**Task 3 Documentation**

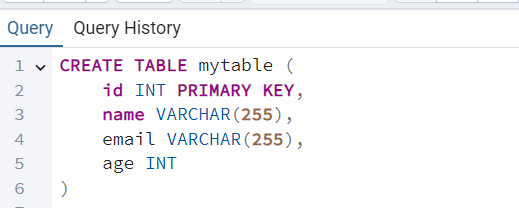
* **Enable CDC in source table**
* **Create producer and consumer python scripts (Producer to insert data to topics and Consumer to fetch data from topics and insert into sink table)**
* **Verify whether changes in the source table are reflected in the topic and in the sink table**

**CDC:**

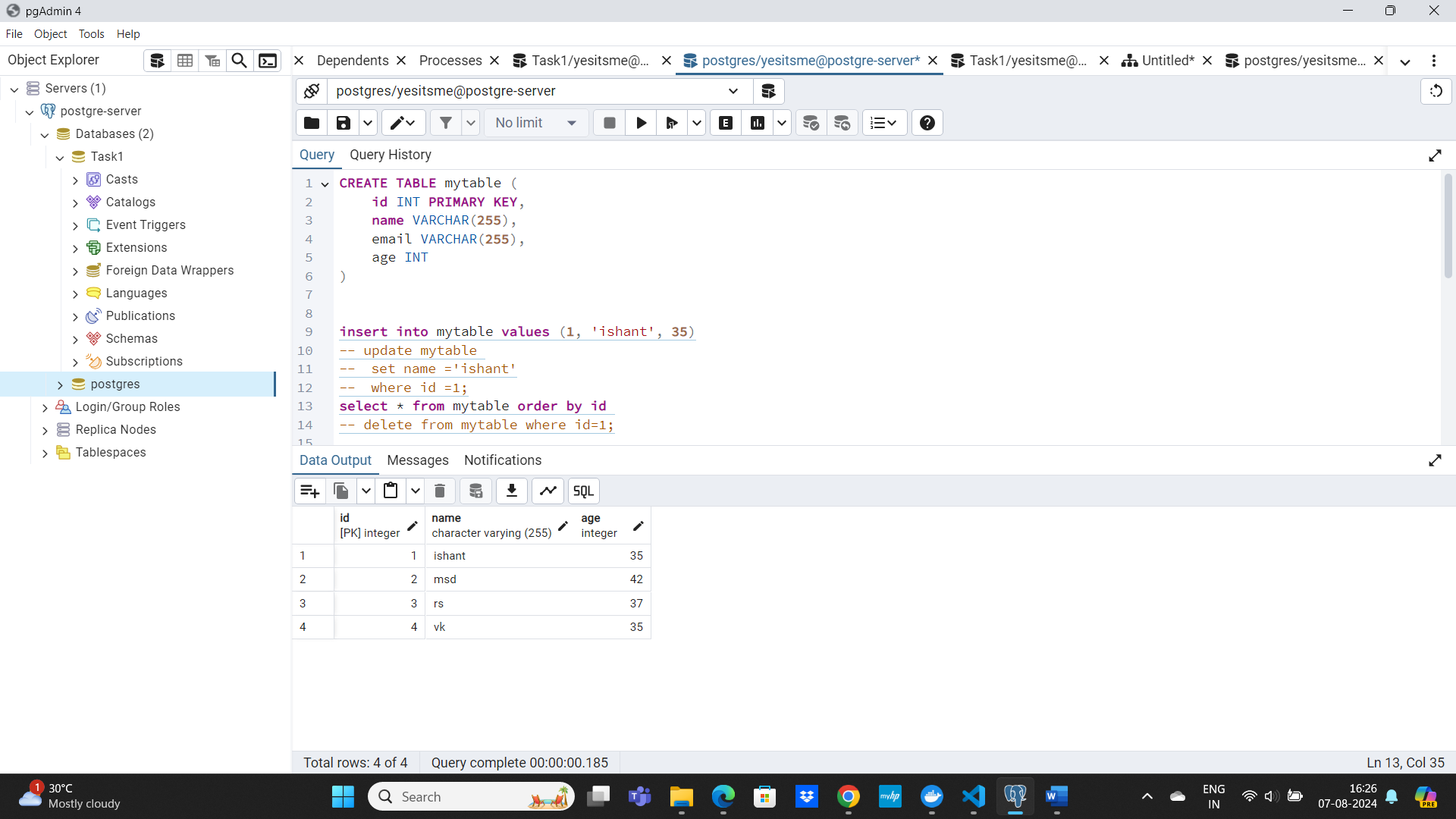
**Change Data Capture (CDC) is a technology in SQL Server that allows you to track changes (inserts, updates, and deletes) of tables made in your database.**

**Step 1:**

* **Created table on source as ‘mytable’**

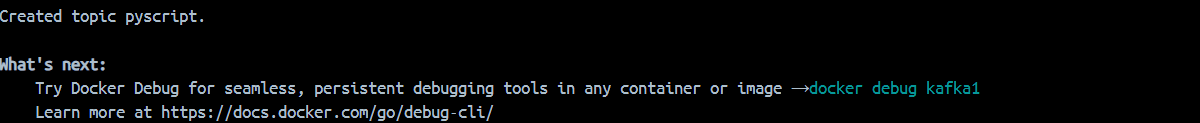
****

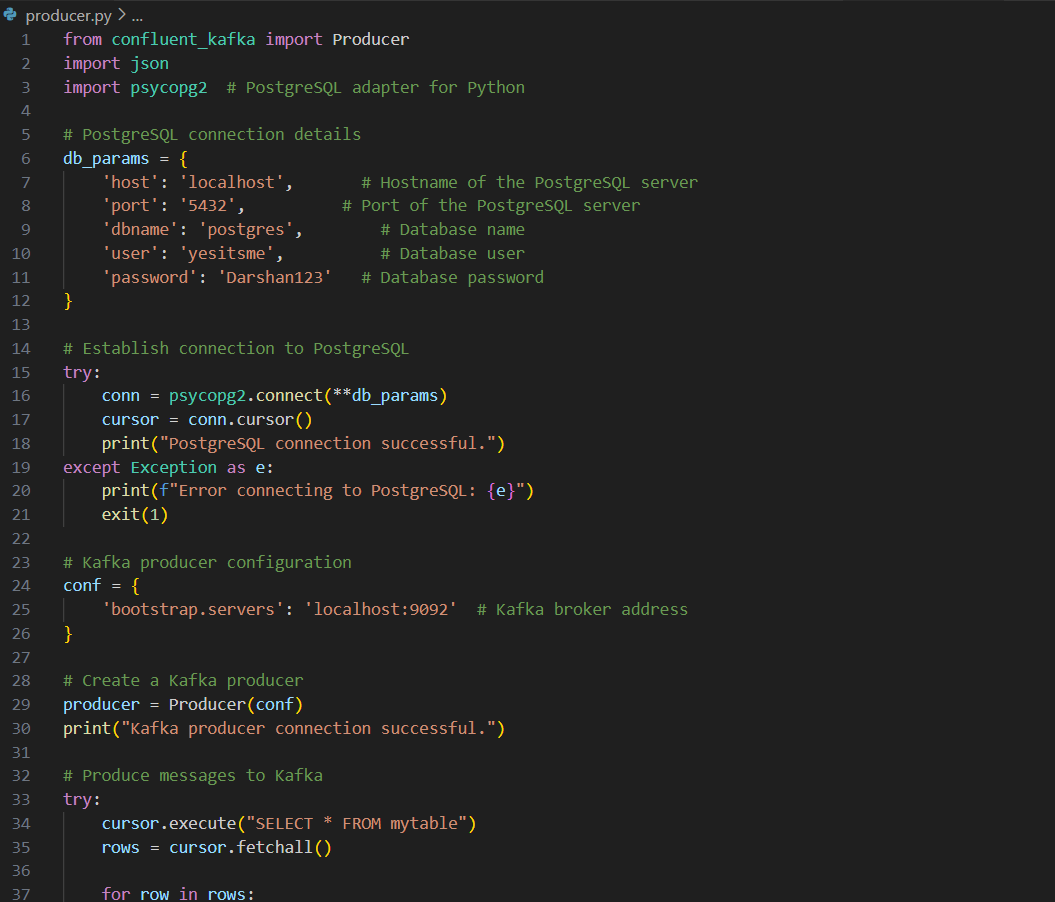
* **Result in database server:**

****

**Step 2: Made a python script for producer to send the data to consumer.**

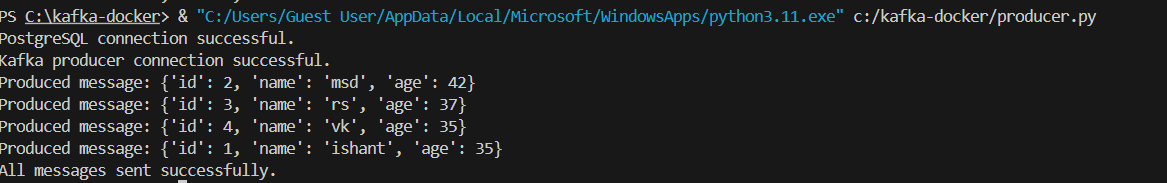
* **Created a new topic as ‘pyscript’.**



****

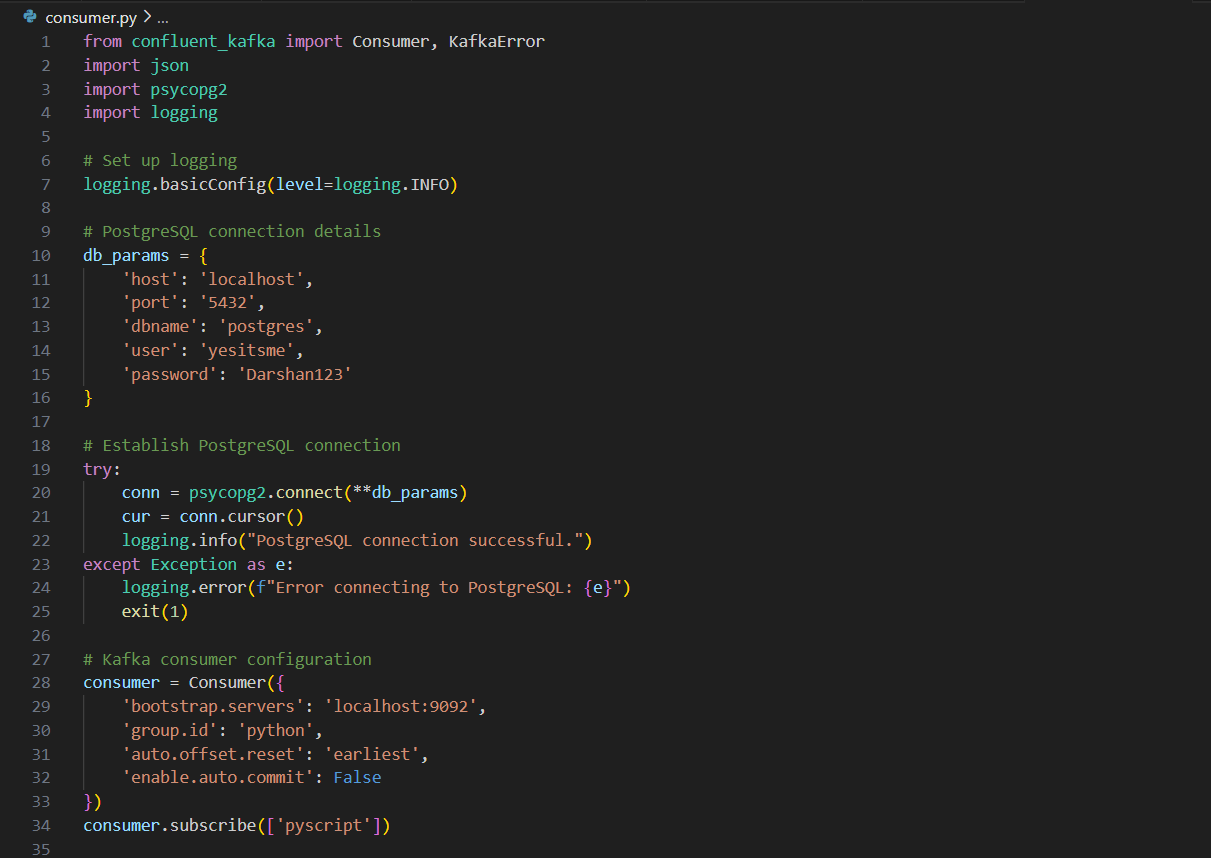
****

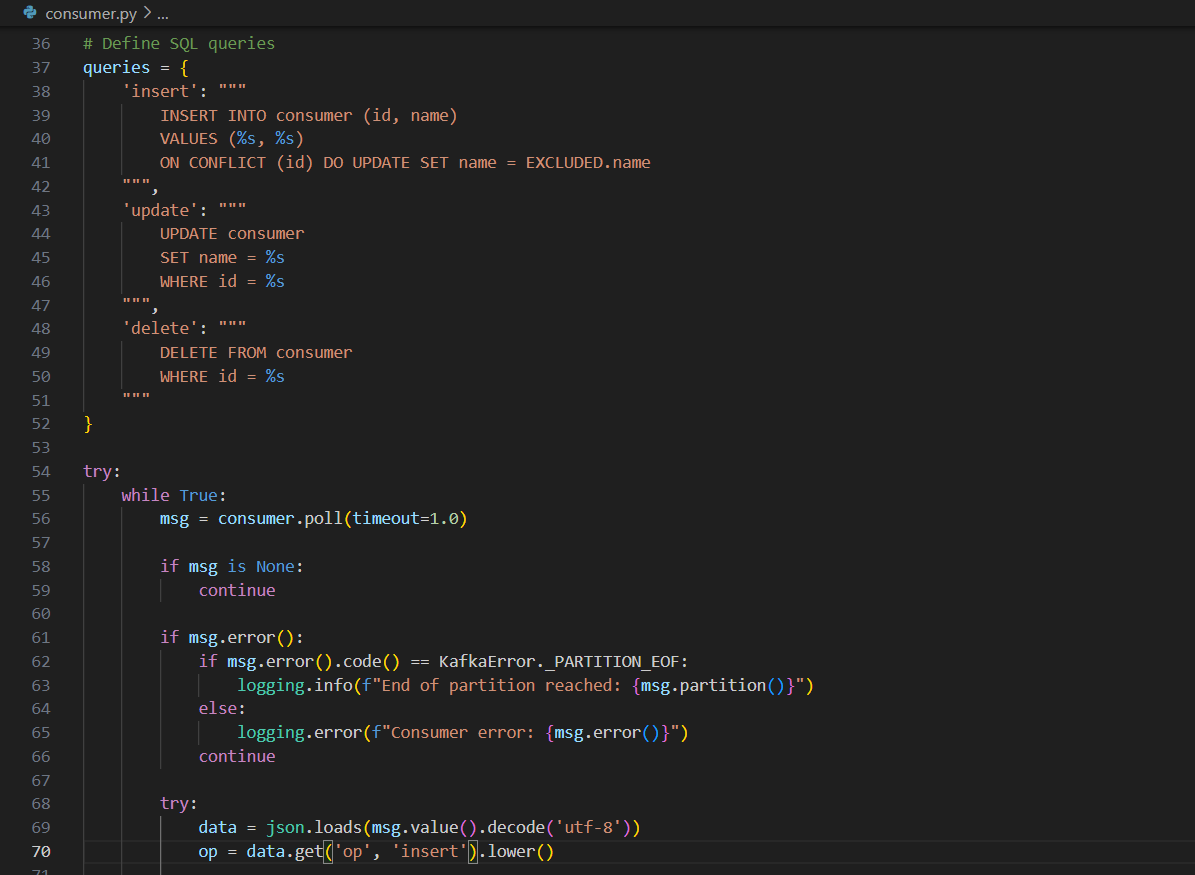
**Output After Insert:**

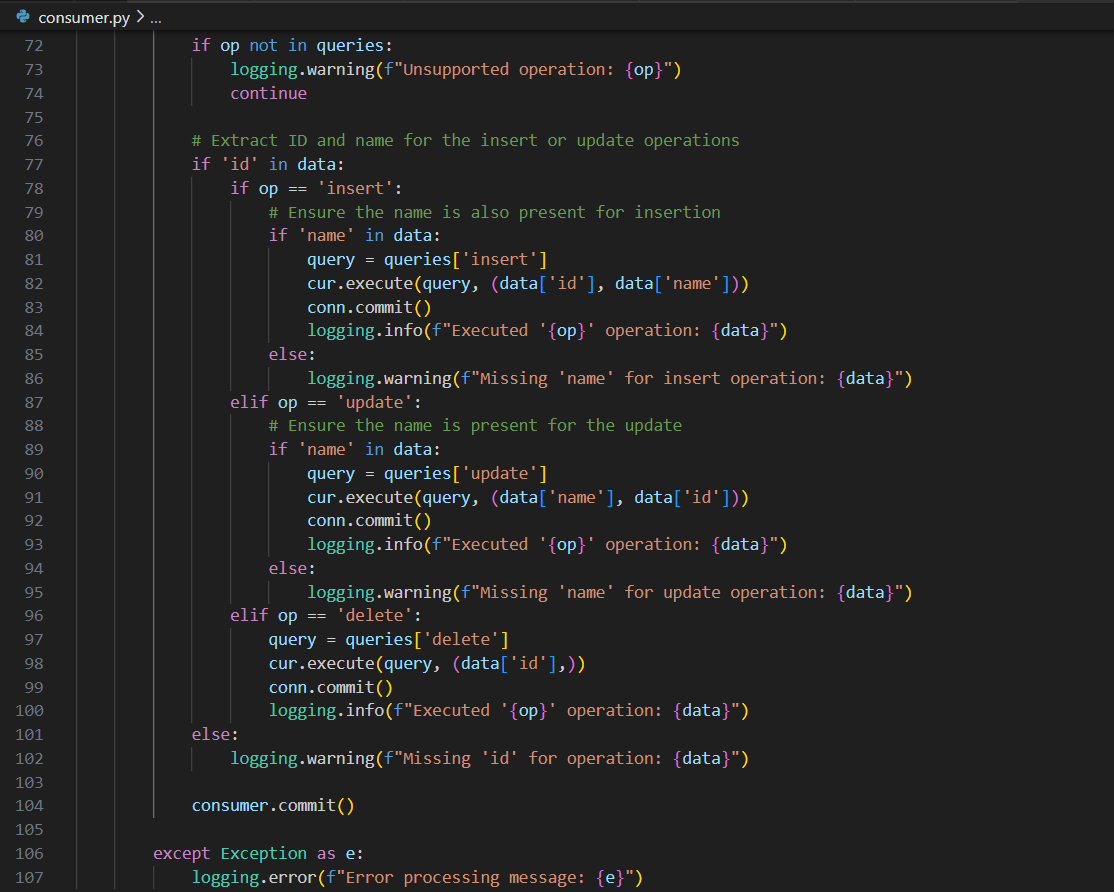
****

**Step 3: Made a consumer python script for hold the live data which is getting updaed by doing changes.**

**In consumer made a table: ‘consumer’**

****

****

****

****

1. **Imports Required Libraries:**

* **confluent\_kafka**: Used to create a Kafka consumer to read messages from Kafka topics.
* **json**: Used to parse JSON-encoded messages from Kafka.
* **psycopg2**: A PostgreSQL database adapter for Python.
* **logging**: The standard logging library for Python to log output and errors.

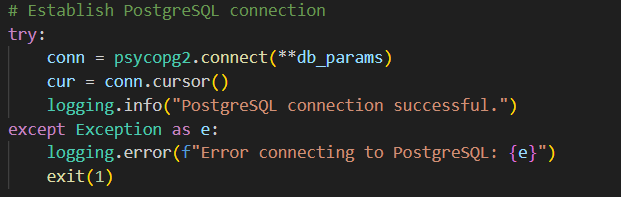
**2. Logging Setup**

Here, logging is configured to display messages at the INFO level and above.

**3. Database Connection Configuration**

This dictionary contains the parameters needed to connect to the PostgreSQL database. It specifies the host, port, database name, user, and password.

**4. Database Connection Establishment**

****

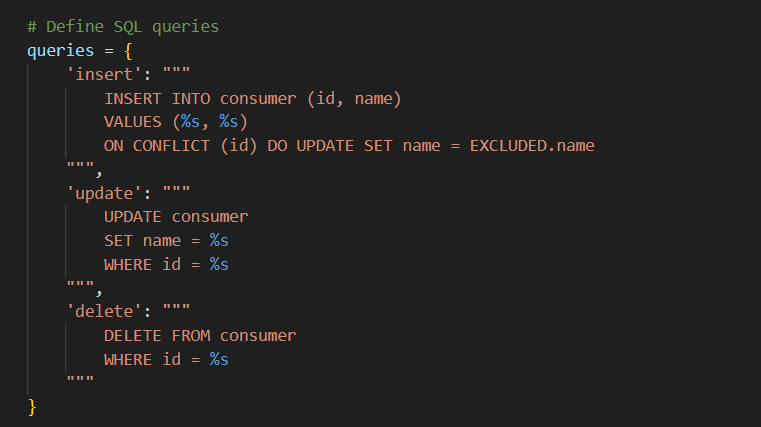
* The script attempts to establish a connection to the PostgreSQL database using the parameters defined earlier. If successful, it creates a cursor for executing SQL commands and logs a success message.
* If the connection fails, an error message is logged and the script exits.

**5. Kafka Consumer Configuration**

****

* A Kafka consumer is initialized with the bootstrap server address, a consumer group ID, and settings for handling offsets.
* The consumer subscribes to the Kafka topic named "pyscript".

**6. SQL Queries**

****

* This dictionary contains SQL query templates for insert, update, and delete operations for a table named consumer.

**7. Main Processing Loop**

****

* A continuous loop is initiated to poll Kafka for messages with a timeout of 1 second. If there are no messages, the loop continues.

**8. Message Processing**

Inside the loop:

* The script checks if msg is None (no messages) or if there is an error with the message (e.g., end of partition reached).
* If no error occurs, it attempts to decode the message value from bytes to a JSON object.

**Operation Handling**

****

* Each message (assumed to be in JSON format) is parsed, and the operation type (op) is retrieved from the message.

**Conditional Processing**

* Based on the value of op, the script executes the corresponding SQL query (insert, update, or delete):
  + For **insert** and **update** operations, it checks for the required fields (id and name) before executing the query.
  + For **delete**, it only requires the id.

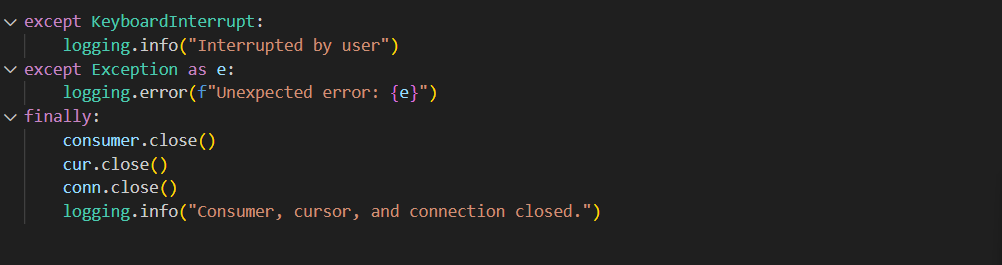


* If the operation is not recognized, a warning is logged.

**9. Error Handling**

* The script includes error handling both for message processing and database operations, logging any errors encountered.

**10. Graceful Shutdown**

****

* This part handles graceful shutdowns of the consumer and database connection, ensuring all resources are properly closed and relevant logs are generated.

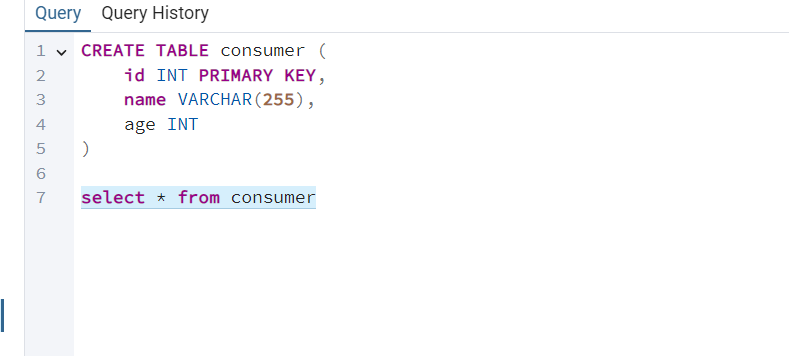
**Summary**

The script is a robust example of an ETL (Extract, Transform, Load) process using a Kafka producer to provide updates and a PostgreSQL database to store processed data. It efficiently handles different operations based on the incoming messages and implements error handling and logging throughout the process.

**Conclusion**

In summary, this script sets up a Kafka consumer that listens for messages on a specific topic, processes the received JSON messages, checks if a corresponding record already exists in a PostgreSQL database, and if not, inserts that record into the database. If no messages are received or an error occurs, it shuts down cleanly by closing the database connection.

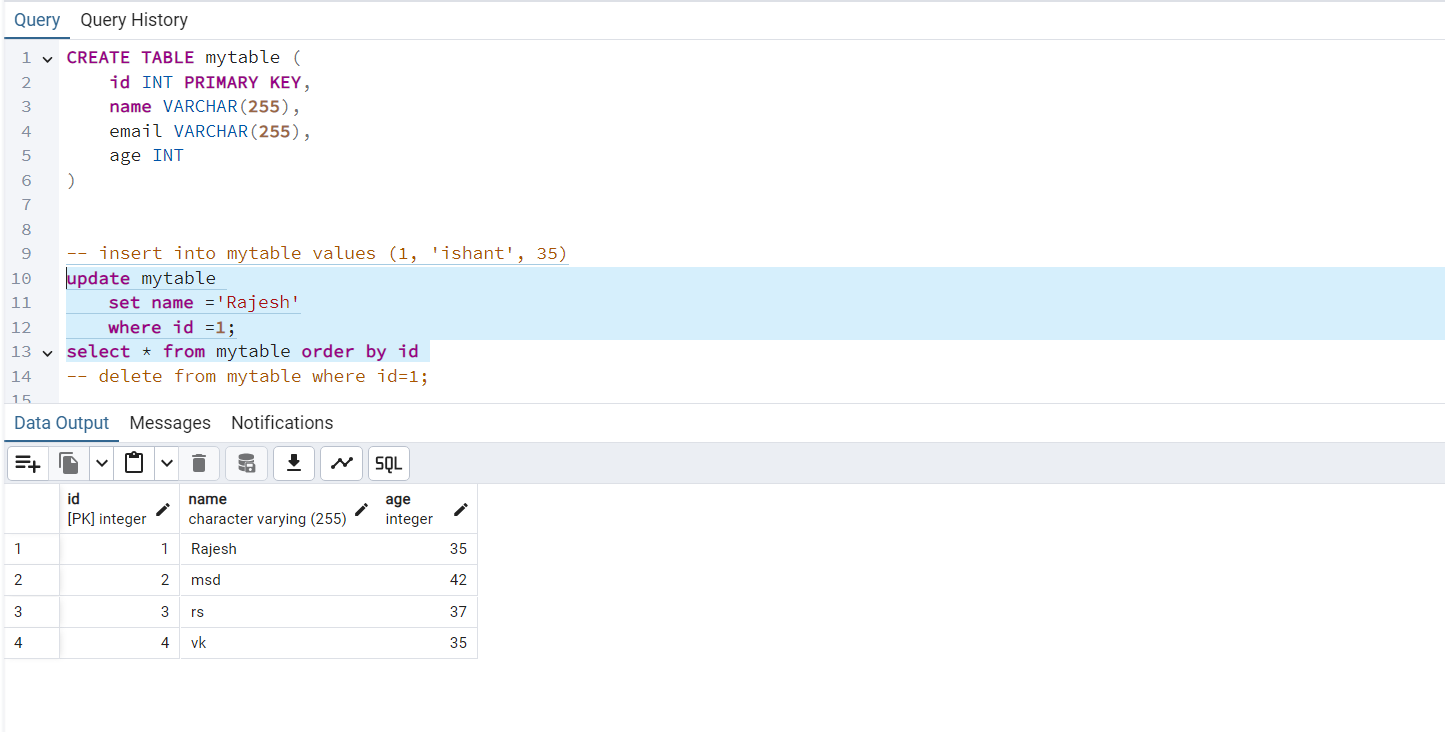
**Step 4: Now, will do some operations in producers table ‘mytable’, it will reflect updation in consumer table ‘consumer’.**

****

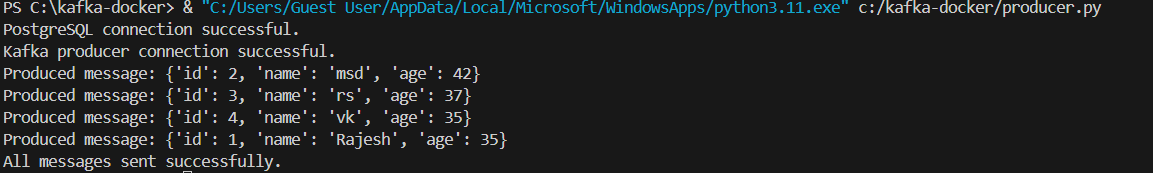
**After insert, also implemented update and delete, it is reflecting all record as once we run the python file.**

**1.After update:**

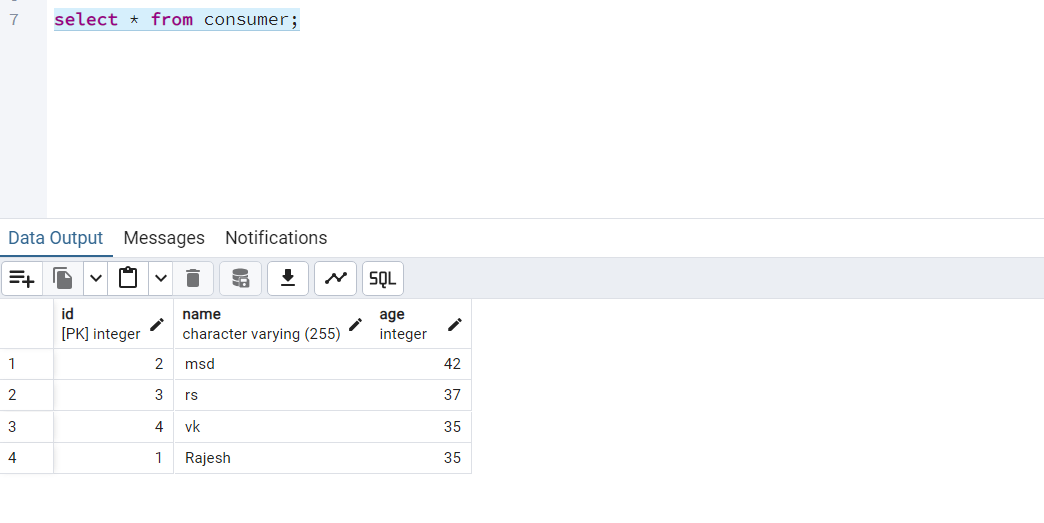
**Output In PostgreSQL (Producer):**

****

**Output In terminal:**

****

**Output In PostgreSQL (Consumer):**

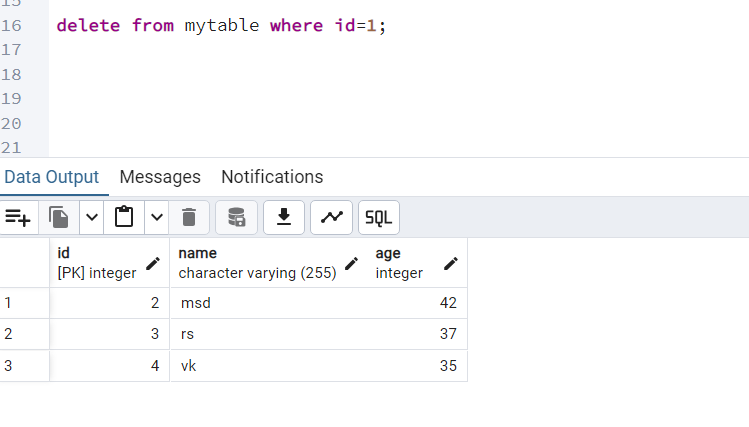
****

**Output In Terminal (Consumer):**

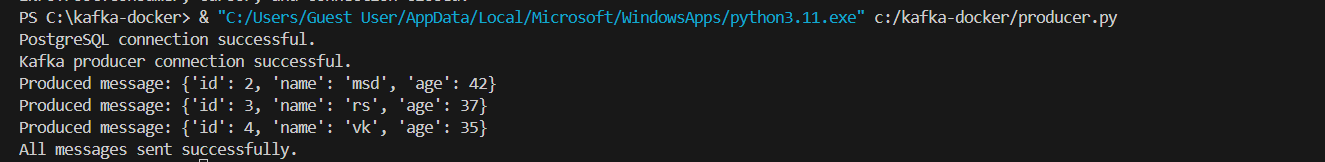
****

**2.After Delete:**

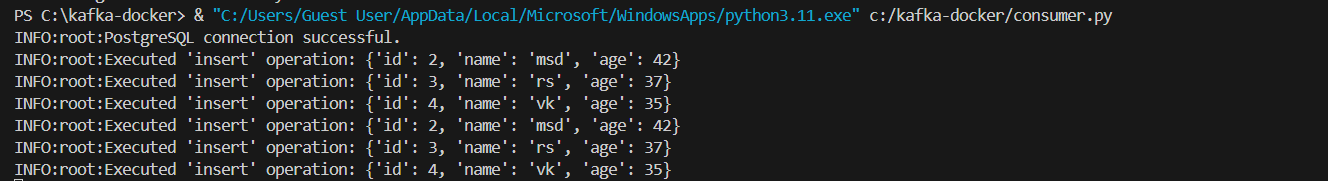
**Output In PostgreSQL 0(Producer):**

****

**Output In Terminal (Producer):**

****

**Output In Terminal (Consumer):**

****

**Issues I got :**

1. **Locahost was not found, so I run my dockerfile and source sink again as well.**
2. **While in consumer I was not able to add configuration like 3-4 types of operations at a time (insert, update, delete), then I make it in if-elif-else by configuring operations or can do with trigger as well.**