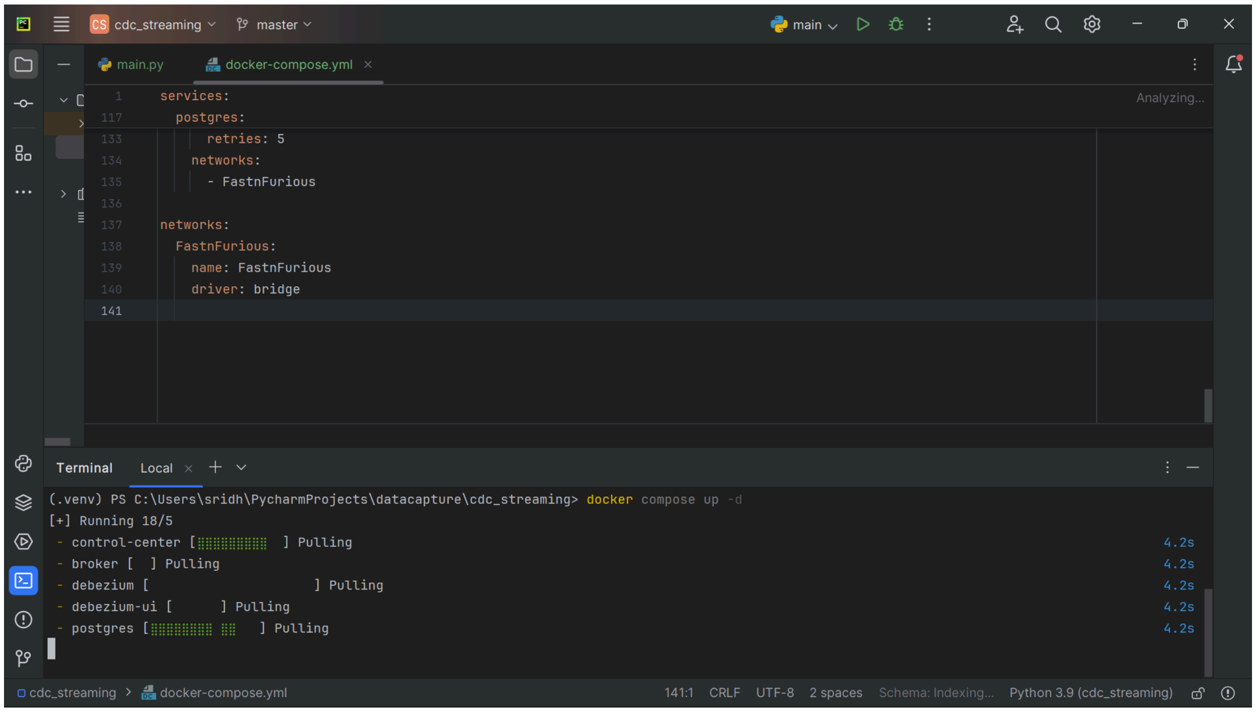
I created a new project in the PyCharm IDE and named it “cdc_streaming”. I created a docker-compose.yml file with the below configuration

```
services:
 zookeeper:
   image: confluentinc/cp-zookeeper:7.4.0
   hostname: zookeeper
   container_name: zookeeper
   ports:
     - '2181:2181'
   environment:
     ZOOKEEPER_CLIENT_PORT: 2181
     ZOOKEEPER_TICK_TIME: 2000
   healthcheck:
     test: echo srvr | nc zookeeper 2181 || exit 1
     start_period: 10s
     retries: 20
     interval: 10s
   networks:
     - FastnFurious

 broker:
   image: confluentinc/cp-kafka:7.4.0
   hostname: broker
   container_name: broker
   ports:
     - '29892:29092'
     - '9892:9092'
     - '9101:9101'
   environment:
     KAFKA_BROKER_ID: 1
     KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
     KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,
PLAINTEXT_HOST:PLAINTEXT
     KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:29092,
PLAINTEXT_HOST://localhost:9092
     KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
     KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
     KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
     KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
     KAFKA_AUTO_CREATE_TOPICS_ENABLE: 'true'
     KAFKA_JMX_PORT: 9101
     KAFKA_JMX_HOSTNAME: localhost
   healthcheck:
     test: nc -z localhost 9092 || exit -1
     start_period: 15s
     interval: 5s
     timeout: 10s
     retries: 10
   networks:
    - FastnFurious

control-center:
  image: confluentinc/cp-enterprise-control-center:7.4.0
  hostname: control-center
  container_name: control-center
  depends_on:
    broker:
      condition: service_healthy
  ports:
    - "9021:9021"
  environment:
    CONTROL_CENTER_BOOTSTRAP_SERVERS: 'broker:29092'
    CONTROL_CENTER_REPLICATION_FACTOR: 1
    CONTROL_CENTER_INTERNAL_TOPICS_PARTITIONS: 1
    CONTROL_CENTER_MONITORING_INTERCEPTOR_TOPIC_PARTITIONS: 1
    CONFLUENT_METRICS_TOPIC_REPLICATION: 1
    CONFLUENT_METRICS_ENABLE: 'false'
    PORT: 9021
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:9021/health"]
    interval: 30s
    timeout: 10s
    retries: 5
  networks:
    - FastnFurious

debezium:
  image: debezium/connect:2.1
  restart: always
  container_name: debezium
  hostname: debezium
  depends_on:
    postgres:
      condition: service_healthy
    broker:
      condition: service_healthy
  ports:
    - '8093:8083'
  environment:
    BOOTSTRAP_SERVERS: broker:29092
    CONNECT_REST_ADVERTISED_HOST_NAME: debezium
    GROUP_ID: 1
    CONFIG_STORAGE_TOPIC: connect_configs
    STATUS_STORAGE_TOPIC: connect_statuses
    OFFSET_STORAGE_TOPIC: connect_offsets
    KEY_CONVERTER: org.apache.kafka.connect.json.JsonConverter
    VALUE_CONVERTER: org.apache.kafka.connect.json.JsonConverter
    ENABLE_DEBEZIUM_SCRIPTING: 'true'
```

```
   healthcheck:
     test: ["CMD", "curl", "--silent", "--fail", "-X", "GET",
"http://localhost:8083/connectors"]
     start_period: 10s
     interval: 10s
     timeout: 5s
     retries: 5
   networks:
     - FastnFurious
 debezium-ui:
   image: debezium/debezium-ui:latest
   restart: always
   container_name: debezium-ui
   hostname: debezium-ui
   depends_on:
     debezium:
       condition: service_healthy
   ports:
     - '8080:8080'
   environment:
     KAFKA_CONNECT_URIS: http://debezium:8083
   networks:
     FastnFurious:
 postgres:
   image: postgres:latest
   restart: always
   container_name: postgres
   hostname: postgres
   ports:
     - '5432:5432'
   environment:
     POSTGRES_USER: postgres
     POSTGRES_PASSWORD: postgres
     POSTGRES_DB: financial_db
   command: ['postgres', '-c', 'wal_level=logical']
   healthcheck:
     test: ['CMD', 'psql', '-U', 'postgres', '-c', 'SELECT 1']
     interval: 10s
     timeout: 5s
     retries: 5
   networks:
     - FastnFurious

networks:
 FastnFurious:
   name: FastnFurious
   driver: bridge
```

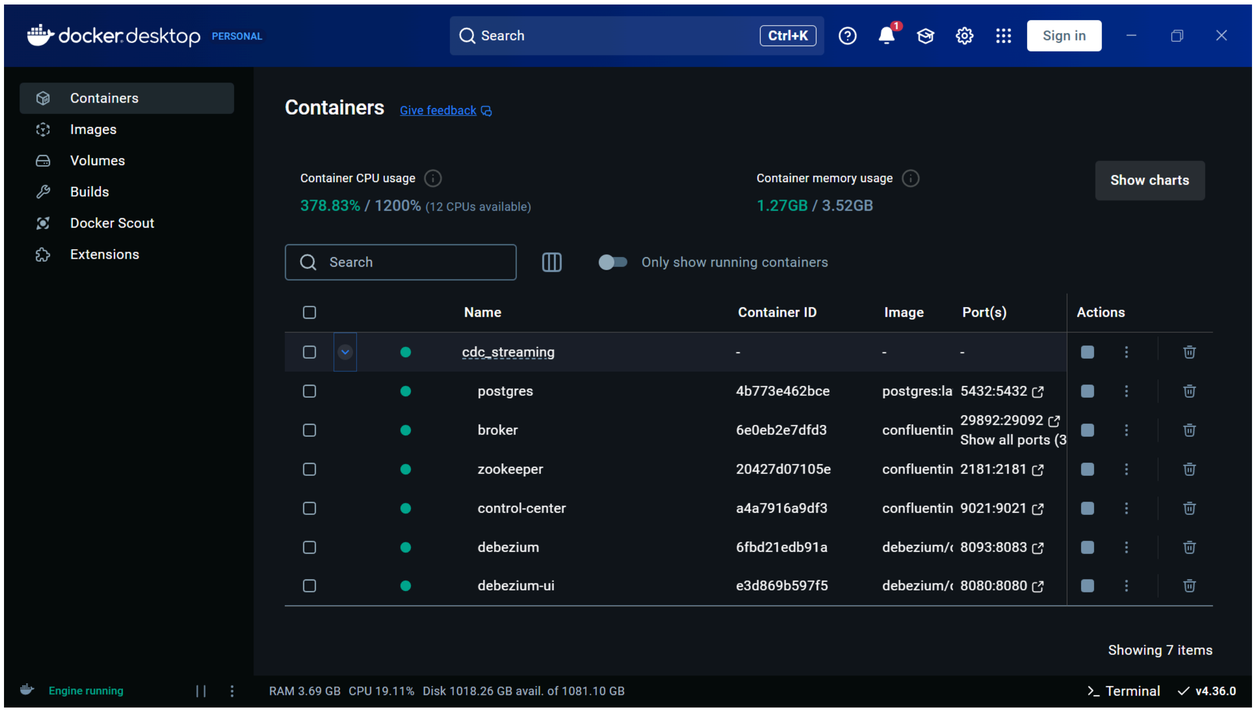
Once I have written all the dependencies, I started the docker containers by entering the “docker compose up -d” in the terminal


cdc_streaming
master
main
main.py
docker-compose.yml
Analyzing...
Terminal
Local

```
services:
  postgres:
      retries: 5
    networks:
      - FastnFurious

networks:
  FastnFurious:
    name: FastnFurious
    driver: bridge
```

```
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming> docker compose up -d
[+] Running 18/5
 - control-center [⣿⣿⣿⣿⣿⣿⣿⣿⣿] Pulling                              4.2s
 - broker [  ] Pulling                                               4.2s
 - debezium [                  ] Pulling                             4.2s
 - debezium-ui [      ] Pulling                                      4.2s
 - postgres [⣿⣿⣿⣿⣿⣿⣿⣿ ⣿⣿   ] Pulling                                 4.2s
```

cdc_streaming > docker-compose.yml
141:1 CRLF UTF-8 2 spaces Schema: Indexing... Python 3.9 (cdc_streaming)



Containers

Container CPU usage
378.83% / 1200% (12 CPUs available)

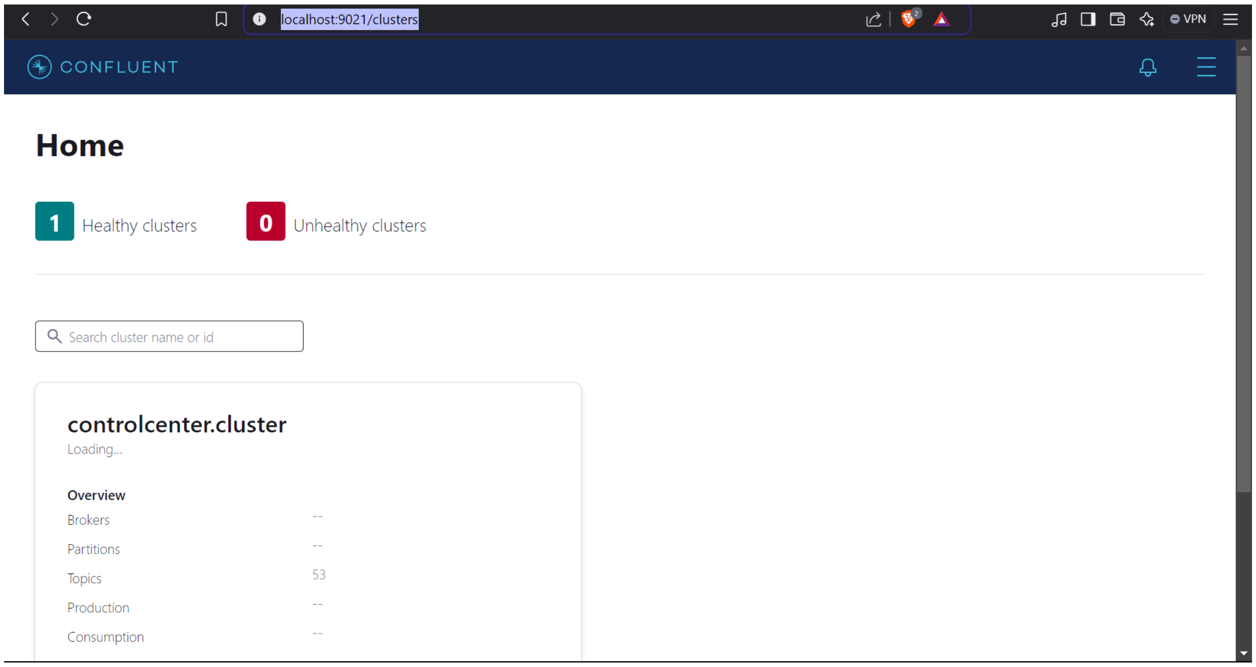
Container memory usage
1.27GB / 3.52GB

Name	Container ID	Image	Port(s)	Actions
cdc_streaming	-	-	-	
postgres	4b773e462bce	postgres:la	5432:5432	
broker	6e0eb2e7dfd3	confluentin	29892:29092 Show all ports (3	
zookeeper	20427d07105e	confluentin	2181:2181	
control-center	a4a7916a9df3	confluentin	9021:9021	
debezium	6fbd21edb91a	debezium/	8093:8083	
debezium-ui	e3d869b597f5	debezium/	8080:8080	

Showing 7 items

RAM 3.69 GB CPU 19.11% Disk 1018.26 GB avail. of 1081.10 GB

Debezium helps us to do change data capture from database level. It goes into the database level logs and picks the changes that are happening and ships it to kafka brokers. Debezium-ui helps us to visualize the changes in real time. From the above, I can see that the control center is at port 9021. I can visit the web page http://localhost:9021/clusters and see the control center UI



localhost:9021/clusters

CONFLUENT

Home

1 Healthy clusters

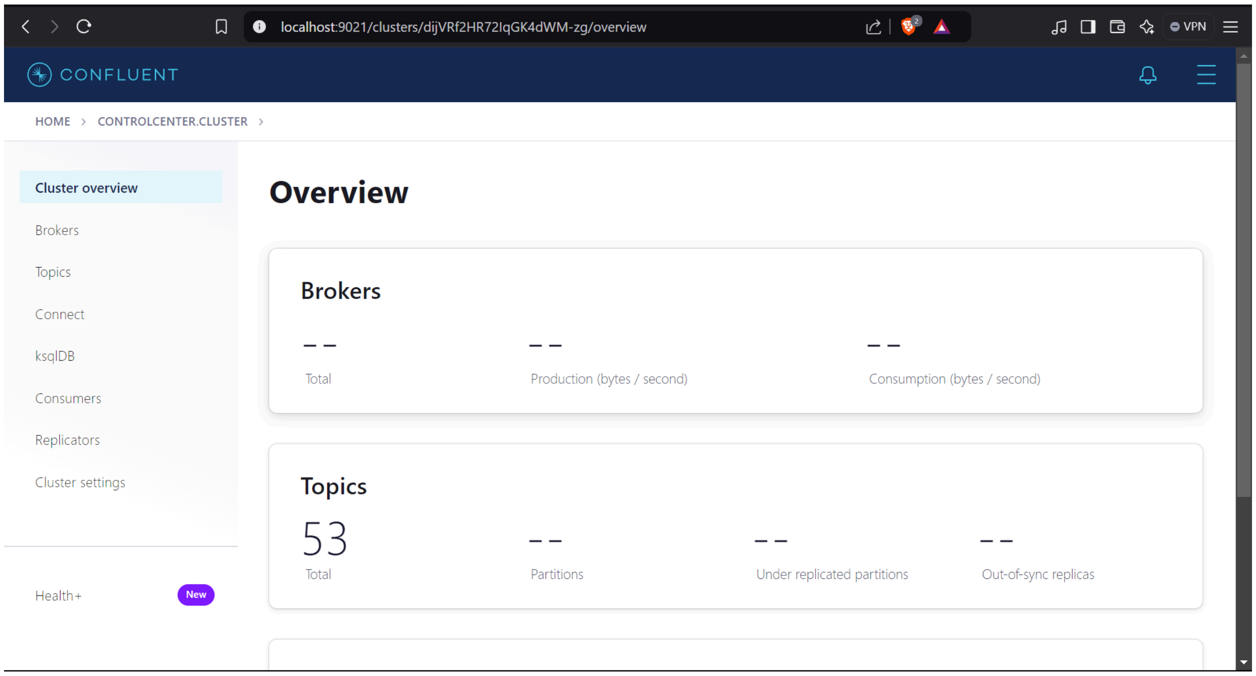
0 Unhealthy clusters

Search cluster name or id

controlcenter.cluster

Loading...

Overview	
Brokers	--
Partitions	--
Topics	53
Production	--
Consumption	--



localhost:9021/clusters/dijVRf2HR72IqGK4dWM-zg/overview

CONFLUENT

HOME › CONTROLCENTER.CLUSTER ›

Overview

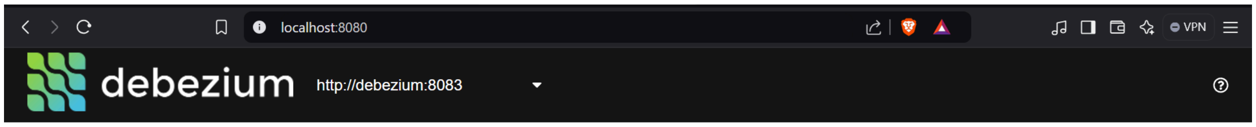
Brokers

Total	Production (bytes / second)	Consumption (bytes / second)
--	--	--

Topics

Total	Partitions	Under replicated partitions	Out-of-sync replicas
53	--	--	--

Similarly , I can see that the debezium UI is running at port 8080


localhost:8080

http://debezium:8083

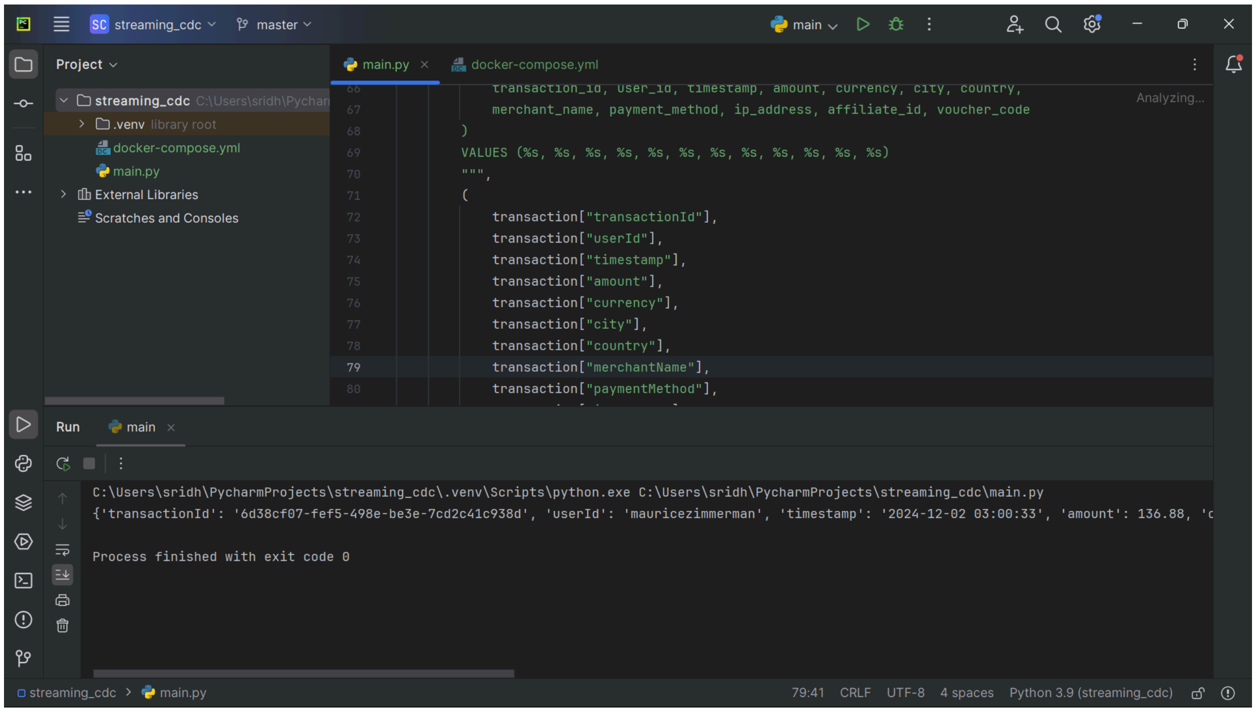
No connectors

Please click 'Create a connector' to create a new connector.

Now, I set up the database and fetched the live data into postgresDB using the below python script

```python
import random
from datetime import datetime
from faker import Faker
import psycopg2

fake = Faker()

def generate_transaction():
   user = fake.simple_profile()
   return {
       "transactionId": fake.uuid4(),
       "userId": user['username'],
       "timestamp": datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S'),
       "amount": round(random.uniform(10, 1000), 2),
       "currency": random.choice(['USD', 'GBP']),
       'city': fake.city(),
       "country": fake.country(),
       "merchantName": fake.company(),
       "paymentMethod": random.choice(['credit_card', 'debit_card',
'online_transfer']),
       "ipAddress": fake.ipv4(),
       "voucherCode": random.choice(['', 'DISCOUNT10', '']),
       'affiliateId': fake.uuid4()
   }

def create_table(conn):
   cursor = conn.cursor()
   cursor.execute(
       """
       CREATE TABLE IF NOT EXISTS transactions(
           transaction_id VARCHAR(255) PRIMARY KEY,
           user_id VARCHAR(255),
           timestamp TIMESTAMP,
           amount DECIMAL,
           currency VARCHAR(255),
           city VARCHAR(255),
           country VARCHAR(255),
           merchant_name VARCHAR(255),
           payment_method VARCHAR(255),
           ip_address VARCHAR(255),
           voucher_code VARCHAR(255),
           affiliate_id VARCHAR(255)
       )
       """
   )
   cursor.close()
   conn.commit()

if __name__ == "__main__":
   conn = psycopg2.connect(
       host='localhost',
       database='financial_db',
       user='postgres',
       password='postgres',
       port=5432
   )

   create_table(conn)

   transaction = generate_transaction()
   cur = conn.cursor()
   print(transaction)

   cur.execute(
       """
       INSERT INTO transactions(
           transaction_id, user_id, timestamp, amount, currency, city, country,
           merchant_name, payment_method, ip_address, affiliate_id,
voucher_code
       )
       VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
       """,
       (
           transaction["transactionId"],
           transaction["userId"],
           transaction["timestamp"],
           transaction["amount"],
           transaction["currency"],
           transaction["city"],
           transaction["country"],
           transaction["merchantName"],
           transaction["paymentMethod"],
           transaction["ipAddress"],
           transaction["affiliateId"],
           transaction["voucherCode"]
       )
   )
   cur.close()
   conn.commit()
```

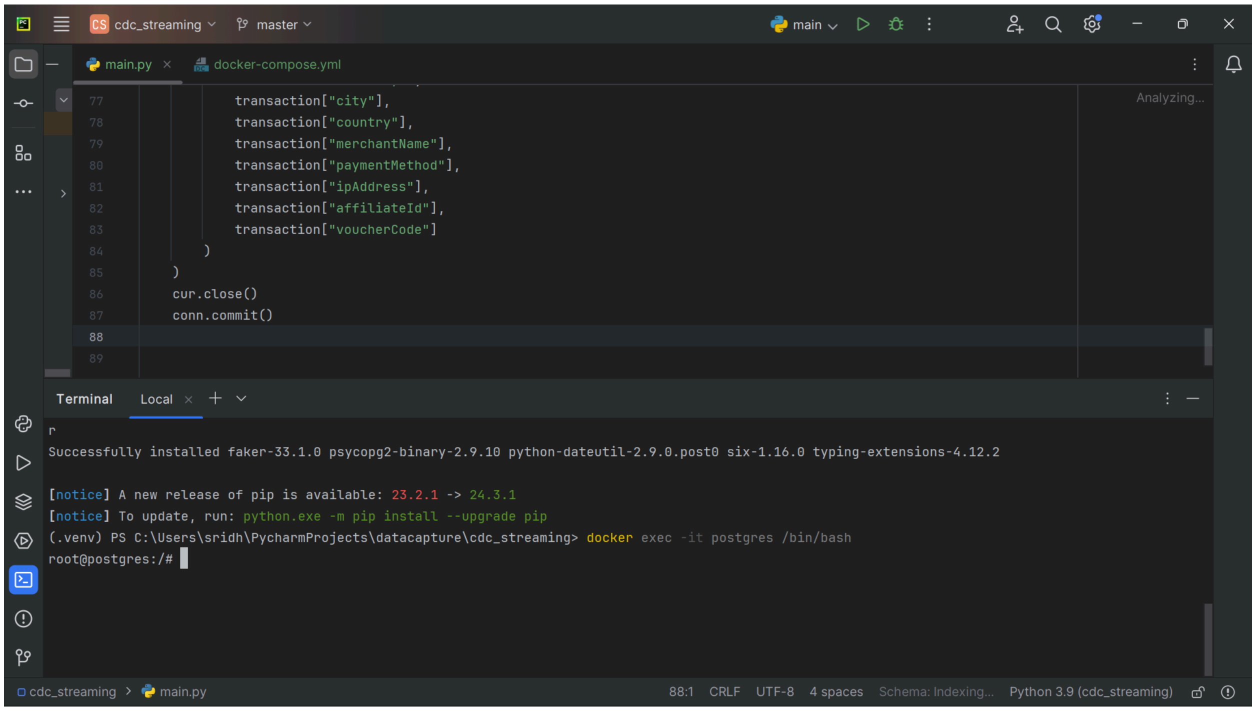
I can see that data is inserted into postgres DB



```
            transaction_id, user_id, timestamp, amount, currency, city, country,
            merchant_name, payment_method, ip_address, affiliate_id, voucher_code
        )
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """,
        (
            transaction["transactionId"],
            transaction["userId"],
            transaction["timestamp"],
            transaction["amount"],
            transaction["currency"],
            transaction["city"],
            transaction["country"],
            transaction["merchantName"],
            transaction["paymentMethod"],
```

```
C:\Users\sridh\PycharmProjects\streaming_cdc\.venv\Scripts\python.exe C:\Users\sridh\PycharmProjects\streaming_cdc\main.py
{'transactionId': '6d38cf07-fef5-498e-be3e-7cd2c41c938d', 'userId': 'mauricezimmerman', 'timestamp': '2024-12-02 03:00:33', 'amount': 136.88, 'c

Process finished with exit code 0
```

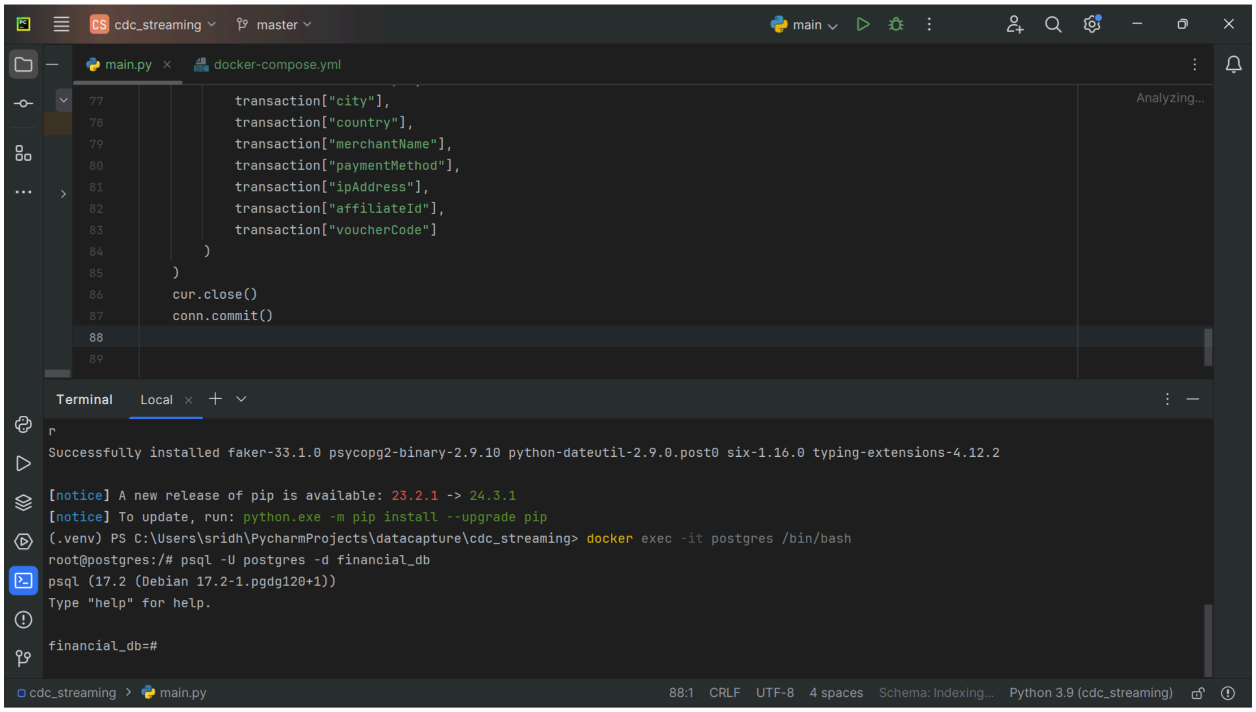
Now, I executed “docker exec -it postgres /bin/bash”



```
                transaction["city"],
                transaction["country"],
                transaction["merchantName"],
                transaction["paymentMethod"],
                transaction["ipAddress"],
                transaction["affiliateId"],
                transaction["voucherCode"]
            )
        )
        cur.close()
        conn.commit()
```

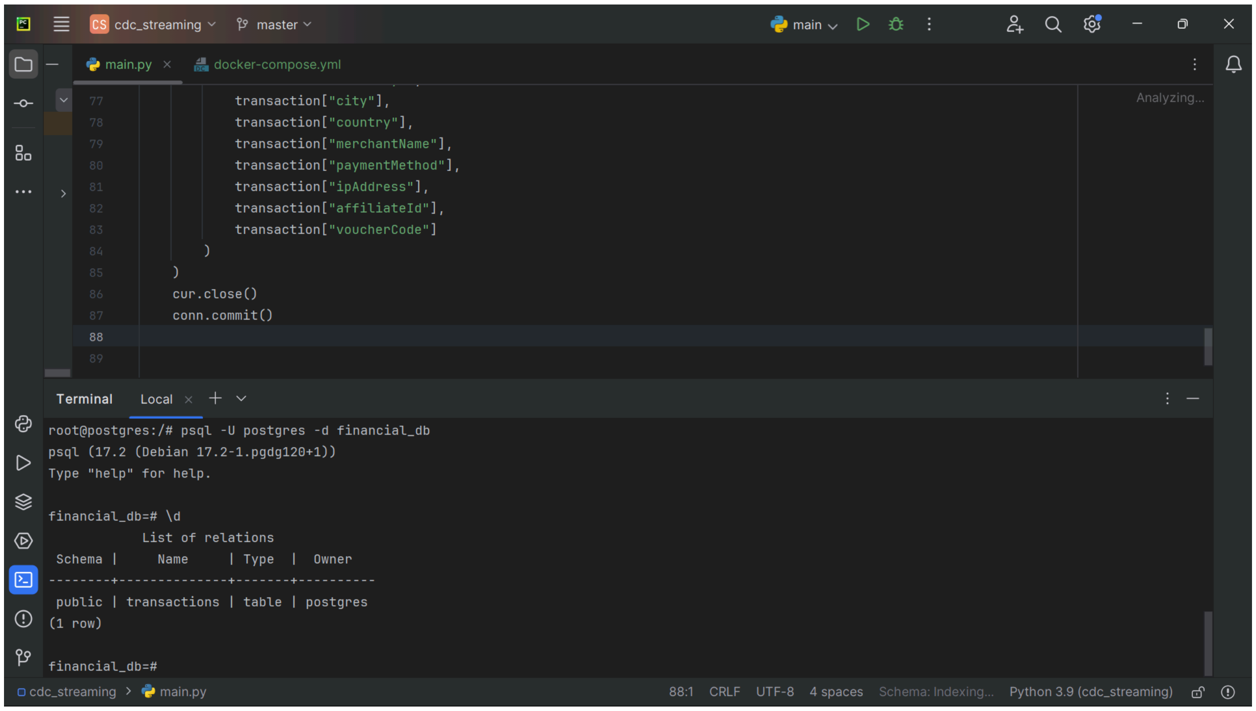
```
r
Successfully installed faker-33.1.0 psycopg2-binary-2.9.10 python-dateutil-2.9.0.post0 six-1.16.0 typing-extensions-4.12.2

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming> docker exec -it postgres /bin/bash
root@postgres:/#
```

To connect to the psql db with the user “postgres” and the database name is financial_db



```
                transaction["city"],
                transaction["country"],
                transaction["merchantName"],
                transaction["paymentMethod"],
                transaction["ipAddress"],
                transaction["affiliateId"],
                transaction["voucherCode"]
            )
        )
        cur.close()
        conn.commit()
```

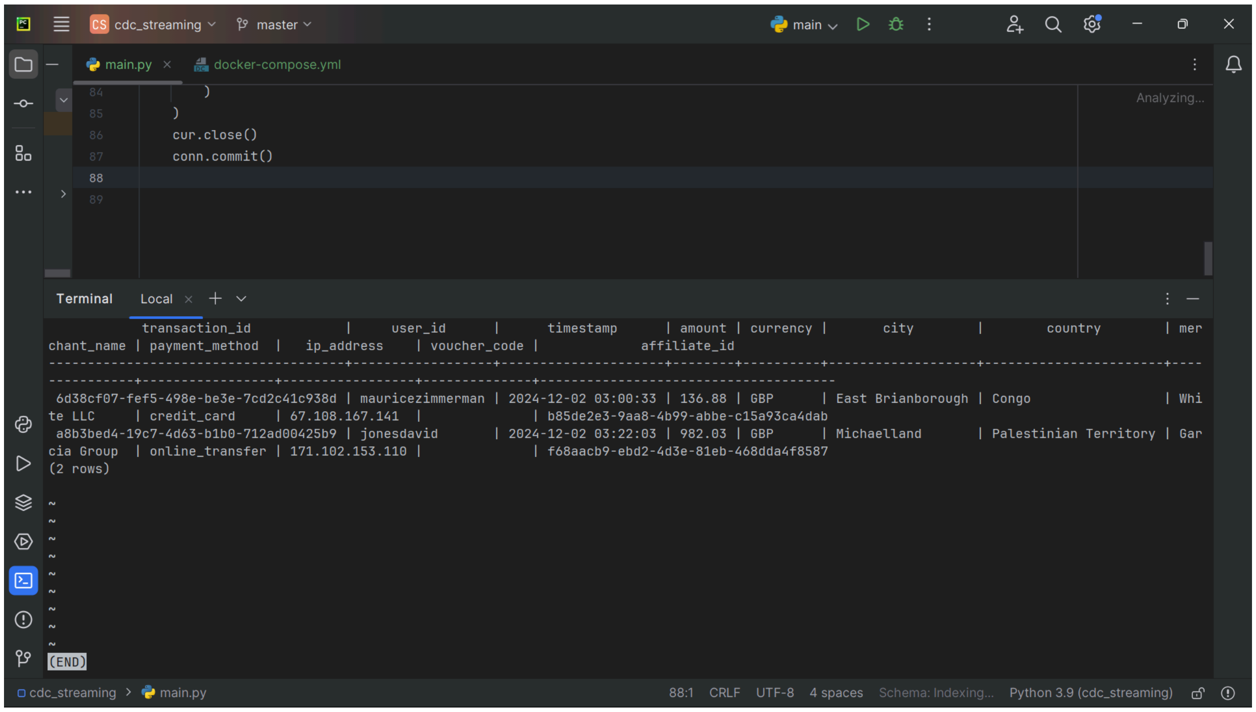
```
r
Successfully installed faker-33.1.0 psycopg2-binary-2.9.10 python-dateutil-2.9.0.post0 six-1.16.0 typing-extensions-4.12.2

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming> docker exec -it postgres /bin/bash
root@postgres:/# psql -U postgres -d financial_db
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

financial_db=#
```



```
                transaction["city"],
                transaction["country"],
                transaction["merchantName"],
                transaction["paymentMethod"],
                transaction["ipAddress"],
                transaction["affiliateId"],
                transaction["voucherCode"]
            )
        )
        cur.close()
        conn.commit()
```

```
root@postgres:/# psql -U postgres -d financial_db
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

financial_db=# \d
            List of relations
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | transactions | table | postgres
(1 row)

financial_db=#
```

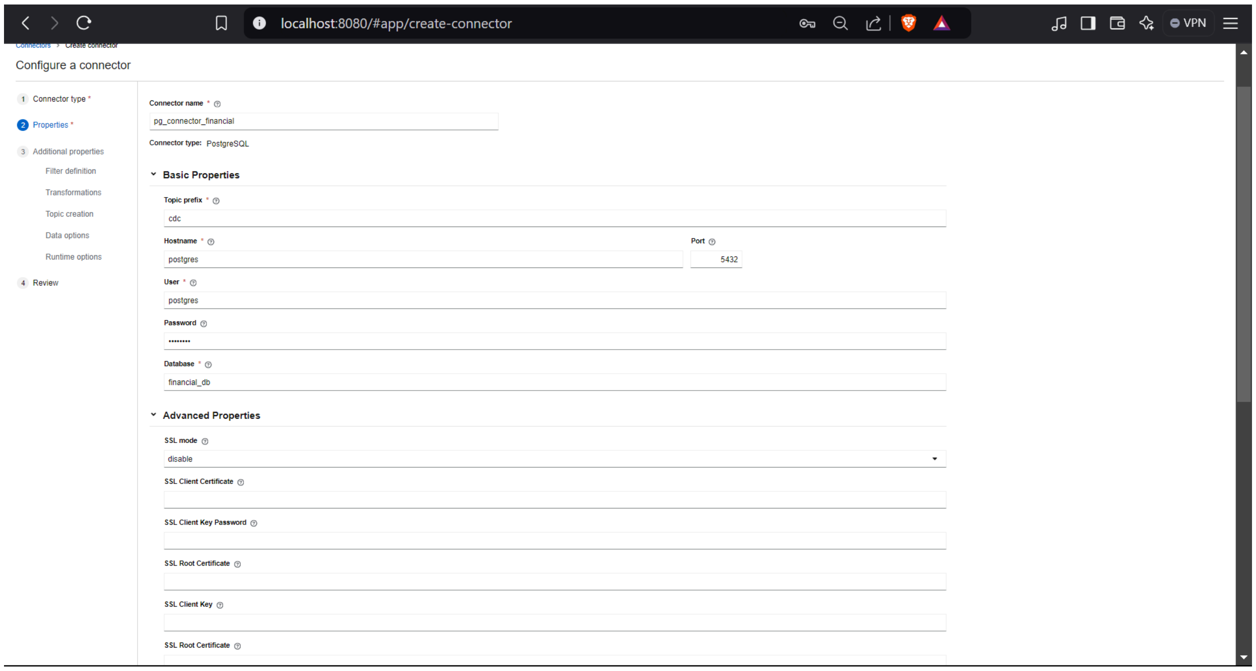
To check the data, SELECT * FROM transactions;



```
            )
        )
        cur.close()
        conn.commit()
```

```
            transaction_id             |     user_id      |      timestamp      | amount | currency |        city        |        country         | mer
chant_name | payment_method |   ip_address    | voucher_code |             affiliate_id
--------------------------------------+------------------+---------------------+--------+----------+--------------------+------------------------+----
-----------+-----------------+-----------------+--------------+--------------------------------------
 6d38cf07-fef5-498e-be3e-7cd2c41c938d | mauricezimmerman | 2024-12-02 03:00:33 | 136.88 | GBP      | East Brianborough | Congo                  | Whi
te LLC     | credit_card     | 67.108.167.141  |              | b85de2e3-9aa8-4b99-abbe-c15a93ca4dab
 a8b3bed4-19c7-4d63-b1b0-712ad00425b9 | jonesdavid       | 2024-12-02 03:22:03 | 982.03 | GBP      | Michaelland        | Palestinian Territory | Gar
cia Group  | online_transfer | 171.102.153.110 |              | f68aacb9-ebd2-4d3e-81eb-468dda4f8587
(2 rows)

~
~
~
~
~
~
~
~
~
(END)
```

Now, I created a connector for postgreSQL database in debezium with the following configuration


localhost:8080/#app/create-connector

Connectors > Create connector

Configure a connector

1. Connector type *
2. Properties *
3. Additional properties
   - Filter definition
   - Transformations
   - Topic creation
   - Data options
   - Runtime options
4. Review

Connector name *: pg_connector_financial

Connector type: PostgreSQL

Basic Properties

Topic prefix *: cdc

Hostname *: postgres

Port: 5432

User *: postgres

Password: ••••••••

Database *: financial_db

Advanced Properties

SSL mode: disable

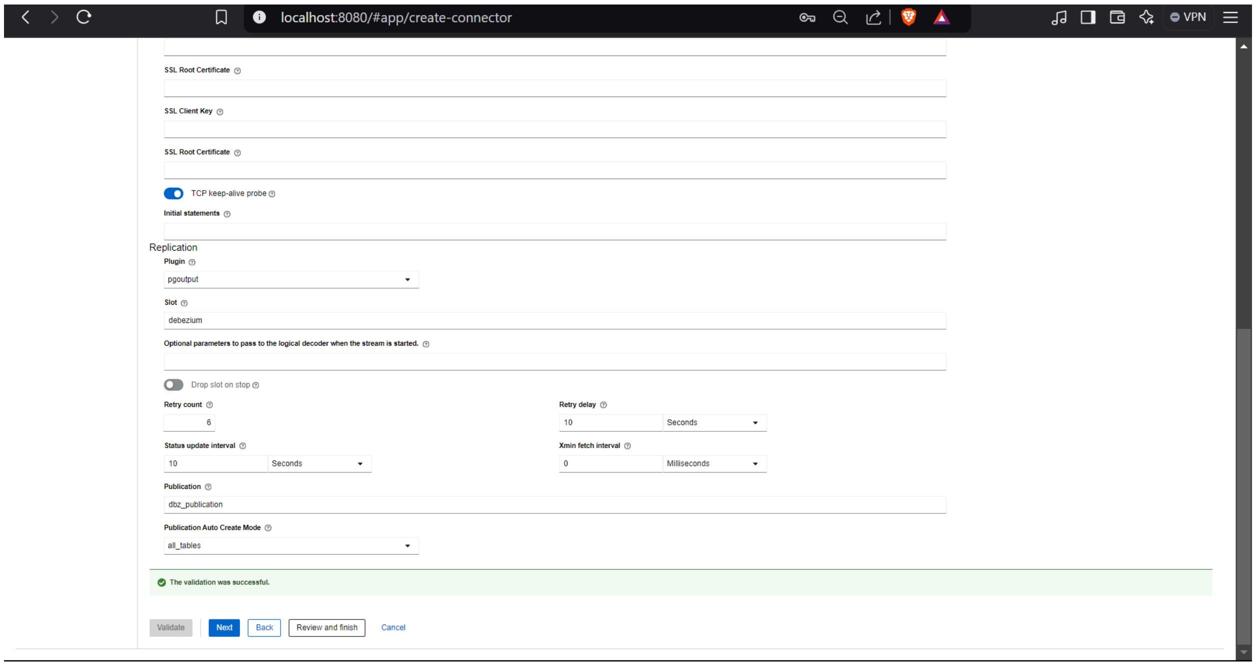
SSL Client Certificate

SSL Client Key Password

SSL Root Certificate

SSL Client Key

SSL Root Certificate


localhost:8080/#app/create-connector

SSL Root Certificate

SSL Client Key

SSL Root Certificate

TCP keep-alive probe

Initial statements

Replication

Plugin: pgoutput

Slot: debezium

Optional parameters to pass to the logical decoder when the stream is started.

Drop slot on stop

Retry count: 6

Retry delay: 10 Seconds

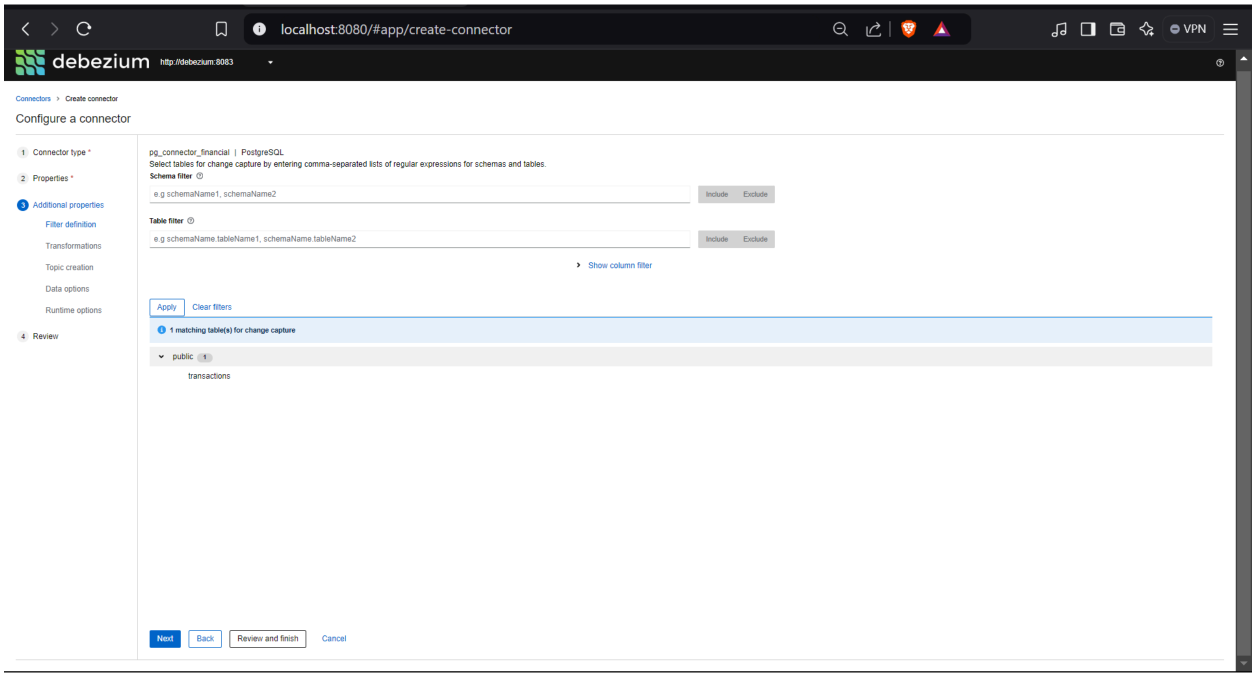
Status update interval: 10 Seconds

Xmin fetch interval: 0 Milliseconds

Publication: dbz_publication

Publication Auto Create Mode: all_tables

The validation was successful.


localhost:8080/#app/create-connector
debezium
http://debezium:8083
Connectors > Create connector

Configure a connector

pg_connector_financial | PostgreSQL

Select tables for change capture by entering comma-separated lists of regular expressions for schemas and tables.

Schema filter: e.g schemaName1, schemaName2

Table filter: e.g schemaName.tableName1, schemaName.tableName2

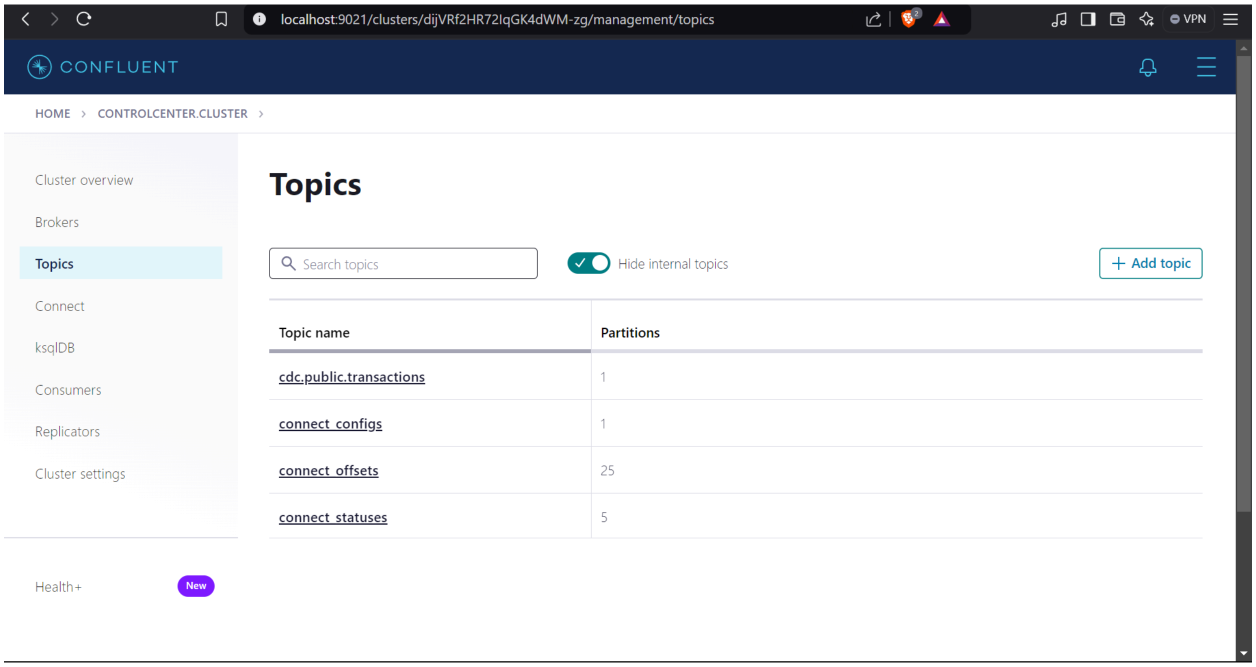
1 matching table(s) for change capture

public 1
transactions

I used the below JSON to create the connector

```
{
  "topic.prefix": "cdc",
  "database.hostname": "postgres",
  "database.user": "postgres",
  "database.password": "********",
  "database.dbname": "financial_db",
  "plugin.name": "pgoutput"
}
```

At this point, I am able to replicate the postgreSQL database in Debezium, the next step is to replicate the debezium data in kafka broker. I can see one topic as shown below

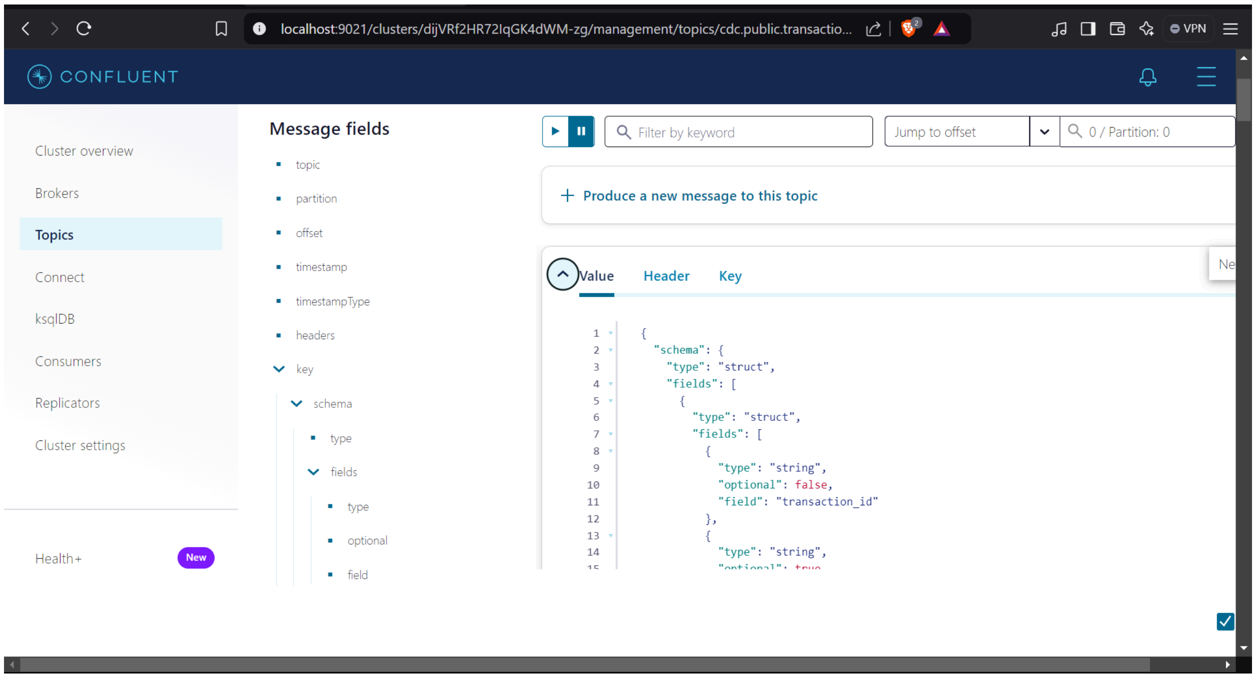

localhost:9021/clusters/dijVRf2HR72lqGK4dWM-zg/management/topics

CONFLUENT

HOME › CONTROLCENTER.CLUSTER ›

Topics

Topic name	Partitions
cdc.public.transactions	1
connect_configs	1
connect_offsets	25
connect_statuses	5


localhost:9021/clusters/dijVRf2HR72lqGK4dWM-zg/management/topics/cdc.public.transactio...

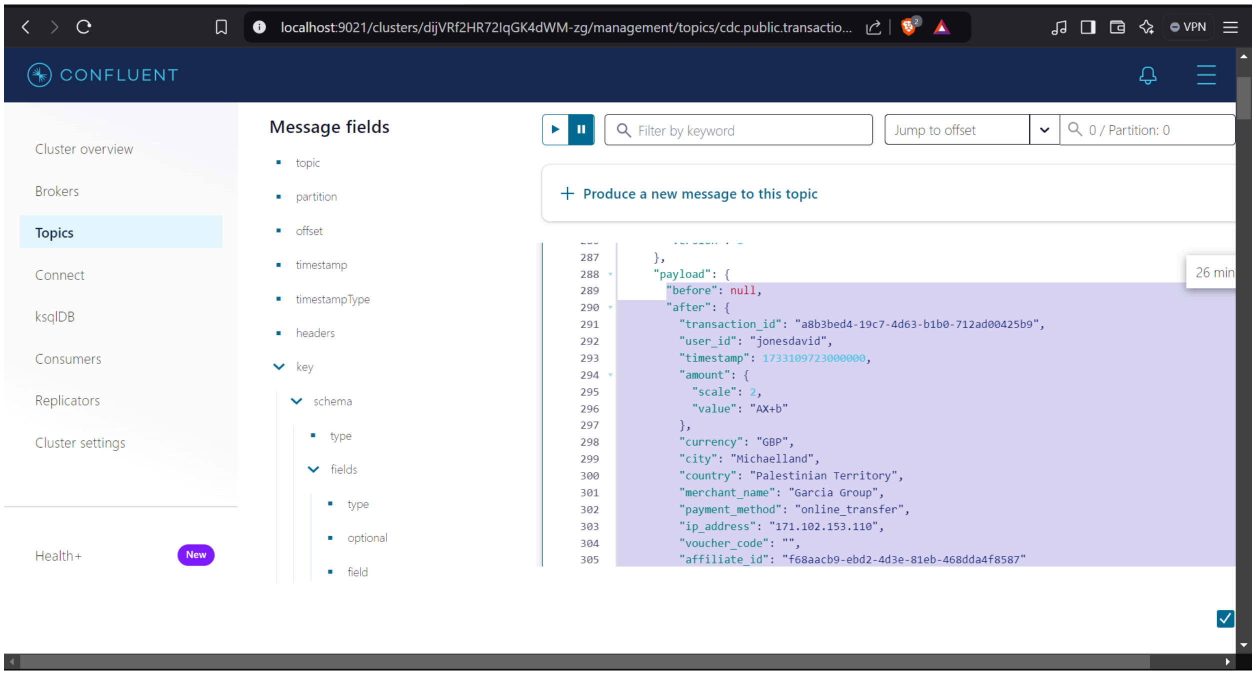
CONFLUENT

Message fields

- topic
- partition
- offset
- timestamp
- timestampType
- headers
- key
  - schema
    - type
    - fields
      - type
      - optional
      - field

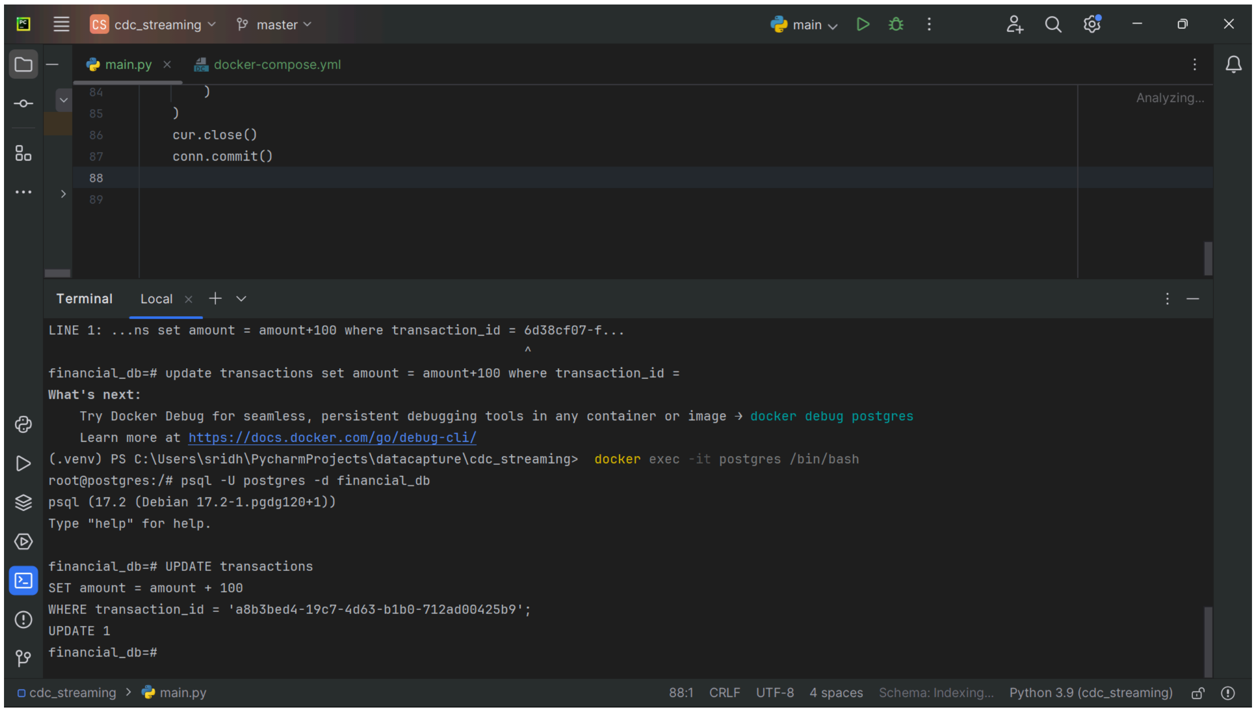
```
{
  "schema": {
    "type": "struct",
    "fields": [
      {
        "type": "struct",
        "fields": [
          {
            "type": "string",
            "optional": false,
            "field": "transaction_id"
          },
          {
            "type": "string",
```

Here, I can see that the value for “before” is null, and “after” has some value as below. Also, the “amount” is not represented properly.



```
    },
    "payload": {
      "before": null,
      "after": {
        "transaction_id": "a8b3bed4-19c7-4d63-b1b0-712ad00425b9",
        "user_id": "jonesdavid",
        "timestamp": 1733109723000000,
        "amount": {
          "scale": 2,
          "value": "AX+b"
        },
        "currency": "GBP",
        "city": "Michaelland",
        "country": "Palestinian Territory",
        "merchant_name": "Garcia Group",
        "payment_method": "online_transfer",
        "ip_address": "171.102.153.110",
        "voucher_code": "",
        "affiliate_id": "f68aacb9-ebd2-4d3e-81eb-468dda4f8587"
```

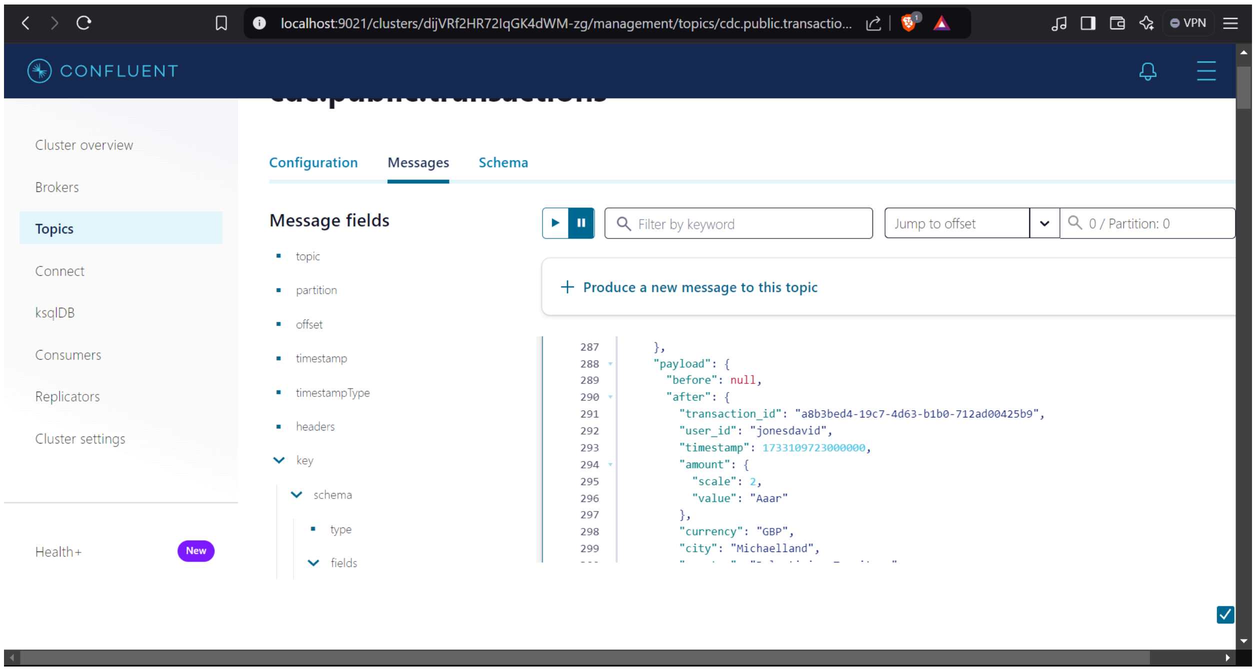
Now, I updated the "amount" as
SET amount = amount + 100
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';



```
        )
    )
    cur.close()
    conn.commit()
```

```
LINE 1: ...ns set amount = amount+100 where transaction_id = 6d38cf07-f...
                                                             ^
financial_db=# update transactions set amount = amount+100 where transaction_id =
What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug postgres
    Learn more at https://docs.docker.com/go/debug-cli/
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming>  docker exec -it postgres /bin/bash
root@postgres:/# psql -U postgres -d financial_db
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

financial_db=# UPDATE transactions
SET amount = amount + 100
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';
UPDATE 1
financial_db=#
```

I can still see the "null" value in "before"



```
    },
    "payload": {
      "before": null,
      "after": {
        "transaction_id": "a8b3bed4-19c7-4d63-b1b0-712ad00425b9",
        "user_id": "jonesdavid",
        "timestamp": 1733109723000000,
        "amount": {
          "scale": 2,
          "value": "Aaar"
        },
        "currency": "GBP",
        "city": "Michaelland",
```

I used below SQL command to create a logical replica

ALTER TABLE transactions REPLICA IDENTITY FULL;

It is used in PostgreSQL to configure the **replica identity** for the `transactions` table. This setting determines how the `UPDATE` and `DELETE` operations on the table are logged in the Write-Ahead Log (WAL) for replication purposes.

What It Does:

- **`REPLICA IDENTITY FULL`**:
  - This setting ensures that **all columns** of a row are included in the WAL entry when the row is updated or deleted.
  - This is required for logical replication or Change Data Capture (CDC) scenarios where all columns are needed to identify and replicate changes.

Use Cases:

1. **Logical Replication**:
   - Logical replication streams the changes (inserts, updates, deletes) to a subscriber. For updates and deletes, having the full row helps to uniquely identify and replicate the changes.
2. **Change Data Capture (CDC)**:

    - If you are capturing changes using tools like `Debezium`, `Striim`, or any custom solution, setting `REPLICA IDENTITY FULL` ensures that even tables without a primary key or unique index can still be used in replication.
3. **Tables Without a Primary Key**:
    - If the table does not have a `PRIMARY KEY` or `UNIQUE` index, PostgreSQL cannot use the default identity (`DEFAULT` or `INDEX`); in such cases, `FULL` is necessary.

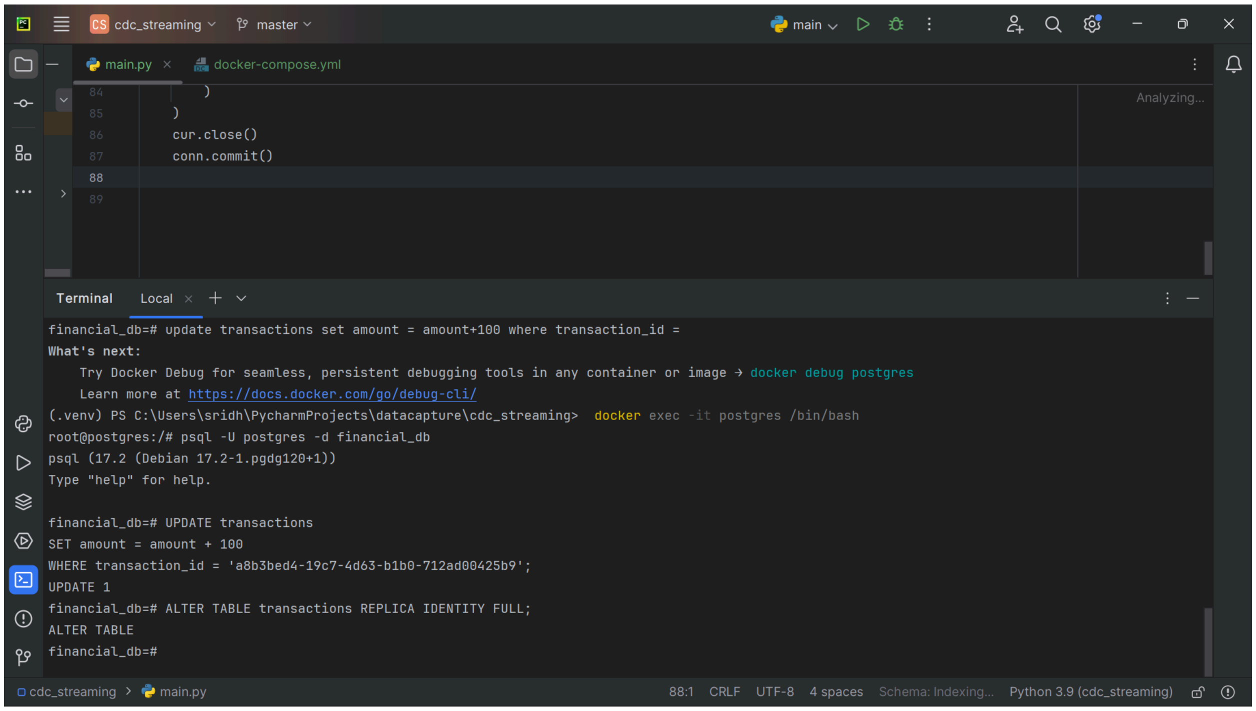
Replica Identity Options:

1. **DEFAULT**:
    - Only logs the primary key column(s) for identifying rows during updates or deletes. This is the default behavior.
2. **INDEX**:
    - Uses a specific unique index (specified by the user) for identifying rows.
3. **FULL**:
    - Logs the entire row's data (both before and after the change).
4. **NOTHING**:
    - Logs no identifying information for updates or deletes. This is generally not used unless replication is disabled.

Example Scenario:

Suppose you are running a Change Data Capture process on the `transactions` table, and it does not have a primary key. In this case:

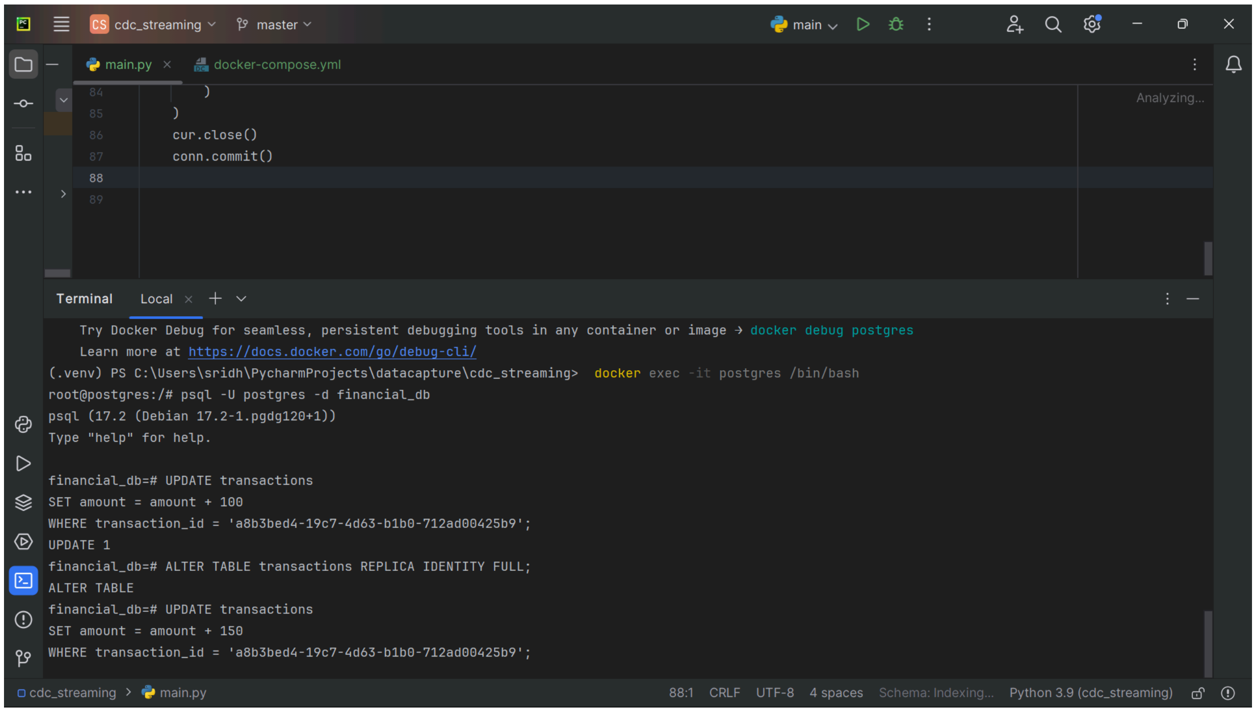
1. Without `REPLICA IDENTITY FULL`, updates or deletes may fail to log enough information to identify the rows.
2. By setting `REPLICA IDENTITY FULL`, you ensure the entire row is included in the WAL, enabling accurate replication.



```
        )
    )
    cur.close()
    conn.commit()
```

```
financial_db=# update transactions set amount = amount+100 where transaction_id =
What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug postgres
    Learn more at https://docs.docker.com/go/debug-cli/
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming>  docker exec -it postgres /bin/bash
root@postgres:/# psql -U postgres -d financial_db
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

financial_db=# UPDATE transactions
SET amount = amount + 100
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';
UPDATE 1
financial_db=# ALTER TABLE transactions REPLICA IDENTITY FULL;
ALTER TABLE
financial_db=#
```

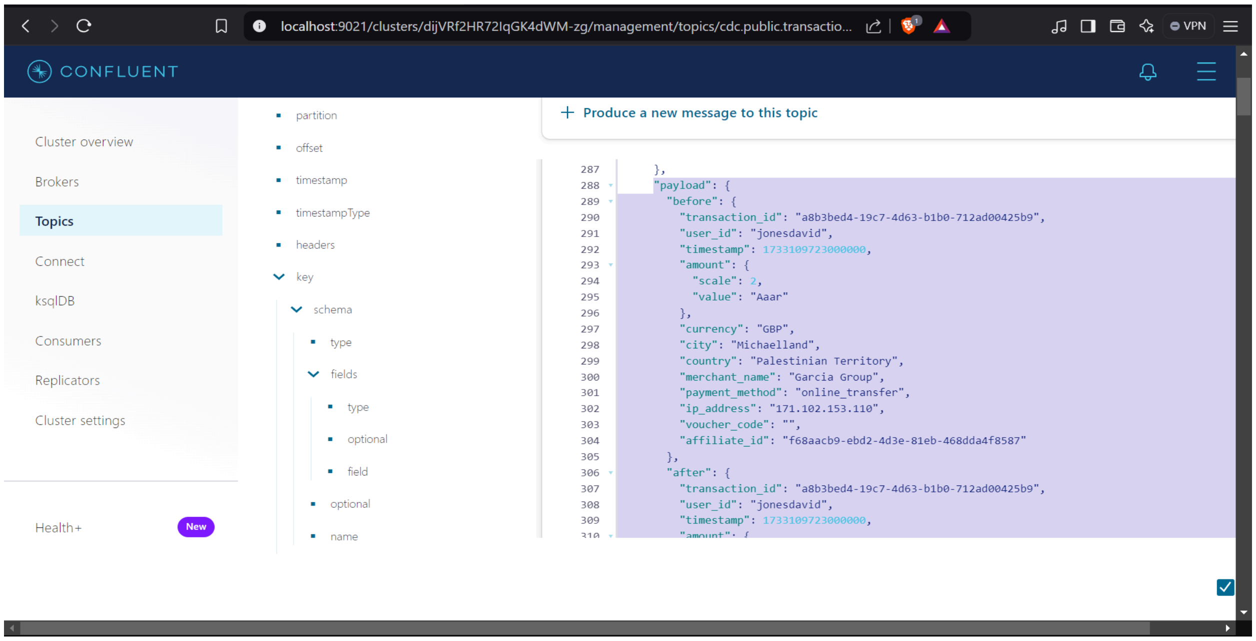
Now, I updated the amount again to add 150 more by using the below SQL command



```
        )
    )
    cur.close()
    conn.commit()
```

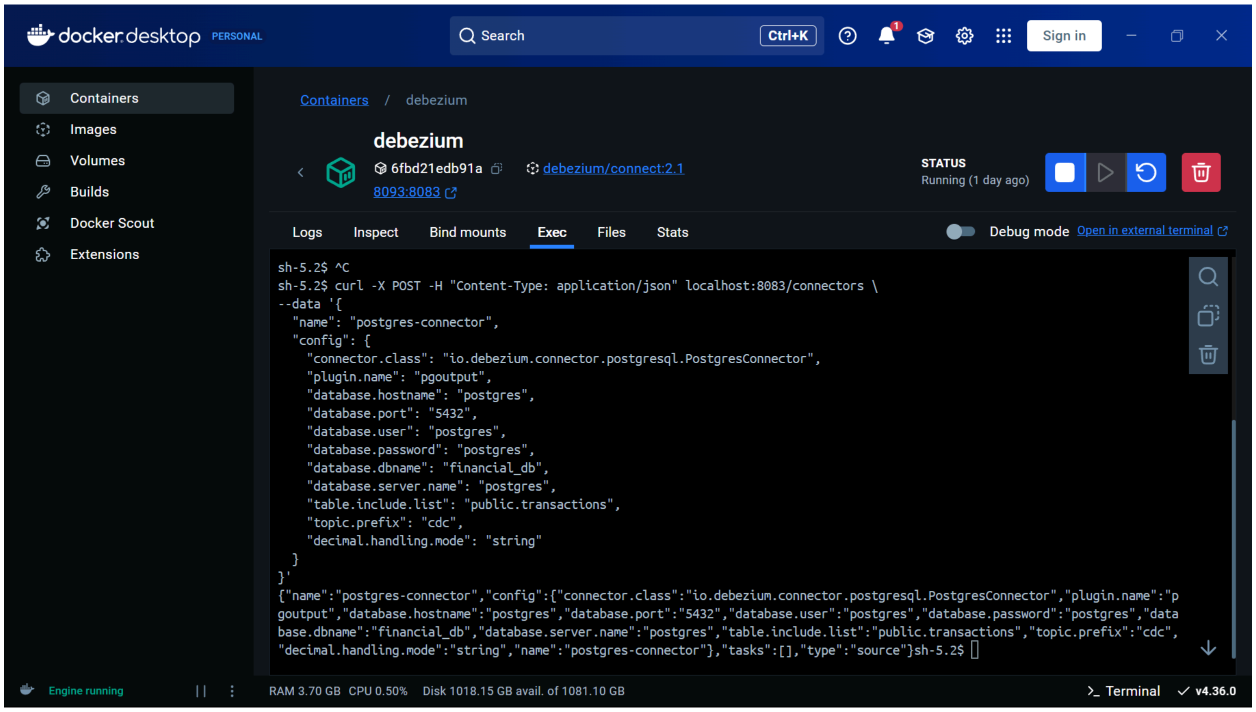
```
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug postgres
    Learn more at https://docs.docker.com/go/debug-cli/
(.venv) PS C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming>  docker exec -it postgres /bin/bash
root@postgres:/# psql -U postgres -d financial_db
psql (17.2 (Debian 17.2-1.pgdg120+1))
Type "help" for help.

financial_db=# UPDATE transactions
SET amount = amount + 100
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';
UPDATE 1
financial_db=# ALTER TABLE transactions REPLICA IDENTITY FULL;
ALTER TABLE
financial_db=# UPDATE transactions
SET amount = amount + 150
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';
```

Now, I observed the offset 3, I can see that a new value propagated for “before”, but still the issue with the “amount” column exists



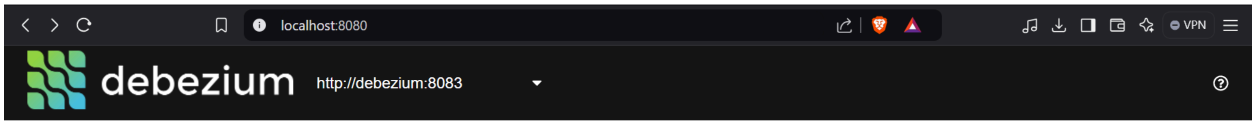
```
    },
    "payload": {
      "before": {
        "transaction_id": "a8b3bed4-19c7-4d63-b1b0-712ad00425b9",
        "user_id": "jonesdavid",
        "timestamp": 1733109723000000,
        "amount": {
          "scale": 2,
          "value": "Aaar"
        },
        "currency": "GBP",
        "city": "Michaelland",
        "country": "Palestinian Territory",
        "merchant_name": "Garcia Group",
        "payment_method": "online_transfer",
        "ip_address": "171.102.153.110",
        "voucher_code": "",
        "affiliate_id": "f68aacb9-ebd2-4d3e-81eb-468dda4f8587"
      },
      "after": {
        "transaction_id": "a8b3bed4-19c7-4d63-b1b0-712ad00425b9",
        "user_id": "jonesdavid",
        "timestamp": 1733109723000000,
        "amount": {
```

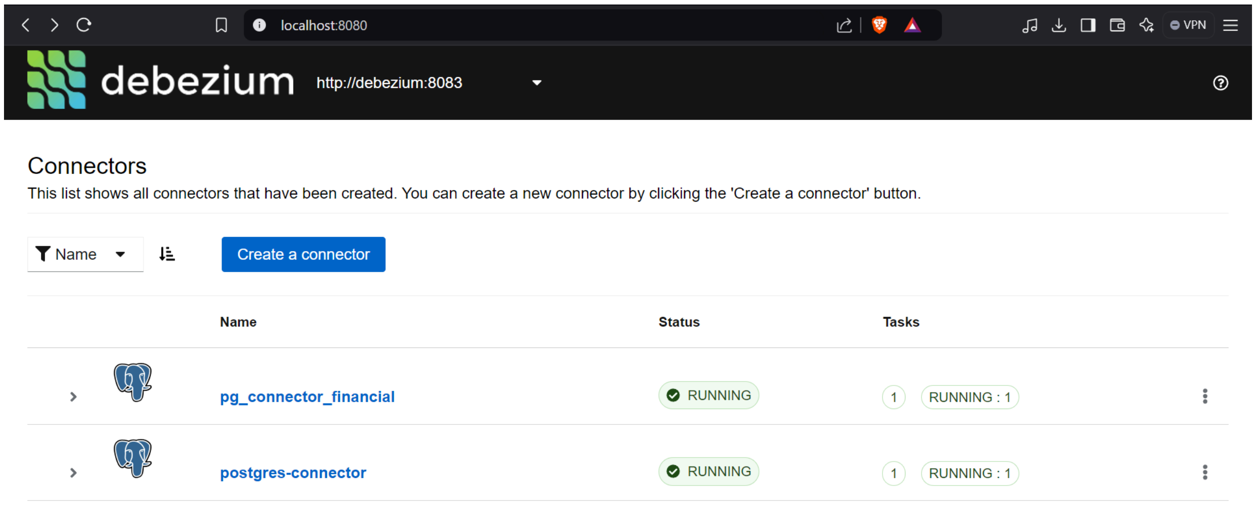
The issue with the amount column is because of the way Debezium handles the decimal values. Instead of doing the conversion exclusively into decimal, I changed the connector value. I added a property in the JSON “decimal.handling.mode”: “string” as below



```
sh-5.2$ ^C
sh-5.2$ curl -X POST -H "Content-Type: application/json" localhost:8083/connectors \
--data '{
  "name": "postgres-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "plugin.name": "pgoutput",
    "database.hostname": "postgres",
    "database.port": "5432",
    "database.user": "postgres",
    "database.password": "postgres",
    "database.dbname": "financial_db",
    "database.server.name": "postgres",
    "table.include.list": "public.transactions",
    "topic.prefix": "cdc",
    "decimal.handling.mode": "string"
  }
}'
{"name":"postgres-connector","config":{"connector.class":"io.debezium.connector.postgresql.PostgresConnector","plugin.name":"p
goutput","database.hostname":"postgres","database.port":"5432","database.user":"postgres","database.password":"postgres","data
base.dbname":"financial_db","database.server.name":"postgres","table.include.list":"public.transactions","topic.prefix":"cdc",
"decimal.handling.mode":"string","name":"postgres-connector"},"tasks":[],"type":"source"}sh-5.2$ 
```

Now, I can see one more connector in the debezium UI with the decimal handling mode present in the properties.





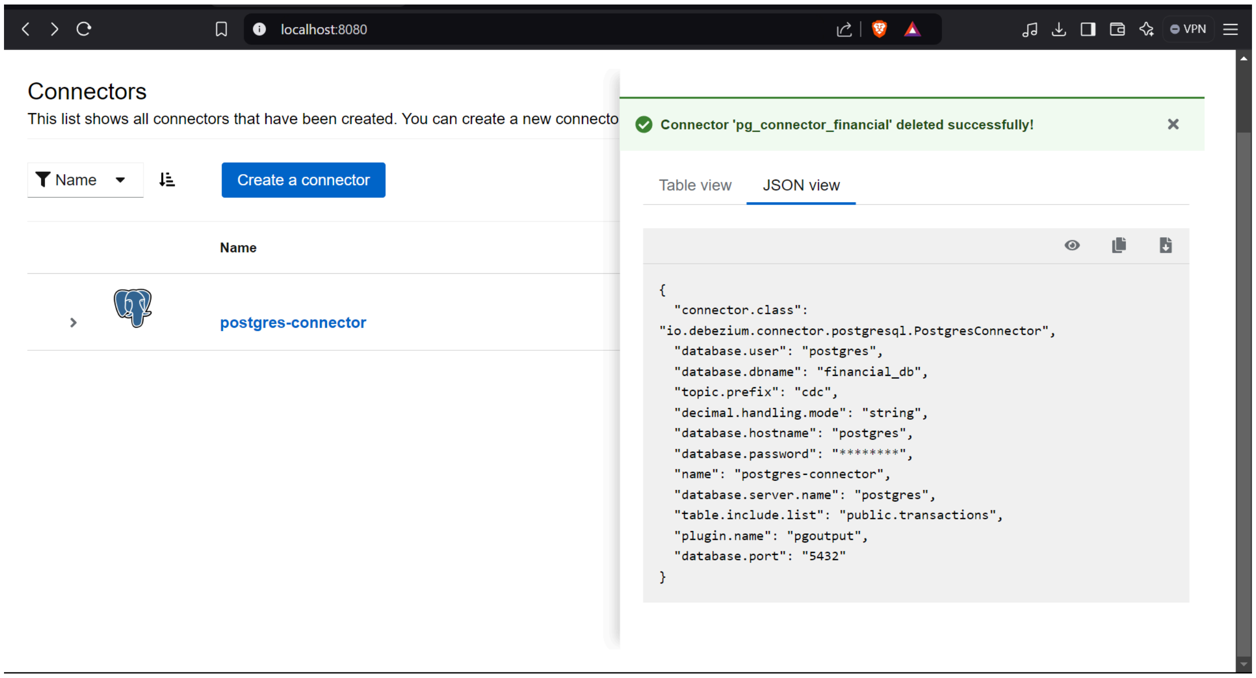
localhost:8080

http://debezium:8083

Connectors

This list shows all connectors that have been created. You can create a new connector by clicking the 'Create a connector' button.

Name	Status	Tasks
pg_connector_financial	RUNNING	1 RUNNING : 1
postgres-connector	RUNNING	1 RUNNING : 1



localhost:8080

Connectors

This list shows all connectors that have been created. You can create a new connecto

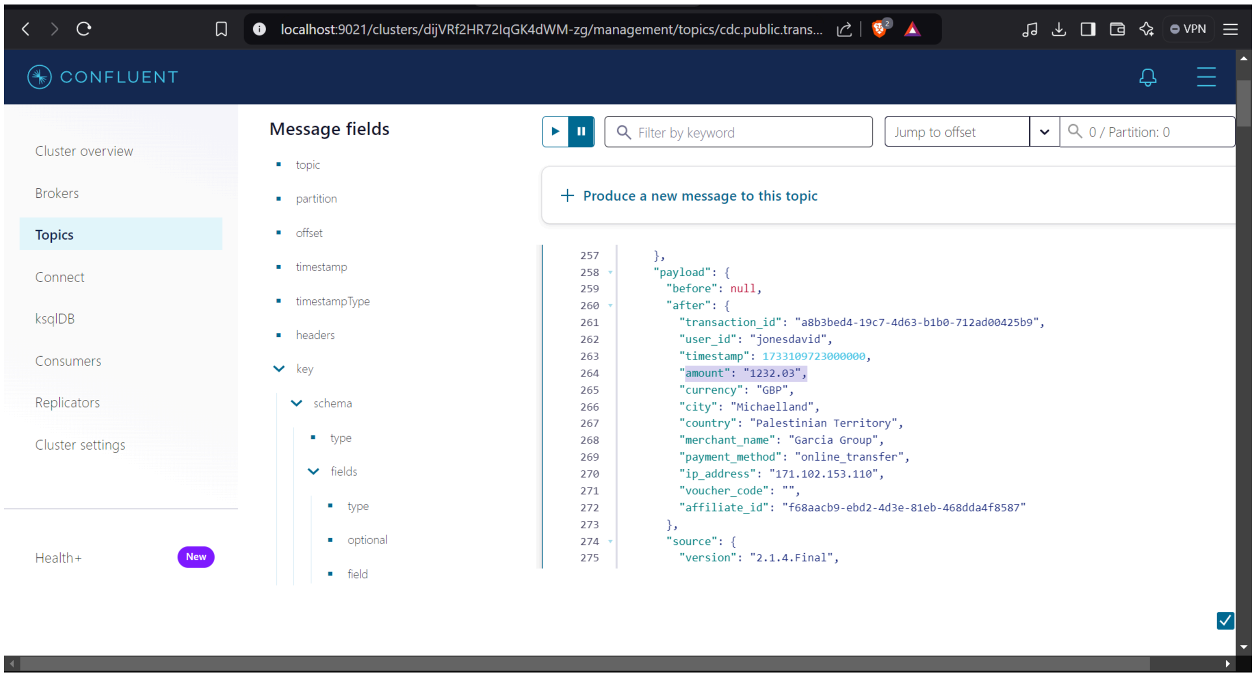
Name
postgres-connector

Connector 'pg_connector_financial' deleted successfully!

Table view JSON view

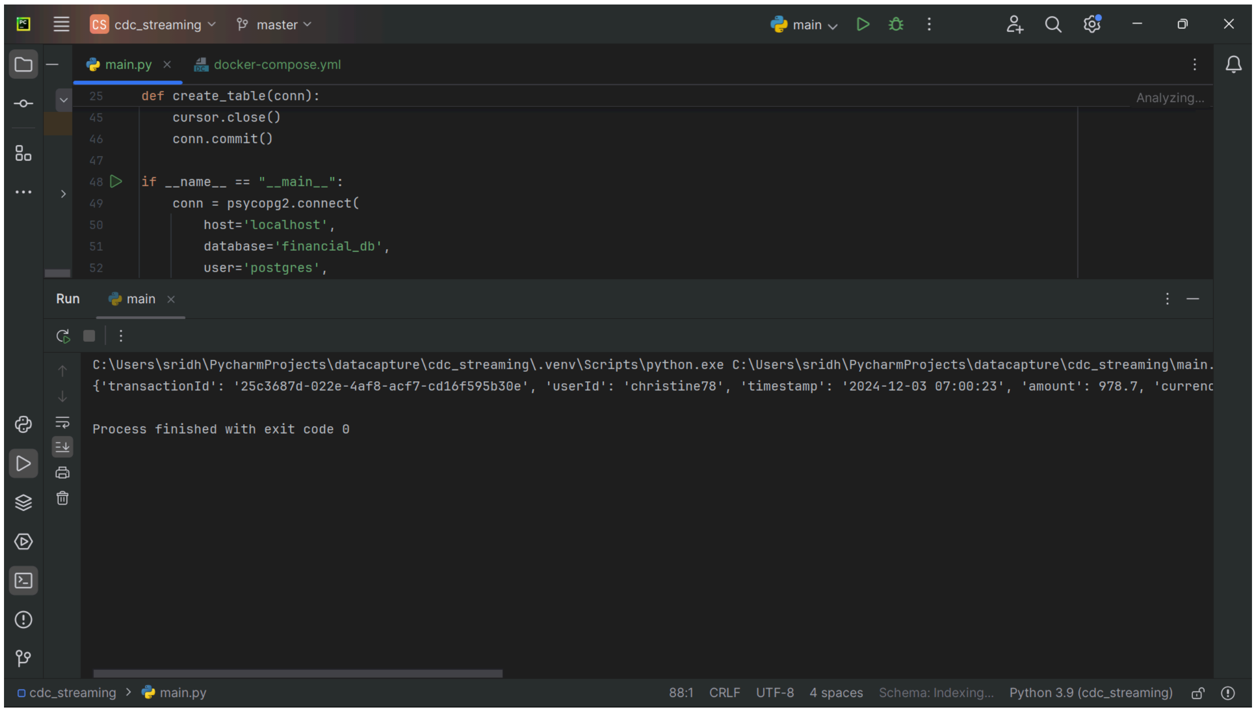
```
{
  "connector.class":
"io.debezium.connector.postgresql.PostgresConnector",
  "database.user": "postgres",
  "database.dbname": "financial_db",
  "topic.prefix": "cdc",
  "decimal.handling.mode": "string",
  "database.hostname": "postgres",
  "database.password": "********",
  "name": "postgres-connector",
  "database.server.name": "postgres",
  "table.include.list": "public.transactions",
  "plugin.name": "pgoutput",
  "database.port": "5432"
}
```

I observed the “amount” in offset 5 and I can see that amount is handled properly



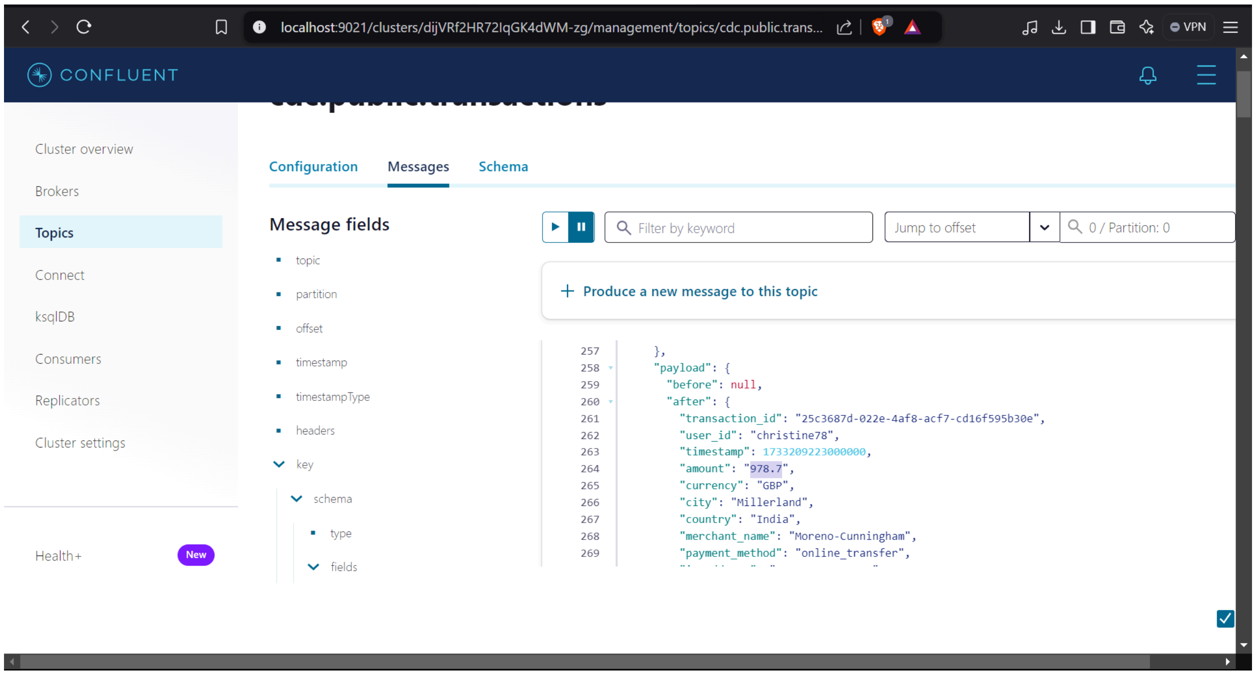
```
    },
    "payload": {
      "before": null,
      "after": {
        "transaction_id": "a8b3bed4-19c7-4d63-b1b0-712ad00425b9",
        "user_id": "jonesdavid",
        "timestamp": 1733109723000000,
        "amount": "1232.03",
        "currency": "GBP",
        "city": "Michaelland",
        "country": "Palestinian Territory",
        "merchant_name": "Garcia Group",
        "payment_method": "online_transfer",
        "ip_address": "171.102.153.110",
        "voucher_code": "",
        "affiliate_id": "f68aacb9-ebd2-4d3e-81eb-468dda4f8587"
      },
      "source": {
        "version": "2.1.4.Final",
```

The data is being properly shipped into kafka. So, if I insert a new record now, I should be able to see the both the before and after values and the amount being handled properly
I inserted one more record by running the main script again as below



```python
def create_table(conn):
        cursor.close()
        conn.commit()

if __name__ == "__main__":
    conn = psycopg2.connect(
        host='localhost',
        database='financial_db',
        user='postgres',
```

```
C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming\.venv\Scripts\python.exe C:\Users\sridh\PycharmProjects\datacapture\cdc_streaming\main.
{'transactionId': '25c3687d-022e-4af8-acf7-cd16f595b30e', 'userId': 'christine78', 'timestamp': '2024-12-03 07:00:23', 'amount': 978.7, 'currenc

Process finished with exit code 0
```



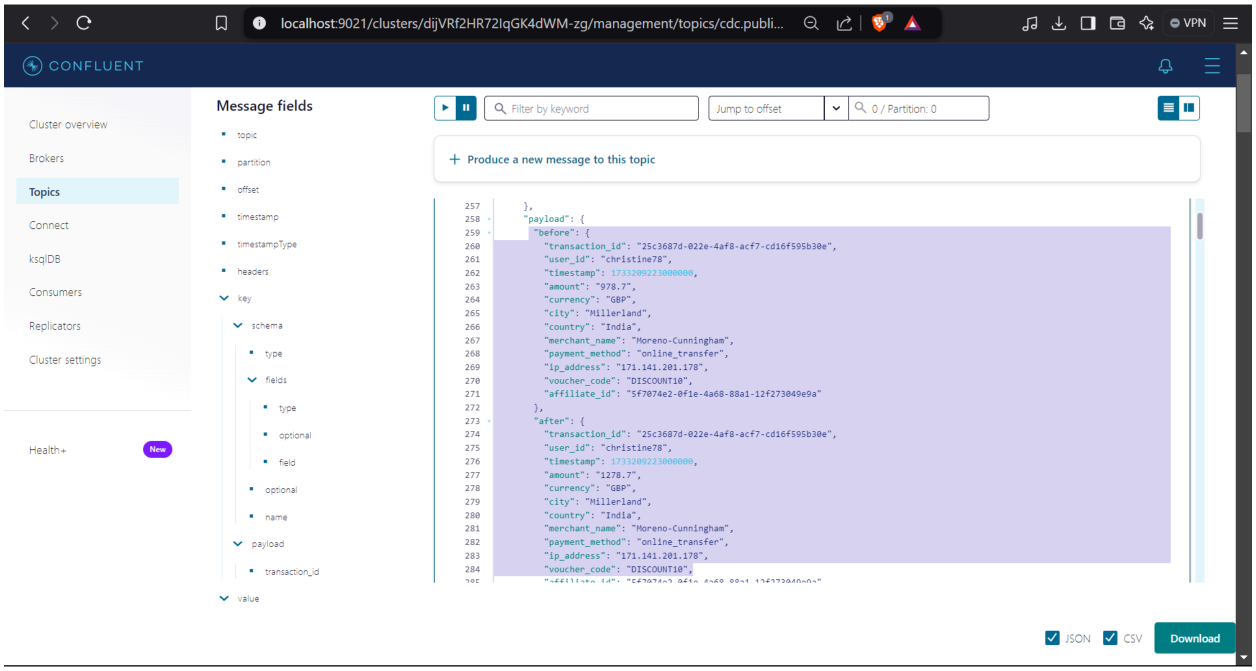
```
    },
    "payload": {
      "before": null,
      "after": {
        "transaction_id": "25c3687d-022e-4af8-acf7-cd16f595b30e",
        "user_id": "christine78",
        "timestamp": 1733209223000000,
        "amount": "978.7",
        "currency": "GBP",
        "city": "Millerland",
        "country": "India",
        "merchant_name": "Moreno-Cunningham",
        "payment_method": "online_transfer",
```

The record has a transaction_id = 25c3687d-022e-4af8-acf7-cd16f595b30e and the amount value = 978.7

Now, If i update the same record, the amount value should be updated and I should see both "before" and "after" values. I executed the below SQL statement in the terminal

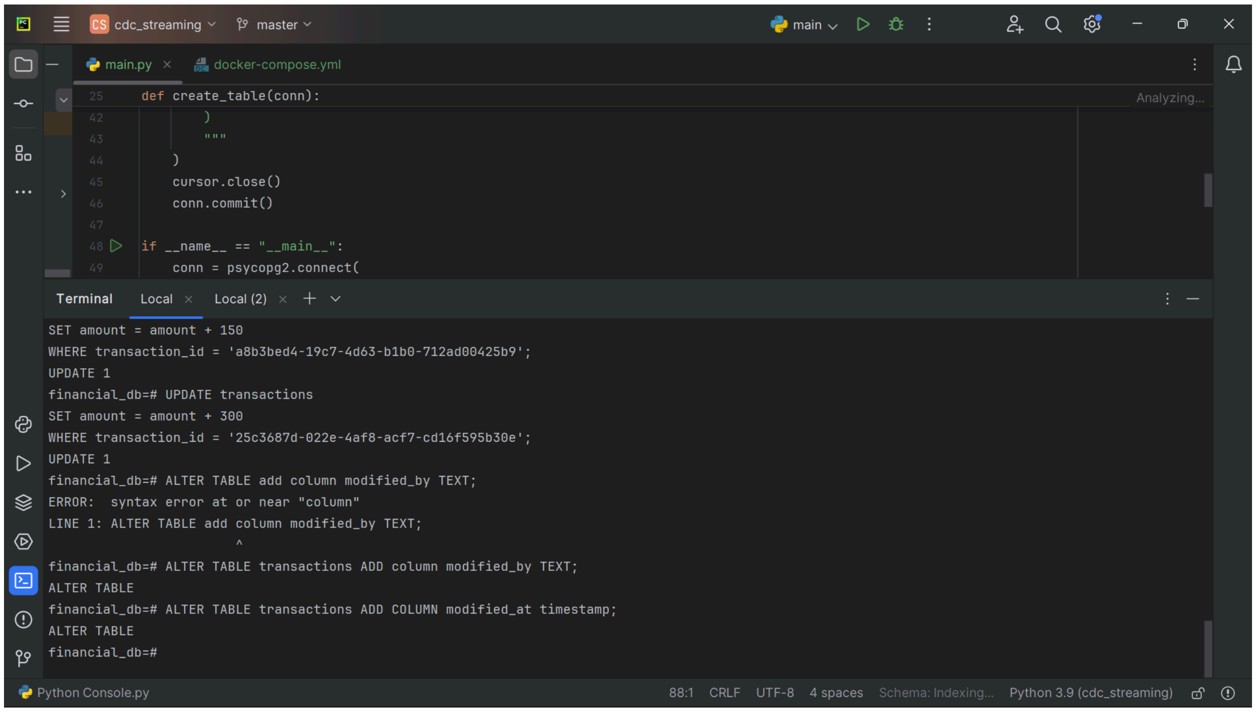
```
UPDATE transactions
SET amount = amount + 300
WHERE transaction_id = '25c3687d-022e-4af8-acf7-cd16f595b30e';
```

I can see the values as below



```
    },
    "payload": {
      "before": {
        "transaction_id": "25c3687d-022e-4af8-acf7-cd16f595b30e",
        "user_id": "christine78",
        "timestamp": 1733209223000000,
        "amount": "978.7",
        "currency": "GBP",
        "city": "Millerland",
        "country": "India",
        "merchant_name": "Moreno-Cunningham",
        "payment_method": "online_transfer",
        "ip_address": "171.141.201.178",
        "voucher_code": "DISCOUNT10",
        "affiliate_id": "5f7074e2-0f1e-4a68-88a1-12f273049e9a"
      },
      "after": {
        "transaction_id": "25c3687d-022e-4af8-acf7-cd16f595b30e",
        "user_id": "christine78",
        "timestamp": 1733209223000000,
        "amount": "1278.7",
        "currency": "GBP",
        "city": "Millerland",
        "country": "India",
        "merchant_name": "Moreno-Cunningham",
        "payment_method": "online_transfer",
        "ip_address": "171.141.201.178",
        "voucher_code": "DISCOUNT10",
```

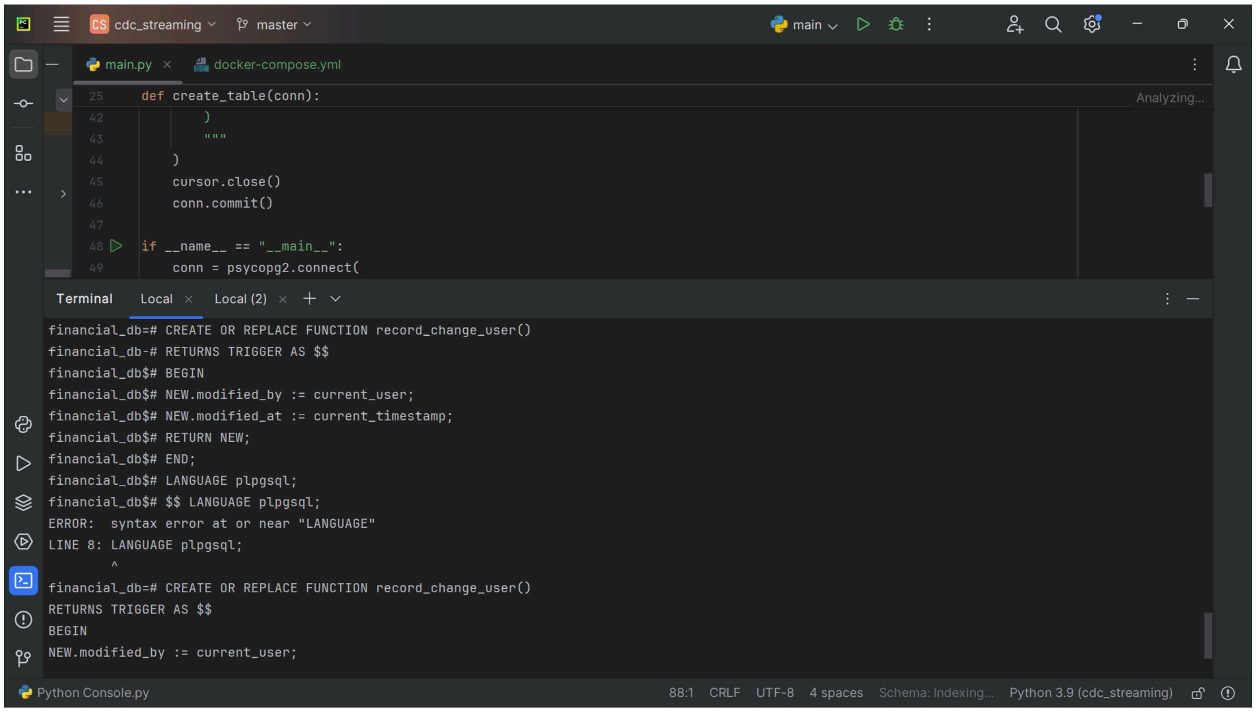
At this point, I can see the changes on the database and I can also see how debezium is tracking those changes. But I am still clueless about the user who changed the data and the timestamp at which the change happened.
I added two columns in the financial_db with names modified_by (TEXT data type), modified_at (TIMESTAMP data type).



```
def create_table(conn):
            )
            """
    )
    cursor.close()
    conn.commit()

if __name__ == "__main__":
    conn = psycopg2.connect(
```

```
SET amount = amount + 150
WHERE transaction_id = 'a8b3bed4-19c7-4d63-b1b0-712ad00425b9';
UPDATE 1
financial_db=# UPDATE transactions
SET amount = amount + 300
WHERE transaction_id = '25c3687d-022e-4af8-acf7-cd16f595b30e';
UPDATE 1
financial_db=# ALTER TABLE add column modified_by TEXT;
ERROR:  syntax error at or near "column"
LINE 1: ALTER TABLE add column modified_by TEXT;
                        ^
financial_db=# ALTER TABLE transactions ADD column modified_by TEXT;
ALTER TABLE
financial_db=# ALTER TABLE transactions ADD COLUMN modified_at timestamp;
ALTER TABLE
financial_db=#
```

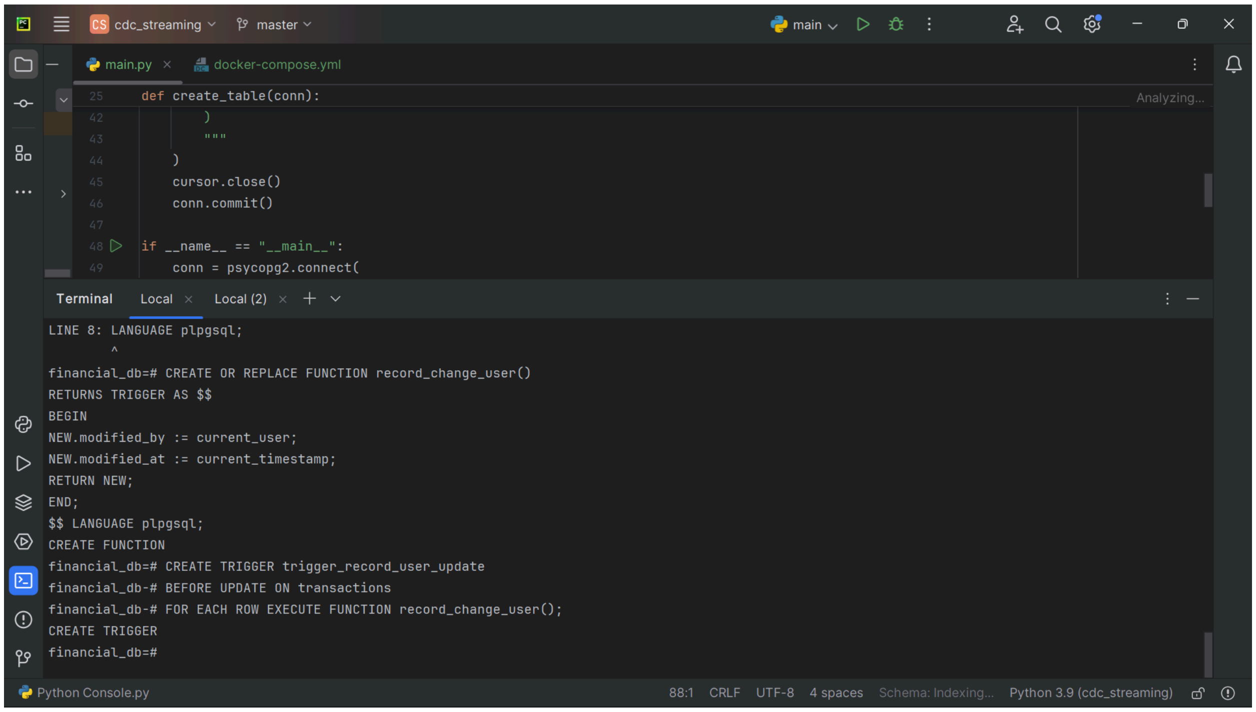
I created a function with the name record_change_user()



```
def create_table(conn):
            )
            """
        )
        cursor.close()
        conn.commit()

if __name__ == "__main__":
        conn = psycopg2.connect(
```

```
financial_db=# CREATE OR REPLACE FUNCTION record_change_user()
financial_db-# RETURNS TRIGGER AS $$
financial_db$# BEGIN
financial_db$# NEW.modified_by := current_user;
financial_db$# NEW.modified_at := current_timestamp;
financial_db$# RETURN NEW;
financial_db$# END;
financial_db$# LANGUAGE plpgsql;
financial_db$# $$ LANGUAGE plpgsql;
ERROR:  syntax error at or near "LANGUAGE"
LINE 8: LANGUAGE plpgsql;
        ^
financial_db=# CREATE OR REPLACE FUNCTION record_change_user()
RETURNS TRIGGER AS $$
BEGIN
NEW.modified_by := current_user;
```

I configured a trigger trigger_record_user_update to be triggered before any update on the transactions table as below

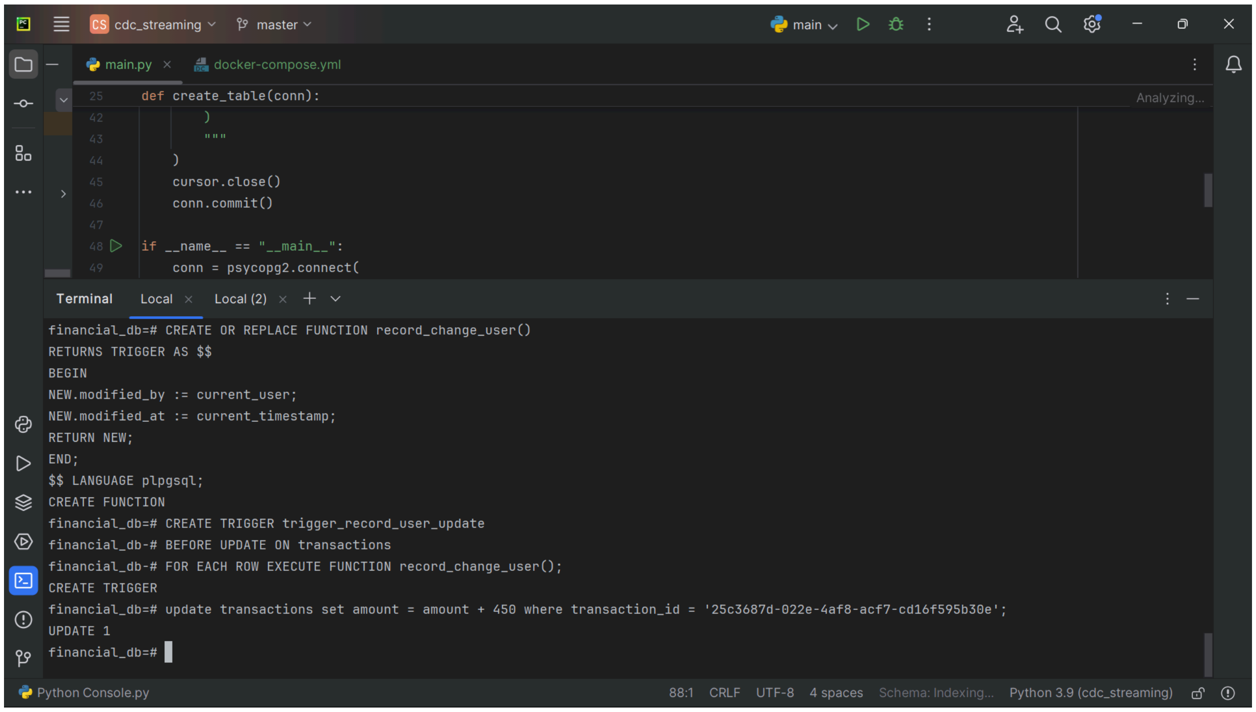


```
def create_table(conn):
            )
            """
        )
        cursor.close()
        conn.commit()

if __name__ == "__main__":
        conn = psycopg2.connect(
```

```
LINE 8: LANGUAGE plpgsql;
        ^
financial_db=# CREATE OR REPLACE FUNCTION record_change_user()
RETURNS TRIGGER AS $$
BEGIN
NEW.modified_by := current_user;
NEW.modified_at := current_timestamp;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
financial_db=# CREATE TRIGGER trigger_record_user_update
financial_db-# BEFORE UPDATE ON transactions
financial_db-# FOR EACH ROW EXECUTE FUNCTION record_change_user();
CREATE TRIGGER
financial_db=#
```

For the transaction with “transaction_id” = “25c3687d-022e-4af8-acf7-cd16f595b30e”, the “amount” value in “before” and “after” were “978.7” and “1278.7” . If I update this transaction again, I should be able to track the user who updated this transaction and when it was updated. I should also be able to see changes in amount value on the kafka brokers

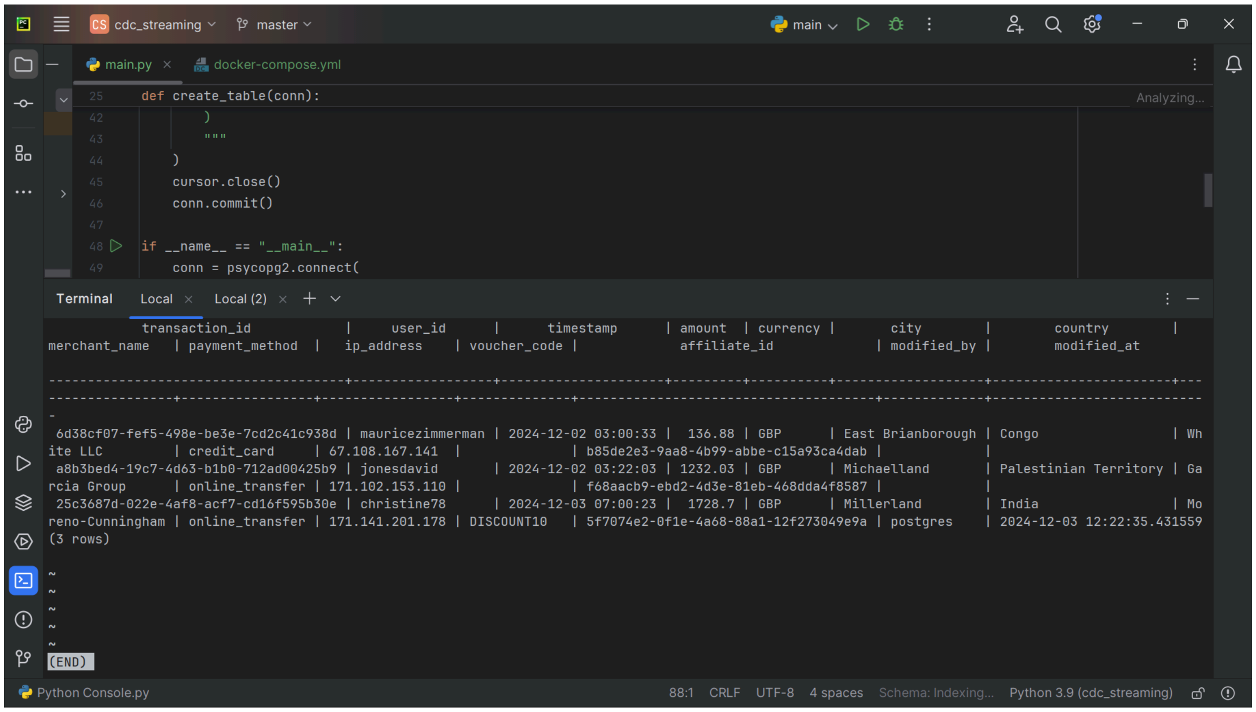
I updated the “amount” column in transactions as below



```
def create_table(conn):
            )
            """
        )
        cursor.close()
        conn.commit()

if __name__ == "__main__":
        conn = psycopg2.connect(
```

```
financial_db=# CREATE OR REPLACE FUNCTION record_change_user()
RETURNS TRIGGER AS $$
BEGIN
NEW.modified_by := current_user;
NEW.modified_at := current_timestamp;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
financial_db=# CREATE TRIGGER trigger_record_user_update
financial_db-# BEFORE UPDATE ON transactions
financial_db-# FOR EACH ROW EXECUTE FUNCTION record_change_user();
CREATE TRIGGER
financial_db=# update transactions set amount = amount + 450 where transaction_id = '25c3687d-022e-4af8-acf7-cd16f595b30e';
UPDATE 1
financial_db=# 
```

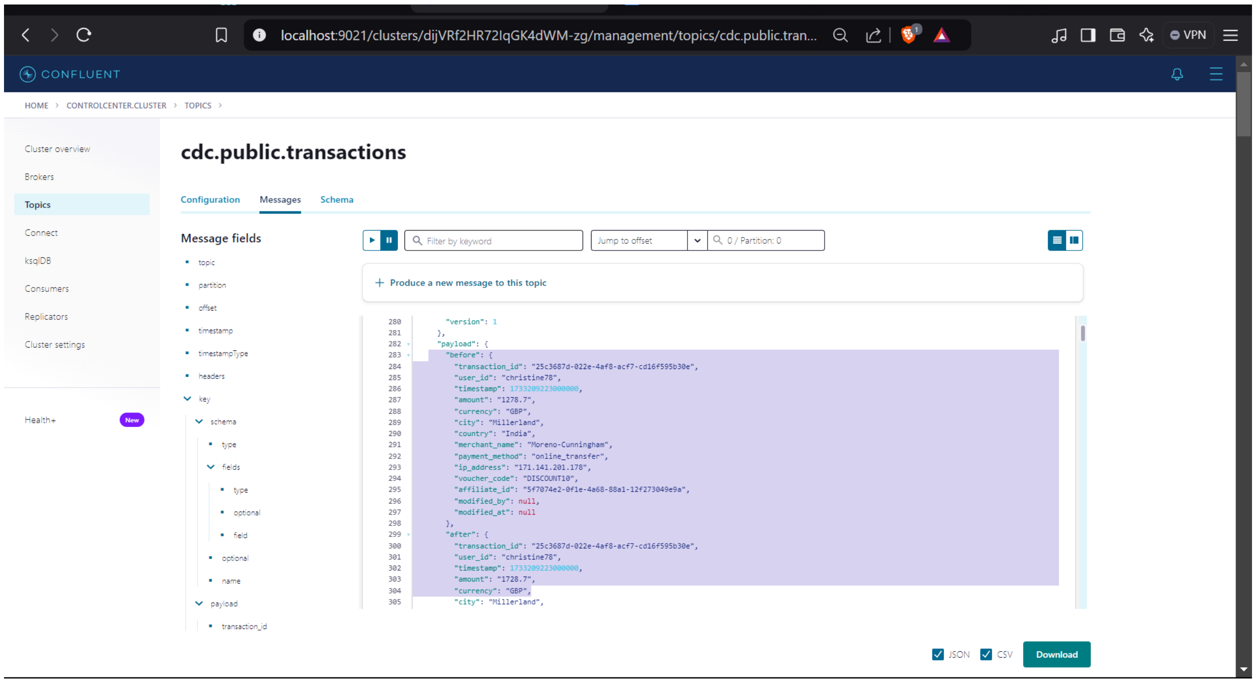
Now, If i see the values populated in columns (modified_by, modified_at), I can see the below results



```
def create_table(conn):
            )
            """
        )
        cursor.close()
        conn.commit()

if __name__ == "__main__":
        conn = psycopg2.connect(
```

```
            transaction_id             |     user_id      |      timestamp      | amount  | currency |       city        |        country        |
merchant_name    | payment_method  |   ip_address    | voucher_code |             affiliate_id             | modified_by |        modified_at

--------------------------------------+------------------+---------------------+---------+----------+-------------------+-----------------------+---
---------------+-----------------+-----------------+--------------+--------------------------------------+-------------+----------------------------
-
 6d38cf07-fef5-498e-be3e-7cd2c41c938d | mauricezimmerman | 2024-12-02 03:00:33 |  136.88 | GBP      | East Brianborough | Congo                 | Wh
ite LLC          | credit_card     | 67.108.167.141  |              | b85de2e3-9aa8-4b99-abbe-c15a93ca4dab |             |
 a8b3bed4-19c7-4d63-b1b0-712ad00425b9 | jonesdavid       | 2024-12-02 03:22:03 | 1232.03 | GBP      | Michaelland       | Palestinian Territory | Ga
rcia Group       | online_transfer | 171.102.153.110 |              | f68aacb9-ebd2-4d3e-81eb-468dda4f8587 |             |
 25c3687d-022e-4af8-acf7-cd16f595b30e | christine78      | 2024-12-03 07:00:23 |  1728.7 | GBP      | Millerland        | India                 | Mo
reno-Cunningham | online_transfer | 171.141.201.178 | DISCOUNT10   | 5f7074e2-0f1e-4a68-88a1-12f273049e9a | postgres    | 2024-12-03 12:22:35.431559
(3 rows)

~
~
~
~
~
(END)
```

Also, on the control center UI, I observe the values for “amount” for offset 8 would be 1278.7 and 1728.7 in “before” and “after”



```
      "version": 1
    },
    "payload": {
      "before": {
        "transaction_id": "25c3687d-022e-4af8-acf7-cd16f595b30e",
        "user_id": "christine78",
        "timestamp": 1733209223000000,
        "amount": "1278.7",
        "currency": "GBP",
        "city": "Millerland",
        "country": "India",
        "merchant_name": "Moreno-Cunningham",
        "payment_method": "online_transfer",
        "ip_address": "171.141.201.178",
        "voucher_code": "DISCOUNT10",
        "affiliate_id": "5f7074e2-0f1e-4a68-88a1-12f273049e9a",
        "modified_by": null,
        "modified_at": null
      },
      "after": {
        "transaction_id": "25c3687d-022e-4af8-acf7-cd16f595b30e",
        "user_id": "christine78",
        "timestamp": 1733209223000000,
        "amount": "1728.7",
        "currency": "GBP",
        "city": "Millerland",
```

Now, I am able to track the changes in the “amount” column. This is how I can observe the change data capture.