Лабораторная работа 4-1. Сравнение подходов хранения больших данных

Цель работы: сравнить производительность и эффективность различных подходов к хранению и обработке больших данных на примере реляционной базы данных PostgreSQL и документо-ориентированной базы данных MongoDB.

Оборудование и программное обеспечение

- Компьютер с операционной системой Ubuntu (или любой другой ОС с поддержкой Docker).
- Docker и Docker Compose.
- Python 3.x.
- Jupyter Notebook или JupyterLab.
- Библиотеки Python: psycopg2-binary, pymongo, pandas, matplotlib, sqlalchemy.

  Конфигурация https://github.com/BosenkoTM/BigDataAnalitic/tree/main/lw/2025/lw_4_1

Теоретическая часть

В современном мире объемы данных растут экспоненциально, что приводит к необходимости использования эффективных методов их хранения и обработки.

Существует два основных подхода к хранению больших данных:

1. **Реляционные базы данных (например, PostgreSQL)**. Основаны на реляционной модели данных, где информация хранится в строго структурированных таблицах с заранее определенной схемой. Связи между таблицами устанавливаются с помощью внешних ключей. PostgreSQL является мощной объектно-реляционной СУБД с открытым исходным кодом, поддерживающей ACID-транзакции, сложные запросы и обладающей высокой степенью расширяемости.
2. **NoSQL базы данных (например, MongoDB)**. Представляют собой широкий класс систем управления базами данных, которые отличаются от традиционных реляционных СУБД. MongoDB — это документо-ориентированная СУБД, которая хранит данные в гибких, JSON-подобных документах. Схема данных не является фиксированной, что позволяет легко хранить иерархические и полуструктурированные данные. NoSQL-системы часто отдают предпочтение масштабируемости и производительности в ущерб строгой согласованности данных (в сравнении с реляционными СУБД).

Каждый из этих подходов имеет свои преимущества и недостатки, которые мы исследуем в ходе выполнения лабораторной работы.

Практическая часть

Шаг 1. Подготовка окружения с помощью Docker

Для упрощения развертывания и управления базами данных мы будем использовать Docker Compose. Этот подход позволяет запустить PostgreSQL, pgAdmin, MongoDB и Mongo Express в изолированных контейнерах одной командой.

1.1. Создайте файл docker-compose.yml следующего содержания:

```
version: '3.8'
services:
  mongodb:
    image: mongo:latest
    container_name: mongodb
    environment:
      MONGO_INITDB_ROOT_USERNAME: mongouser
      MONGO_INITDB_ROOT_PASSWORD: mongopass
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    networks:
      - db_network

  mongo-express:
    image: mongo-express:latest
    container_name: mongo-express
    restart: always
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: mongouser
      ME_CONFIG_MONGODB_ADMINPASSWORD: mongopass
      ME_CONFIG_MONGODB_SERVER: mongodb
    ports:
      - "8081:8081"
    depends_on:
      - mongodb
    networks:
      - db_network

  postgresql:
    image: postgres:latest
```

```
    container_name: postgresql
    environment:
      POSTGRES_USER: pguser
      POSTGRES_PASSWORD: pgpass
      POSTGRES_DB: studpg # Сразу создаем базу данных studpg
    ports:
      - "5432:5432"
    volumes:
      - pg_data:/var/lib/postgresql/data
    networks:
      - db_network

  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: pgadmin
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@example.com
      PGADMIN_DEFAULT_PASSWORD: admin
    ports:
      - "5050:80"
    depends_on:
      - postgresql
    networks:
      - db_network

networks:
  db_network:
    driver: bridge

volumes:
  mongo_data:
  pg_data:
```

1.2. В терминале, находясь в директории с файлом docker-compose.yml, выполните команду:

```
sudo docker compose up -d
```

1.3. Установите необходимые библиотеки Python:

```
pip install psycopg2-binary pymongo pandas matplotlib sqlalchemy
```

Шаг 2: Настройка баз данных

После запуска контейнеров у вас будет доступ к следующим сервисам:

- **PostgreSQL**: localhost:5432
- **pgAdmin** (веб-интерфейс для PostgreSQL): http://localhost:5050
- **MongoDB**: localhost:27017
- **Mongo Express** (веб-интерфейс для MongoDB): http://localhost:8081

Для всех вариантов:

1. **PostgreSQL**:
    - База данных studpg уже создана согласно docker-compose.yml.
    - Подключитесь к pgAdmin (Email: admin@example.com, Пароль: admin).
    - Добавьте новый сервер:
        - Host name/address: postgresql
        - Port: 5432
        - Maintenance database: studpg
        - Username: pguser
        - Password: pgpass
    - Откройте Query Tool для базы studpg и выполните SQL-запрос для создания пользователя student:
      ```
      CREATE USER student WITH PASSWORD 'Stud2024!!!';
      GRANT ALL PRIVILEGES ON DATABASE studpg TO student;
      ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO student;
      ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON SEQUENCES TO student;
      ```
    - В дальнейшем для подключения из Python используйте пользователя student.
2. **MongoDB**:
    - Создайте базу данных studmongo. Это можно сделать автоматически при первом подключении и вставке данных из вашего Python-скрипта.

Шаг 3: Выполнение заданий

Выберите вариант из таблицы ниже и выполните три задания: для PostgreSQL, для MongoDB и сравнительный анализ в Jupyter Notebook.

Общий подход к выполнению:

1. **Сгенерируйте данные**. Напишите скрипт на Python для генерации данных в соответствии с вашим вариантом. Объем данных должен составлять не менее 10 000 записей, если иное не указано в задании.
2. **Работа с PostgreSQL**:
    - Определите реляционную схему (CREATE TABLE).
    - Напишите скрипт для вставки сгенерированных данных.

- Реализуйте запрос для вашего задания.
- Измерьте время выполнения запроса.

3. **Работа с MongoDB**:
   - Определите структуру документа.
   - Напишите скрипт для вставки сгенерированных данных.
   - Реализуйте запрос (чаще всего, с помощью aggregation framework).
   - Измерьте время выполнения запроса.
4. **Анализ в Jupyter Notebook**:
   - Объедините все шаги в одном .ipynb файле.
   - Представьте результаты измерений в виде таблицы Pandas.
   - Постройте график (например, столбчатую диаграмму) для визуального сравнения производительности.
   - Напишите выводы по результатам сравнения для вашего конкретного случая.

Варианты заданий

Сформируйте данные объемом не менее 10 000 записей для обычных данных и 100 000 для "больших данных".

№	Задание для PostgreSQL	Задание для MongoDB	Анализ в Jupyter Notebook
1	Продажи в интернет-магазине. Создать таблицы products, categories, orders. Выполнить сложный агрегационный запрос: найти 5 самых продаваемых товаров в каждой категории за последний месяц.	Продажи в интернет-магазине. Создать коллекцию orders, где информация о товарах и категориях вложена в документ заказа. Выполнить агрегационный запрос для поиска 5 самых продаваемых товаров в каждой категории.	Сравнить время выполнения сложных агрегационных запросов. Визуализировать результат. Сделать вывод о применимости для аналитических отчетов.
2	База клиентов. Создать таблицу clients (email, name, phone). Выполнить поиск клиента по email. Затем создать B-Tree индекс на поле email и снова измерить время поиска.	База клиентов. Создать коллекцию clients. Выполнить поиск по email. Затем создать индекс на поле email и снова измерить время поиска.	Сравнить влияние индексирования на скорость поиска в обеих СУБД. Построить график, показывающий время до и после создания индекса для каждой системы.

3	Логистика. Создать таблицы shipments (грузы), warehouses (склады), routes (маршруты). Загрузить 100 000 записей. Оценить размер базы данных на диске.	Логистика. Создать коллекцию shipments с вложенной информацией о складах и маршрутах. Загрузить 100 000 записей. Оценить размер коллекции на диске.	Сравнить потребление дискового пространства для хранения одинакового набора данных. Сравнить время выполнения простого запроса (например, найти все грузы > 100 кг).
4	Финансовые транзакции. Создать таблицу transactions (from_account, to_account, amount, timestamp). Измерить время вставки 100 000 записей с помощью executemany.	Финансовые транзакции. Создать коллекцию transactions. Измерить время вставки 100 000 записей с помощью insert_many.	Сравнить производительность массовой вставки данных. Построить столбчатую диаграмму. Проанализировать, какая СУБД лучше справляется с быстрой записью данных.
5	CRM-система. Создать таблицу customers с полем status. Выполнить операцию UPDATE, изменяющую статус для 10% случайно выбранных клиентов. Измерить время.	CRM-система. Создать коллекцию customers с полем status. Выполнить операцию update_many , изменяющую статус для 10% клиентов. Измерить время.	Сравнить производительность массовых обновлений данных. Проанализировать, как механизм хранения данных влияет на скорость операций UPDATE.
6	Полнотекстовый поиск. Создать таблицу documents с полем content (тип TEXT). Заполнить 10 000 записей текстом. Выполнить поиск по ключевому слову с использованием to_tsvector.	Полнотекстовый поиск. Создать коллекцию documents с текстовым полем content. Создать текстовый индекс. Выполнить поиск по ключевому слову с использованием оператора $text.	Сравнить производительность полнотекстового поиска. Сравнить удобство настройки и использования в обеих СУБД.
7	Геоданные. Создать таблицу locations с	Геоданные. Создать коллекцию locations с	Сравнить производительность

	использованием расширения PostGIS (тип GEOMETRY). Найти все объекты в радиусе 5 км от заданной точки.	полем координат в формате GeoJSON. Создать 2dsphere индек с. Найти все объекты в радиусе 5 км от точки.	геопространственных запросов. Оценить простоту реализации подобных запросов в каждой из систем.
8	Временные ряды (биржевые котировки). Создать таблицу quotes (ticker, price, timestamp). Найти среднюю цену для каждого тикера за последний час.	Временные ряды. Создать коллекцию quotes. Написать агрегационный запрос для поиска средней цены для каждого тикера за последний час.	Сравнить производительность обработки временных рядов. Построить график времени выполнения. Сделать вывод о применимости для задач мониторинга и анализа котировок.
9	Управление персоналом (JOIN). Создать таблицы employees и depart ments. Выполнить запрос JOIN для получения списка сотрудников с названиями их отделов.	Управление персоналом (вложенность). Создать коллекцию employees с вложенным документом department . Выполнить запрос для получения списка сотрудников и их отделов.	Сравнить производительность "соединения" данных. Проанализировать разницу между JOIN в SQL и работой с вложенными документами в NoSQL.
10	Работа с JSON. Создать таблицу products с полем attributes типа JSON B. Найти все товары, у которых есть атрибут "color" со значением "blue".	Работа с JSON. Создать коллекцию products, где атрибуты хранятся как вложенный объект. Найти все товары, у которых attributes.color равен "blue".	Сравнить производительность и удобство запросов к вложенным JSON-структурам.
11	Вложенные структуры (управление проектами). Создать таблицы projects и tasks (с project_id). Найти все проекты, в которых есть хотя бы одна задача со статусом "срочно".	Вложенные структуры (управление проектами). Создать коллекцию projects с массивом вложенных задач tasks. Найти все проекты, где есть задача со статусом "срочно".	Сравнить производительность запросов к иерархическим данным. Проанализировать, какой подход (реляционный или документо-

			ориентированный) удобнее для таких структур.
12	Сортировка. Создать таблицу logs (100 000 записей) с полем timestamp. Выполнить запрос SELECT * FROM logs ORDER BY timestamp DESC LIMIT 1000.	Сортировка. Создать коллекцию logs (100 000 записей). Выполнить запрос find().sort("timestamp", -1).limit(1000).	Сравнить производительность операций сортировки большого объема данных. Проанализировать, как индексирование поля сортировки влияет на результат в обеих СУБД.
13	Группировка данных. Создать таблицу webevents (user_id, event_type, timestamp). Сгруппировать данные для подсчета количества событий каждого типа для каждого пользователя.	Группировка данных. Создать коллекцию webevents. Написать агрегационный запрос с $group для подсчета количества событий каждого типа для каждого пользователя.	Сравнить производительность операций группировки (GROUP BY vs $group). Сделать вывод о применимости для построения аналитических отчетов.
14	Удаление данных. Создать таблицу archive (100 000 записей). Измерить время выполнения операции DELETE для удаления 20% записей по условию.	Удаление данных. Создать коллекцию archive (100 000 записей). Измерить время выполнения delete_many для удаления 20% записей по условию.	Сравнить производительность массового удаления данных. Проанализировать, как СУБД справляются с освобождением места и поддержкой производительности после удаления.
15	Анализ логов. Создать таблицу app_logs (level, message, timestamp). Найти все логи с уровнем 'ERROR' за последние 24 часа, содержащие слово 'database'.	Анализ логов. Создать коллекцию app_logs. Найти все логи с уровнем 'ERROR' за последние 24 часа, используя $regex для поиска слова 'database'.	Сравнить производительность запросов к большим объемам лог-файлов, включающих фильтрацию по дате и текстовому содержанию.

16	Интернет вещей (IoT). Создать таблицу sensor_data (sensor _id, temperature, timestamp). Найти максимальную температуру для каждого сенсора за все время.	Интернет вещей (IoT). Создать коллекцию sensor_data. Написать агрегационный запрос для поиска максимальной температуры для каждого сенсора.	Сравнить производительность агрегационных запросов на данных временных рядов от множества источников.
17	Социальная сеть. Создать таблицы users и posts (с use r_id). Подсчитать количество постов для каждого пользователя, у которого более 10 постов.	Социальная сеть. Создать коллекцию posts (с aut hor_id). Написать агрегацию для подсчета постов каждого автора и отфильтровать тех, у кого их более 10.	Сравнить производительность запросов GROUP BY с HAVING против агрегационного конвейера $group -> $match.
18	Бронирование отелей. Создать таблицы hotels, rooms, book ings. Найти все отели, в которых есть свободные номера в заданном диапазоне дат.	Бронирование отелей. Создать коллекцию hotels с вложенным массивом rooms, где для каждой комнаты есть массив bookings. Написать запрос для поиска отелей со свободными номерами.	Сравнить сложность реализации и производительность запроса на поиск доступности, который требует проверки отсутствия пересекающихся записей.
19	Бинарные данные (файлы). Создать таблицу files с полем data типа bytea. Вставить и затем извлечь 100 файлов по 1 МБ. Измерить среднее время.	Бинарные данные (файлы). Использовать GridFS для хранения 100 файлов по 1 МБ. Измерить среднее время вставки и извлечения одного файла.	Сравнить подходы к хранению и извлечению бинарных данных. Проанализировать производительность и удобство использования bytea и GridFS.
20	Уникальные значения. Создать таблицу visits (ip_address, url). Найти количество	Уникальные значения. Создать коллекцию visits. Использовать	Сравнить производительность подсчета уникальных

	уникальных IP-адресов, посетивших сайт. Использовать COUNT(DIS TINCT ip_address).	агрегацию с $group по ip_address и затем подсчитать количество полученных групп для определения уникальных посетителей.	значений в большом наборе данных.
21	Работа с массивами. Создать таблицу articles с полем tags типа text[] (масс ив). Найти все статьи, у которых есть тег 'python'.	Работа с массивами. Создать коллекцию articles с полем tags (массив строк). Найти все статьи, у которых в массиве тегов есть 'python'.	Сравнить производительность и синтаксис запросов к элементам массива в PostgreSQL и MongoDB.
22	Сложная фильтрация. Создать таблицу products (price, category, rating). Найти товары из категории 'electronics' с ценой от 100 до 500 и рейтингом > 4.5.	Сложная фильтрация. Создать коллекцию products. Найти товары, соответствующие нескольким условиям ($and или неявный 'и'): категория, диапазон цен, рейтинг.	Сравнить производительность запросов с несколькими фильтрами. Создать составные индексы и проверить, как они влияют на скорость в обеих СУБД.
23	Оконные функции. Создать таблицу sales (employee_id, sale_amount, sale_date). С помощью оконной функции вычислить скользящее среднее продаж для каждого сотрудника за 3 дня.	Оконные функции. Создать коллекцию sales. С помощью агрегационного конвейера и оператора $setWindow Fields вычислить скользящее среднее продаж для каждого сотрудника за 3 дня.	Сравнить производительность и сложность реализации аналитических запросов с использованием оконных функций.
24	Целостность данных. Создать таблицы authors и books с внешним ключом. Попробовать вставить	Целостность данных. Создать коллекции authors и bo oks. Реализовать логику проверки	Проанализировать и сравнить подходы к обеспечению целостности данных: на уровне БД

	книгу с несуществующим автором. Попробовать удалить автора, у которого есть книги.	целостности на уровне приложения (перед вставкой книги проверить наличие автора).	(PostgreSQL) и на уровне приложения (для MongoDB). Производительность не измерять, сделать акцент на надежности.
25	Скорость чтения случайных записей. Создать таблицу items (100 000 записей). Выполнить 1000 запросов на чтение одной случайной записи по её id. Измерить общее время.	Скорость чтения случайных записей. Создать коллекцию items. Выполнить 1000 запросов на чтение одного случайного документа по его _id. Измерить общее время.	Сравнить производительность точечных чтений (Point Reads) по первичному ключу/индексу.
26	Управление запасами. Создать таблицу inventory (product_id, warehouse_id, quantity). Найти товары, количество которых на любом складе меньше 10.	Управление запасами. Создать коллекцию products с массивом stock ([{warehouse_id, quantity}]). Найти товары, где количество в одном из элементов массива stock меньше 10.	Сравнить производительность запросов к данным в формате "один-ко-многим", реализованных через отдельную таблицу и через вложенный массив.
27	AB-тестирование. Создать таблицу experiments (user_id, experiment_name, group ('A' или 'B'), metric). Посчитать среднее значение метрики для групп A и B в каждом эксперименте.	AB-тестирование. Создать коллекцию experiments. Написать агрегацию для группировки по experiment_name и group для подсчета среднего значения метрики.	Сравнить производительность типичного аналитического запроса для анализа результатов AB-тестов.
28	Мультиязычные данные. Создать таблицу content с полями title_en, title_fr, body_en, body_fr. Выполнить поиск по title_en и body_en.	Мультиязычные данные. Создать коллекцию content со вложенным объектом i18n ({en: {title, body}, fr: {title,	Сравнить удобство и производительность работы с локализованными данными при использовании "плоской" структуры с

		body}}). Выполнить поиск по i18n.en.title и i18n.en.body.	префиксами и иерархической структуры.
29	Аудит изменений. Создать таблицу documents и document_history (с document_id). При обновлении документа вставлять старую версию в history. Измерить время сложной операции обновления + вставки.	Аудит изменений. Создать коллекцию documents, где каждая версия документа хранится в массиве history. При обновлении добавлять новую версию в массив. Измерить время обновления (с $push).	Сравнить производительность и сложность реализации паттерна "аудит изменений" или "версионирование" данных в реляционном и документо-ориентированном подходе.
30	Рекомендательная система. Создать таблицы users, products, views (user_id, product_id). Найти товары, которые просматривал пользователь X, а также другие пользователи, смотревшие те же товары.	Рекомендательная система. Создать коллекции users и products. В документе пользователя хранить массив просмотренных viewed_products. Написать сложный агрегационный запрос для поиска "похожих" пользователей.	Сравнить производительность и сложность реализации простого запроса для коллаборативной фильтрации ("users who viewed this also viewed

Требования к отчету

Отчет должен быть представлен в виде одного файла Jupyter Notebook (lw.ipynb), который включает в себя:

1. **Титульный лист**. В виде ячейки Markdown с информацией о студенте и номере варианта.
2. **Цель работы**.
3. **Краткая теоретическая справка**. Описание PostgreSQL и MongoDB, их ключевые различия.
4. **Архитектура решения и потоки данных**.
    - Опишите, как вы структурировали данные в PostgreSQL (схема таблиц, связи).
    - Опишите, как вы структурировали данные в MongoDB (структура документа).
    - Приведите диаграмму или словесное описание потоков данных: от генерации данных в Python до загрузки в каждую БД, выполнения запросов и анализа результатов.
5. **Практическая часть**.
    - Код для создания таблиц в PostgreSQL и описание структуры документов в MongoDB.
    - Код на Python для генерации и вставки данных в обе СУБД.
    - Код запросов для обеих СУБД и код для измерения времени их выполнения.
6. **Результаты и анализ**.
    - Таблица (Pandas DataFrame) со временем выполнения запросов.
    - График, визуально сравнивающий производительность.
    - Подробные выводы, объясняющие полученные результаты. Почему одна СУБД оказалась быстрее другой в данном конкретном задании? Какие особенности архитектуры каждой СУБД повлияли на результат?

Примеры кода для генерации данных для каждого задания

```
import random
from datetime import datetime, timedelta
import json
import psycopg2
from pymongo import MongoClient
from faker import Faker

fake = Faker()

# Подключение к PostgreSQL
pg_conn = psycopg2.connect(
    dbname="your_database",
    user="your_username",
    password="your_password",
    host="localhost"
)
pg_cursor = pg_conn.cursor()

# Подключение к MongoDB
mongo_client = MongoClient('mongodb://localhost:27017/')
mongo_db = mongo_client['your_database']

# 1. Данные о продажах в интернет-магазине
def generate_sales_data(n_records):
    sales = []
    for _ in range(n_records):
        sale = {
            "order_id": fake.uuid4(),
            "customer_id": fake.uuid4(),
            "product_id": fake.uuid4(),
            "quantity": random.randint(1, 10),
            "price": round(random.uniform(10, 1000), 2),
            "date": fake.date_time_this_year()
        }
        sales.append(sale)
    return sales

# 2. База данных клиентов компании
def generate_customer_data(n_records):
    customers = []
    for _ in range(n_records):
        customer = {
            "id": fake.uuid4(),
            "name": fake.name(),
            "email": fake.email(),
```

```
            "phone": fake.phone_number(),
            "address": fake.address(),
            "registration_date": fake.date_time_this_decade()
        }
        customers.append(customer)
    return customers

# 3. Данные логистической компании
def generate_logistics_data(n_records):
    logistics = []
    for _ in range(n_records):
        logistic = {
            "delivery_id": fake.uuid4(),
            "from_address": fake.address(),
            "to_address": fake.address(),
            "weight": round(random.uniform(0.1, 100), 2),
            "status": random.choice(["In Transit", "Delivered", "Pending"]),
            "estimated_delivery": fake.future_date()
        }
        logistics.append(logistic)
    return logistics

# 4. Транзакции финансовой организации
def generate_financial_transactions(n_records):
    transactions = []
    for _ in range(n_records):
        transaction = {
            "transaction_id": fake.uuid4(),
            "account_from": fake.iban(),
            "account_to": fake.iban(),
            "amount": round(random.uniform(10, 10000), 2),
            "currency": random.choice(["USD", "EUR", "GBP"]),
            "timestamp": fake.date_time_this_year()
        }
        transactions.append(transaction)
    return transactions

# 5. Данные CRM-системы
def generate_crm_data(n_records):
    crm_data = []
    for _ in range(n_records):
        record = {
            "contact_id": fake.uuid4(),
            "name": fake.name(),
            "company": fake.company(),
            "email": fake.company_email(),
            "phone": fake.phone_number(),
            "last_contact": fake.date_time_this_month(),
```

```
            "status": random.choice(["Lead", "Prospect", "Customer", "Churned"])
        }
        crm_data.append(record)
    return crm_data

# 6. Данные для системы управления складскими запасами
def generate_inventory_data(n_records):
    inventory = []
    for _ in range(n_records):
        item = {
            "item_id": fake.uuid4(),
            "name": fake.word(),
            "category": fake.word(),
            "quantity": random.randint(0, 1000),
            "price": round(random.uniform(1, 1000), 2),
            "warehouse": fake.city(),
            "last_updated": fake.date_time_this_month()
        }
        inventory.append(item)
    return inventory

# 7. Техническая документация IT-компании
def generate_technical_docs(n_records):
    docs = []
    for _ in range(n_records):
        doc = {
            "doc_id": fake.uuid4(),
            "title": fake.sentence(),
            "content": fake.text(),
            "author": fake.name(),
            "created_date": fake.date_time_this_year(),
            "category": random.choice(["API", "Database", "Frontend", "Backend", "DevOps"]),
            "version": f"{random.randint(1,5)}.{random.randint(0,9)}"
        }
        docs.append(doc)
    return docs

# 8. Данные отслеживания корпоративного транспорта
def generate_vehicle_tracking_data(n_records):
    vehicles = []
    for _ in range(n_records):
        vehicle = {
            "vehicle_id": fake.license_plate(),
            "timestamp": fake.date_time_this_month(),
            "latitude": fake.latitude(),
            "longitude": fake.longitude(),
            "speed": random.uniform(0, 120),
            "fuel_level": random.uniform(0, 100),
```

```
            "status": random.choice(["Moving", "Stopped", "Maintenance"])
        }
        vehicles.append(vehicle)
    return vehicles

# 9. Биржевые котировки
def generate_stock_data(n_records):
    stocks = []
    for _ in range(n_records):
        stock = {
            "symbol": fake.currency_code(),
            "timestamp": fake.date_time_this_year(),
            "open": round(random.uniform(10, 1000), 2),
            "high": round(random.uniform(10, 1000), 2),
            "low": round(random.uniform(10, 1000), 2),
            "close": round(random.uniform(10, 1000), 2),
            "volume": random.randint(1000, 1000000)
        }
        stocks.append(stock)
    return stocks

# 10. Данные системы управления персоналом
def generate_hr_data(n_records):
    employees = []
    for _ in range(n_records):
        employee = {
            "employee_id": fake.uuid4(),
            "name": fake.name(),
            "department": fake.job(),
            "position": fake.job(),
            "hire_date": fake.date_this_decade(),
            "salary": random.randint(30000, 150000),
            "performance_score": round(random.uniform(1, 5), 1)
        }
        employees.append(employee)
    return employees

# 11. База данных учета материальных ценностей
def generate_asset_data(n_records):
    assets = []
    for _ in range(n_records):
        asset = {
            "asset_id": fake.uuid4(),
            "name": fake.word(),
            "category": random.choice(["IT", "Furniture", "Vehicle", "Equipment"]),
            "purchase_date": fake.date_this_decade(),
            "purchase_price": round(random.uniform(100, 10000), 2),
            "current_value": round(random.uniform(50, 9000), 2),
```

```
            "location": fake.city(),
            "status": random.choice(["In Use", "In Storage", "Maintenance", "Disposed"])
        }
        assets.append(asset)
    return assets

# 12. Данные системы управления проектами
def generate_project_data(n_records):
    projects = []
    for _ in range(n_records):
        project = {
            "project_id": fake.uuid4(),
            "name": fake.catch_phrase(),
            "start_date": fake.date_this_year(),
            "end_date": fake.date_between(start_date='+30d', end_date='+1y'),
            "status": random.choice(["Planning", "In Progress", "On Hold", "Completed"]),
            "budget": random.randint(10000, 1000000),
            "manager": fake.name(),
            "tasks": [
                {
                    "task_id": fake.uuid4(),
                    "name": fake.bs(),
                    "status": random.choice(["To Do", "In Progress", "Done"]),
                    "assigned_to": fake.name()
                } for _ in range(random.randint(3, 10))
            ]
        }
        projects.append(project)
    return projects

# 13. Данные с IoT-устройств
def generate_iot_data(n_records):
    iot_data = []
    for _ in range(n_records):
        data = {
            "device_id": fake.uuid4(),
            "timestamp": fake.date_time_this_month(),
            "temperature": round(random.uniform(-10, 40), 1),
            "humidity": round(random.uniform(0, 100), 1),
            "pressure": round(random.uniform(900, 1100), 1),
            "battery_level": random.randint(0, 100)
        }
        iot_data.append(data)
    return iot_data
```

```
# 14. Данные о поведении пользователей веб-сайта
def generate_user_behavior_data(n_records):
    behaviors = []
    for _ in range(n_records):
        behavior = {
            "session_id": fake.uuid4(),
            "user_id": fake.uuid4(),
            "timestamp": fake.date_time_this_month(),
            "page_visited": f"/{fake.uri_path()}",
            "time_spent": random.randint(5, 300),
            "device": random.choice(["Desktop", "Mobile", "Tablet"]),
            "browser": random.choice(["Chrome", "Firefox", "Safari", "Edge"]),
            "action": random.choice(["Click", "Scroll", "Type", "Hover"])
        }
        behaviors.append(behavior)
    return behaviors

# 15. Данные системы онлайн-бронирования
def generate_booking_data(n_records):
    bookings = []
    for _ in range(n_records):
        booking = {
            "booking_id": fake.uuid4(),
            "user_id": fake.uuid4(),
            "service": random.choice(["Hotel", "Flight", "Car", "Restaurant"]),
            "date": fake.date_between(start_date='today', end_date='+1y'),
            "status": random.choice(["Confirmed", "Pending", "Cancelled"]),
            "price": round(random.uniform(50, 1000), 2),
            "payment_method": random.choice(["Credit Card", "PayPal", "Bank Transfer"])
        }
        bookings.append(booking)
    return bookings

# 16. Данные отзывов клиентов
def generate_customer_reviews(n_records):
    reviews = []
    for _ in range(n_records):
        review = {
            "review_id": fake.uuid4(),
            "product_id": fake.uuid4(),
            "user_id": fake.uuid4(),
            "rating": random.randint(1, 5),
            "comment": fake.text(),
            "date": fake.date_time_this_year(),
            "helpful_votes": random.randint(0, 100)
        }
        reviews.append(review)
    return reviews
```

```
# 17. Данные для системы ранжирования поставщиков
def generate_supplier_ranking_data(n_records):
    suppliers = []
    for _ in range(n_records):
        supplier = {
            "supplier_id": fake.uuid4(),
            "name": fake.company(),
            "product_quality": round(random.uniform(1, 10), 1),
            "delivery_time": round(random.uniform(1, 10), 1),
            "price_competitiveness": round(random.uniform(1, 10), 1),
            "customer_service": round(random.uniform(1, 10), 1),
            "total_orders": random.randint(10, 1000),
            "return_rate": round(random.uniform(0, 0.1), 3)
        }
        suppliers.append(supplier)
    return suppliers

# 18. Данные цифровых активов компании
def generate_digital_assets(n_records):
    assets = []
    for _ in range(n_records):
        asset = {
            "asset_id": fake.uuid4(),
            "name": fake.file_name(),
            "type": random.choice(["Image", "Video", "Document", "Audio"]),
            "size": random.randint(1000000, 100000000),  # size in bytes
            "created_date": fake.date_time_this_year(),
            "last_modified": fake.date_time_this_month(),
            "owner": fake.name(),
            "tags": [fake.word() for _ in range(random.randint(1, 5))]
        }
        assets.append(asset)
    return assets

# 19. Данные социальных связей в корпоративной сети
def generate_social_network_data(n_records):
    users = [fake.uuid4() for _ in range(n_records)]
    connections = []
    for user in users:
        num_connections = random.randint(1, 20)
        for _ in range(num_connections):
            connection = {
                "user_id": user,
                "connected_user_id": random.choice(users),
                "connection_type": random.choice(["Colleague", "Manager", "Subordinate", "Project
Team"]),
                "connection_strength": random.randint(1, 10),
```

```
            "last_interaction": fake.date_time_this_year()
        }
        connections.append(connection)
    return connections

# 20. Данные финансовой отчетности предприятия
def generate_financial_reports(n_records):
    reports = []
    for _ in range(n_records):
        report = {
            "report_id": fake.uuid4(),
            "company_id": fake.uuid4(),
            "year": random.randint(2010, 2023),
            "quarter": random.randint(1, 4),
            "revenue": round(random.uniform(1000000, 1000000000), 2),
            "expenses": round(random.uniform(800000, 900000000), 2),
            "profit": round(random.uniform(100000, 100000000), 2),
            "assets": round(random.uniform(1000000, 10000000000), 2),
            "liabilities": round(random.uniform(500000, 5000000000), 2),
            "equity": round(random.uniform(500000, 5000000000), 2)
        }
        reports.append(report)
    return reports

# 21. Данные мониторинга производственных процессов
def generate_production_monitoring_data(n_records):
    data = []
    for _ in range(n_records):
        record = {
            "timestamp": fake.date_time_this_month(),
            "machine_id": fake.uuid4(),
            "temperature": round(random.uniform(20, 80), 1),
            "pressure": round(random.uniform(1, 10), 2),
            "production_rate": random.randint(100, 1000),
            "defect_rate": round(random.uniform(0, 0.05), 3),
            "energy_consumption": round(random.uniform(100, 1000), 2),
            "status": random.choice(["Running", "Idle", "Maintenance", "Error"])
        }
        data.append(record)
    return data

# 22. Данные деловых документов в формате XML
def generate_business_documents_xml(n_records):
    documents = []
    for _ in range(n_records):
        document = f"""
        <document>
            <id>{fake.uuid4()}</id>
```

```
            <type>{random.choice(['Invoice', 'Contract', 'Report', 'Proposal'])}</type>
            <date>{fake.date_this_year()}</date>
            <author>{fake.name()}</author>
            <content>{fake.paragraph()}</content>
            <metadata>
                <department>{fake.job()}</department>
                <priority>{random.choice(['Low', 'Medium', 'High'])}</priority>
                <status>{random.choice(['Draft', 'Under Review', 'Approved', 'Rejected'])}</status>
            </metadata>
        </document>
        """
        documents.append(document)
    return documents

# 25. Логи корпоративной сети
def generate_network_logs(n_records):
    log_types = ["INFO", "WARNING", "ERROR", "CRITICAL"]
    logs = []
    for _ in range(n_records):
        log = {
            "timestamp": fake.date_time_this_month(),
            "ip_address": fake.ipv4(),
            "user_id": fake.uuid4(),
            "action": fake.word(),
            "status": random.choice(["SUCCESS", "FAILURE"]),
            "log_level": random.choice(log_types),
            "message": fake.sentence()
        }
        logs.append(log)
    return logs

# Пример использования:
sales_data = generate_sales_data(100000)
mongo_db.sales.insert_many(sales_data)

customer_data = generate_customer_data(1000)
pg_cursor.executemany("""
    INSERT INTO customers (id, name, email, phone, address, registration_date)
    VALUES (%(id)s, %(name)s, %(email)s, %(phone)s, %(address)s, %(registration_date)s)
""", customer_data)
pg_conn.commit()

# Закрытие соединений
pg_cursor.close()
pg_conn.close()
mongo_client.close()
```