2	20	40
3	10	10

Q49. Write a query to split employee salaries into quartiles.

Answer:

SELECT EmployeeID, FirstName, Salary,

NTILE(4) OVER (ORDER BY Salary) AS SalaryQuartile

FROM Employees;Copy Code

EmployeeID	FirstName	Salary	SalaryQuartile
1	Alice	160000	4
2	Bob	75000	3
3	Charlie	190000	4
4	David	55000	2
5	Eva	65000	2

Q50. Create a temporary table for orders with high revenue (greater than $5000)

Answer:

CREATE TEMPORARY TABLE HighRevenueOrders AS

SELECT o.OrderID, o.Quantity, p.Price, (o.Quantity * p.Price) AS Revenue

FROM Orders o

JOIN Products p ON o.ProductID = p.ProductID

WHERE (o.Quantity * p.Price) > 5000;Copy Code

OrderID	Quantity	Price	Revenue
1	10	1200	12000
2	25	800	20000

able: 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;

3. Let us consider the following schema:

Table: Activity

Column Name	Type
playerId	int
deviceId	int
eventDate	date
gamesplayed	int

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) AS seq
  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;
```

You can download a PDF version of Sql Query Interview Questions.

4. 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	customeId
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.

5. 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;
```

6. 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

trans_id	account_number	amount	transacted_on
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;

7. Given the following schema:

Table: Employee

Column Name	Type
id	int
name	varcahar
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);
```

Start Your Coding Journey With Tracks

Master Data Structures and Algorithms

Topic Buckets

Mock Assessments

Reading Material

8. 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' **GROUP BY** gender;

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

1. 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 IS NULL)
      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
```

2. 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;
```

3. 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

id	name	salary	departmentId
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.

4. 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))
```

Discover your path to a **Successful Tech Career!**

Answer 4 simple questions & get a career plan tailored for you

Interview Process

CTC & Designation

Projects on the Job

5. 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

Id	Company	Salary
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