BUILD END-TO-END DATA PIPELINE FOR ONLINE PAYMENT FRAUD

Group 4

1. Asti Nuswantari
2. Fahrul Rozi Meiduan
3. Rifqi Manufi Fathur Rahman

Mentor: Fiqri Wicaksono

OUTLINES

01

BUSINESS OVERVIEW

PROBLEM STATEMENT

Financial institutions increasingly rely upon data-driven methods for developing fraud detection systems, which are able to automatically detect and block fraudulent transactions. From a machine learning perspective, the task of detecting suspicious transactions is a binary classification problem and therefore many techniques can be applied.

In this case, company needs to know about why our consumer have negative trend. We as a data specialist should analysis the problem. Lists like this one:

- The transaction healthy or not
- Build Data Pipeline
- Build Data Mart & Dashboard

02

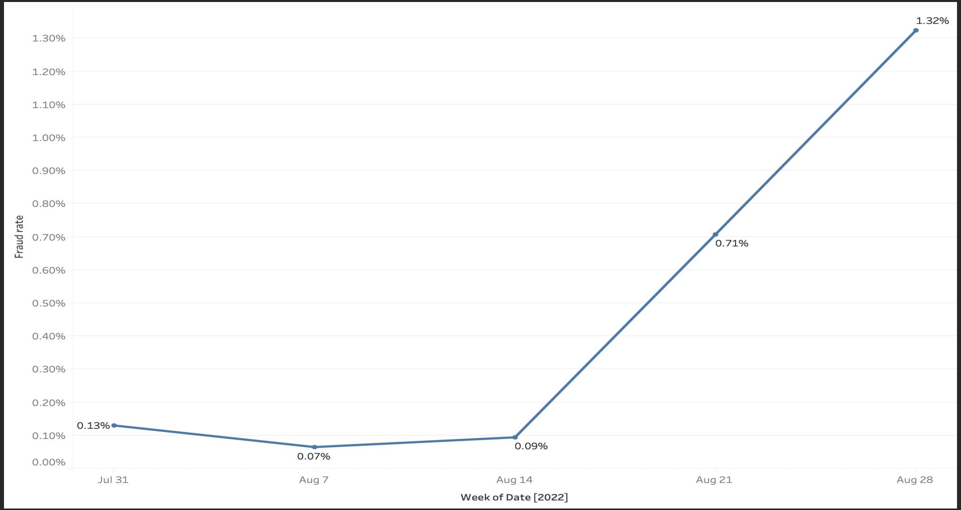
DATA UNDERSTANDING

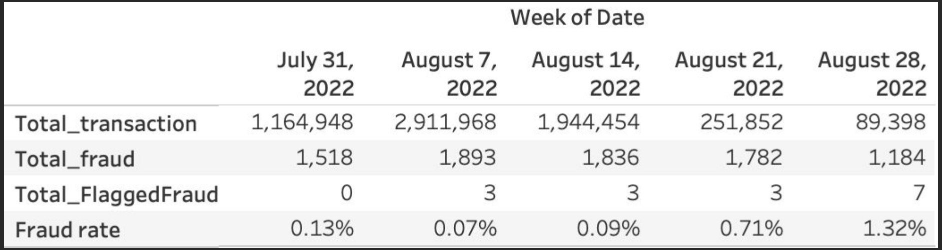
DATA DESCRIPTION

Description:

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
11. Datetime (added column) : data timestamp

DATA UNDERSTANDING





	Week of Date				
	July 31, 2022	August 7, 2022	August 14, 2022	August 21, 2022	August 28, 2022
Total_transaction	1,164,948	2,911,968	1,944,454	251,852	89,398
Total_fraud	1,518	1,893	1,836	1,782	1,184
Total_FlaggedFraud	0	3	3	3	7
Fraud rate	0.13%	0.07%	0.09%	0.71%	1.32%

● Fraud rate

Increasing fraud rate by week

● Transaction

Decreasing transaction behavior

03

GOALS & STRATEGIES

GOALS

Serve ready to analysis data from our data which is online payment fraud data into several tables that has been normalized and visualize data distribution

STRATEGIES

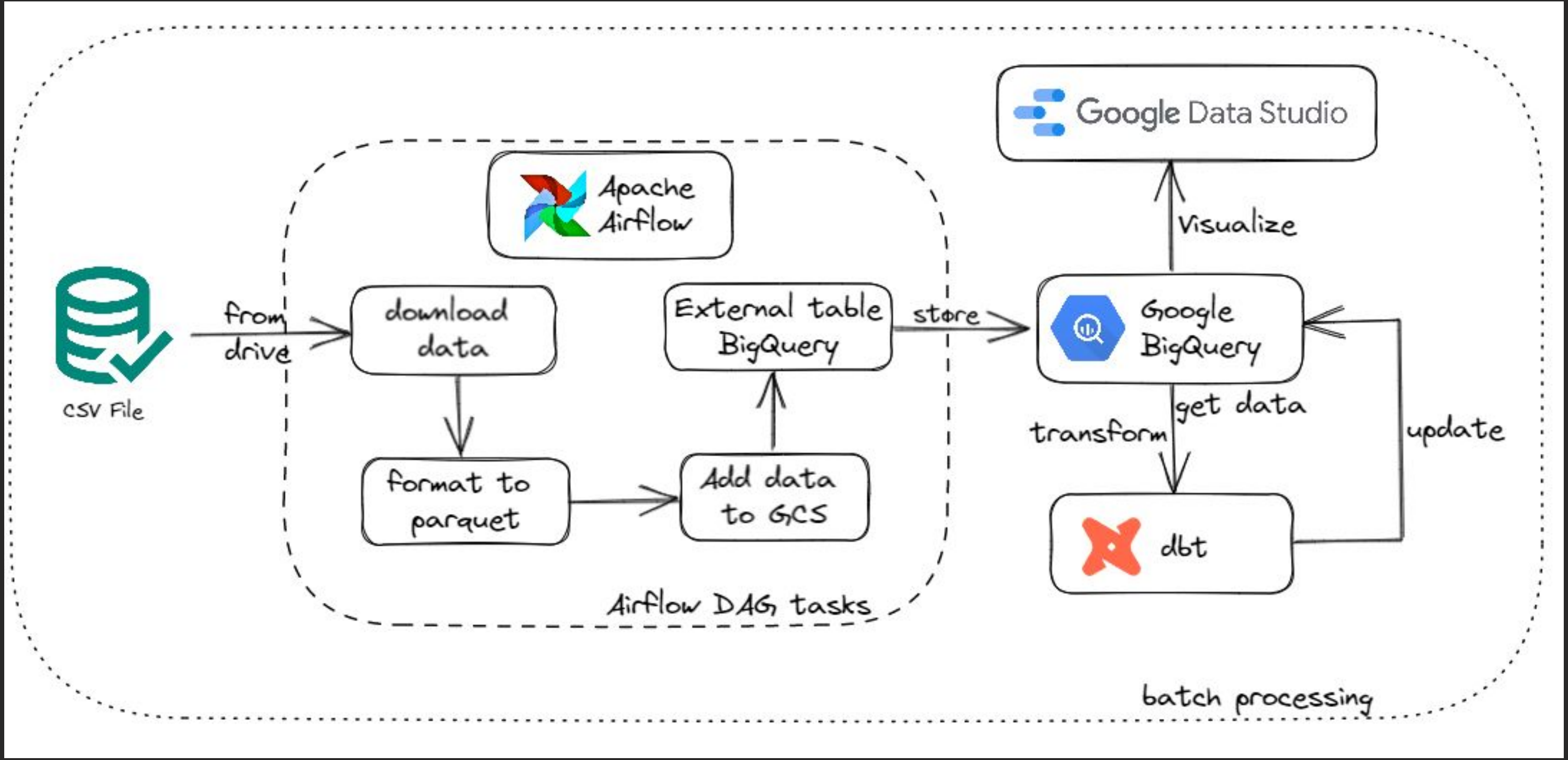
build an end-to-end data pipeline from ingestion data with Airflow to bigQuery until create dashboard

04

CLOUD & DATA INGESTION

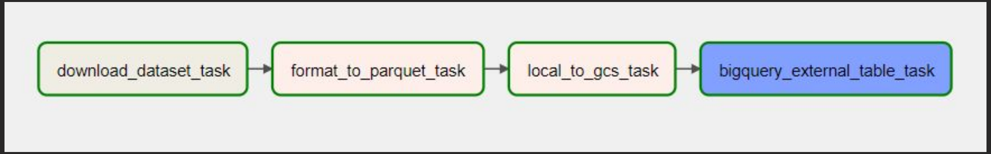
Batch & Stream Processing

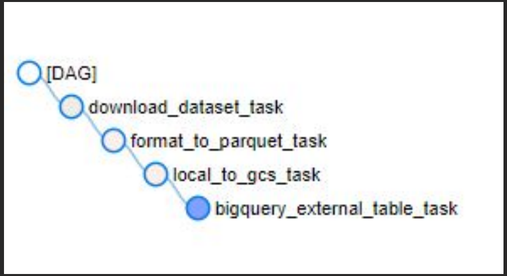
Batch Processing Workflow



Airflow DAGs

- Since our initialize data is in zip file, we tried to move the extracted file data to another gdrive.
- DAG is consist of 4 tasks:
  - download_dataset_task
  - format_to_parquet_task
  - local_to_gcs_task
  - bigquery_external_table_task
- DAG in airflow

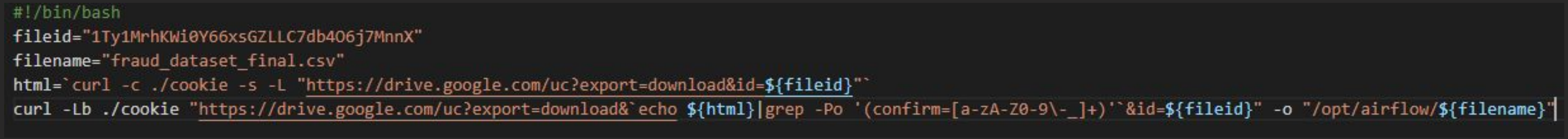




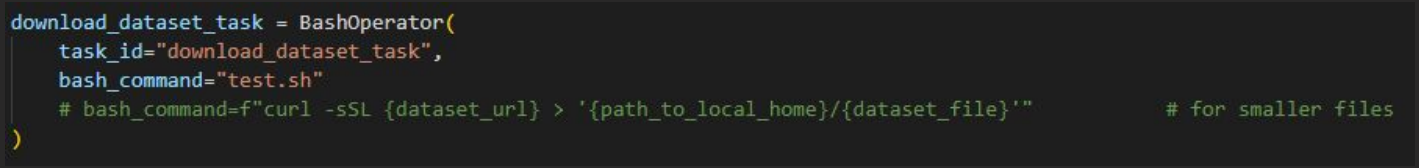
Airflow DAGs

download_dataset_task

- In this step, we download dataset from google drive with name fraud_dataset_final.csv
- Since the file is big in size, tried to download the data from test.sh file and run the bashOperator command in airflow.
- test.sh and download_dataset_task code looks like this:



```bash
#!/bin/bash
fileid="1Ty1MrhKWi0Y66xsGZLLC7db4O6j7MnnX"
filename="fraud_dataset_final.csv"
html=`curl -c ./cookie -s -L "https://drive.google.com/uc?export=download&id=${fileid}"`
curl -Lb ./cookie "https://drive.google.com/uc?export=download&`echo ${html}|grep -Po '(confirm=[a-zA-Z0-9\-_]+)'`&id=${fileid}" -o "/opt/airflow/${filename}"
```

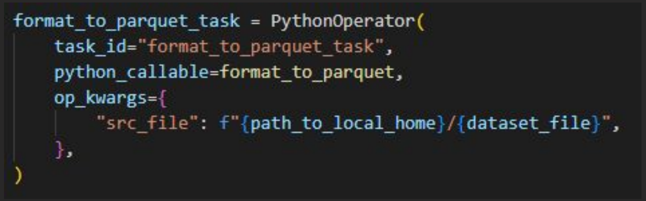


```python
download_dataset_task = BashOperator(
    task_id="download_dataset_task",
    bash_command="test.sh"
    # bash_command=f"curl -sSL {dataset_url} > '{path_to_local_home}/{dataset_file}'"           # for smaller files
)
```

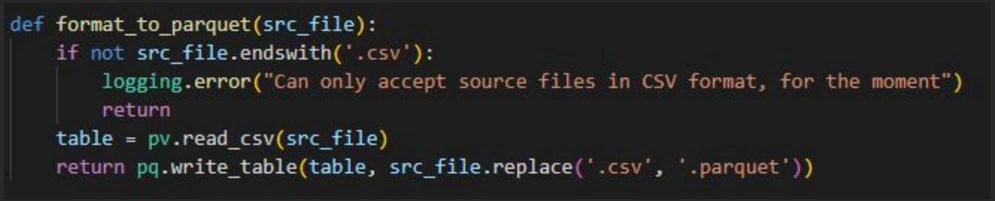
Airflow DAGs

format_to_parquet_task

- In this step, we tried to convert csv dataset to parquet.
- Dataset file size is around 591MB.
- Since the data is too big, we decided to convert our dataset to parquet so we can save up some space and time during uploading data to GCS.
- format_to_parquet_task and format_to_parquet function looks like this:



```python
format_to_parquet_task = PythonOperator(
    task_id="format_to_parquet_task",
    python_callable=format_to_parquet,
    op_kwargs={
        "src_file": f"{path_to_local_home}/{dataset_file}",
    },
)
```

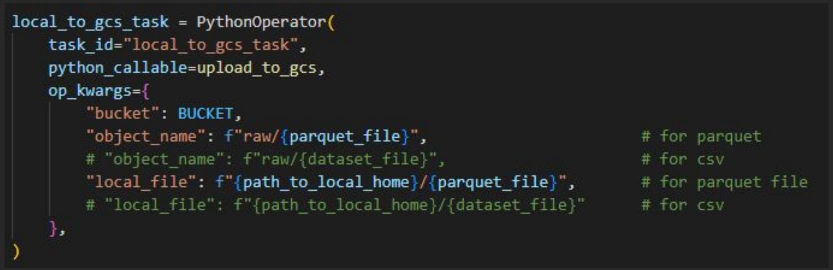


```python
def format_to_parquet(src_file):
    if not src_file.endswith('.csv'):
        logging.error("Can only accept source files in CSV format, for the moment")
        return
    table = pv.read_csv(src_file)
    return pq.write_table(table, src_file.replace('.csv', '.parquet'))
```

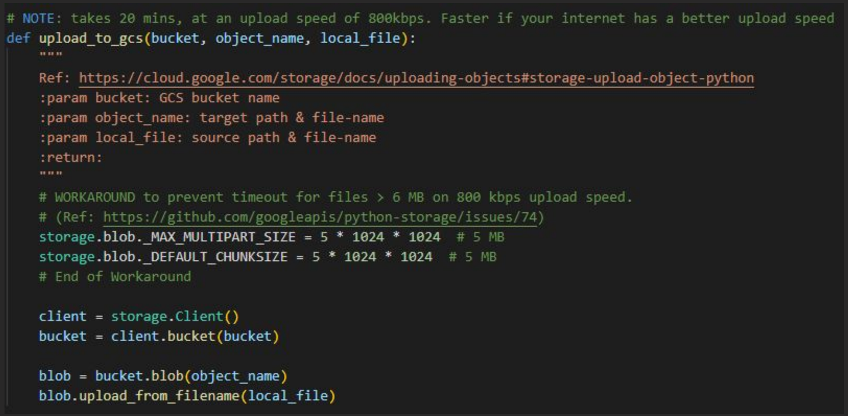
Airflow DAGs

local_to_gcs_task

- In this step, we tried to upload the parquet file to Google Cloud Storage (GCS)
- local_to_gcs_task will call upload_to_gcs function.
- This will upload our parquet file to GCS based on object assigned, in this case data will be available in raw/fraud_dataset_final.parquet
- local_to_gcs_task and upload_to_gcs function looks like this:



```python
local_to_gcs_task = PythonOperator(
    task_id="local_to_gcs_task",
    python_callable=upload_to_gcs,
    op_kwargs={
        "bucket": BUCKET,
        "object_name": f"raw/{parquet_file}",                       # for parquet
        # "object_name": f"raw/{dataset_file}",                     # for csv
        "local_file": f"{path_to_local_home}/{parquet_file}",       # for parquet file
        # "local_file": f"{path_to_local_home}/{dataset_file}"      # for csv
    },
)
```

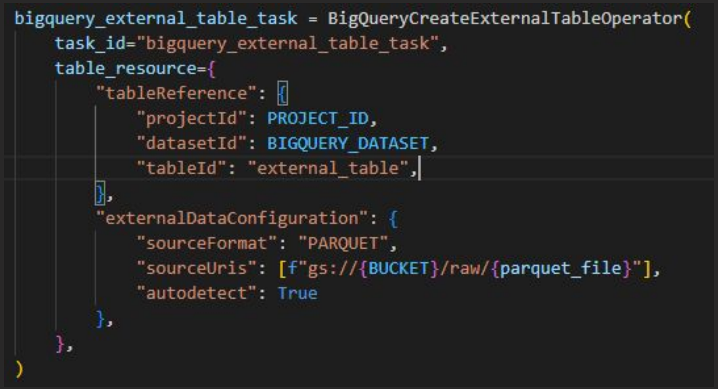


```python
# NOTE: takes 20 mins, at an upload speed of 800kbps. Faster if your internet has a better upload speed
def upload_to_gcs(bucket, object_name, local_file):
    """
    Ref: https://cloud.google.com/storage/docs/uploading-objects#storage-upload-object-python
    :param bucket: GCS bucket name
    :param object_name: target path & file-name
    :param local_file: source path & file-name
    :return:
    """
    # WORKAROUND to prevent timeout for files > 6 MB on 800 kbps upload speed.
    # (Ref: https://github.com/googleapis/python-storage/issues/74)
    storage.blob._MAX_MULTIPART_SIZE = 5 * 1024 * 1024  # 5 MB
    storage.blob._DEFAULT_CHUNKSIZE = 5 * 1024 * 1024  # 5 MB
    # End of Workaround

    client = storage.Client()
    bucket = client.bucket(bucket)

    blob = bucket.blob(object_name)
    blob.upload_from_filename(local_file)
```

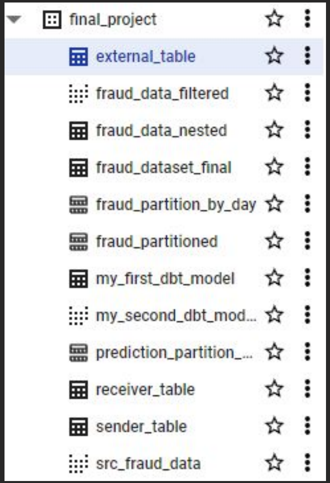
Airflow DAGs

bigquery_external_table_task

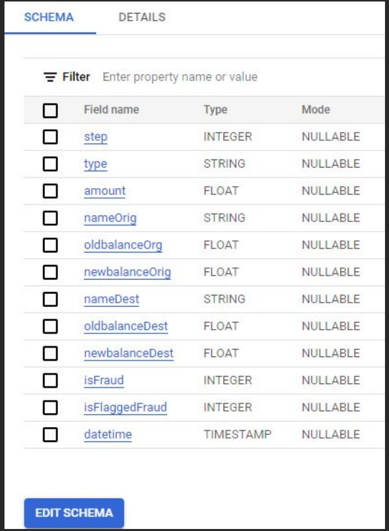
- In this step, we transfer data from our GCS to BigQuery.
- Output from this task is external table in BigQuery.
- bigquery_external_table_task looks like this:



```
bigquery_external_table_task = BigQueryCreateExternalTableOperator(
    task_id="bigquery_external_table_task",
    table_resource={
        "tableReference": {
            "projectId": PROJECT_ID,
            "datasetId": BIGQUERY_DATASET,
            "tableId": "external_table",
        },
        "externalDataConfiguration": {
            "sourceFormat": "PARQUET",
            "sourceUris": [f"gs://{BUCKET}/raw/{parquet_file}"],
            "autodetect": True
        },
    },
)
```

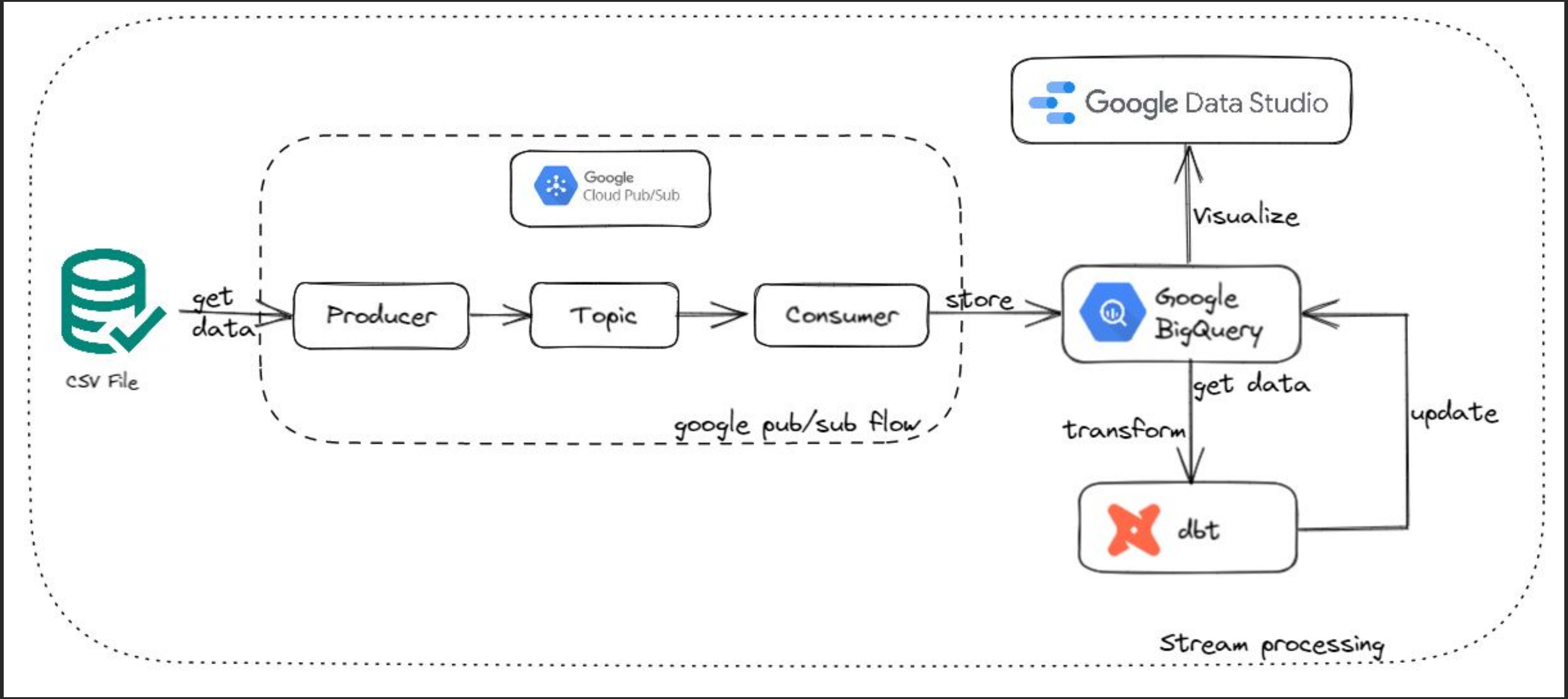


- final_project
  - external_table
  - fraud_data_filtered
  - fraud_data_nested
  - fraud_dataset_final
  - fraud_partition_by_day
  - fraud_partitioned
  - my_first_dbt_model
  - my_second_dbt_mod...
  - prediction_partition_...
  - receiver_table
  - sender_table
  - src_fraud_data



Field name	Type	Mode
step	INTEGER	NULLABLE
type	STRING	NULLABLE
amount	FLOAT	NULLABLE
nameOrig	STRING	NULLABLE
oldbalanceOrg	FLOAT	NULLABLE
newbalanceOrig	FLOAT	NULLABLE
nameDest	STRING	NULLABLE
oldbalanceDest	FLOAT	NULLABLE
newbalanceDest	FLOAT	NULLABLE
isFraud	INTEGER	NULLABLE
isFlaggedFraud	INTEGER	NULLABLE
datetime	TIMESTAMP	NULLABLE

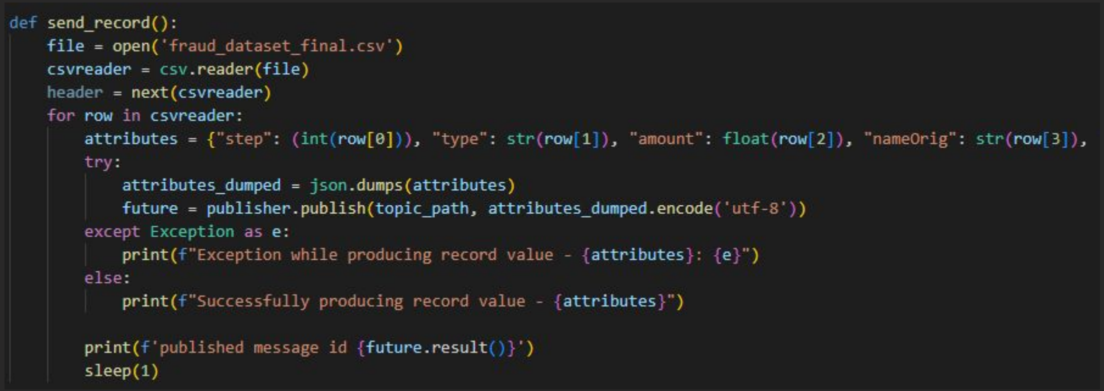
Stream Processing Workflow



Stream Processing PubSub

Publisher.py

- Also known as producer
- This step will get the dataset and send the data based on our schema to consumer
- Data send to using topic that is connected to Google Cloud Platform (pubsub section)
- In code, function to send data is called as send_record()

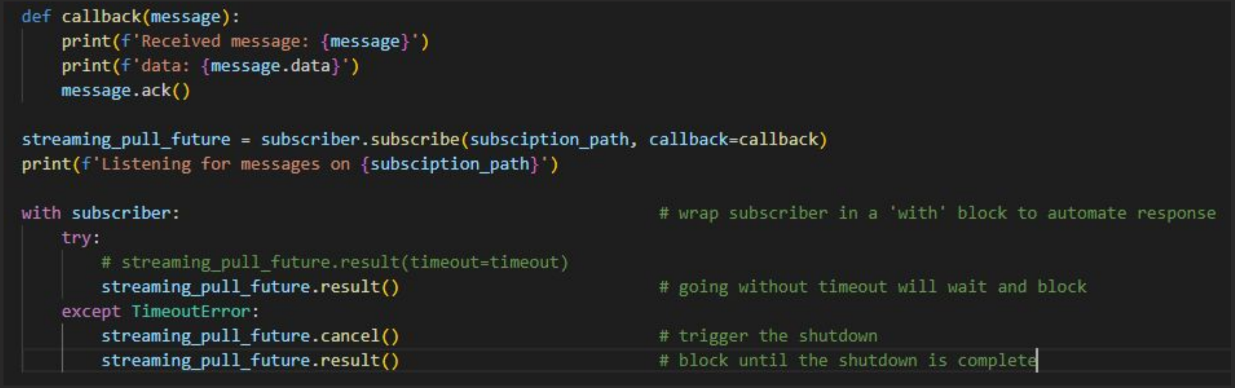


```python
def send_record():
    file = open('fraud_dataset_final.csv')
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        attributes = {"step": (int(row[0])), "type": str(row[1]), "amount": float(row[2]), "nameOrig": str(row[3]),
        try:
            attributes_dumped = json.dumps(attributes)
            future = publisher.publish(topic_path, attributes_dumped.encode('utf-8'))
        except Exception as e:
            print(f"Exception while producing record value - {attributes}: {e}")
        else:
            print(f"Successfully producing record value - {attributes}")

        print(f'published message id {future.result()}')
        sleep(1)
```

Stream Processing PubSub

Subscriber.py

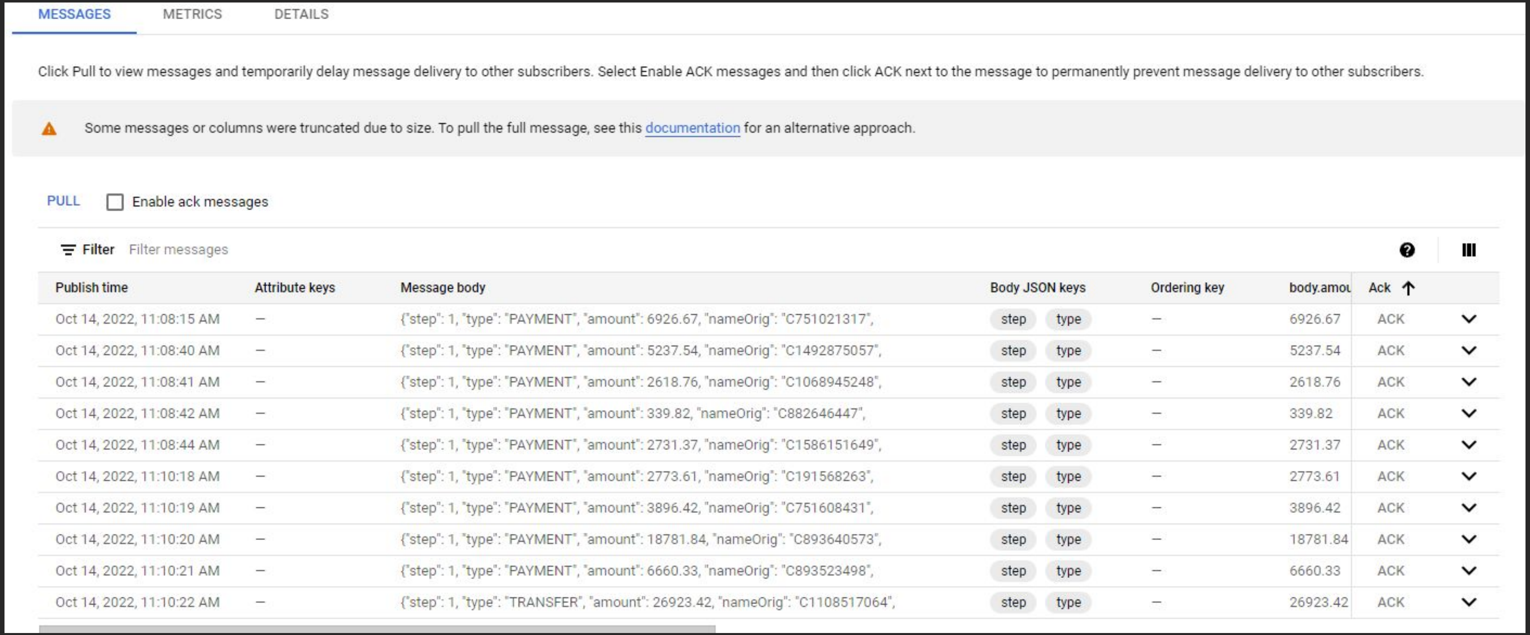
- Also known as consumer
- This step will listen the data that is send by our producer.
- You can also check the data that is send by producer in Google Cloud PubSub section.
- To get the data, just pull message in sub topic



```
def callback(message):
    print(f'Received message: {message}')
    print(f'data: {message.data}')
    message.ack()

streaming_pull_future = subscriber.subscribe(subsciption_path, callback=callback)
print(f'Listening for messages on {subsciption_path}')

with subscriber:                                          # wrap subscriber in a 'with' block to automate response
    try:
        # streaming_pull_future.result(timeout=timeout)
        streaming_pull_future.result()                    # going without timeout will wait and block
    except TimeoutError:
        streaming_pull_future.cancel()                    # trigger the shutdown
        streaming_pull_future.result()                    # block until the shutdown is complete
```

Stream Processing PubSub

Example of pulling message from sub topic Google PubSub



MESSAGES METRICS DETAILS

Click Pull to view messages and temporarily delay message delivery to other subscribers. Select Enable ACK messages and then click ACK next to the message to permanently prevent message delivery to other subscribers.

Some messages or columns were truncated due to size. To pull the full message, see this documentation for an alternative approach.

PULL ☐ Enable ack messages

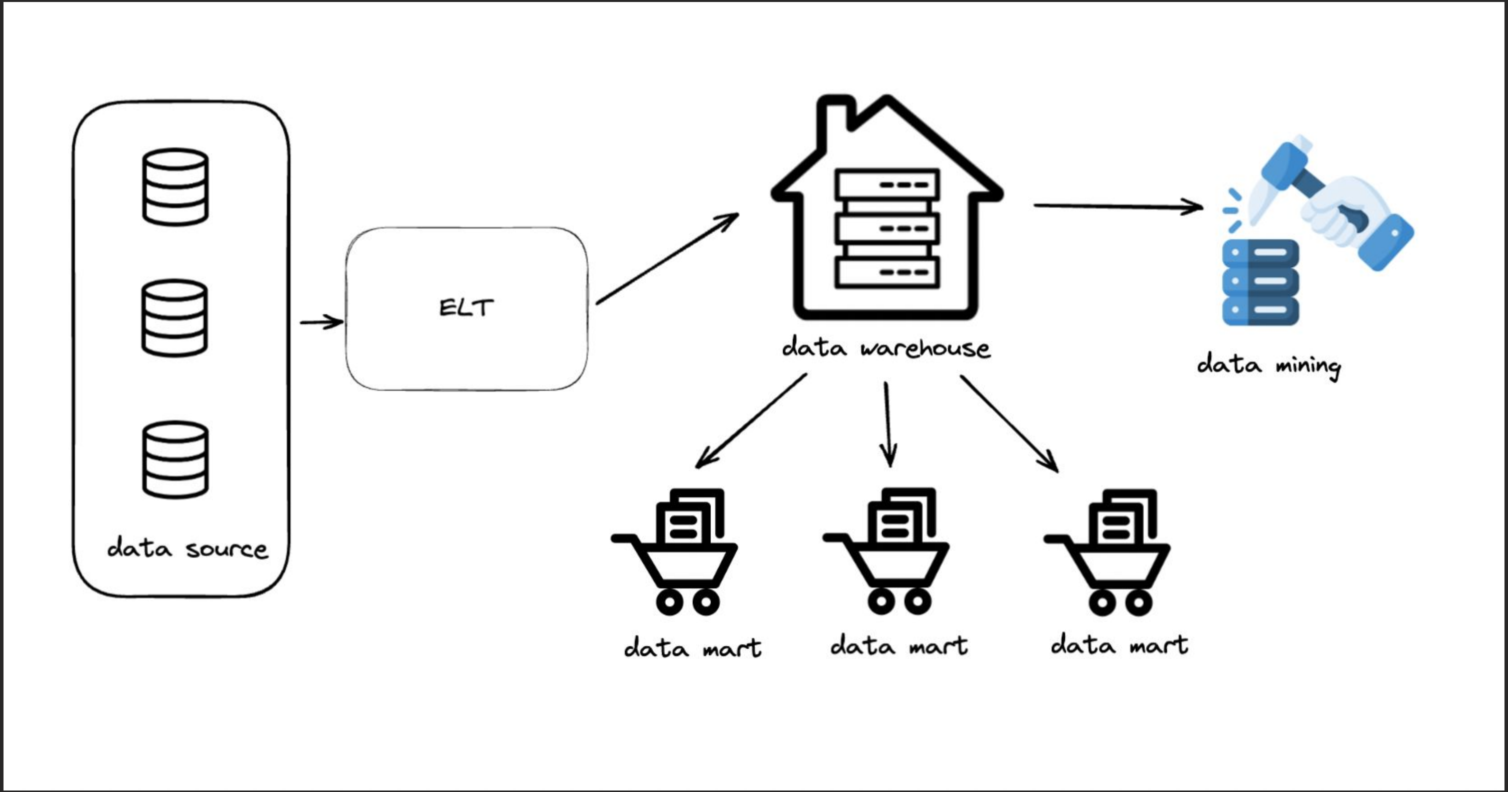
Filter Filter messages

Publish time	Attribute keys	Message body	Body JSON keys	Ordering key	body.amou	Ack ↑	
Oct 14, 2022, 11:08:15 AM	–	{"step": 1, "type": "PAYMENT", "amount": 6926.67, "nameOrig": "C751021317",	step type	–	6926.67	ACK	⌄
Oct 14, 2022, 11:08:40 AM	–	{"step": 1, "type": "PAYMENT", "amount": 5237.54, "nameOrig": "C1492875057",	step type	–	5237.54	ACK	⌄
Oct 14, 2022, 11:08:41 AM	–	{"step": 1, "type": "PAYMENT", "amount": 2618.76, "nameOrig": "C1068945248",	step type	–	2618.76	ACK	⌄
Oct 14, 2022, 11:08:42 AM	–	{"step": 1, "type": "PAYMENT", "amount": 339.82, "nameOrig": "C882646447",	step type	–	339.82	ACK	⌄
Oct 14, 2022, 11:08:44 AM	–	{"step": 1, "type": "PAYMENT", "amount": 2731.37, "nameOrig": "C1586151649",	step type	–	2731.37	ACK	⌄
Oct 14, 2022, 11:10:18 AM	–	{"step": 1, "type": "PAYMENT", "amount": 2773.61, "nameOrig": "C191568263",	step type	–	2773.61	ACK	⌄
Oct 14, 2022, 11:10:19 AM	–	{"step": 1, "type": "PAYMENT", "amount": 3896.42, "nameOrig": "C751608431",	step type	–	3896.42	ACK	⌄
Oct 14, 2022, 11:10:20 AM	–	{"step": 1, "type": "PAYMENT", "amount": 18781.84, "nameOrig": "C893640573",	step type	–	18781.84	ACK	⌄
Oct 14, 2022, 11:10:21 AM	–	{"step": 1, "type": "PAYMENT", "amount": 6660.33, "nameOrig": "C893523498",	step type	–	6660.33	ACK	⌄
Oct 14, 2022, 11:10:22 AM	–	{"step": 1, "type": "TRANSFER", "amount": 26923.42, "nameOrig": "C1108517064",	step type	–	26923.42	ACK	⌄

05

DATA WAREHOUSE & DATA TRANSFORMATION

DATA WAREHOUSE DESIGN



TRANSFORMATION DATA

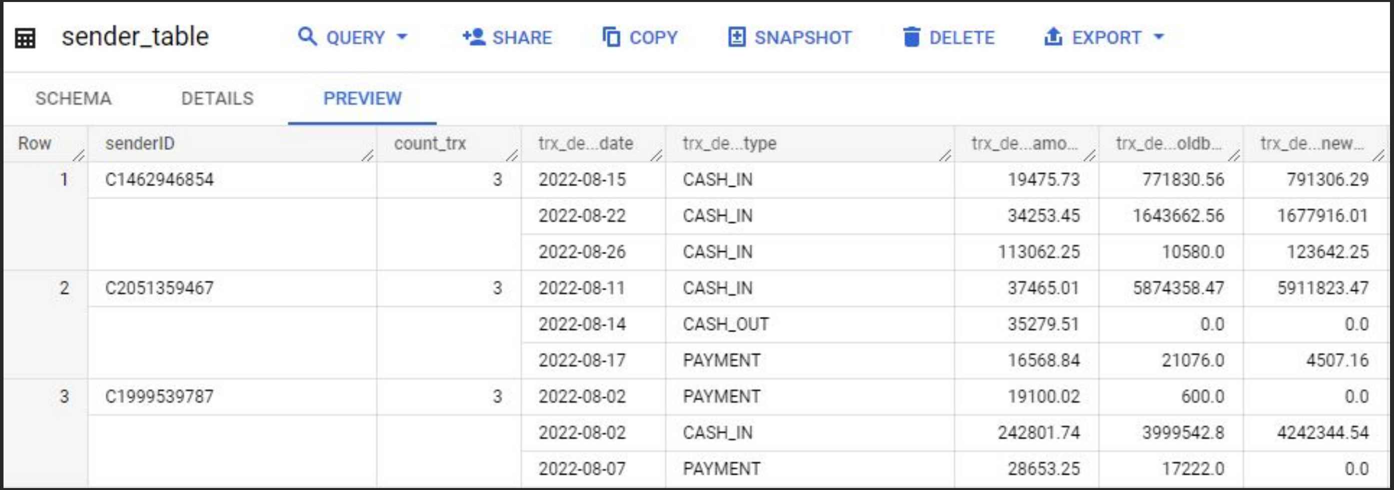
We used dbt to transform data.

- Add ‘datetime’ column based on step
- Extract date and time from datetime column
- Replace Orig with ‘Sender’
- Add ‘difsender’ by calculating the difference of newbalanceSend and oldbalanceSend
- Add ‘difReceiver’ by calculating the difference of newbalanceReceive and oldbalanceReceive

```sql
create or replace table
`complete-aviary-362903.final_project_iykra.fraud` as
  select step, type,
  amount,
  nameOrig as senderID,
  oldbalanceOrg as oldbalanceSend,
  newbalanceOrig as newbalanceSend,
  abs(newbalanceOrig-oldbalanceOrg) as difsender,
  nameDest as ReceiverID,
  oldbalanceDest as oldbalancereceive,
  newbalanceDest as newbalancereceive,
  abs(newbalanceDest-oldbalanceDest) as difreceiver,
  isFraud,
  isFlaggedFraud,
  `datetime`,
  extract(date from `datetime`) as `date`,
  extract(time from `datetime`) as `time`
  from
`complete-aviary-362903.final_project_iykra.fraud_dataset
_final`
  order by `datetime`;
```

TRANSFORMATION DATA

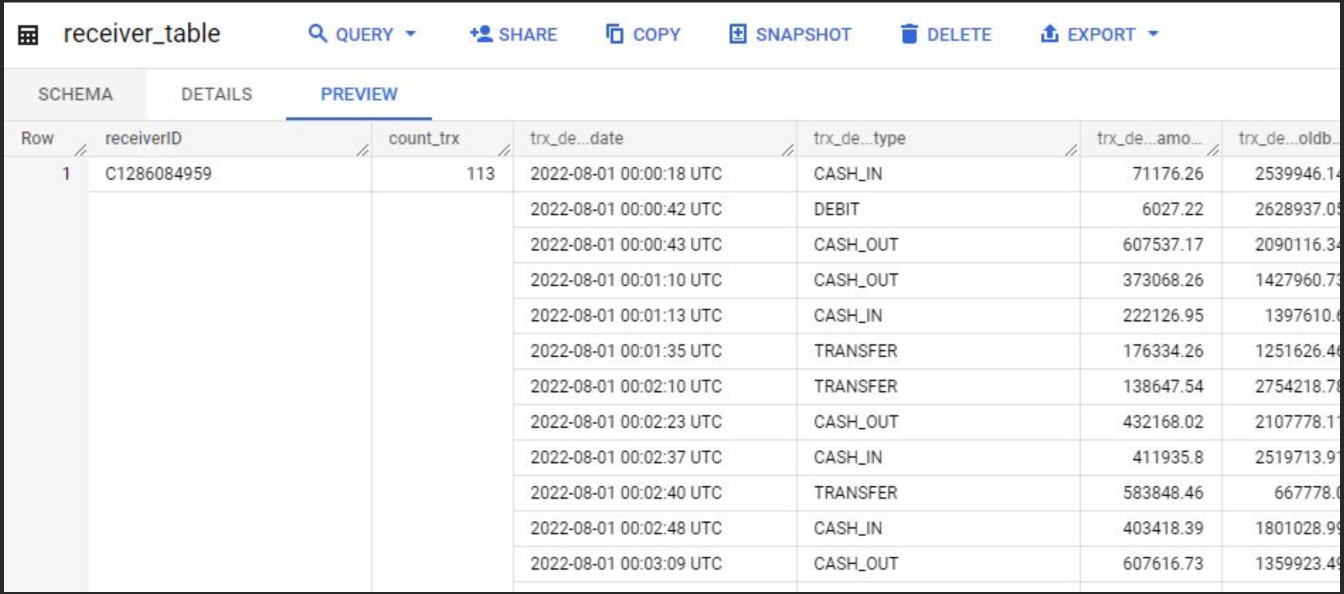
Normalization data with separating sender_table from raw data



Row	senderID	count_trx	trx_de...date	trx_de...type	trx_de...amo...	trx_de...oldb...	trx_de...new...
1	C1462946854	3	2022-08-15	CASH_IN	19475.73	771830.56	791306.29
			2022-08-22	CASH_IN	34253.45	1643662.56	1677916.01
			2022-08-26	CASH_IN	113062.25	10580.0	123642.25
2	C2051359467	3	2022-08-11	CASH_IN	37465.01	5874358.47	5911823.47
			2022-08-14	CASH_OUT	35279.51	0.0	0.0
			2022-08-17	PAYMENT	16568.84	21076.0	4507.16
3	C1999539787	3	2022-08-02	PAYMENT	19100.02	600.0	0.0
			2022-08-02	CASH_IN	242801.74	3999542.8	4242344.54
			2022-08-07	PAYMENT	28653.25	17222.0	0.0

TRANSFORMATION DATA

Normalization data with separating receiver_table from raw data



receiver_table

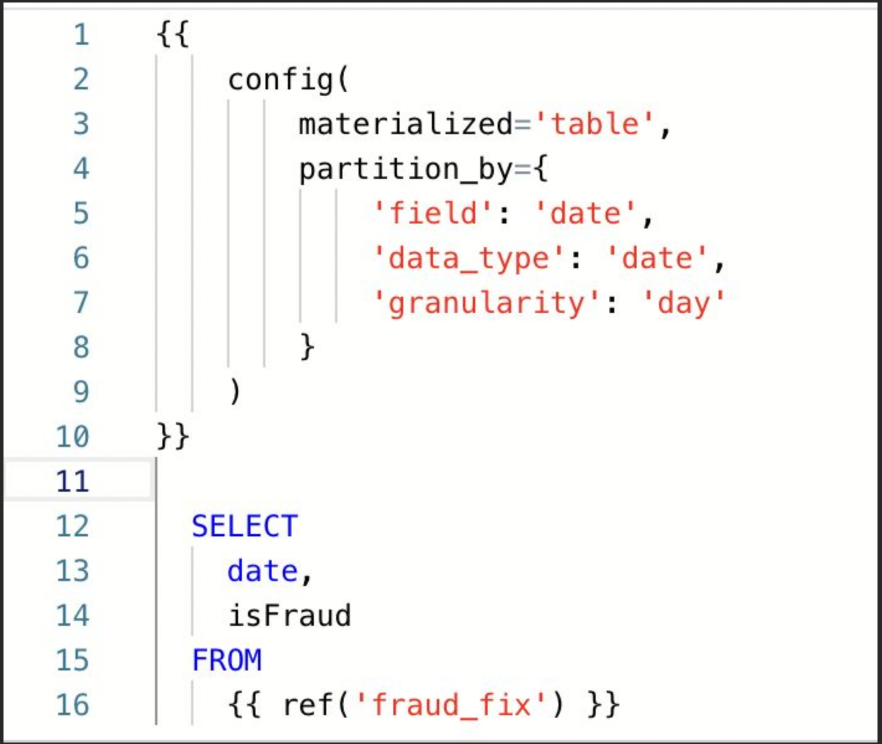
QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW

Row	receiverID	count_trx	trx_de...date	trx_de...type	trx_de...amo...	trx_de...oldb...
1	C1286084959	113	2022-08-01 00:00:18 UTC	CASH_IN	71176.26	2539946.14
			2022-08-01 00:00:42 UTC	DEBIT	6027.22	2628937.05
			2022-08-01 00:00:43 UTC	CASH_OUT	607537.17	2090116.34
			2022-08-01 00:01:10 UTC	CASH_OUT	373068.26	1427960.73
			2022-08-01 00:01:13 UTC	CASH_IN	222126.95	1397610.6
			2022-08-01 00:01:35 UTC	TRANSFER	176334.26	1251626.46
			2022-08-01 00:02:10 UTC	TRANSFER	138647.54	2754218.78
			2022-08-01 00:02:23 UTC	CASH_OUT	432168.02	2107778.11
			2022-08-01 00:02:37 UTC	CASH_IN	411935.8	2519713.91
			2022-08-01 00:02:40 UTC	TRANSFER	583848.46	667778.0
			2022-08-01 00:02:48 UTC	CASH_IN	403418.39	1801028.99
			2022-08-01 00:03:09 UTC	CASH_OUT	607616.73	1359923.49

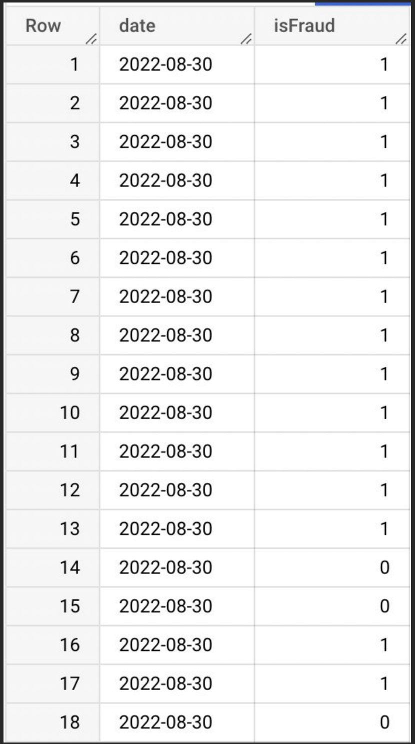
TRANSFORMATION DATA

Data Fraud Partitioned by day



```
{{
    config(
        materialized='table',
        partition_by={
            'field': 'date',
            'data_type': 'date',
            'granularity': 'day'
        }
    )
}}

SELECT
  date,
  isFraud
FROM
  {{ ref('fraud_fix') }}
```

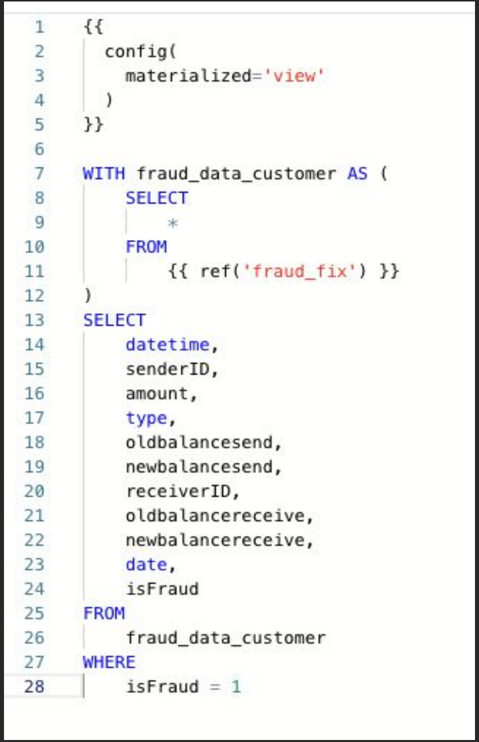
Table Result



Row	date	isFraud
1	2022-08-30	1
2	2022-08-30	1
3	2022-08-30	1
4	2022-08-30	1
5	2022-08-30	1
6	2022-08-30	1
7	2022-08-30	1
8	2022-08-30	1
9	2022-08-30	1
10	2022-08-30	1
11	2022-08-30	1
12	2022-08-30	1
13	2022-08-30	1
14	2022-08-30	0
15	2022-08-30	0
16	2022-08-30	1
17	2022-08-30	1
18	2022-08-30	0

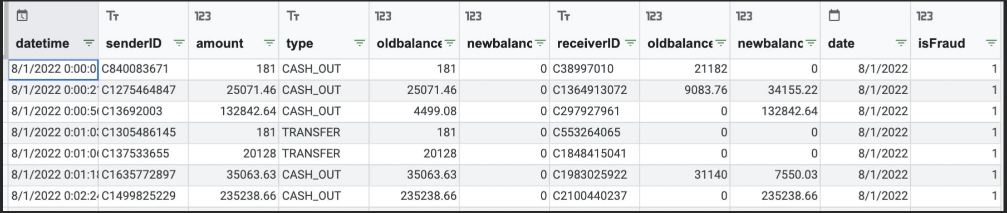
TRANSFORMATION DATA

Fraud Data Only



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

WITH fraud_data_customer AS (
    SELECT
        *
    FROM
        {{ ref('fraud_fix') }}
)
SELECT
    datetime,
    senderID,
    amount,
    type,
    oldbalancesend,
    newbalancesend,
    receiverID,
    oldbalancereceive,
    newbalancereceive,
    date,
    isFraud
FROM
    fraud_data_customer
WHERE
    isFraud = 1
```

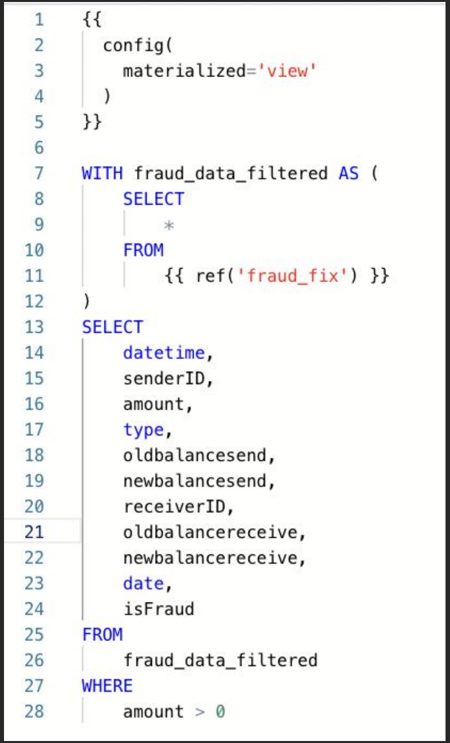
Table Result



datetime	senderID	amount	type	oldbalance	newbalanc	receiverID	oldbalance	newbalanc	date	isFraud
8/1/2022 0:00:0	C840083671	181	CASH_OUT	181	0	C38997010	21182	0	8/1/2022	1
8/1/2022 0:00:2	C1275464847	25071.46	CASH_OUT	25071.46	0	C1364913072	9083.76	34155.22	8/1/2022	1
8/1/2022 0:00:5	C13692003	132842.64	CASH_OUT	4499.08	0	C297927961	0	132842.64	8/1/2022	1
8/1/2022 0:01:0	C1305486145	181	TRANSFER	181	0	C553264065	0	0	8/1/2022	1
8/1/2022 0:01:0	C137533655	20128	TRANSFER	20128	0	C1848415041	0	0	8/1/2022	1
8/1/2022 0:01:1	C1635772897	35063.63	CASH_OUT	35063.63	0	C1983025922	31140	7550.03	8/1/2022	1
8/1/2022 0:02:2	C1499825229	235238.66	CASH_OUT	235238.66	0	C2100440237	0	235238.66	8/1/2022	1

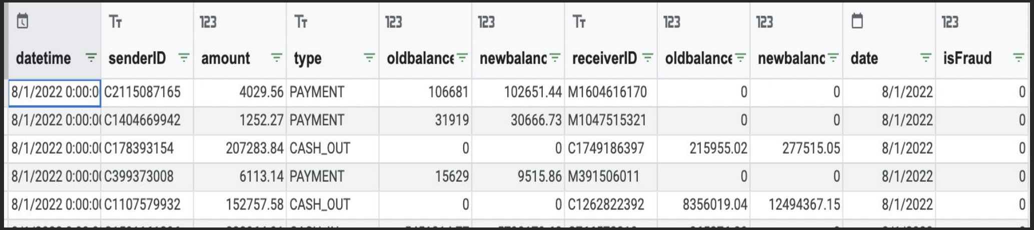
TRANSFORMATION DATA

Data Fraud Filtered



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

WITH fraud_data_filtered AS (
    SELECT
        *
    FROM
        {{ ref('fraud_fix') }}
)
SELECT
    datetime,
    senderID,
    amount,
    type,
    oldbalancesend,
    newbalancesend,
    receiverID,
    oldbalancereceive,
    newbalancereceive,
    date,
    isFraud
FROM
    fraud_data_filtered
WHERE
    amount > 0
```

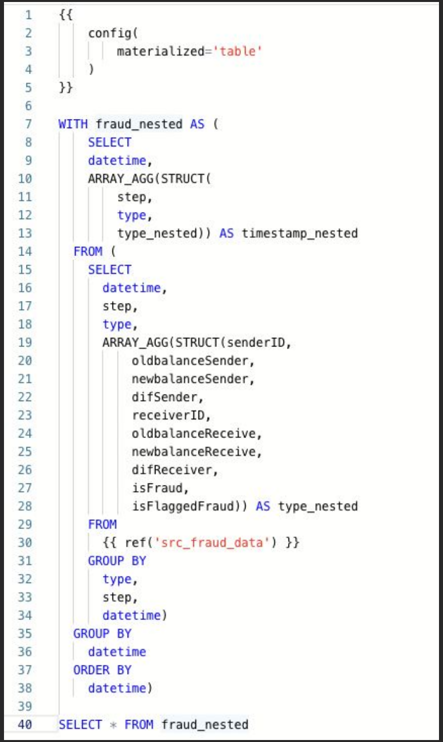
Table Result



datetime	senderID	amount	type	oldbalance	newbalanc	receiverID	oldbalance	newbalanc	date	isFraud
8/1/2022 0:00:0	C2115087165	4029.56	PAYMENT	106681	102651.44	M1604616170	0	0	8/1/2022	0
8/1/2022 0:00:0	C1404669942	1252.27	PAYMENT	31919	30666.73	M1047515321	0	0	8/1/2022	0
8/1/2022 0:00:0	C178393154	207283.84	CASH_OUT	0	0	C1749186397	215955.02	277515.05	8/1/2022	0
8/1/2022 0:00:0	C399373008	6113.14	PAYMENT	15629	9515.86	M391506011	0	0	8/1/2022	0
8/1/2022 0:00:0	C1107579932	152757.58	CASH_OUT	0	0	C1262822392	8356019.04	12494367.15	8/1/2022	0

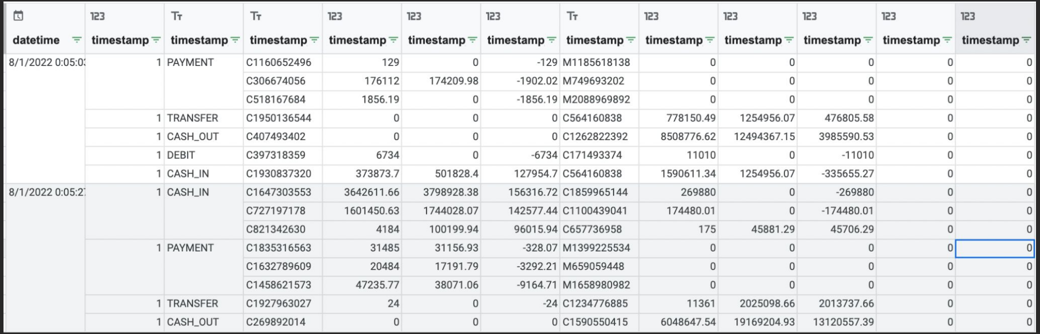
TRANSFORMATION DATA

Data Fraud Nested



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

WITH fraud_nested AS (
    SELECT
    datetime,
    ARRAY_AGG(STRUCT(
        step,
        type,
        type_nested)) AS timestamp_nested
  FROM (
    SELECT
     datetime,
     step,
     type,
     ARRAY_AGG(STRUCT(senderID,
         oldbalanceSender,
         newbalanceSender,
         difSender,
         receiverID,
         oldbalanceReceive,
         newbalanceReceive,
         difReceiver,
         isFraud,
         isFlaggedFraud)) AS type_nested
    FROM
     {{ ref('src_fraud_data') }}
    GROUP BY
     type,
     step,
     datetime)
  GROUP BY
    datetime
  ORDER BY
    datetime)

SELECT * FROM fraud_nested
```

Table Result



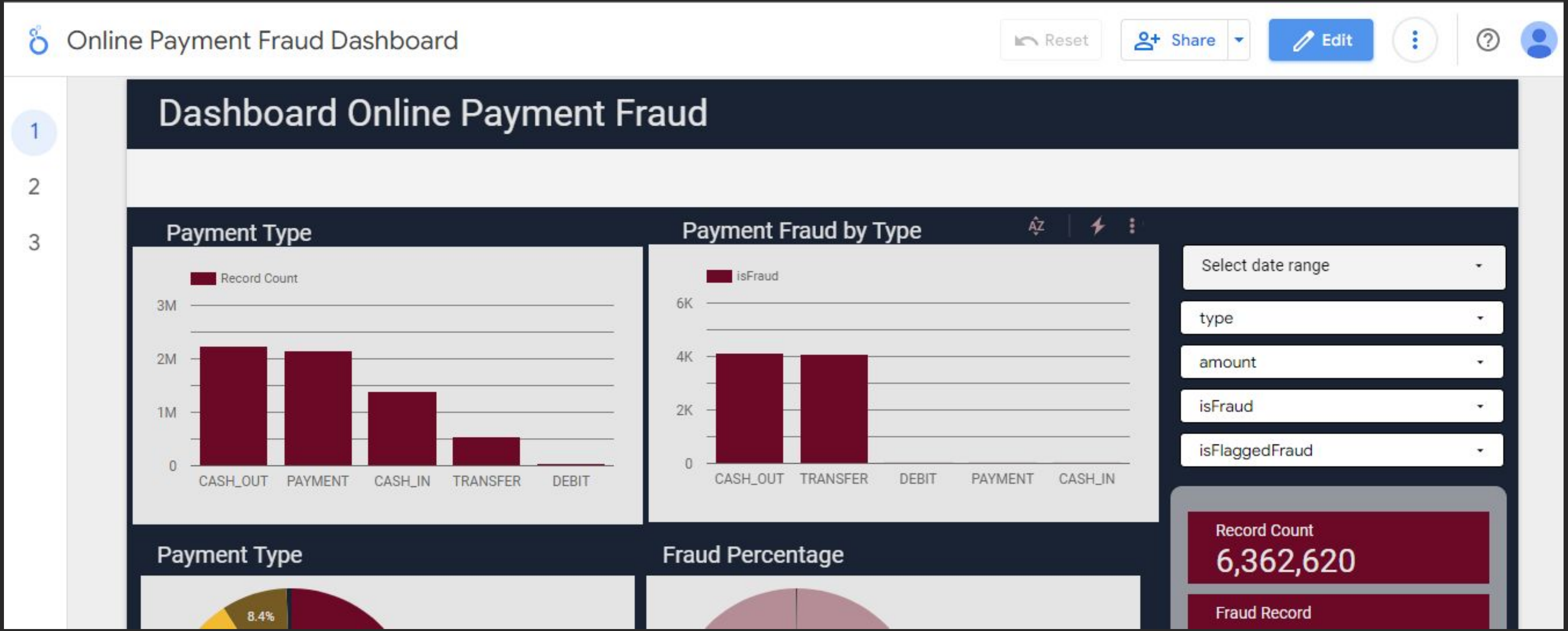
datetime	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp	timestamp
8/1/2022 0:05:0	1	PAYMENT	C1160652496	129	0	-129	M1185618138	0	0	0	0	0
			C306674056	176112	174209.98	-1902.02	M749693202	0	0	0	0	0
			C518167684	1856.19	0	-1856.19	M2088969892	0	0	0	0	0
	1	TRANSFER	C1950136544	0	0	0	C564160838	778150.49	1254956.07	476805.58	0	0
	1	CASH_OUT	C407493402	0	0	0	C1262822392	8508776.62	12494367.15	3985590.53	0	0
	1	DEBIT	C397318359	6734	0	-6734	C171493374	11010	0	-11010	0	0
	1	CASH_IN	C1930837320	373873.7	501828.4	127954.7	C564160838	1590611.34	1254956.07	-335655.27	0	0
8/1/2022 0:05:2	1	CASH_IN	C1647303553	3642611.66	3798928.38	156316.72	C1859965144	269880	0	-269880	0	0
			C727197178	1601450.63	1744028.07	142577.44	C1100439041	174480.01	0	-174480.01	0	0
			C821342630	4184	100199.94	96015.94	C657736958	175	45881.29	45706.29	0	0
	1	PAYMENT	C1835316563	31485	31156.93	-328.07	M1399225534	0	0	0	0	0
			C1632789609	20484	17191.79	-3292.21	M659059448	0	0	0	0	0
			C1458621573	47235.77	38071.06	-9164.71	M1658980982	0	0	0	0	0
	1	TRANSFER	C1927963027	24	0	-24	C1234776885	11361	2025098.66	2013737.66	0	0
	1	CASH_OUT	C269892014	0	0	0	C1590550415	6048647.54	19169204.93	13120557.39	0	0

06

DASHBOARD & CODE

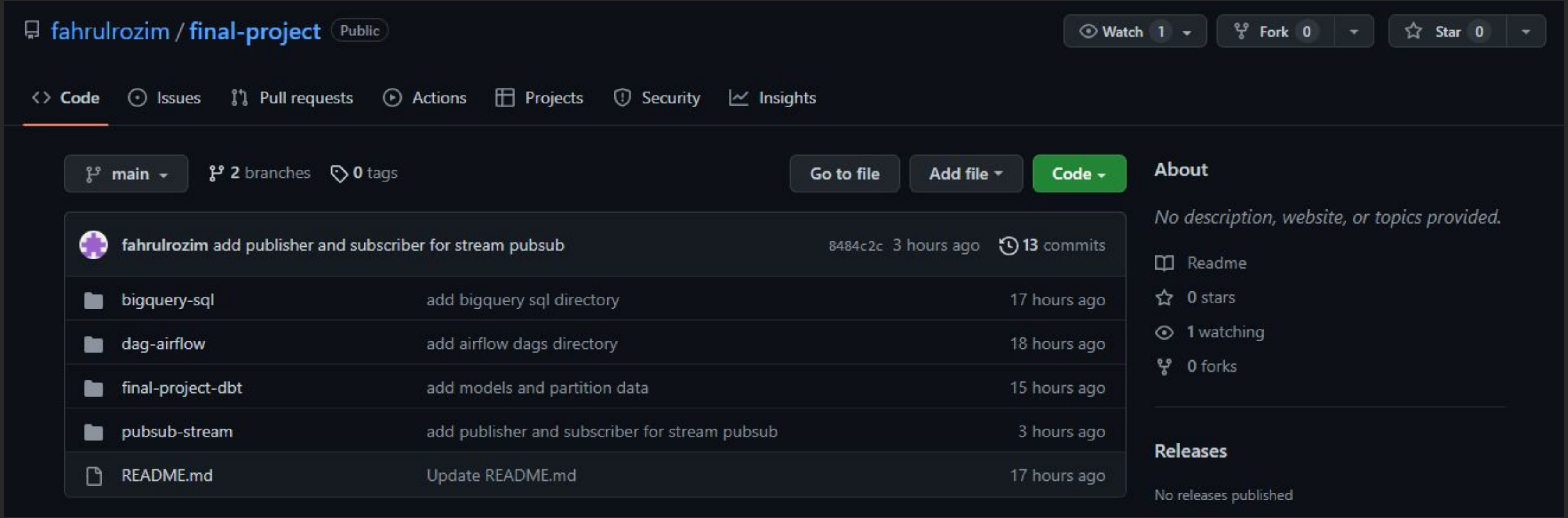
DASHBOARD

You can access dashboard at: **bit.ly/dashboardfraud**



SOURCE CODE

You can access github at: https://github.com/fahrulrozim/final-project



No description, website, or topics provided.

fahrulrozim add publisher and subscriber for stream pubsub — 8484c2c 3 hours ago — 13 commits

bigquery-sql	add bigquery sql directory	17 hours ago
dag-airflow	add airflow dags directory	18 hours ago
final-project-dbt	add models and partition data	15 hours ago
pubsub-stream	add publisher and subscriber for stream pubsub	3 hours ago
README.md	Update README.md	17 hours ago

THANK

YOU

“Data telling the truth, people tell stories and hopes.”

–Someone Famous