TOP 10 SQL INTERVIEW QUESTIONS

MySQL Delete Duplicate Rows but keep one

- Delete the duplicate rows but keep latest: **using GROUP BY and MAX**

- Delete the duplicate rows but keep latest: **using JOINS**

- Delete the duplicate row but keep oldest: **using JOINS**

- Delete the duplicate row but keep oldest: **using ROW_NUMBER()**

Build Composite Index (Multiple-Column Indexes)

- Can have indexes on multiple columns in Mysql?
- What is Composite Index? Give a relevant example.

- In Mysql, An index may consist of up to how many columns?
- Multiple column index is built to support which primary use case in Mysql?

Clustered and Non Clustered Index

- Differentiate between the two with an example?
- When to Use a Clustered Index? Show with an example.
- When to Use a Non-clustered Index? Show with an example.
- How many clustered indexes & non clustered indexes are allowed per table?
- Should Primary Keys be Clustered or Non-clustered Index?
- Which is Faster: Clustered or Non-clustered Index?

What is the difference between delete, truncate & drop?

- Which of these is DDL & DML command?
- Can we use the "where" clause with each one of them?
- Which of the above is executed by taking table lock?
- In which of the three one can "not" retrieve data even after the rollback is performed?
- Which of the above is executed by taking row-level lock?
- What is the primary use case of each one of the above mentioned?

Employees Earning More Than Their Managers

Table: `Employee`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| salary      | int     |
| managerId   | int     |
+-------------+---------+
id is the primary key column for this table.
Each row of this table indicates the ID of an employee, their name, salary, and the ID of their
manager.
```

Example 1:

Input:

Employee table:

```
+----+-------+--------+-----------+
| id | name  | salary | managerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | Null      |
| 4  | Max   | 90000  | Null      |
+----+-------+--------+-----------+
```

Output:

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

Explanation: Joe is the only employee who earns more than his manager.

Rank Scores

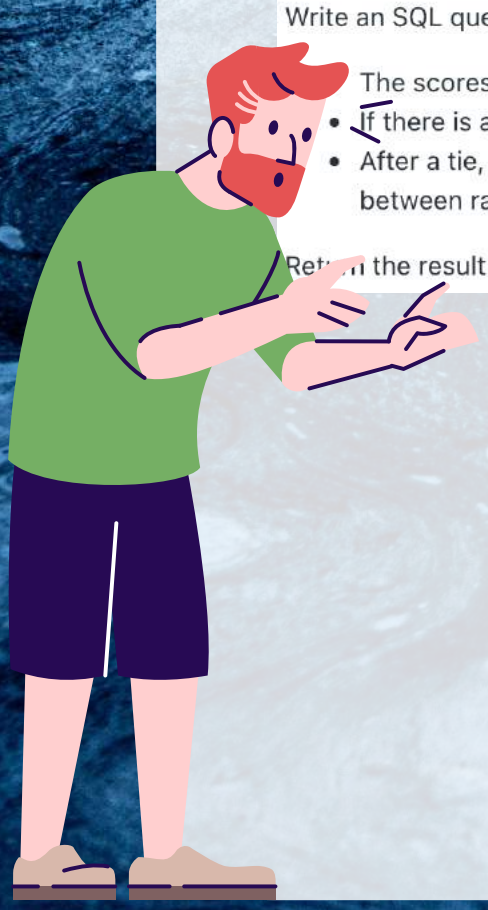
Table: `Scores`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| score       | decimal |
+-------------+---------+
id is the primary key for this table.
Each row of this table contains the score of a game. Score is a floating point value with two decimal
places.
```

Write an SQL query to rank the scores. The ranking should be calculated according to the following rules:

- The scores should be ranked from the highest to the lowest.
- If there is a tie between two scores, both should have the same ranking.
- After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by `score` in descending order.



Example 1:

Input:

Scores table:

```
+----+-------+
| id | score |
+----+-------+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+----+-------+
```

Output:

```
+-------+------+
| score | rank |
+-------+------+
| 4.00  | 1    |
| 4.00  | 1    |
| 3.85  | 2    |
| 3.65  | 3    |
| 3.65  | 3    |
| 3.50  | 4    |
+-------+------+
```

Trips & Users

Table: `Trips`

```
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| id          | int      |
| client_id   | int      |
| driver_id   | int      |
| city_id     | int      |
| status      | enum     |
| request_at  | date     |
+-------------+----------+
id is the primary key for this table.
The table holds all taxi trips. Each trip has a unique id, while client_id and driver_id are foreign
keys to the users_id at the Users table.
Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').
```

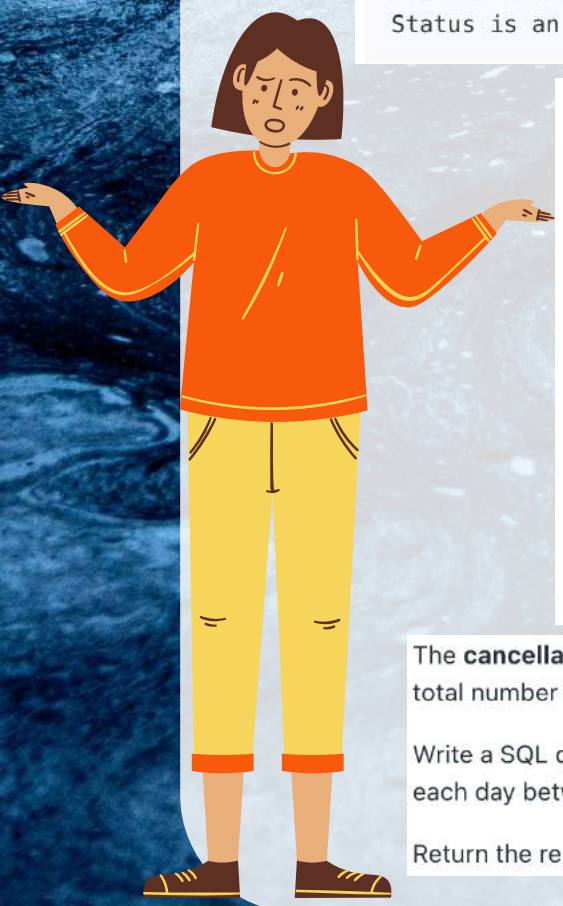


Table: `Trips`

```
+-------------+----------+
| Column Name | Type     |
+-------------+----------+
| id          | int      |
| client_id   | int      |
| driver_id   | int      |
| city_id     | int      |
| status      | enum     |
| request_at  | date     |
+-------------+----------+
id is the primary key for this table.
The table holds all taxi trips. Each trip has a unique id, while client_id and driver_id are foreign
keys to the users_id at the Users table.
Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').
```

The **cancellation rate** is computed by dividing the number of canceled (by client or driver) requests with unbanned users by the total number of requests with unbanned users on that day.

Write a SQL query to find the **cancellation rate** of requests with unbanned users (**both client and driver must not be banned**) each day between `"2013-10-01"` and `"2013-10-03"`. Round `Cancellation Rate` to **two decimal** points.

Return the result table in **any order**.

Consecutive Numbers

Table: `Logs`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| num         | varchar |
+-------------+---------+
id is the primary key for this table.
```

Write an SQL query to find all numbers that appear at least three times consecutively.

Return the result table in **any order**.

The query result format is in the following example.

Example 1:

Input:
Logs table:

```
+----+-----+
| id | num |
+----+-----+
| 1  | 1   |
| 2  | 1   |
| 3  | 1   |
| 4  | 2   |
| 5  | 1   |
| 6  | 2   |
| 7  | 2   |
+----+-----+
```

Output:

```
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
```

Explanation: 1 is the only number that appears consecutively for at least three times.

Duplicate Emails

Table: `Person`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| email       | varchar |
+-------------+---------+
id is the primary key column for this table.
Each row of this table contains an email. The emails will not contain uppercase letters.
```

Write an SQL query to report all the duplicate emails.

Return the result table in **any order**.

Example 1:

Input:
Person table:
```
+----+---------+
| id | email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```
Output:
```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```
Explanation: a@b.com is repeated two times.

Customer who never orders

Table: `Customers`

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
+-------------+---------+
id is the primary key column for this table.
Each row of this table indicates the ID and name of a customer.
```

Table: `Orders`

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| id          | int  |
| customerId  | int  |
+-------------+------+
id is the primary key column for this table.
customerId is a foreign key of the ID from the Customers table.
Each row of this table indicates the ID of an order and the ID of the customer who ordered it.
```

Write an SQL query to report all customers who never order anything.

Return the result table in **any order**.