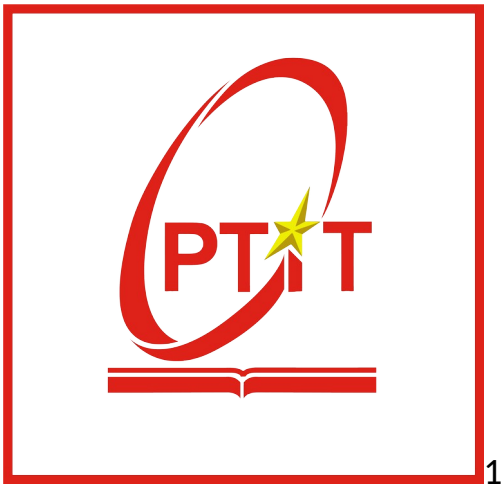
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



1

Báo cáo hàng tuần

Môn học: Thực tập cơ sở

Giảng viên: Kim Ngọc Bách

Họ và tên: Nguyễn Hữu Phúc

Mã SV: B22DCAT224

Lớp: E22CQCN04-B

Báo cáo tuần 7

I. Sửa lại code Python để tiết kiệm RAM trong quá trình chạy

File database_config.py

```
import os
from dataclasses import dataclass


from dotenv import load_dotenv
from typing import Dict




class DatabaseConfig():
  def validate(self) -> None:
      for key, value in self.__dict__.items():
          if value is None:
              raise ValueError(f"----------Missing config for
{key}-------------")


@dataclass
class MongoDBConfig(DatabaseConfig):
  uri : str
  db_name : str


def get_database_config() -> Dict[str,DatabaseConfig]:
  load_dotenv()
  config = {
      "mongodb" :  MongoDBConfig(
          uri = os.getenv("MONGO_URI"),
          db_name = os.getenv("MONGO_DB_NAME")
      )
  }


  for db, setting in config.items():
      print(type(setting))
      setting.validate()


  return config
```

```
db_config = get_database_config()
print(db_config)
```

File mongodb_connect.py

```
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure
from config.database_config import get_database_config
from database.schema_manager import create_mongodb_schema,
validate_mongodb_schema
import psutil


class MongoDBConnect:
    def __init__(self, mongo_uri, db_name):
        self.mongo_uri = mongo_uri
        self.db_name = db_name
        self.client = None
        self.db = None

    def connect(self):
        try:
            self.client = MongoClient(self.mongo_uri)
            self.client.server_info()  # Test connection
            self.db = self.client[self.db_name]
            print(f"Connected to MongoDB: {self.db_name}")
            return self.db
        except ConnectionFailure as e:
            raise Exception(f"Failed to connect to MongoDB:
{e}") from e

    def close(self):
        if self.client:
            self.client.close()
            print("MongoDB connection closed")

    def __enter__(self):
        self.connect()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close()
```

```python
def add_id_to_records(db):
    """Add incremental id field (1, 2, 3, ...) to all records
in the summary collection."""
    index = 1  # Start id from 1
    records_processed = 0

    print(f"Starting to add id field to all records...")
    print(f"Initial memory usage:
{psutil.Process().memory_info().rss / 1024 / 1024} MB")

    # Use cursor with small batch_size to reduce RAM
    cursor = db.summary.find({}, {"_id": 1}).batch_size(1000)

    try:
        for doc in cursor:
            # Update each record with id
            db.summary.update_one(
                {"_id": doc["_id"]},
                {"$set": {"id": index}}
            )
            records_processed += 1
            index += 1

            # Print progress every 100,000 records
            if records_processed % 100000 == 0:
                print(
                    f"Processed {records_processed} records,
memory usage: {psutil.Process().memory_info().rss / 1024 /
1024} MB")

    finally:
        cursor.close()

    print(f"Total records processed and updated:
{records_processed}")
    print(f"Final memory usage:
{psutil.Process().memory_info().rss / 1024 / 1024} MB")


def query_product_views(db):
    """Query and display 5 documents with id field."""
    projection = {
```

```
        "id": 1,  # Include new id field
        "time_stamp": 1,
        "current_url": 1,
        "referrer_url": 1,
        "collection": 1,
        "product_id": 1,
        "option": 1,
        "_id": 0  # Exclude _id
    }
    cursor = db.summary.find({},
projection).limit(5).batch_size(5)

    documents = []
    try:
        for doc in cursor:
            documents.append(doc)
    finally:
        cursor.close()

    return documents
```

File schema_manager.py

```
from pymongo import MongoClient
from pymongo.errors import CollectionInvalid

def create_mongodb_schema(db):
    if "summary" not in db.list_collection_names():
        try:
            db.create_collection("summary", validator={
                "$jsonSchema": {
                    "bsonType": "object",
                    "required": ["_id", "time_stamp",
"current_url", "collection", "product_id"],
                    "properties": {
                        "_id": {"bsonType": ["objectId",
"string"]},
                        "time_stamp": {"bsonType": ["int",
"long"]},
                        "ip": {"bsonType": ["string", "null"]},
                        "user_agent": {"bsonType": ["string",
"null"]},
                        "resolution": {"bsonType": ["string",
"null"]},
                        "user_id_db": {"bsonType": ["string",
"null"]},
```

```
                        "device_id": {"bsonType": ["string",
"null"]},
                        "api_version": {"bsonType": ["string",
"null"]},
                        "store_id": {"bsonType": ["string",
"null"]},
                        "local_time": {"bsonType": ["string",
"null"]},
                        "show_recommendation": {"bsonType":
["string", "bool", "null"]},
                        "current_url": {"bsonType": "string"},
                        "referrer_url": {"bsonType": ["string",
"null"]},
                        "email_address": {"bsonType":
["string", "null"]},
                        "recommendation": {"bsonType": ["bool",
"null"]},
                        "utm_source": {"bsonType": ["bool",
"string", "null"]},
                        "utm_medium": {"bsonType": ["bool",
"string", "null"]},
                        "collection": {"bsonType": "string"},
                        "product_id": {"bsonType": "string"},
                        "option": {
                            "bsonType": ["array", "null"],
                            "items": {
                                "bsonType": "object",
                                "properties": {
                                    "option_label":
{"bsonType": ["string", "null"]},
                                    "option_id": {"bsonType":
["string", "null"]},
                                    "value_label": {"bsonType":
["string", "null"]},
                                    "value_id": {"bsonType":
["string", "null"]}
                                }
                            }
                        }
                    }
                }
            })
            db.summary.create_index("product_id")
            print("Created summary collection with schema")
```

```
        except CollectionInvalid:
            print("summary collection already exists")
    else:
        print("summary collection already exists, skipping
creation")

def validate_mongodb_schema(db):
    collections = db.list_collection_names()
    print("Collections:", collections)
    if "summary" not in collections:
        raise ValueError("Missing summary collection in
MongoDB")
    if db.summary.find_one() is None:
        print("Warning: summary collection is empty")
    else:
        print("summary collection contains documents")
    print("Validated schema for summary collection")
```

File main.py

```
from database.mongodb_connect import MongoDBConnect,
query_product_views, add_id_to_records
from database.schema_manager import create_mongodb_schema,
validate_mongodb_schema
from config.database_config import get_database_config

def main():
    configMongo = get_database_config()
    with MongoDBConnect(configMongo["mongodb"].uri,
configMongo["mongodb"].db_name) as mongo_client:
        db = mongo_client.db
        # Ensure schema exists and validate
        create_mongodb_schema(db)
        validate_mongodb_schema(db)

        # Add id field to all records
        add_id_to_records(db)

        # Query and display documents
        documents = query_product_views(db)
        print("Retrieved documents:")
        for doc in documents:
            print(doc)

if __name__ == "__main__":
```

```
    main()
```

II. Tích hợp Kafka vào quá trình xử lí

kafka_producer.py

```
from confluent_kafka import Producer
import json
import os
from dotenv import load_dotenv

class KafkaProducer:
    def __init__(self):
        load_dotenv()
        self.bootstrap_servers = os.getenv("KAFKA_BOOTSTRAP_SERVERS",
"localhost:9092")
        self.topic = os.getenv("KAFKA_TOPIC", "product_views")
        self.producer_config = {
            'bootstrap.servers': self.bootstrap_servers,
            'client.id': 'mongodb-to-kafka-producer'
        }
        self.producer = Producer(self.producer_config)

    def delivery_report(self, err, msg):
        """Callback để báo cáo trạng thái gửi message."""
        if err is not None:
            print(f"Message delivery failed: {err}")
        else:
            print(f"Message delivered to {msg.topic()}
[{msg.partition()}]")

    def send_to_kafka(self, data):
        """Gửi dữ liệu tới Kafka topic."""
        try:
            # Chuyển dữ liệu thành JSON string
            data_json = json.dumps(data, default=str)
            self.producer.produce(
                self.topic,
                value=data_json.encode('utf-8'),
                callback=self.delivery_report
            )
            self.producer.flush()
        except Exception as e:
            print(f"Error sending to Kafka: {e}")
```

```
    def close(self):
        """Đóng producer."""
        self.producer.flush()
        print("Kafka producer closed")
```

2. Sửa file mongodb_connect

```
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure
from config.database_config import get_database_config
from database.schema_manager import create_mongodb_schema,
validate_mongodb_schema
import psutil

class MongoDBConnect:
  def __init__(self, mongo_uri, db_name):
      self.mongo_uri = mongo_uri
      self.db_name = db_name
      self.client = None
      self.db = None

  def connect(self):
      try:
          self.client = MongoClient(self.mongo_uri)
          self.client.server_info()  # Test connection
          self.db = self.client[self.db_name]
          print(f"Connected to MongoDB: {self.db_name}")
          return self.db
      except ConnectionFailure as e:
          raise Exception(f"Failed to connect to MongoDB: {e}") from e

  AGRITECH SOLUTIONS LIMITED
  def close(self):
      if self.client:
          self.client.close()
          print("MongoDB connection closed")

  def __enter__(self):
      self.connect()
      return self

  def __exit__(self, exc_type, exc_val, exc_tb):
      self.close()
```

```
def add_id_to_records(db):
  """Add incremental id field (1, 2, 3, ...) to all records in the
summary collection."""
  index = 1  # Start id from 1
  records_processed = 0

  print(f"Starting to add id field to all records...")
  print(f"Initial memory usage: {psutil.Process().memory_info().rss /
1024 / 1024} MB")

  # Use cursor with small batch_size to reduce RAM
  cursor = db.summary.find({}, {"_id": 1}).batch_size(1000)

  try:
      for doc in cursor:
          # Update each record with id
          db.summary.update_one(
              {"_id": doc["_id"]},
              {"$set": {"id": index}}
          )
          records_processed += 1
          index += 1

          # Print progress every 100,000 records
          if records_processed % 100000 == 0:
              print(
                  f"Processed {records_processed} records, memory
usage: {psutil.Process().memory_info().rss / 1024 / 1024} MB")

  finally:
      cursor.close()

  print(f"Total records processed and updated: {records_processed}")
  print(f"Final memory usage: {psutil.Process().memory_info().rss /
1024 / 1024} MB")

def query_product_views(db, batch_size=1000):
  """Query processed documents from summary collection (with id field)
and yield them in batches."""
  projection = {
      "id": 1,  # Include new id field
      "time_stamp": 1,
      "current_url": 1,
```

```
        "referrer_url": 1,
        "collection": 1,
        "product_id": 1,
        "option": 1,
        "_id": 0  # Exclude _id
    }
    # Only query documents that have the id field
    cursor = db.summary.find({"id": {"$exists": True}},
projection).batch_size(batch_size)

    try:
        for doc in cursor:
            yield doc
    finally:
        cursor.close()
```

3. Sửa file main.y

```
from database.mongodb_connect import MongoDBConnect,
query_product_views, add_id_to_records
from database.schema_manager import create_mongodb_schema,
validate_mongodb_schema
from config.database_config import get_database_config
from kafka_producer import KafkaProducer
import psutil

def main():
    configMongo = get_database_config()
    kafka_producer = KafkaProducer()

    try:
        with MongoDBConnect(configMongo["mongodb"].uri,
configMongo["mongodb"].db_name) as mongo_client:
            db = mongo_client.db
            # Ensure schema exists and validate
            create_mongodb_schema(db)
            validate_mongodb_schema(db)

            # Add id field to all records
            add_id_to_records(db)

            # Query processed documents and send to Kafka
            print("Starting to query processed documents and send to
Kafka...")
```

```
        print(f"Initial memory usage:
{psutil.Process().memory_info().rss / 1024 / 1024} MB")
        records_processed = 0

        for doc in query_product_views(db, batch_size=1000):
            print(f"Sending processed document to Kafka: {doc}")
            kafka_producer.send_to_kafka(doc)
            records_processed += 1

            # Print progress every 1000 records
            if records_processed % 1000 == 0:
                print(
                    f"Sent {records_processed} processed records to
Kafka, memory usage: {psutil.Process().memory_info().rss / 1024 / 1024}
MB")

        print(f"Total processed records sent to Kafka:
{records_processed}")
        print(f"Final memory usage:
{psutil.Process().memory_info().rss / 1024 / 1024} MB")

    finally:
        kafka_producer.close()

if __name__ == "__main__":
    main()
```