1831. Maximum Transaction Each Day

[Leetcode Link]

Problem

In this problem, we are given a table `Transactions` with the following columns: `transaction_id`, `day`, and `amount`. Our task is to write an SQL query that reports the transaction IDs with the maximum amount for their respective days. If multiple transactions have the same amount on the same day, include all of them. The result should be sorted in ascending order by `transaction_id`.

Example

Consider the following `Transactions` table:

```
+----------------+---------------------+--------+
| transaction_id | day                 | amount |
+----------------+---------------------+--------+
| 8              | 2021-4-3 15:57:28   | 57     |
| 9              | 2021-4-28 08:47:25  | 21     |
| 1              | 2021-4-29 13:28:30  | 58     |
| 5              | 2021-4-28 16:39:59  | 40     |
| 6              | 2021-4-29 23:39:28  | 58     |
+----------------+---------------------+--------+
```

The expected result table would be:

```
+----------------+
| transaction_id |
+----------------+
| 1              |
| 5              |
| 6              |
| 8              |
+----------------+
```

Here's the walk through for the example above:

- "2021-4-3" has only one transaction with ID 8, so we add 8 to the result table.
- "2021-4-28" has two transactions with IDs 5 and 9. The transaction with ID 5 has an amount of 40, while the transaction with ID 9 has an amount of 21. We only include the transaction with ID 5 as it has the maximum amount this day.
- "2021-4-29" has two transactions with IDs 1 and 6. Both transactions have the same amount of 58, so we include both in the result table.

Finally, the result table is sorted by `transaction_id`.

Approach

To solve this problem, we can use the following approach:

1. Select the date (without time) for each transaction and group transactions by day.
2. For each day, find the transaction(s) with the maximum amount.
3. Order the result table by `transaction_id`.

Solution in SQL

We can implement this approach using SQL:

```sql
WITH daily_transactions AS (
  SELECT transaction_id, DATE(day) AS date, amount
  FROM Transactions
)

SELECT transaction_id
FROM daily_transactions
WHERE (date, amount) IN (
  SELECT date, MAX(amount)
  FROM daily_transactions
  GROUP BY date
)
ORDER BY transaction_id;
```

Explanation

1. We first create a Common Table Expression (CTE) `daily_transactions` where we store the `transaction_id`, date (without time), and `amount` of each transaction.
2. Next, we select `transaction_id` from the `daily_transactions` table, where the `(date, amount)` tuple is in the result of finding the maximum amount for each date.
3. Finally, we order the result table by `transaction_id`.## Solutions in Python, JavaScript and Java

Normally, SQL problems should be solved using SQL queries. However, if you need to implement it in a programming language, you can do that with the following code snippets for Python, JavaScript, and Java.

Python

```python
from collections import defaultdict
import datetime


def max_transactions(transactions):
    daily_transactions = defaultdict(list)

    for transaction in transactions:
        transaction_id, day, amount = transaction
        date = day.date()
        daily_transactions[date].append((transaction_id, amount))

    result = []
    for date_transactions in daily_transactions.values():
        max_amount = max(transaction[1] for transaction in date_transactions)
        max_transactions = [transaction[0] for transaction in date_transactions if transaction[1] == max_amount]
        result.extend(max_transactions)

    result.sort()
    return result


transactions = [
    (8, datetime.datetime(2021, 4, 3, 15, 57, 28), 57),
    (9, datetime.datetime(2021, 4, 28, 8, 47, 25), 21),
    (1, datetime.datetime(2021, 4, 29, 13, 28, 30), 58),
    (5, datetime.datetime(2021, 4, 28, 16, 39, 59), 40),
    (6, datetime.datetime(2021, 4, 29, 23, 39, 28), 58),
]

print(max_transactions(transactions))
```

JavaScript

```javascript
function max_transactions(transactions) {
    const daily_transactions = {};

    transactions.forEach(([transaction_id, day, amount]) => {
        const date = day.toISOString().substring(0, 10);
        if (!daily_transactions[date]) daily_transactions[date] = [];
        daily_transactions[date].push([transaction_id, amount]);
    });

    const result = [];
    Object.values(daily_transactions).forEach(date_transactions => {
        const max_amount = Math.max(...date_transactions.map(transaction => transaction[1]));
        const max_transactions = date_transactions.filter(transaction => transaction[1] === max_amount).map(transaction => transactic
        result.push(...max_transactions);
    });

    result.sort((a, b) => a - b);
    return result;
}

const transactions = [
    [8, new Date('2021-04-03T15:57:28'), 57],
    [9, new Date('2021-04-28T08:47:25'), 21],
    [1, new Date('2021-04-29T13:28:30'), 58],
    [5, new Date('2021-04-28T16:39:59'), 40],
    [6, new Date('2021-04-29T23:39:28'), 58],
];

console.log(max_transactions(transactions));
```

Java

```java
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class MaxTransactions {
    public static List<Integer> max_transactions(List<Object[]> transactions) {
        Map<LocalDate, List<Object[]>> daily_transactions = new HashMap<>();

        for (Object[] transaction : transactions) {
            Integer transaction_id = (Integer) transaction[0];
            LocalDateTime day = (LocalDateTime) transaction[1];
            Integer amount = (Integer) transaction[2];
            LocalDate date = day.toLocalDate();

            daily_transactions.putIfAbsent(date, new ArrayList<>());
            daily_transactions.get(date).add(new Object[]{transaction_id, amount});
        }

        List<Integer> result = new ArrayList<>();

        for (List<Object[]> date_transactions : daily_transactions.values()) {
            int max_amount = date_transactions.stream().mapToInt(transaction -> (int) transaction[1]).max().orElse(0);
            date_transactions.stream()
                    .filter(transaction -> (int) transaction[1] == max_amount)
                    .forEach(transaction -> result.add((Integer) transaction[0]));
        }

        result.sort(Integer::compareTo);
        return result;
    }

    public static void main(String[] args) {
        List<Object[]> transactions = List.of(
                new Object[]{8, LocalDateTime.of(2021, 4, 3, 15, 57, 28), 57},
                new Object[]{9, LocalDateTime.of(2021, 4, 28, 8, 47, 25), 21},
                new Object[]{1, LocalDateTime.of(2021, 4, 29, 13, 28, 30), 58},
                new Object[]{5, LocalDateTime.of(2021, 4, 28, 16, 39, 59), 40},
                new Object[]{6, LocalDateTime.of(2021, 4, 29, 23, 39, 28), 58}
        );

        System.out.println(max_transactions(transactions));
    }
}
```

These code snippets implement the same algorithm in different programming languages. It first processes the transactions by day, then finds the maximum amount transactions for each day, and finally combines and sorts the resulting transactions.