

A CASE STUDY

**N̄26**

BY

FIDANSOY, ALTAR

## Contents

<b>Contents</b>	<b>i</b>
<b>List of Tables</b>	<b>ii</b>
<b>List of Figures</b>	<b>iii</b>
<b>List of Listings</b>	<b>iv</b>
<b>1 Analytics Engineering Challenge</b>	<b>1</b>
1.1 Task 1	1
1.1.1 Solution 1	2
1.2 Task 2	3
1.2.1 Solution 2	4

List of Tables

1	Output Data. . . . .	1
---	----------------------	---

List of Figures

1	Task 1 Solution Output . . . . .	3
2	Task 2 SQL Output . . . . .	4
3	Task 2 Python Output . . . . .	7

## List of Listings

1	Transactions table in N26 schema. . . . .	1
2	Users table in N26 schema. . . . .	1
3	Solution Query. . . . .	2
4	Task 2 Query. . . . .	3
5	Data read function. . . . .	5
6	Cleaning the data by applying requested filters. . . . .	5
7	Joining the two data sets. . . . .	6
8	Calling the functions and printing the output. . . . .	6

# 1 Analytics Engineering Challenge

For the tasks, assume that we have a database with the following schema:

```

1 CREATE TABLE n26.transactions (
2     transaction_id UUID,
3     date DATE,
4     user_id UUID,
5     is_blocked BOOL,
6     transaction_amount INTEGER,
7     transaction_category_id INTEGER
8 );

```

List of Listings 1: Transactions table in N26 schema.

```

1 CREATE TABLE n26users (
2     user_id UUID,
3     is_active BOOLEAN
4 );

```

List of Listings 2: Users table in N26 schema.

Example data for these tables is stored in the corresponding CSV files transactions.csv and users.csv, which can be generated from the generate\_data.py script.

## 1.1 Task 1

We want to compute a DWH table for transactions. For every user transaction, it should compute the number of transactions the user had within the previous seven days. The resulting table should look something like the following example:

transaction_id	user_id	date	no_txn_last_7days
ef05-4247	becf-457e	2020-01-01	0
c8d1-40ca	becf-457e	2020-01-05	1
fc2b-4b36	becf-457e	2020-01-07	2
3725-48c4	becf-457e	2020-01-15	0
5f2a-47c2	becf-457e	2020-01-16	1
7541-412c	5728-4f1c	2020-01-01	0
3deb-47d7	5728-4f1c	2020-01-12	0

Table 1: Output Data.

**Your task is now to write a SQL query**, which computes this table based on the transactions table from the described schema. What would the query planner of the database need to consider in order to optimize the query?

Please note that it doesn't matter if the day of the current transaction is included or excluded for the calculation of the previous seven days, both solutions are fine. It also doesn't matter which SQL implementation the solution is using.

### 1.1.1 Solution 1

In the first step, I created transactions and users tables in my personal database under "n26" schema. Afterwards, I filled these tables with regenerated data sets. In order to increase the performance, I avoided to use select all and instead I only called required columns in a CTE which is called TRANSACTIONS. Then, I left join CTE with itself based on user id and date. However, for the date, I used previous 7 days for the each transaction. Finally, I grouped by the data based on transaction id, user id and date.

```
1 WITH TRANSACTIONS AS (  
2     SELECT  
3         transaction_id,  
4         user_id,  
5         CONVERT(DATE, date) AS date  
6     FROM n26.transactions)  
7 SELECT  
8     a.*,  
9     COUNT(b.transaction_id) AS no_txn_last_7days  
10 FROM TRANSACTIONS a  
11 LEFT JOIN TRANSACTIONS b ON a.user_id = b.user_id  
12     AND b.date BETWEEN DATEADD(DAY, -7, a.date)  
13     AND DATEADD(DAY, -1, a.date)  
14 GROUP BY a.transaction_id, a.user_id, a.date  
15 ORDER BY a.user_id, a.date
```

List of Listings 3: Solution Query.

By running the query above, I obtained the below output (see Figure 1). In this solution, same day transactions are not considered and that why you will see 0 in first two rows

because for this specific user, there is no any other historical transaction earlier than 2023-07-01. In order to validate the output, I checked the row number 23 (2023-07-08) and has 22 transactions which matches the number of transactions starting from 2023-07-01.

	ABC transaction_id	ABC user_id	date	123 no_txn_last_7days
1	50996370-f01b-42e8-88ae-80dc29bfe407	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-01	0
2	347351f8-9308-441a-ba18-0939911494bd	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-01	0
3	3e39bb21-7343-4a8b-86a0-c4baae4608e5	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-02	2
4	2901cefb-937c-48ce-9047-326b91aa1559	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-02	2
5	3c5a8c4d-9ee1-4ca6-ba3f-e44729c7ea6b	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-02	2
6	d787c710-7f05-436a-8059-6bc76b7a781c	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-03	5
7	1e543041-5b8e-4a69-9934-76e475faaf7	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-03	5
8	04fe14f8-d5e2-45c8-8777-f060148012b3	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-03	5
9	3f2e6807-ff39-4d5f-ad22-8c6a6f860f22	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-04	8
10	34d24702-65d3-42df-8465-525a13e37033	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-04	8
11	aef0141e-3328-4bf1-88fe-654d3191c103	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-04	8
12	4d690b37-0371-40c0-bfba-fc237b35025d	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-05	11
13	00d05903-e81e-43c1-b277-cba73073a482	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-05	11
14	9ee93658-02e2-4a6f-a658-c5d954144334	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-05	11
15	0b5cbabf-fb19-44eb-9adf-2677c5515666	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-06	14
16	9ecb4fab-1162-42ec-81dd-bf8a9e1fafeb	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-06	14
17	3e4cc19e-9b08-4f91-9a6f-53f9ee873b23	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
18	436cb4fe-e1ac-4ed4-a212-3f05e88d08d8	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
19	61bd00fb-fe5b-45a5-8d0a-73d829841269	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
20	e1a89275-0e49-423e-8e65-a4a0191ddbba	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
21	96e60380-cc4e-4ea2-83da-91a861562071	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
22	e149c9cf-a908-4504-8028-52c1825698c5	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-07	16
23	5db18201-d804-4464-a347-38f7510485af	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-08	22
24	3d7c106a-f7bd-4e40-b55a-e72e66798435	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-08	22
25	adf8dd0e-9d64-4e34-b7a4-3a9a213a969b	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-08	22
26	7211ab9d-7028-4790-b968-80f15f601293	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-09	23
27	9d383942-2962-4b68-8746-17c3aa6e3d80	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-10	21
28	93d63c91-66ca-412f-8a66-d6d32ab13199	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-10	21
29	29f0448a-aff2-4b1a-8d0e-5787066bbff7	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-10	21
30	a798258d-e2cd-49f4-a914-4b8e4521ac70	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-11	21
31	81538083-abcc-4c20-b16d-d547281e2d3a	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-11	21
32	4e856541-b17c-44e2-accf-77b59196c882	00508d90-86f4-435a-a7e1-9b8f4c6fe2ee	2023-07-11	21

Figure 1: Task 1 Solution Output

## 1.2 Task 2

We want to compute the result of the following query:

```

1  SELECT t.transaction_category_id,
2         sum(t.amount) AS sum_amount,
3         count(DISTINCT t.user_id) AS num_users
4  FROM transactions t JOIN users u USING (user_id)
5  WHERE t.is_blocked = False AND u.is_active = 1
6  GROUP BY t.transaction_category_id
7  ORDER BY sum_amount DESC;
```

List of Listings 4: Task 2 Query.



But unfortunately the database query planner cannot optimize the query well enough in order for us to get the results.

**Your task is now to write a Python program** using only the **standard libraries** (i.e. any external package that would require installation should not be used), which reads data from the CSV files `transactions.csv` and `users.csv`, and computes the equivalent result of the SQL query. The result should be printed to `stdout`.

Please note that the scope of the task is **not to parse the SQL query** or to generalize the computation in any way, but only to write a program which computes the result of this one specific query.

### 1.2.1 Solution 2

The query in Listing 4, generated the output in below (see Figure 2). This output will be use for validating the python output at the end of the this section.

	ABC transaction_category_id	123 sum_amount	123 num_users
1	2	4.851,02	92
2	3	4.845,06	81
3	0	4.318,86	79
4	5	4.042,7	79
5	1	3.981,19	79
6	10	3.842,18	73
7	8	3.779,1	79
8	7	3.774,9	81
9	9	3.738,91	73
10	6	3.689,34	72
11	4	3.036,74	62

Figure 2: Task 2 SQL Output

```
1 import csv
2
3 def read_csv(filename: str) -> list[list]:
4     with open(filename, 'r') as transactions_file:
5         data = [line.strip().split(',') \
6                 for line in transactions_file.readlines()]
7     return data
```

List of Listings 5: Data read function.

In the first step, I imported csv library which is an internal python library. Afterwards, I created a function which reads csv data and returns it as a python list by splitting it.

```
1 def cleansed(transactions_data: list[list], users_data: list[list]) ->
   tuple[list, list]:
2     transactions = {}
3     users = {}
4     for row in transactions_data[1:]:
5         if row[3] == 'FALSE':
6             transaction_id, date, user_id, is_blocked, \
7             transaction_amount, transaction_category_id = row
8             transactions[transaction_id] = {
9                 'transaction_category_id': transaction_category_id,
10                'transaction_amount': float(transaction_amount),
11                'user_id': user_id,
12                'is_blocked': is_blocked
13            }
14     for row in users_data[1:]:
15         if row[1] == 'TRUE':
16             user_id, is_active = row
17             users[user_id] = {
18                 'is_active': is_active
19             }
20     return transactions, users
```

List of Listings 6: Cleaning the data by applying requested filters.

Afterwards, I cleaned the data by applying the requested conditions. Cleansed function filters out the blocked transactions and inactive users.

```

1 def merge_and_sort(transactions: list[dict], users: list[dict]) -> list[
    dict]:
2     result = {}
3     for transaction_id, transaction_data in transactions.items():
4         user_id = transaction_data['user_id']
5         if user_id in users:
6             transaction_category_id = \
7                 transaction_data['transaction_category_id']
8             amount = transaction_data['transaction_amount']
9             if transaction_category_id not in result:
10                result[transaction_category_id] = \
11                    {'sum_amount': 0, 'num_users': set()}
12
13                result[transaction_category_id]['sum_amount'] = \
14                    result[transaction_category_id]['sum_amount'] + amount
15                result[transaction_category_id]['num_users'].add(user_id)
16
17    sorted_result = sorted(result.items(), \
18        key=lambda x: x[1]['sum_amount'], reverse=True)
19    return sorted_result

```

List of Listings 7: Joining the two data sets.

By using the function above, I basically merged users and transactions data based on category id only. Thus, for each iteration, I sum the amount and count the users.

```

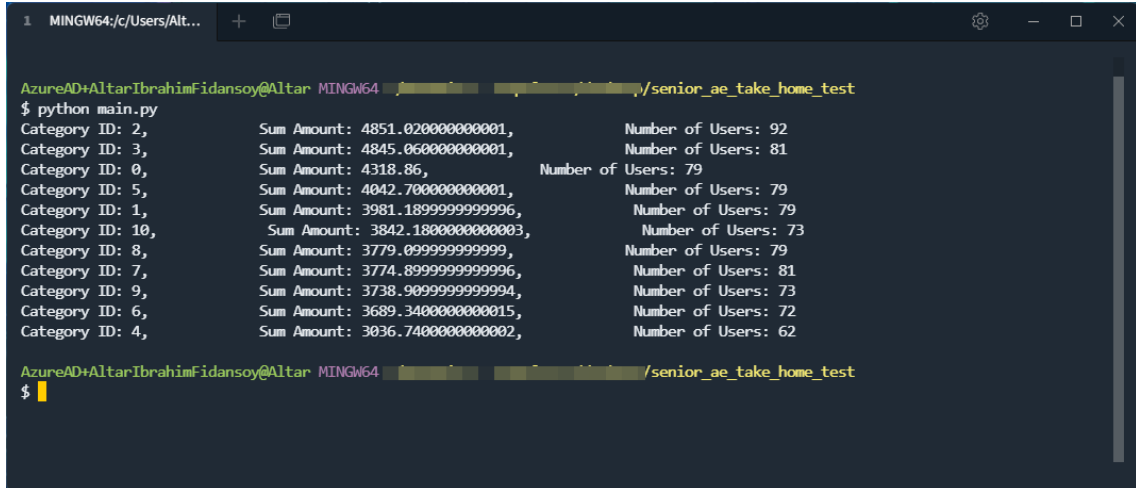
1 if __name__ == '__main__':
2     transactions_data = read_csv("./CSV/transactions.csv")
3     users_data = read_csv("./CSV/users.csv")
4     transactions, users = cleansed(transactions_data, users_data)
5     data = merge_and_sort(transactions, users)
6     for category_id, category_data in data:
7         sum_amount = category_data['sum_amount']
8         num_users = len(category_data['num_users'])
9         print(f"Category ID: {category_id}, \
10             Sum Amount: {sum_amount}, \
11             Number of Users: {num_users}")

```

List of Listings 8: Calling the functions and printing the output.

Finally, in the main function, I called all the functions which are explained above. First,

read\_csv function is called for both transactions and users data sets. Then, the cleansed function is called to clean these data sets. Afterwards, merge\_and\_sort function is called to obtain results. At the end results are printed to screen.



```
MINGW64/c/Users/Alt... + [icon] [icon] [icon]
AzureAD+AltarIbrahimFidansoy@Altar MINGW64 [redacted] /senior_ae_take_home_test
$ python main.py
Category ID: 2,          Sum Amount: 4851.020000000001,      Number of Users: 92
Category ID: 3,          Sum Amount: 4845.060000000001,      Number of Users: 81
Category ID: 0,          Sum Amount: 4318.86,                Number of Users: 79
Category ID: 5,          Sum Amount: 4042.700000000001,      Number of Users: 79
Category ID: 1,          Sum Amount: 3981.1899999999996,      Number of Users: 79
Category ID: 10,         Sum Amount: 3842.1800000000003,      Number of Users: 73
Category ID: 8,          Sum Amount: 3779.0999999999999,      Number of Users: 79
Category ID: 7,          Sum Amount: 3774.8999999999996,      Number of Users: 81
Category ID: 9,          Sum Amount: 3738.9099999999994,      Number of Users: 73
Category ID: 6,          Sum Amount: 3689.3400000000015,      Number of Users: 72
Category ID: 4,          Sum Amount: 3036.7400000000002,      Number of Users: 62

AzureAD+AltarIbrahimFidansoy@Altar MINGW64 [redacted] /senior_ae_take_home_test
$
```

Figure 3: Task 2 Python Output

Figure 3, shows exactly the same numbers with SQL output which is given in Figure 2. All data files, scripts and documents are can be found in this [GitHub](#) repository.