iykra

IYKRA **Final Project**

ft.

Online Payment Fraud data

Group 4

DE BOIS

FUTURE DE

OUR TEAM

Amin Pohan

Fahmi Hamzah

Laurenzius Julio

Suryo Hamukti

TABLE OF CONTENTS

01

PROBLEM DESCRIPTION

PROBLEM DESCRIPTION

Online payment systems have helped many people to make payments instantly. But on the other hand, it also increases payment fraud. That is why detecting online payment fraud is very important for financial technology companies to ensure that customers are not getting charged for the products and services they never pay.

As data engineers, we need to build a data engineering infrastructure to turn raw data into dashboards that can later be used to see the company's transactions data health.

Primary Objectives:

- Build an end-to-end data pipeline
- Build Dashboard
- Check Transaction’s Health

02

DATA UNDERSTANDING

DATA UNDERSTANDING

- 1. step: represents a unit of time where 1 step equals 1 hour
- 2. type: type of online transaction
- 3. amount: the amount of the transaction
- 4. nameOrig: customer starting the transaction
- 5. oldbalanceOrg: balance before the transaction
- 6. newbalanceOrig: balance after the transaction
- 7. nameDest: recipient of the transaction
- 8. oldbalanceDest: initial balance of recipient before the transaction
- 9. newbalanceDest: the new balance of recipient after the transaction
- 10. isFraud: fraud transaction

03

DATA PREPROCESS

- First off, it is a good idea to check for nulls before proceeding with anything.

```
# check if there are any nulls or missing values in the dataset
df.isna().sum()
```

[10]

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Huh, not even a single missing value, cool.

- Next up, let's see what kind of data we're dealing with

```
df.head()
```

[6]

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

Okay, I guess.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

- Surprisingly enough, this dataset does not have any mistyped column, so that's a relief.

```
[5]
df['step'].unique()
df['step'].value_counts()
```

```
19     51352
18     49579
187    49083
235    47491
307    46968
       ...
432        4
706        4
693        4
112        2
662        2
Name: step, Length: 743, dtype: int64
```

```
[4]
# 743 steps equal roughly 31 days which can be used as timestamp
x = 743/24
print(x)
```

```
30.958333333333332
```

It's like they wanted us to figure out how to add the timestamp by ourselves

- Right off the bat, we noticed that there were no columns of any explicit timestamp, but! There is one column called **step** that represents a unit of time where 1 step equals 1 hour, which can be used to create a timestamp field. We'll hop on that a little bit later, though.

```
[7]
# check how many rows are confirmed to be fraud
df['isFraud'].value_counts()

0    6354407
1       8213
Name: isFraud, dtype: int64
```

```
[3]
# check how many rows are flagged for fraud
df['isFlaggedFraud'].value_counts()

0    6362604
1         16
Name: isFlaggedFraud, dtype: int64
```

```
[11]
# check how many rows of flagged for fraud is actually confirmed to be fraud
df['isFraud'][df['isFlaggedFraud']==1].value_counts()

1    16
Name: isFraud, dtype: int64
```

I wonder what the algorithm for fraud flagging and the actual fraud confirmation looks like.

- Next up, let's actually check how many fraud transactions there are in this dataset, as well as how many were flagged for fraud and how many were actually confirmed to be fraud.
- There are **8213** counts of fraudulent transactions and **16** flagged for fraud, in which all those **16** flagged for fraud were **indeed fraudulent transactions**.

- Alright, back on to the creation of timestamp utilizing the **step** column.

```
# initialize the days and hours
num_days = 7
num_hours = 24
df['days'] = df['step']%num_days
df['hours'] = df['step']%num_hours - 1
df['day_trans'] = df['step']/24
df['day_trans'] = round(df['day_trans'])
```
[16]

```
# initialize the starting date for the timestamp
df['date'] = pd.to_datetime(df['day_trans'],unit='D',origin=pd.Timestamp('2022-12-01 00:00:00'))
```
[17]

```
# append the hours to the timestamp
df['date_hour'] = df['date'] + pd.TimedeltaIndex(df['hours'], unit='H')
```
[18]

```
# initialize the random seconds for the minute
n = df.shape[0]
rand_list=[]
for i in range(n):
    rand_list.append(random.randint(0,360))

df['random_seconds'] = rand_list
```
[20]

```
# append the minute to the timestamp
df['datetime'] = df['date_hour'] + pd.TimedeltaIndex(df['random_seconds'], unit='seconds')
```
[22]

- *Voilà.*

```
df['datetime']
```

[23]

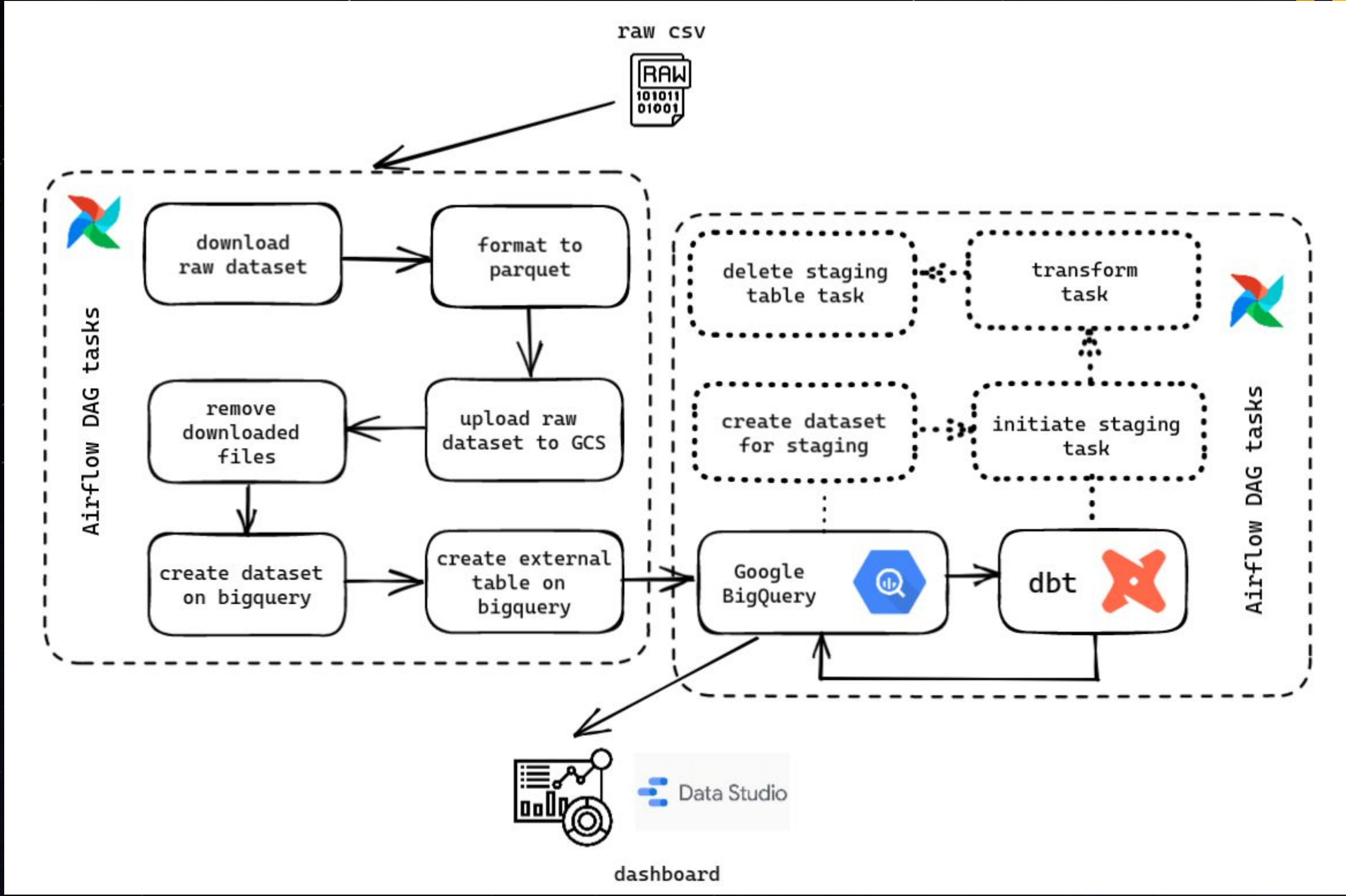
```
0         2022-12-01 00:04:22
1         2022-12-01 00:00:28
2         2022-12-01 00:04:22
3         2022-12-01 00:02:10
4         2022-12-01 00:01:31
                 ...
6362615   2023-01-01 22:04:17
6362616   2023-01-01 22:05:31
6362617   2023-01-01 22:03:17
6362618   2023-01-01 22:04:29
6362619   2023-01-01 22:00:38
Name: datetime, Length: 6362620, dtype: datetime64[ns]
```

```
# drop the placeholder columns leaving the single datetime field as the datetime column.
df.drop(['date','date_hour','random_seconds', 'days', 'hours', 'day_trans'],axis=1,inplace=True)
```

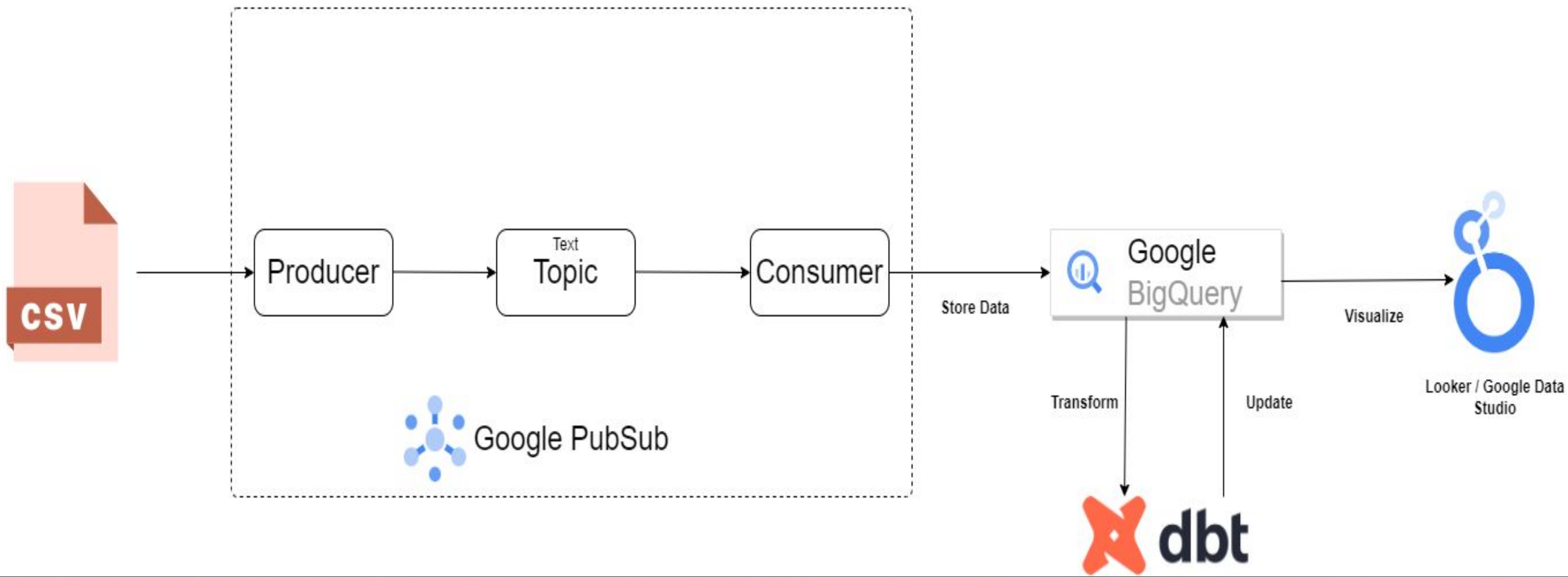
[24]

04

DATA INGESTION & WORKFLOW


raw csv
RAW
101011
01001
Airflow DAG tasks
download raw dataset
format to parquet
upload raw dataset to GCS
remove downloaded files
create dataset on bigquery
create external table on bigquery
delete staging table task
transform task
create dataset for staging
initiate staging task
Google BigQuery
dbt
Airflow DAG tasks
Data Studio
dashboard

Stream Processing WorkFlow



AIRFLOW DAGs

Shall be explained by Fahmi Hamzah

05

DATA TRANSFORMATION

dbt lineage graph

```
{{ config(materialized="view") }}

select
    md5(step || date || nameOrig || nameDest || type || amount) as transaction_id,
    CAST(date AS datetime) AS timestamp,
    CAST(step AS numeric) AS step,
    nameOrig AS IdSender,
    CAST(amount AS numeric) AS Amount,
    {{payment_type_desc ('type') }} AS type_id,
    CAST(oldbalanceOrg AS numeric) AS PrevBalanceSender,
    CAST(newbalanceOrig AS numeric) AS NewBalanceSender,
    nameDest AS IdReceiver,
    CAST(oldbalanceDest AS numeric) AS PrevBalanceReceiver,
    CAST(newbalanceDest  AS numeric) AS NewBalanceReceiver,
    CAST(isFraud AS integer) AS Fraud,
    CAST(isFlaggedFraud AS integer) AS FlaggedFraud

from {{ source("staging", "online_payment_view") }}
```

- stg_onlinepayment
- Staging phase of data transformation utilizing dbt where data types are recast to ensure good data structure as well as proper **naming conventions** for easier data understanding.

```
{{config (materialized="table") }}

SELECT
    transaction_id,
    timestamp,
    IdSender,
    Amount,
    IdReceiver,
    Fraud,
    type_id

FROM {{ref ('stg_onlinepayment')}}
```

- dim_fraud
- Dimension Table for Fraud data

```
{{config(materialized="table")}}

SELECT
    DISTINCT(type_id) AS id,
    CASE
        WHEN type_id =  1 then 'PAYMENT'
        when type_id =  2 then 'CASH_OUT'
        when type_id =  3 then 'CASH_IN'
        when type_id =  4 then 'TRANSFER'
        when type_id =  5 then 'DEBIT'
    end AS payment_type

FROM {{ref ('stg_onlinepayment')}}
```

- dim_type_transaction
- Dimension Table for Payment Types data

cool guy fahmi hamzah for modelling this payment type dimension table.

```
{{ config(materialized="table") }}

SELECT
    IdReceiver,
    COUNT(IdReceiver) AS total_trx,
    Fraud,
    ARRAY_AGG(STRUCT(transaction_id,
            NewBalanceReceiver - PrevBalanceReceiver AS BalanceDiff)) AS trx

FROM
    {{ref ('stg_onlinepayment')}}
GROUP BY
    IdReceiver,
    Fraud
ORDER BY
    total_trx desc, Fraud desc
```

- dim_receiver
- Dimension Table for Receiver data

```
{{ config(materialized="table") }}

SELECT
    IdSender,
    COUNT(IdSender) AS total_trx,
    Fraud,
    ARRAY_AGG(STRUCT(transaction_id,
            NewBalanceSender - PrevBalanceSender AS BalanceDiff)) AS trx

FROM
    {{ref ('stg_onlinepayment')}}
GROUP BY
    IdSender,
    Fraud
ORDER BY
    total_trx desc, Fraud desc
```

- dim_sender
- Dimension Table for Sender data

```
{{config (materialized="table") }}

SELECT
    transaction_id,
    step,
    type_id,
    PrevBalanceSender,
    NewBalanceSender,
    PrevBalanceReceiver,
    NewBalanceReceiver,
    FlaggedFraud
FROM
    {{ ref('stg_onlinepayment') }}
```

- online_payment
- Online Payment Table Data

```sql
{{ config(
  materialized = 'table',
  partition_by={
    "field": "date",
    "data_type": "timestamp",
    "granularity": "day"}
    )
}}
with sender AS(
    SELECT IdSender, total_trx,
    t.transaction_id as transaction_id,
    t.BalanceDiff as BalanceDiff
FROM {{ref ('dim_sender') }},
UNNEST(trx) as t
),
receiver AS (
    SELECT IdReceiver, total_trx,
    t.transaction_id as transaction_id,
    t.BalanceDiff AS BalanceDiff
FROM {{ref ('dim_receiver') }},
UNNEST(trx) as t
)
```

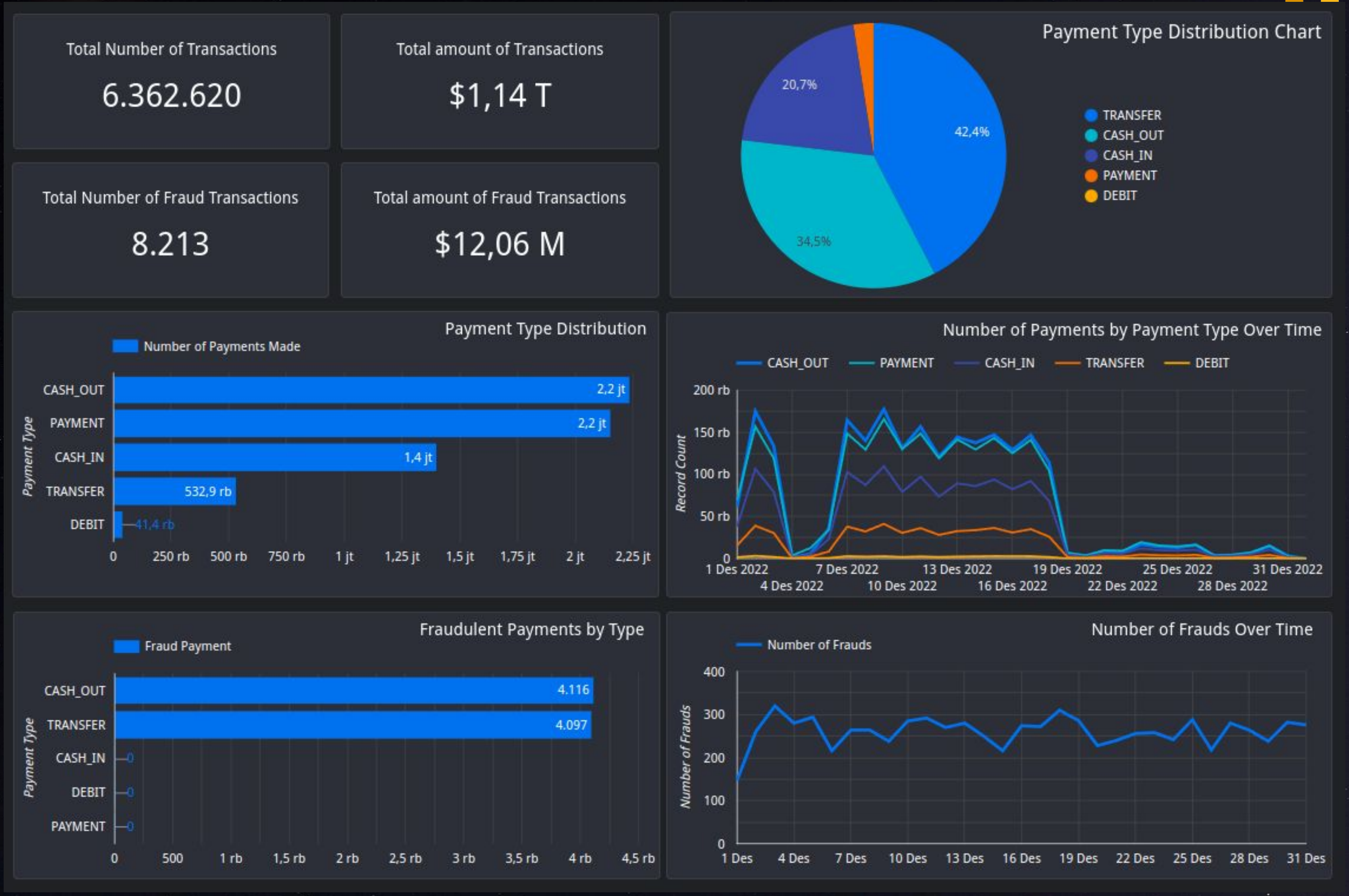
```sql
SELECT
    fraudData.transaction_id AS transaction_id,
    fraudData.timestamp AS date,
    payment.payment_type,
    fraudData.IdSender AS IdSender,
    sender.total_trx AS CountTrxAsSender,
    fraudData.Amount AS AmountOfTrx,
    online_payment.PrevBalanceSender AS PrevBalanceSender,
    online_payment.NewBalanceSender AS NewBalanceSender,
    sender.BalanceDiff AS SendBalanceDiff,
    fraudData.IdReceiver AS IdReceiver,
    receiver.total_trx AS CountTrxAsRcv,
    online_payment.PrevBalanceReceiver AS PrevBalanceReceiver,
    online_payment.NewBalanceReceiver AS NewBalanceReceiver,
    receiver.BalanceDiff AS RcvBalanceDiff,
    fraudData.Fraud

FROM {{ref ('dim_fraud')}} AS fraudData
LEFT JOIN {{ref ('online_payment')}} AS online_payment
    ON fraudData.transaction_id = online_payment.transaction_id
LEFT JOIN {{ref ('dim_type_transaction')}} AS payment
    ON fraudData.type_id = payment.id
LEFT JOIN sender
    ON fraudData.transaction_id = sender.transaction_id
LEFT JOIN receiver
    ON fraudData.transaction_id = receiver.transaction_id
```

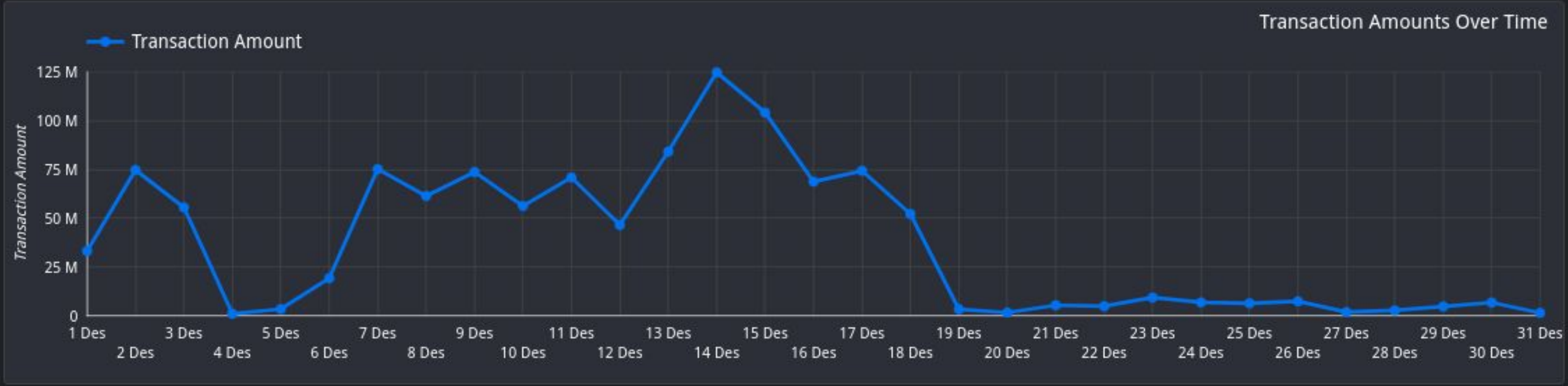
- fact_fraud
- Fact table creation/transformation script where **common table expression** and partitioning is utilized in which the partitioning is done by **day** of the date.
- This fact table is the product of the previously modelled tables.

06

DASHBOARD



Total Number of Transactions
6.362.620
Total amount of Transactions
$1,14 T
Total Number of Fraud Transactions
8.213
Total amount of Fraud Transactions
$12,06 M
Payment Type Distribution Chart
42,4%
34,5%
20,7%
TRANSFER
CASH_OUT
CASH_IN
PAYMENT
DEBIT
Payment Type Distribution
Number of Payments Made
Payment Type
CASH_OUT 2,2 jt
PAYMENT 2,2 jt
CASH_IN 1,4 jt
TRANSFER 532,9 rb
DEBIT 41,4 rb
0 250 rb 500 rb 750 rb 1 jt 1,25 jt 1,5 jt 1,75 jt 2 jt 2,25 jt
Number of Payments by Payment Type Over Time
CASH_OUT PAYMENT CASH_IN TRANSFER DEBIT
Record Count
200 rb 150 rb 100 rb 50 rb 0
1 Des 2022 4 Des 2022 7 Des 2022 10 Des 2022 13 Des 2022 16 Des 2022 19 Des 2022 22 Des 2022 25 Des 2022 28 Des 2022 31 Des 2022
Fraudulent Payments by Type
Fraud Payment
Payment Type
CASH_OUT 4.116
TRANSFER 4.097
CASH_IN 0
DEBIT 0
PAYMENT 0
0 500 1 rb 1,5 rb 2 rb 2,5 rb 3 rb 3,5 rb 4 rb 4,5 rb
Number of Frauds Over Time
Number of Frauds
Number of Frauds
400 300 200 100 0
1 Des 4 Des 7 Des 10 Des 13 Des 16 Des 19 Des 22 Des 25 Des 28 Des 31 Des



Summary Table Based on Total Amount

Payment Type	Average Amount	Total Amount	Confirmed Fraud	Flagged for Fraud	Record Count
TRANSFER	910.647,01	485.291.987.263,16	4.097	16	532.909
CASH_OUT	176.273,96	394.412.995.224,49	4.116	0	2.237.500
CASH_IN	168.920,24	236.367.391.912,46	0	0	1.399.284
PAYMENT	13.057,6	28.093.371.138,37	0	0	2.151.495
DEBIT	5.483,67	227.199.221,28	0	0	41.432

The dashboard can be publicly accessed using the link below:
https://datastudio.google.com/reporting/85df6c5e-d1ba-4bbf-a5ef-8cce3aa15422

SOURCE CODE:

bit.ly/FinalProjectDF8_Grup4

"Sucking at something is the first step to becoming sorta good at something"

-Jake The Dog

THANKS!

Do you have any questions?

Do tell!