# **Real-Time Data Processing with Confluent Kafka, MySQL, and Avro**

**Table of Contents**

[**Real-Time Data Processing with Confluent Kafka, MySQL, and Avro** 1](#_Toc171089983)

[**Introduction** 2](#_Toc171089984)

[**Problem Statement and Objective** 2](#_Toc171089985)

[**Problem Statement** 2](#_Toc171089986)

[**Objective** 2](#_Toc171089987)

[**Tools Required** 2](#_Toc171089988)

[**System Architecture** 2](#_Toc171089989)

[**Producer Code Overview** 3](#_Toc171089990)

[**Consumer Code Overview** 6](#_Toc171089991)

[**Testing** 9](#_Toc171089992)

[**Challenges Faced** 12](#_Toc171089993)

[**Conclusion** 12](#_Toc171089994)

[**Future Enhancements** 13](#_Toc171089995)

## **Introduction**

Real-time processing is increasingly becoming integral across various industries and becoming the most important in today’s fast world where everyone wants quick and effective services whether it’s about E-commerce where everyone wants to track status of their delivery in real time or monitoring inventory stock of their favorite mobile phone or about IoT(Internet of Things) where smart devices which track traffic in real time so that they could reach their office or picnic with alternate route without causing to stuck in long traffic just like this there are many examples where real time data processing helping us.

In this assignment we build the real time data stream system which will extract the data from the relational database(MySQL) tables and then with the help of python and confluent Kafka ,Avro serialization process the data by publishing data to the topic in the Kafka broker then after that subscribing the topic with the help of consumer group which contain 5 consumers so that we can parallelize the process after consuming and transforming data with the help of python then load the data to Json file. In this system we make the use of partitioning and number of consumers for faster and efficient handling of data processing.

## **Problem Statement and Objective**

### **Problem Statement**

You are working in a fictitious e-commerce company called "**BuyOnline**" which has a MySQL database that stores product information such as product ID, name, category, price, and updated timestamp. The database gets updated frequently with new products and changes in product information. The company wants to build a real-time system to stream these updates incrementally to a downstream system for real-time analytics and business intelligence.

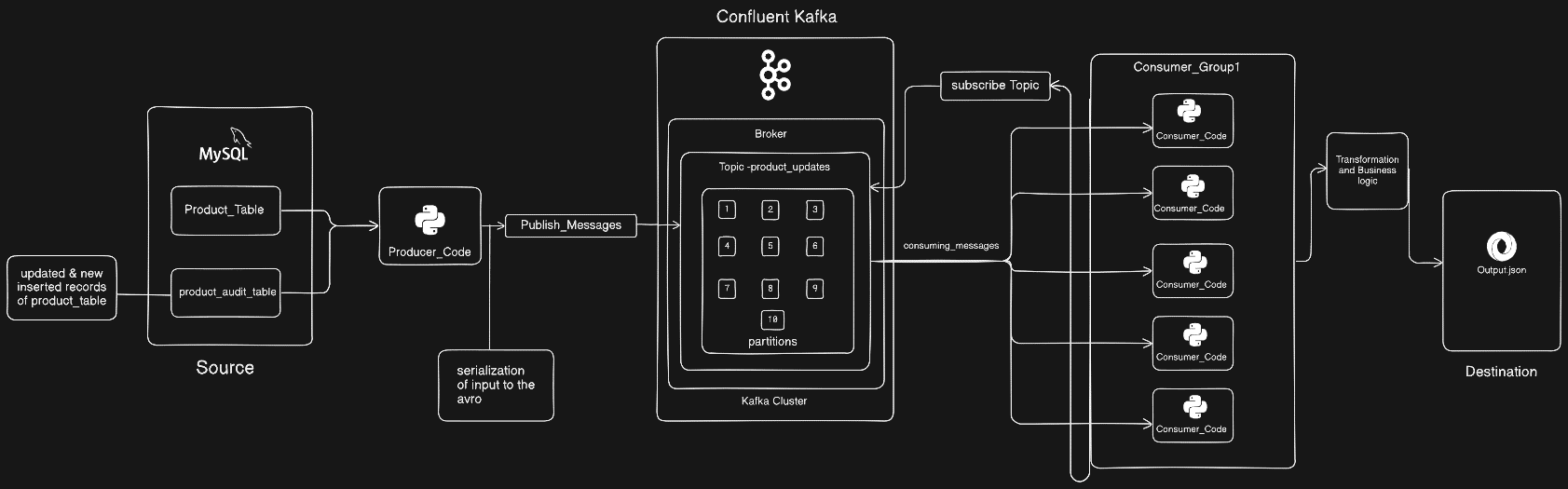
### **Objective**

In this assignment, you will build a Kafka producer and a consumer group that work with a MySQL database, Avro serialization, and multi-partition Kafka topics. The producer will fetch incremental data from a MySQL table and write Avro serialized data into a Kafka topic. The consumers will deserialize this data and append it to separate JSON files.

### **Tools Required**

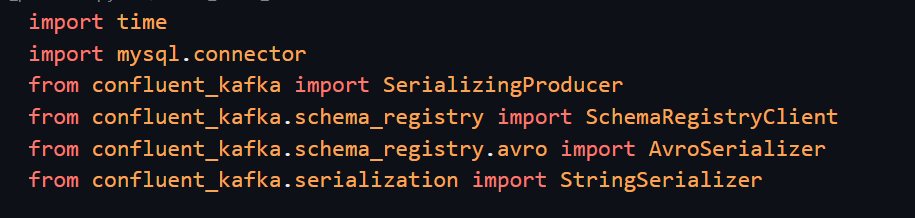
* Python 3.7 or later
* Confluent Kafka Python client
* MySQL Database
* Apache Avro
* A suitable IDE for coding (e.g., PyCharm, Visual Studio Code)

## **System Architecture**



Based on the above architecture diagram, we can see that initially, we had two tables in MySQL: `product\_table` and `product\_audit table`. The `product\_table` is the main table where the company stores information about products, while the `product\_audit\_table` is an additional table created to log new insertions and updates that occur in the main product table, which will serve as our source. Using our producer code, we will fetch all the records initially present in the main `product\_table`. Additionally, we will retrieve any new inserted records and updates from the `product\_audit\_table`. By using the MySQL connector library, we connect MySQL with Python and write queries to fetch these records. We then use the Avro serialization library to serialize the fetched records. We use Avro serialization because it provides schema evolution and efficient memory handling by converting data to binary format. Next, we connect to the Confluent Kafka topic (`product\_updates` with 10 partitions) using the Confluent Kafka library, define our schema, and connect using the Schema Registry client. We then publish these messages to the topic. In the consumer code, we subscribe to that topic by running five instances of consumers from the same consumer group to parallelize the process. In the consumer code, we define some transformations (e.g., price discounts for certain prices) before dumping the consumed records to a JSON file, which is our destination.

## **Producer Code Overview**



First, we import all the required libraries:

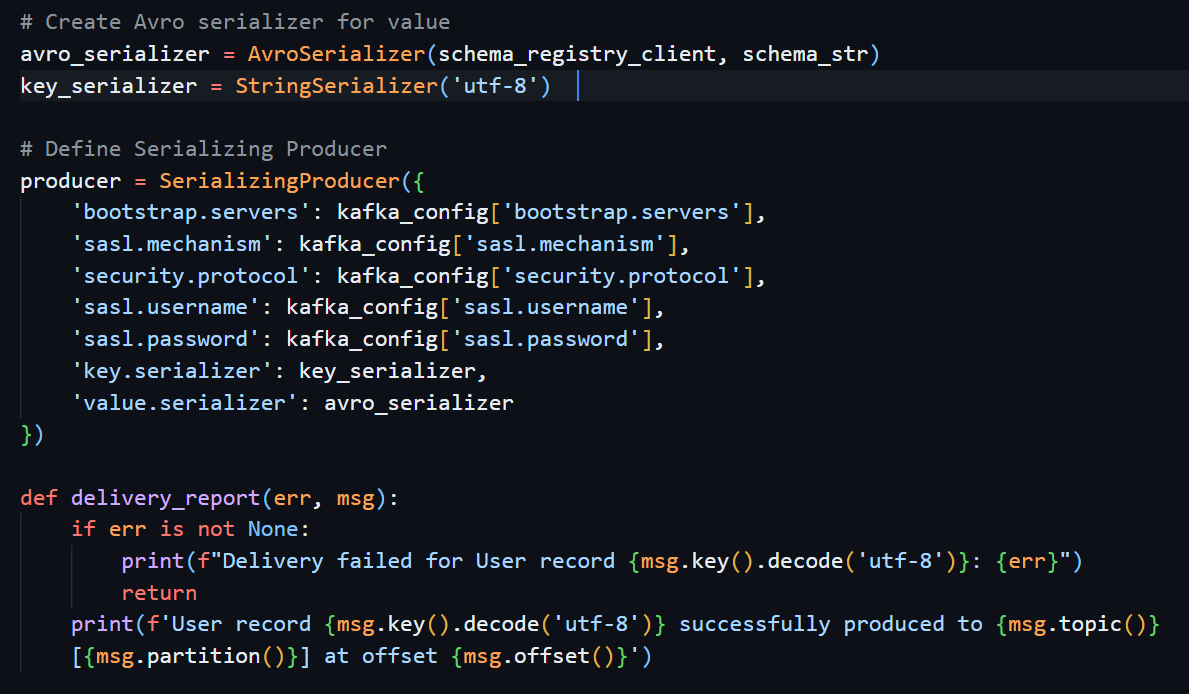
1. **Time** module to handle timestamp data.
2. **Mysql.connector** to connect python with MySQL.
3. **SerializingProducer** as we are serializing our input data before publishing to the topic.
4. **SchemaRegistryClient** to connect our confluent kafka schema we created.
5. **AvroSerializer** as we are using avroserialization for the input data.
6. **StringSerializer** to connect or serialize our key of the schema.

In the next step:



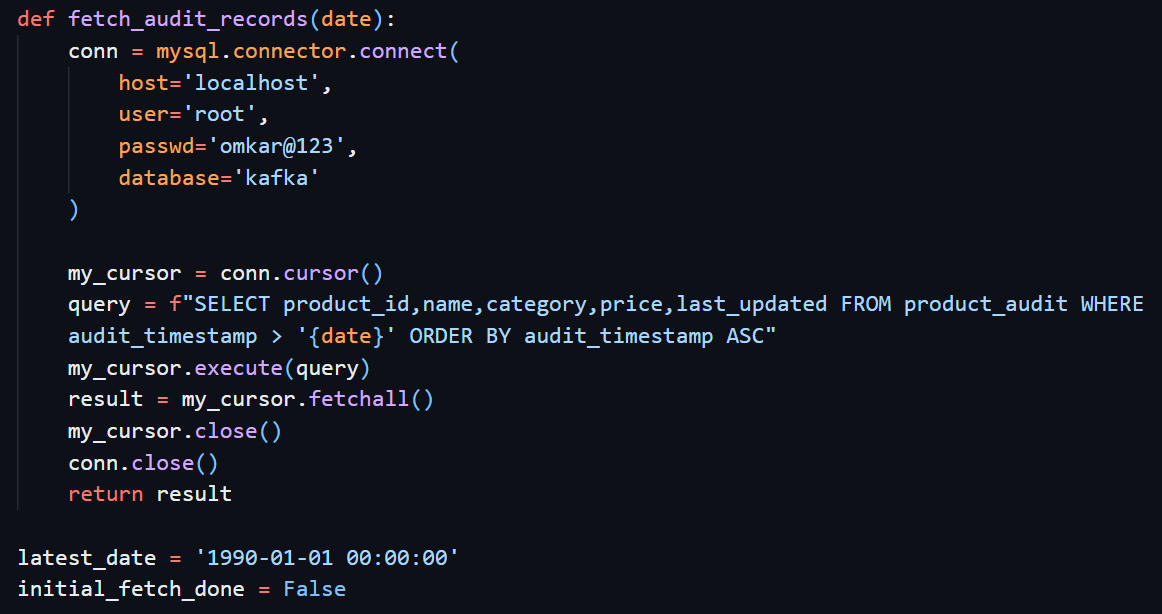
We define our all the required kafka\_configurations in the dictionary which will be required by producer class.

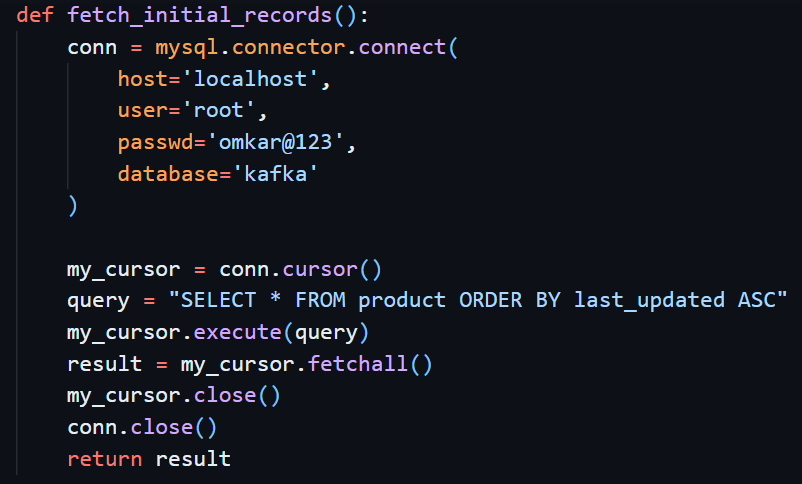
After that we create the object of SchemaRegistryClient and in that we define our confluent kafka endpoint url and api\_key and api\_secret key to connect to our schema. Then we give name of our subject (i.e. schema for value of our messages) after that we store our schema in string format in schema\_str variable. After that we create the avro\_serializer and key\_serializer variable which we will be using in our serializing producer object, we also create the function which will show the error or successful publish message on the console window on the successful publish we will se topic name partition name and offset to which message get published.



Then create two functions:

1. **Fetch\_initial\_records**: which will connect to the MySQL and then select all the records from the product table initially (i.e. before new product insert and update).
2. **Fetch\_audit\_records**: which will again connect to the MySQL and then select similar columns from audit table which contain all the newly inserted or updated records.





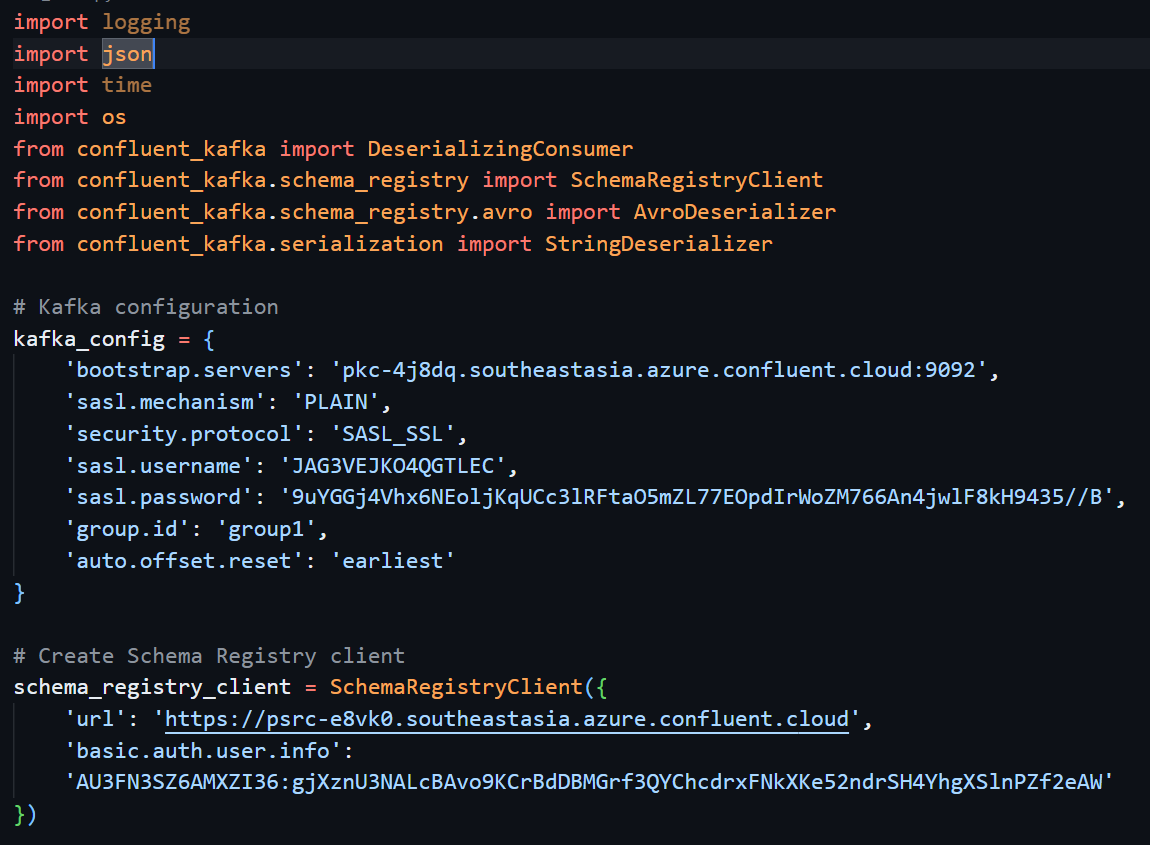
After creating two functions we initialize the latest\_date variable to default date ‘1990-01-01 00:00:00’ which will get updated in the loop and set the initial\_fetch\_done variable (i.e. flag variable) whose initial value we set to false once we fetch all the records from the product table then we will set it to True so that once we load all the data which is initially there in the product table we will go for the product\_audit table for the newly inserted and updated data.



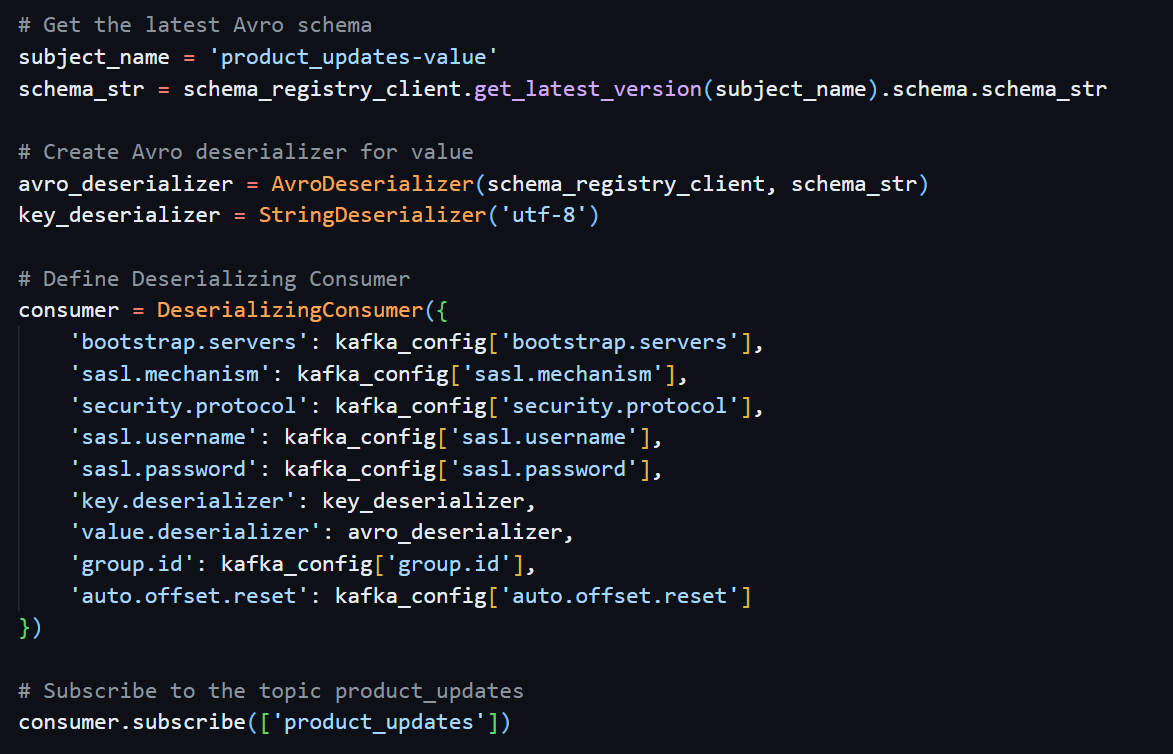
We used try catch block to handle exception and keyboard interruption after getting all the rows from the table we use for loop to loop over the tuples then store the information in dictionary for each iteration so that at every step we will send the data to the producer object where our product\_id will be the key. After each record we use producer.flush() to publish the message to the topic.

## **Consumer Code Overview**

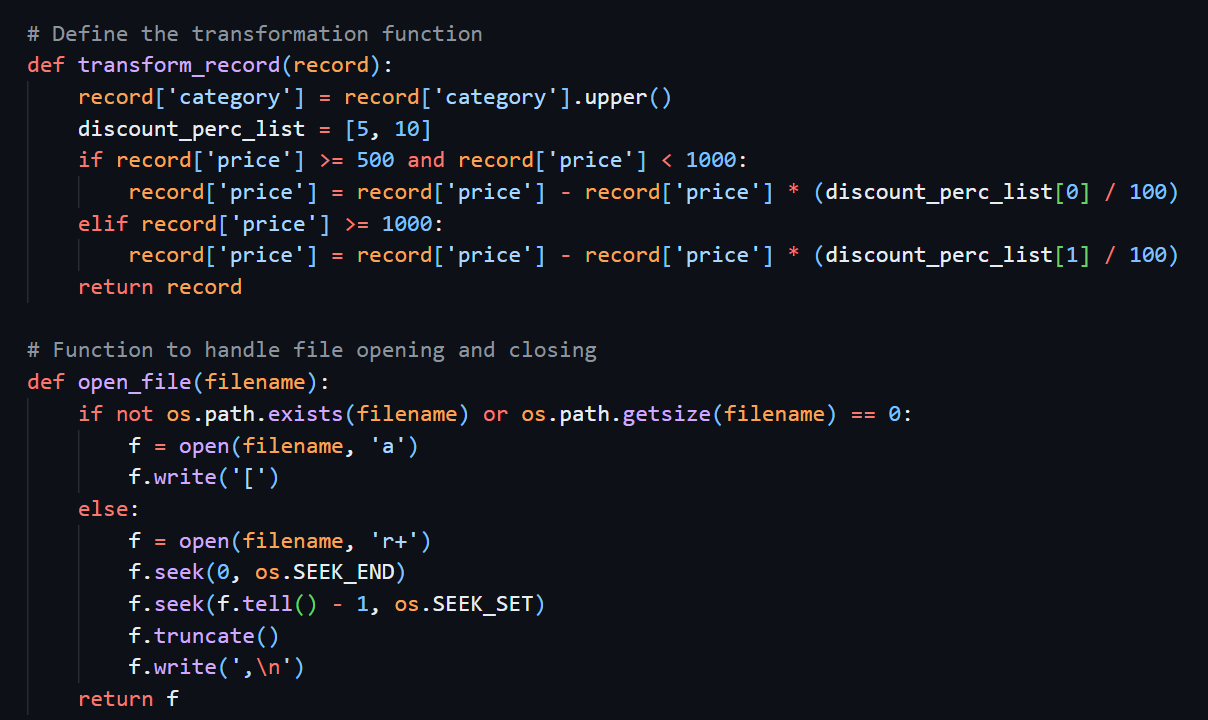
In the consumer code first step is same as the producer code we import all the important library. The only difference in here is that at time of consuming the message we want the raw format which is human readable as at time of publishing the message we serialize the data in **Avro** so in the same way we need **AvroDeserializer** to deserialize the value and **StringDeserializer** to deserialize the key and as we are deserializing is there we need to import **DeserializingConsumer** rather than just normal consumer. Then like producer code we define configuration for the consumer object which we will be creating only thing added here is we need to provide group.id and **auto.offset.Reset** stratergy (in this case we set it to ‘earliest’)



Then we create the schema\_registry\_client and avro\_deserializer for values and key\_deserializer for key then we create the object of the consumer and define our configuration which we created earlier.



After creating consumer object we use subscribe method to subscribe to the topic which we created.



Then we create transformation function called **transform\_record** which will do all the transformation on the records which we send to this function as argument. In this function after receiving records we basically do two transformations.

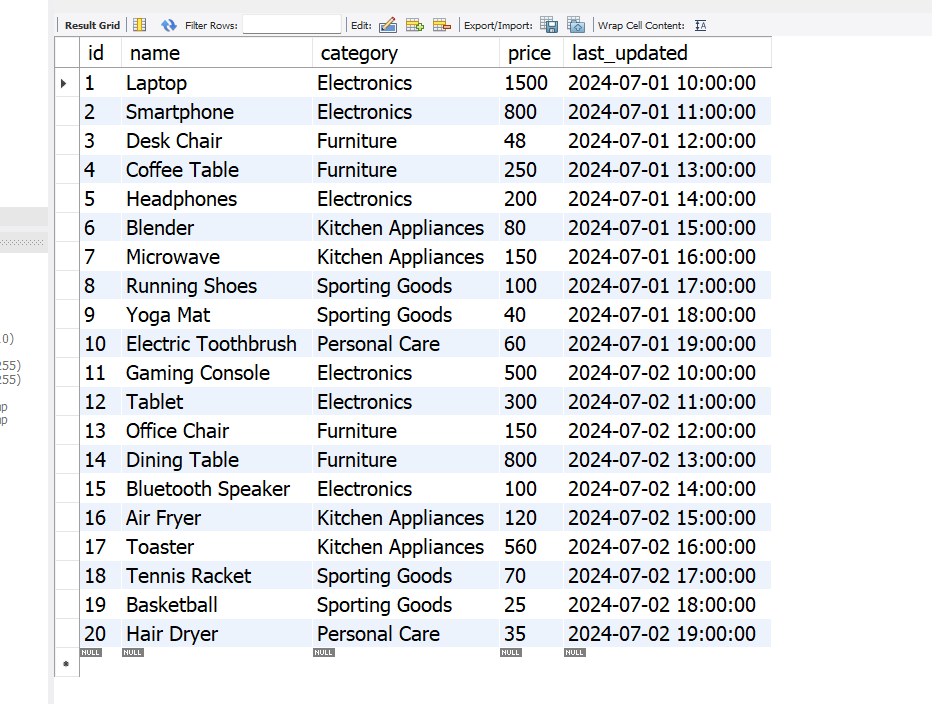
1. Capitalize all the values of the category column.
2. If certain product price lies between 500 and less than 1000 then we will update the price of product with discounted price (i.e. 5% discount) and for the products whose price are above 1000 then we will give 10% discount.

After transformation we will return those records. Then we define another function which will handle file opening (i.e. **open\_file)** which will accept filename as parameter. In this function if file is not created or file has zero content in it we will open the file in the append mode and write open square bracket at start and return file object. If file is already there then we open the file in r+ (i.e. read and write) mode and move the file reference pointer to the end then using f.tell()-1 move the pointer just before the end and then use truncate to remove all the content after that position of the pointer and then write “,” and “\n” character to start the new content on the new line. Then in try catch block we will use **poll** method on the consumer block to fetch messages from the topic If we encounter error then we will print it to the console and if there is no error then we will first use transformation function pass our consumed records to that function after getting transformed record we will use json.dumps() method to convert message to the string format then write those string value in the file context which we open earlier after writing to the file we will show message to the console of written records then in finally block once all the records consumed and consumers stop it will written data to the outpu.json file.



## **Testing**

First create the table in the mysql product table with given schem and then create product\_audit table to log the changes in the product table then create the trigger on product table for insert and update then insert some sample records in the product table.



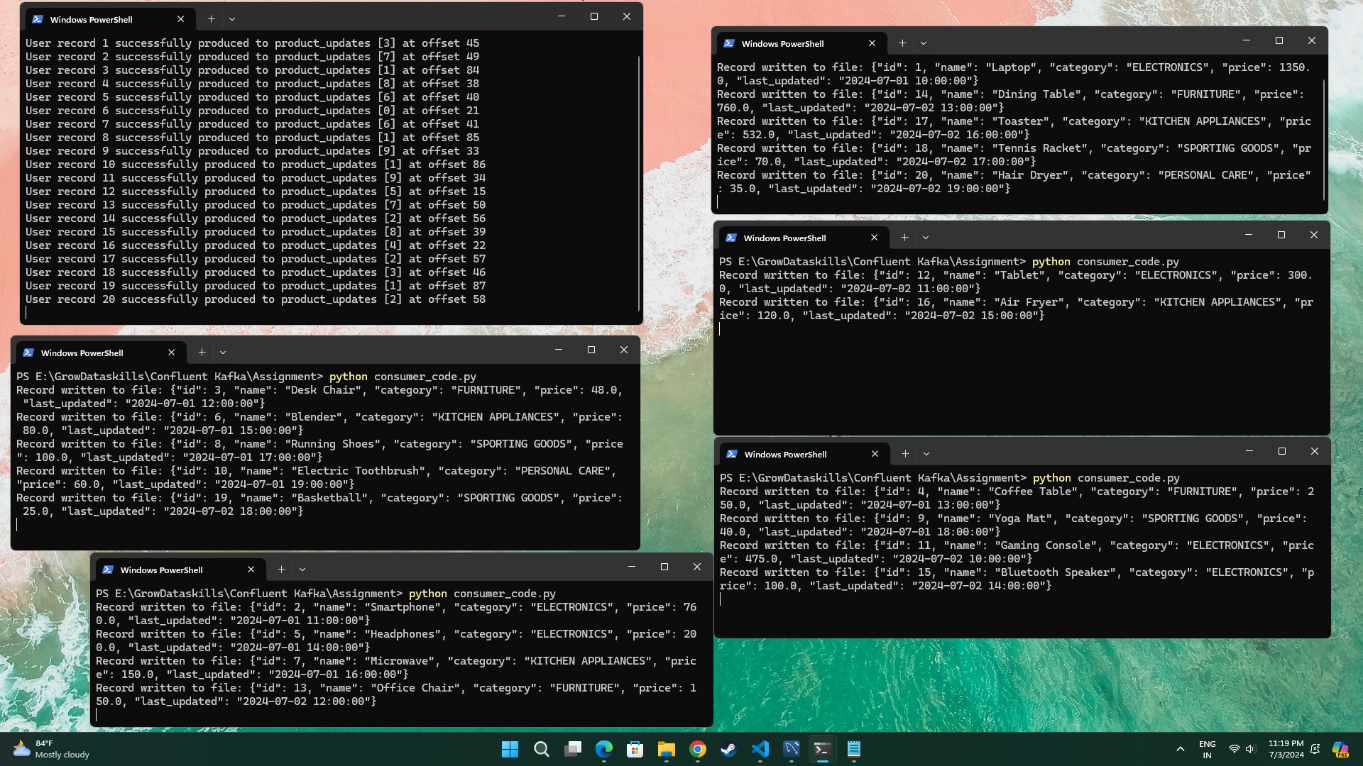
As you can see there are 20 records initially with their last\_updated timestamp

Now we will run our consumers with auto. offset. Reset stratergy as “earliest” and then run the 5 instance of same group (i.e. group1) consumers to process data parallelly after turning on consumers we will run producer so that it will start publish the data and simultaneously consumer start consuming the data.

*Consumer 1*

*Producer*





*Consumer 5*

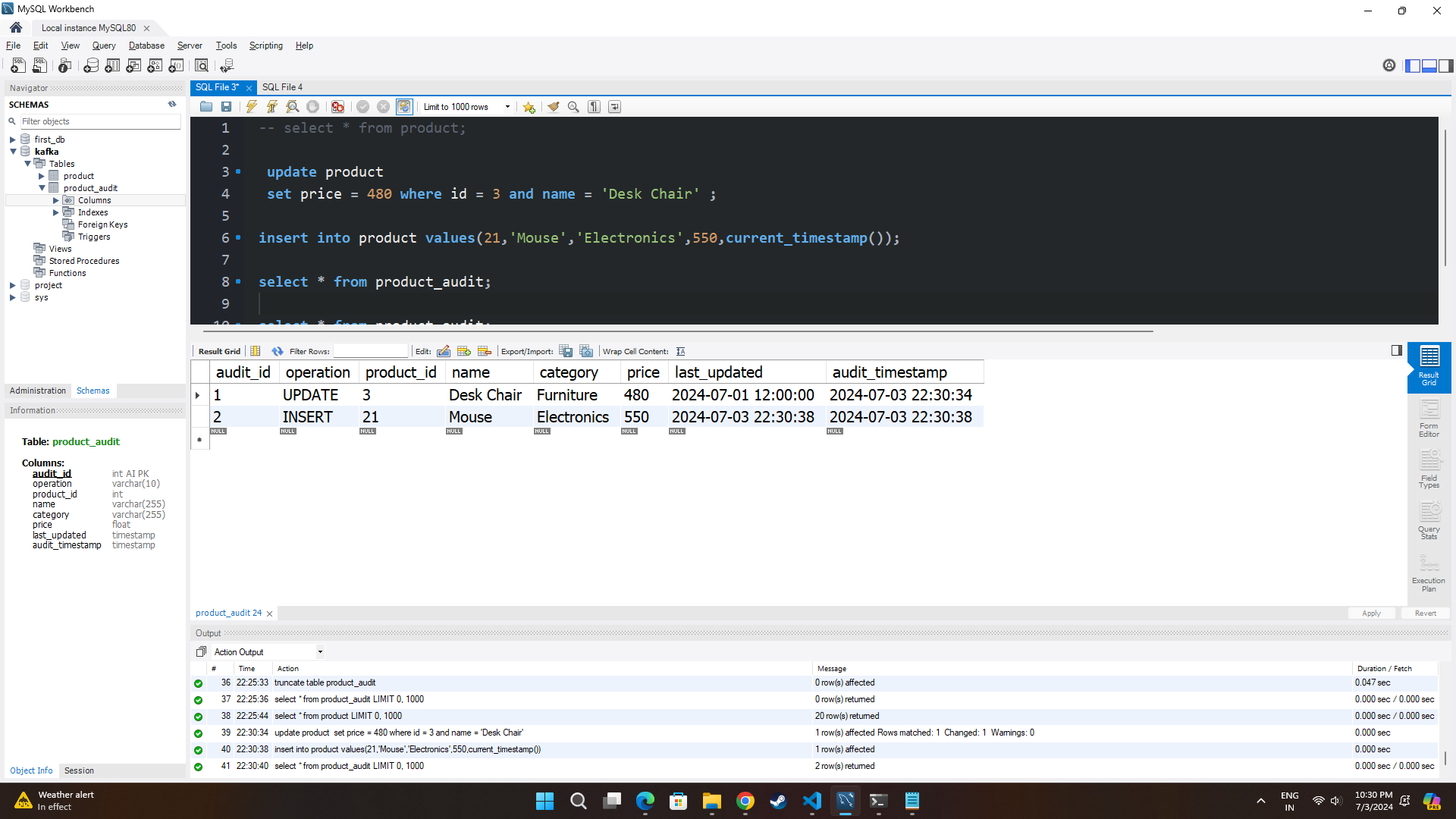
*Consumer 4*

*Consumer 3*

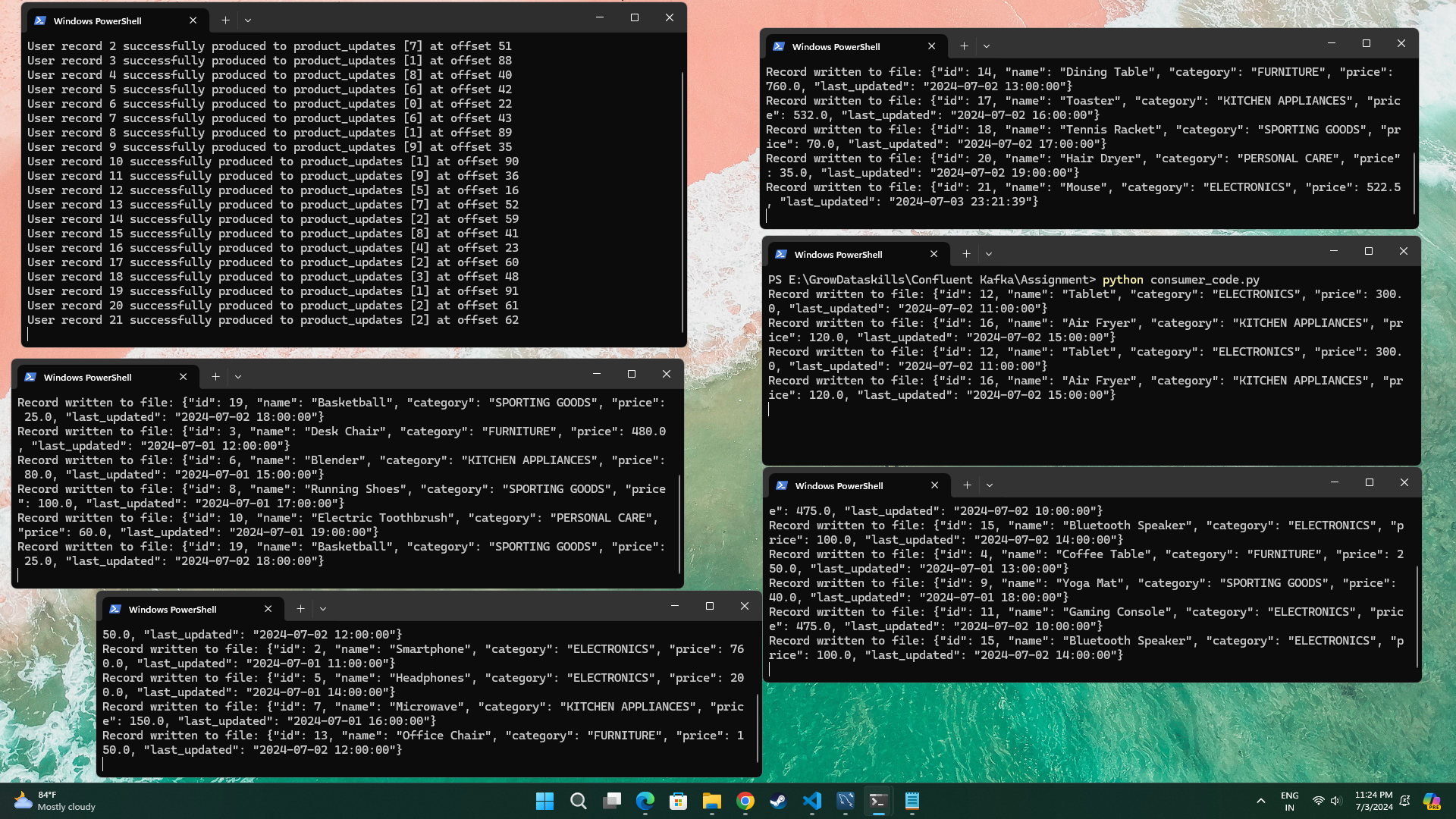
*Consumer 2*



After producer publish all the records of our MySQL table then we can insert new record and update existing record to check whether these changes are getting captured or not. In our case we update the price of product\_id =3 from 48 to 480 and insert new record of product id = 21 they both get reflected in our product\_audit table once we do changes in the main product table due to triggers, we apply. We can see their audit\_timestamps.



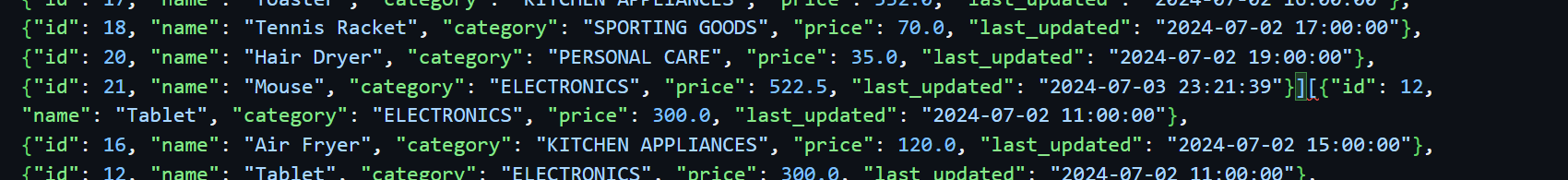
After doing changes we can see that the results from product\_audit table immediately get produce by the producer.



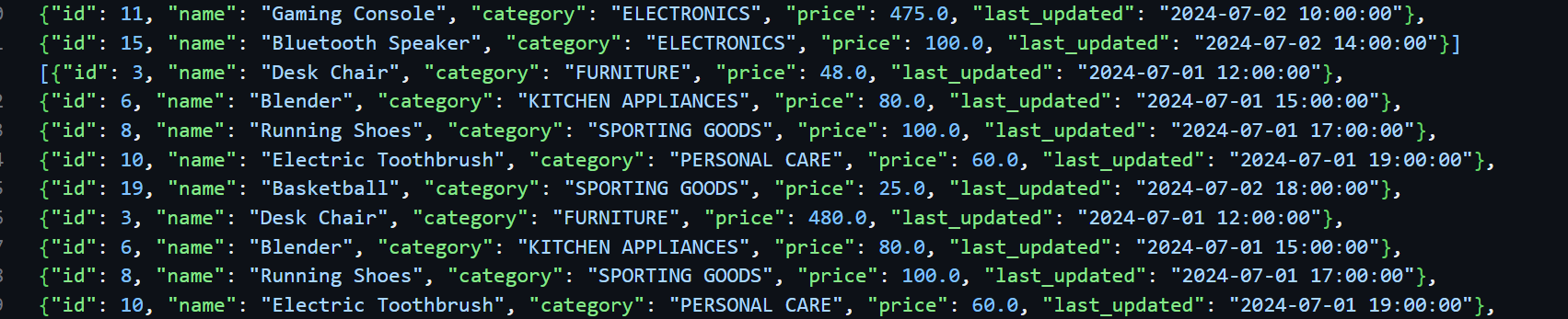
*New inserted record*



You can see from below image in the output.json file product with id 21 gets appended.



And also our updated record get’s added into output Json file from below image you can see that for product id =3 there are two records with different price and timestamp.



## **Challenges Faced**

1. Creating logic to load increment data from MySQL to python as we want to build the system which will not only load the data from MySQL table but also capture the changes that will be happening in the table (e.g. new product added or change in the information of existing product) so we need to come up with logic so thought about create another table which will log the changes and from that table I can get changes.
2. During producer code preparation once, source is ready need to think logic by which we need to extract the data from MySQL table and get the result in required dictionary format for the value. And format the timestamp values from MySQL table so that in next iteration it will get use in query.
3. Most difficult part was to code the consumer code as we need to transform the data and handle the output file (i.e. Json file) then we need to write the transform data to the output Json for each iteration and maintain the format.so for that understand how file handling works spend some time with file handling. But still not getting output in the required format so take some help from ChatGPT.

## **Conclusion**

In this project, we successfully implemented a real-time data streaming pipeline using Confluent Kafka. We built a Kafka producer that fetches incremental data from a MySQL database and writes Avro-serialized data into a Kafka topic. A group of Kafka consumers was developed to deserialize this data and store it in JSON files.

The main components of this project included:

* **Kafka Producer:** Responsible for reading data from the MySQL table and publishing it to the Kafka topic in Avro format.
* **Kafka Consumer Group:** Handles the deserialization of Avro data and transformation before appending it to JSON files.
* **Transformation Logic:** Applied specific business logic to modify the data, such as adjusting prices based on predefined discount percentages.

### **Future Enhancements**

1. **Scalability Improvements:** Implementing advanced techniques for scaling Kafka consumers.
2. **Monitoring and Alerts:** Setting up comprehensive monitoring and alert systems for better observability.
3. **Advanced Transformations:** Adding more complex data transformation logic as per business needs.

For the complete code and further details, please visit the GitHub repository: [GitHub Repository.](https://github.com/omkardesai98/Real-Time-Data-Processing-with-Confluent-Kafka-MySQL-and-Avro)

This project demonstrates the practical application of Kafka in real-time data processing scenarios, showcasing its capability to handle large-scale data streams efficiently. It lays the foundation for building more complex and scalable data processing systems in the future.