**ЛАБОРАТОРНАЯ РАБОТА №10**

РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ СООБЩЕНИЙ

С ИСПОЛЬЗОВАНИЕМ APACHE KAFKA

**1. Установка и настройка Kafka**

• Запустите ZooKeeper и Kafka-брокер (локально или через Docker).

• Создайте топик user\_actions с 3 партициями.

**2. Разработка продюсера**

• Напишите приложение (Java/Python), отправляющее JSON-сообщения в топик. Пример сообщения:

{"user\_id": 101, "action": "purchase", "timestamp": "2023-10-01T12:00:00"}

• Реализуйте ввод сообщений через консоль или чтение из файла.**3. Разработка консьюмера**

• Напишите консьюмер, обрабатывающий сообщения из топика:

Фильтрация по типу действия (например, "purchase").

Вывод статистики в консоль (количество сообщений, частые действия).

**4. Обработка ошибок и масштабирование**

• Настройте Dead Letter Topic (DLT) для некорректных сообщений.

• Запустите 2 консьюмера в одной группе. Продемонстрируйте

распределение сообщений между ними.

**5. Интеграция с внешней системой**

• Сохраняйте данные в PostgreSQL/MySQL.

Был создан файл Producer.py для отправки сообщений в Apache Kafka. Он принимает пользовательский ввод, формирует JSON-объект с user\_id, action и timestamp, и отправляет его в Kafka-топик user\_actions. Используется для логирования и обработки событий в реальном времени

from confluent\_kafka import Producer

import json

import time

from datetime import datetime

import sys

def delivery\_report(err, msg):

if err is not None:

print(f'Message delivery failed: {err}')

else:

print(f'Message delivered to {msg.topic()} [{msg.partition()}]')

def main():

conf = {'bootstrap.servers': 'localhost:9092'}

producer = Producer(conf)

try:

while True:

# Чтение ввода пользователя

user\_input = input("Введите сообщение (user\_id,action) или 'exit' для выхода: ")

if user\_input.lower() == 'exit':

break

try:

user\_id, action = user\_input.split(',')

message = {

'user\_id': int(user\_id),

'action': action.strip(),

'timestamp': datetime.now().isoformat()

}

# Отправка сообщения

producer.produce(

'user\_actions',

key=str(message['user\_id']),

value=json.dumps(message),

callback=delivery\_report

)

producer.flush()

except ValueError:

print("Неверный формат. Используйте: user\_id,action")

except KeyboardInterrupt:

print("\nProducer остановлен")

if \_\_name\_\_ == '\_\_main\_\_':

main()

Был создан файл **Consumer.py** для обработки сообщений из **Apache Kafka**. Он читает события из топика user\_actions, сохраняет их в **PostgreSQL**, отслеживает ошибки и отправляет проблемные сообщения в **Dead Letter Topic (DLT)**. Используется для **анализа пользовательских действий и управления ошибками**.

from confluent\_kafka import Consumer, KafkaException, Producer

import json

from collections import defaultdict

import psycopg2

import sys

from datetime import datetime

producer\_conf = {'bootstrap.servers': 'localhost:9092'}

error\_producer = Producer(producer\_conf)

def get\_db\_connection():

"""Установка соединения с PostgreSQL"""

return psycopg2.connect(

host="localhost",

database="kafka\_events",

user="postgres",

password="1111",

port="5432"

)

def delivery\_report(err, msg):

"""Callback для отслеживания доставки сообщений в DLT"""

if err is not None:

print(f'Message delivery failed: {err}')

else:

print(f'Message delivered to {msg.topic()} [{msg.partition()}]')

def save\_to\_db(data):

"""Сохранение данных в PostgreSQL"""

try:

conn = get\_db\_connection()

cur = conn.cursor()

cur.execute(

"""INSERT INTO user\_actions

(user\_id, action, timestamp)

VALUES (%s, %s, %s)""",

(data['user\_id'], data['action'], data['timestamp'])

)

conn.commit()

print(f"Сохранено в БД: {data}")

except Exception as e:

print(f"Ошибка при сохранении в БД: {e}")

# Отправляем в Dead Letter Topic

error\_producer.produce(

'user\_actions\_dlt',

key=str(data.get('user\_id', 'unknown')),

value=json.dumps({

"original\_data": data,

"error": str(e),

"timestamp": datetime.now().isoformat()

}),

headers={'error\_type': 'db\_error'},

callback=delivery\_report

)

error\_producer.flush()

finally:

if 'cur' in locals(): cur.close()

if 'conn' in locals(): conn.close()

def main():

conf = {

'bootstrap.servers': 'localhost:9092',

'group.id': 'user\_actions\_group',

'auto.offset.reset': 'earliest',

'enable.auto.commit': False

}

consumer = Consumer(conf)

consumer.subscribe(['user\_actions'])

stats = {

'total\_messages': 0,

'actions\_count': defaultdict(int),

'users\_count': defaultdict(int)

}

try:

while True:

msg = consumer.poll(1.0)

if msg is None:

continue

if msg.error():

if msg.error().code() == KafkaException.\_PARTITION\_EOF:

continue

else:

raise KafkaException(msg.error())

try:

data = json.loads(msg.value())

stats['total\_messages'] += 1

stats['actions\_count'][data['action']] += 1

stats['users\_count'][data['user\_id']] += 1

if data['action'] == 'purchase':

print(f" Purchase detected: User {data['user\_id']}")

save\_to\_db(data)

if stats['total\_messages'] % 10 == 0:

print("\n=== Статистика ===")

print(f"Всего обработано: {stats['total\_messages']}")

print("Топ действий:", dict(sorted(

stats['actions\_count'].items(),

key=lambda x: x[1],

reverse=True

)[:3]))

print("Активные пользователи:", dict(sorted(

stats['users\_count'].items(),

key=lambda x: x[1],

reverse=True

)[:3]))

print("=================\n")

consumer.commit(asynchronous=False)

except json.JSONDecodeError as e:

print(f" Ошибка декодирования JSON: {msg.value()}")

error\_producer.produce(

'user\_actions\_dlt',

value=msg.value(),

headers={'error\_type': 'json\_decode'}

)

error\_producer.flush()

except KeyboardInterrupt:

print("\nConsumer остановлен")

finally:

consumer.close()

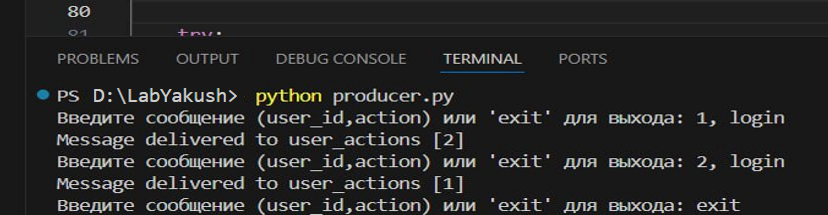
error\_producer.flush()

print("Ресурсы освобождены")

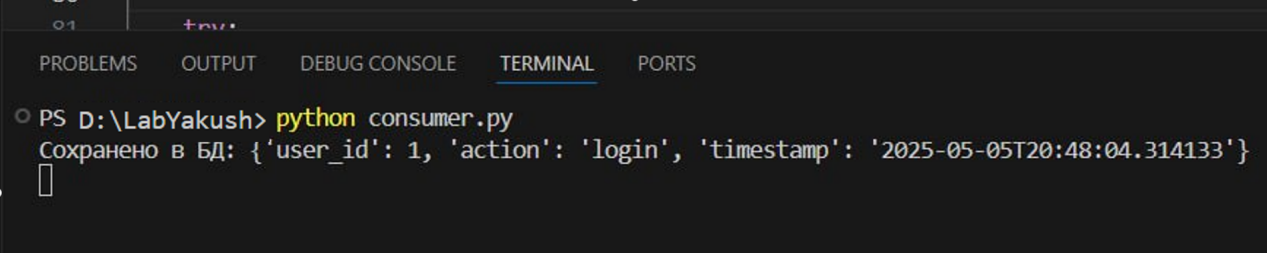
if \_\_name\_\_ == '\_\_main\_\_':

main()

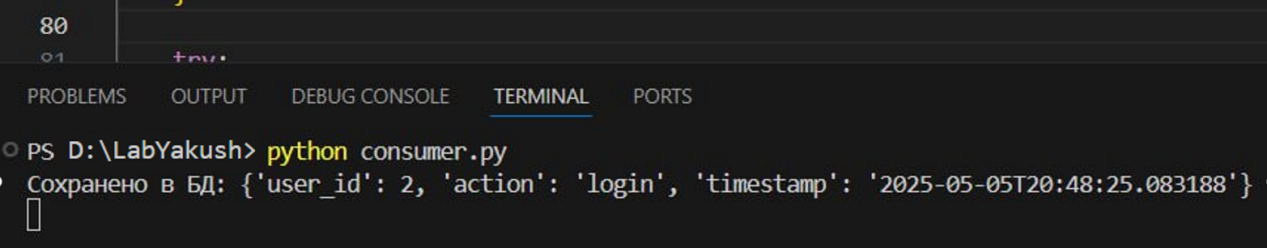
**Далее были отправлены два сообщения**



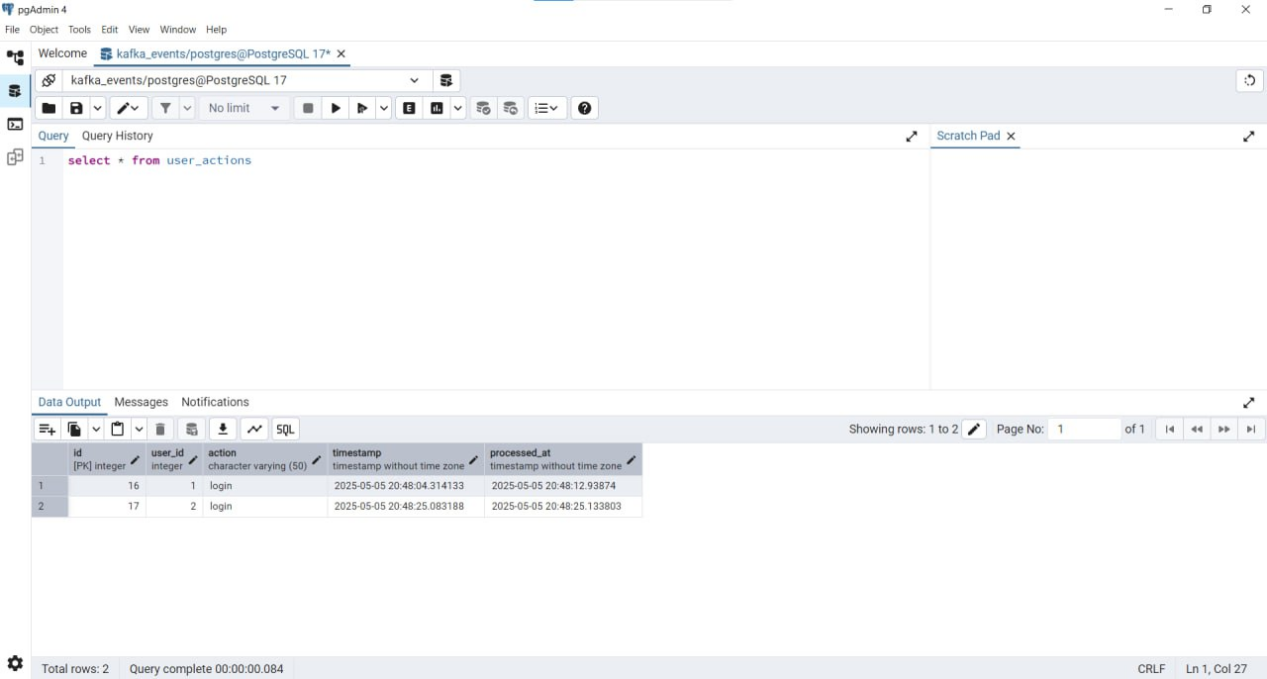
Было отправлено сообщение User1



Было отправлено сообщение User2



База данных выглядит следующим образом:



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. **Зачем Kafka использует ZooKeeper?**

Kafka использует ZooKeeper для управления метаданными кластера, отслеживания состояния брокеров, распределения партиций и избрания лидера. ZooKeeper играет ключевую роль в обеспечении согласованности и отказоустойчивости системы.

1. Как обеспечить порядок сообщений в партиции?

Порядок сообщений в партиции обеспечивается назначением ключа партиционирования при отправке сообщений. Каждый продюсер отправляет сообщения в определённую партицию, а Kafka гарантирует порядок их обработки за счёт последовательности записи. При чтении консьюмеры читают сообщения в том же порядке, в котором они были записаны.

1. **В чём разница между auto.offset.reset=earliest и latest?**

Параметр auto.offset.reset=earliest позволяет консьюмеру начинать чтение с **самого старого** доступного сообщения в партиции, если его предыдущая позиция не найдена. latest, наоборот, заставляет консьюмер начинать с **новых сообщений**, игнорируя старые данные.

1. **Как Kafka обеспечивает отказоустойчивость?**

Kafka обеспечивает отказоустойчивость благодаря **репликации партиций**. Каждая партиция может иметь несколько реплик, а один из брокеров выступает в роли **лидера**, который принимает записи. При сбое лидера ZooKeeper автоматически выбирает новый лидер из реплик. Кроме того, **логическая обработка offset'ов** позволяет консьюмерам восстанавливать свою позицию чтения после сбоев. За счёт горизонтального масштабирования Kafka легко добавляет новые брокеры, сохраняя высокую производительность.