**ЛАБОРАТОРНАЯ РАБОТА №10**

**РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ СООБЩЕНИЙ**

**С ИСПОЛЬЗОВАНИЕМ APACHE KAFKA**

**Цель работы:** Разработка системы асинхронной обработки сообщений с использованием Apache Kafka, включая создание продюсеров, консьюмеров и настройку их взаимодействия.

**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.

Файл Producer.py разработан для передачи сообщений в Apache Kafka. Он получает данные от пользователя, создаёт JSON-объект с параметрами user\_id, action и timestamp, а затем отправляет его в Kafka-топик user\_actions:

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()

Файл Consumer.py разработан для обработки сообщений из Apache Kafka. Он получает события из топика user\_actions, записывает их в базу данных PostgreSQL, отслеживает ошибки и перенаправляет проблемные сообщения в Dead Letter Topic (DLT):

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()

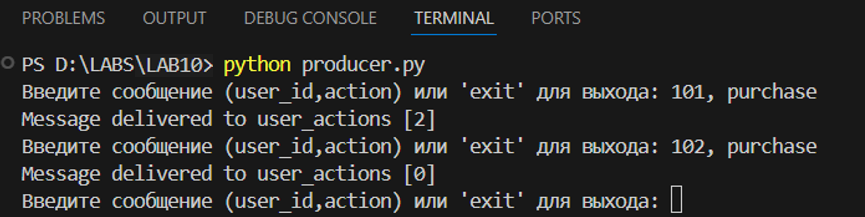
error\_producer.flush()

print("Ресурсы освобождены")

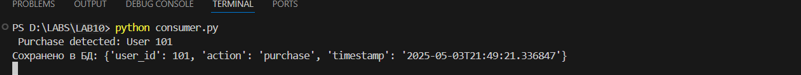
if \_\_name\_\_ == '\_\_main\_\_':

main()

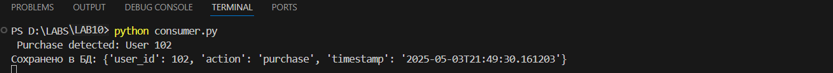
Произведён запуск producer.py и двух экземпляров consumer.py.. В систему отправлены два сообщения: пользователь 101 и пользователь 102.



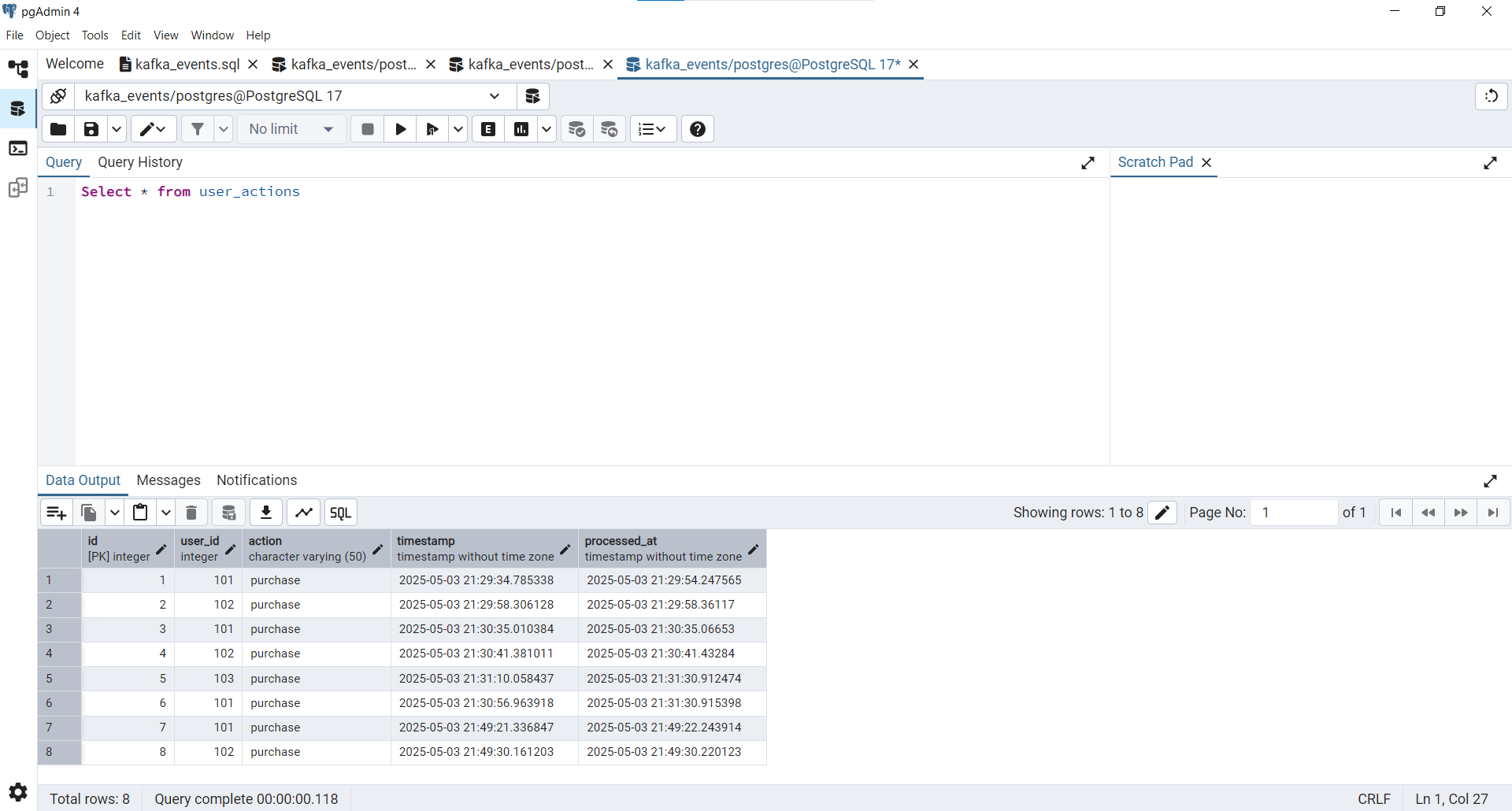
Отправлено сообщение пользователю 1



Отправлено сообщение пользователю 2



База данных:



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Зачем Kafka использует ZooKeeper?

Kafka использует ZooKeeper для управления распределённой системой. ZooKeeper отвечает за координацию брокеров Kafka, отслеживание их состояния, управление лидерами партиций и поддержание отказоустойчивости кластера. Он также помогает в обнаружении новых брокеров, распределении данных и балансировке нагрузки.

1. Как обеспечить порядок сообщений в партиции?

Порядок сообщений в партиции гарантируется за счёт того, что Kafka назначает каждому сообщению уникальный offset. В рамках одной партиции Kafka сохраняет строгий порядок поступления сообщений, а обработка их потребителями выполняется последовательно. Чтобы сохранить порядок при работе нескольких продюсеров, используется один ключ для сообщений (например, key в producer API), что позволяет направлять их в одну и ту же партицию.

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

Параметр auto.offset.reset определяет, с какого места потребитель начнёт чтение сообщений, если его offset отсутствует или не задан.

* earliest — потребитель начнёт чтение с самого первого доступного сообщения в партиции.
* latest — потребитель начнёт чтение с самого последнего записанного сообщения, пропуская предыдущие.

4. Как Kafka обеспечивает отказоустойчивость?

Kafka достигает отказоустойчивости за счёт нескольких механизмов:

* Репликация партиций — данные каждой партиции дублируются на нескольких брокерах, и если один выходит из строя, другая копия остаётся доступной.
* Лидерство партиций — в каждой партиции есть ведущий (leader), который отвечает за чтение и запись данных. В случае сбоя ZooKeeper назначает нового лидера.
* Децентрализованная архитектура — распределённые брокеры работают независимо, снижая нагрузку на отдельные узлы.
* Журнал событий (commit log) — данные записываются в лог и могут быть восстановлены при перезапуске брокера.