Министерство науки и высшего образования РФ Федеральное государственное автономное образовательное учреждение высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий Кафедра программной инженерии

# ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

# НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

|  |
| --- |
| Кафедра программной инженерии |
| место прохождения практики |
|  |

Сравнение скорости CRUD операций (PostgreSQL c SQLite)

|  |
| --- |
| тема |

Руководитель от университета: А.Н. Пупков

подпись, дата

Руководитель от предприятия: А.Н. Пупков

подпись, дата

Студент Д.Д. Сурков

номер группы, зачетной книжки подпись, дата

Красноярск 2025

# ВВЕДЕНИЕ

* 1. **Цели и задачи**

Цель практики: Проведение сравнительного анализа производительности систем управления базами данных PostgreSQL и SQLite, получение практических навыков измерения и оценки скорости выполнения CRUD-операций (Create, Read, Update, Delete) и влияния архитектуры СУБД на эти показатели.

Задачи практики:

* Изучить теоретические основы и принципы CRUD-операций;
* Изучить архитектурные особенности, принципы работы и хранения данных в реляционной клиент-серверной СУБД PostgreSQL и встраиваемой файловой СУБД SQLite;
* Разработать и реализовать программные скрипты для выполнения CRUD-операций и измерения времени их выполнения для каждой из СУБД;
* Настроить рабочее окружение с использованием Docker для проведения экспериментов;
* Провести серию экспериментов по измерению производительности, собрать и обработать полученные данные;
* Проанализировать результаты, сформулировать выводы о различиях в производительности и подготовить итоговый отчет о проделанной работе.

# Задание на практику

Для успешного прохождения практики необходимо выполнить следующие задания:

Теоретическая часть:

1. Описать, что такое CRUD-операции. Дать определение акрониму CRUD (Create, Read, Update, Delete). Для каждой операции привести соответствующую SQL-команду (INSERT, SELECT, UPDATE, DELETE) и кратко описать ее назначение в контексте управления данными.
2. Описать, как работают хранилища данных PostgreSQL и SQLite. Что делать: Изучить и описать ключевые архитектурные различия: клиент-серверная модель PostgreSQL (процессы, сетевое взаимодействие) и встраиваемая модель SQLite (работа через файл, блокировки на уровне файла). Объяснить, как каждая СУБД хранит данные на диске. Обязательно ссылаться на официальную документацию.
3. Провести сравнительный теоретический анализ скорости CRUD-операций. На основе документации и других источников (статей, блогов) проанализировать, почему можно ожидать разницу в производительности. Рассмотреть такие факторы, как: сетевые задержки (у PostgreSQL), механизмы блокировок и параллелизма (MVCC у PostgreSQL против блокировки файла у SQLite), накладные расходы на запуск транзакций.
4. Сделать теоретические выводы. Сформулировать гипотезу: какая из СУБД и в каких операциях (например, массовая вставка, одиночное чтение, сложные обновления) теоретически должна быть быстрее и почему. Связать это с дизайном каждой системы.

Экспериментальная часть:

1. Установить Docker. Скачать и установить Docker Desktop. Убедиться, что он корректно запускается и работает в вашей операционной системе.
2. Настроить окружение для баз данных. Для PostgreSQL найти официальный образ на Docker Hub и скачать его командой docker pull postgres. Запустить контейнер. Для SQLite убедиться, что в выбранном языке программирования (например, Python) есть встроенная или легко устанавливаемая библиотека для работы с ней.
3. Написать скрипты для тестирования. Создать скрипты, которые будут: подключаться к PostgreSQL и SQLite; создавать одинаковую по структуре таблицу; в цикле выполнять каждую из CRUD-операций (например, 1000 вставок, 1000 чтений по ключу, 1000 обновлений, 1000 удалений), замеряя общее время выполнения каждой серии операций.
4. Провести эксперименты. Запустить скрипты несколько раз (например, 5-10 запусков для каждого теста), чтобы нивелировать случайные колебания производительности системы. Перед началом тестов зафиксировать и записать в отчет характеристики вашего ПК (модель CPU, объем RAM, тип диска (SSD/HDD)).
5. Провести статистическую обработку данных. Для каждой серии замеров (например, для 10 запусков теста на вставку в PostgreSQL) рассчитать среднее время выполнения (математическое ожидание) и дисперсию. Дисперсия покажет, насколько стабильны результаты от запуска к запуску.
6. Построить графики. Создать наглядные графики (например, столбчатые диаграммы), сравнивающие среднее время выполнения каждой CRUD-операции (Create, Read, Update, Delete) для PostgreSQL и SQLite. Каждый график должен иметь заголовок и подписанные оси.
7. Сделать выводы по результатам эксперимента. Сравнить полученные результаты с теоретическими предположениями из первой части. Объяснить, почему получились именно такие цифры. Например: "PostgreSQL показал более медленную вставку из-за сетевых задержек, но более быстрое чтение при высокой нагрузке благодаря MVCC, что подтверждается графиками...".

# 2 Календарный график практики

В таблице 1 показан учебный план практики.

Таблица 1 – Учебный план практики

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Этапы практики** | **Виды учебной работы, на практике включает самостоятельную работу студентов, трудоёмкость (в часах) и сроки выполнения** | **Формы контроля** |
| 1 | Подготовительный этап | Ознакомление с заданием на практику, изучение теоретических основ (CRUD, PostgreSQL, SQLite), 14 часов, 24.06.2024 – 25.06.2024 | Общий контроль руководителя |
| Составление и отправка первого промежуточного (титульного) отчета, 4 часа, 26.06.2024 |
| 2 | Экспериментальный  этап | Настройка рабочего окружения (Docker, PostgreSQL), 6 часов, 27.06.2024 | Общий контроль  руководителя |
|  | Разработка и отладка скриптов для тестирования CRUD-операций, 32 часа, 28.06.2024 – 01.07.2024 | Глава 1 отчета по практике |
|  | Проведение серий экспериментов, сбор данных, 24 часа, 02.07.2024 – 03.07.2024 | Общий контроль руководителя |
|  | Статистическая обработка результатов, построение графиков, 12 часов, 04.07.2024 | Общий контроль руководителя |
|  | Составление второго промежуточного (экспериментального) отчета, 4 часа, 05.07.2024 | Общий контроль руководителя |
| 3 | Составление финального отчёта по производственной практике  и его защита | Подготовка и оформление финального отчета по практике, 4 часа, 06.07.2024 | Отчёт о практике |
|  | Подготовка к защите и публичная защита финального отчета, 2 часа, 07.07.2024 | Дифференцированный зачёт |

**3 Теоретическая часть**

**3.1 Описание CRUD операций**

CRUD — это акроним, обозначающий четыре базовые функции, используемые для работы с данными в постоянных хранилищах. Эти операции являются фундаментом для большинства приложений, работающих с базами данных.

* Create (Создание) — операция добавления новых записей в базу данных. В языке SQL этой операции соответствует команда INSERT. Пример: INSERT INTO table\_name (column1, column2) VALUES (value1, value2);
* Read (Чтение) — операция извлечения или чтения данных из базы данных. Данные могут извлекаться как по одной записи, так и группами. В SQL этой операции соответствует команда SELECT. Пример: SELECT column1, column2 FROM table\_name WHERE condition;
* Update (Обновление) — операция изменения существующих записей в базе данных. В SQL этой операции соответствует команда UPDATE. Пример: UPDATE table\_name SET column1 = new\_value1 WHERE condition;
* Delete (Удаление) — операция удаления существующих записей из базы данных. В SQL этой операции соответствует команда DELETE. Пример: DELETE FROM table\_name WHERE condition;

**3.2 Описание работы хранилищ данных**

В данном исследовании сравниваются две популярные реляционные системы управления базами данных (СУБД) с принципиально разной архитектурой: PostgreSQL и SQLite.

PostgreSQL — это мощная, объектно-реляционная СУБД с открытым исходным кодом. Она работает по клиент-серверной модели. Это означает, что база данных функционирует как отдельный процесс (сервер), который постоянно запущен и ожидает подключений от клиентских приложений (в нашем случае — от Python-скрипта).

Ключевые особенности:

* Многопроцессная архитектура: Каждое новое подключение к серверу порождает собственный процесс на сервере, что обеспечивает высокую изоляцию и надежность.
* Сетевое взаимодействие: Клиент и сервер обмениваются данными по сети (даже если они находятся на одной машине), используя стандартные протоколы. Это создает дополнительные накладные расходы на установку соединения и передачу данных.
* Механизм MVCC (Multi-Version Concurrency Control): PostgreSQL использует механизм управления параллельным доступом с помощью многоверсионности. Это позволяет нескольким клиентам одновременно читать данные, не блокируя друг друга при операциях записи, что критически важно для высоконагруженных систем.
* Хранение данных: Данные хранятся в виде строк в файлах на диске, управляемых сервером.

SQLite — это встраиваемая (embedded) реляционная СУБД. В отличие от PostgreSQL, она не является отдельным сервером. SQLite представляет собой библиотеку, которую приложение подключает напрямую. Вся база данных (таблицы, индексы, данные) хранится в одном единственном файле на диске.

Ключевые особенности:

* Бессерверная архитектура: Отсутствует отдельный серверный процесс. Все операции чтения и записи производятся напрямую с файлом базы данных.
* Отсутствие сетевых задержек: Так как нет сетевого взаимодействия, накладные расходы на выполнение операций минимальны.
* Простые блокировки: Для обеспечения целостности данных при одновременной записи SQLite блокирует либо отдельные таблицы, либо весь файл базы данных. Это делает ее менее подходящей для систем с высоким уровнем параллелизма по сравнению с PostgreSQL, но очень эффективной для однопользовательских или низконагруженных приложений.
* Хранение данных: Вся база данных представляет собой один файл.
  1. **Описание используемых программных продуктов**

Для проведения эксперимента был использован следующий набор программных продуктов:

* Docker Desktop: Инструмент для контейнеризации, который позволил запустить СУБД PostgreSQL в изолированной, стандартной среде, не устанавливая ее напрямую в операционную систему. Это обеспечивает чистоту и воспроизводимость эксперимента.
* Python 3: Язык программирования, на котором были написаны скрипты для автоматизации CRUD-операций и измерения времени их выполнения.
* Библиотека psycopg2-binary: Драйвер для языка Python, обеспечивающий возможность подключения и взаимодействия с базой данных PostgreSQL.
* Библиотека sqlite3: Стандартная библиотека Python, входящая в его состав и предназначенная для работы с базами данных SQLite.
* Библиотека numpy: Библиотека для научных вычислений, которая использовалась для точного расчета статистических показателей: математического ожидания (среднего значения) и дисперсии по результатам нескольких запусков тестов.
  1. **Сравнительный теоретический анализ скорости CRUD операций**

Основываясь на архитектурных различиях PostgreSQL и SQLite, можно выдвинуть следующую гипотезу о производительности CRUD-операций:

* Create, Update, Delete: Ожидается, что SQLite покажет значительно более высокую скорость для одиночных операций вставки, обновления и удаления. Причина в отсутствии сетевых задержек. Каждая операция в PostgreSQL требует отправки запроса на сервер, обработки его отдельным процессом и получения ответа. В SQLite же это сводится к прямым файловым операциям, что несравнимо быстрее в однопользовательском режиме.
* Read: Для простых операций чтения по ключу также ожидается преимущество SQLite по тем же причинам (отсутствие сети). PostgreSQL может показать сопоставимые или лучшие результаты только в сложных сценариях с большим количеством одновременных чтений, но в рамках нашего эксперимента (один последовательный клиент) этот фактор не учитывается.
* Стабильность результатов: Дисперсия (разброс значений) у SQLite должна быть ниже, так как на время выполнения влияет меньше внешних факторов (состояние сети, загрузка серверных процессов). У PostgreSQL время может незначительно колебаться из-за фоновых процессов операционной системы и самого сервера.

Таким образом, теоретически для задач, выполняемых в рамках данного исследования, встраиваемая СУБД SQLite должна превосходить по производительности клиент-серверную PostgreSQL.

**4 Экспериментальная часть**

**4.1 Описание тестового стенда и методики эксперимента**

Исследования проводились на персональном компьютере со следующими техническими характеристиками:

* Процессор (CPU): Intel Core i5-8300H, 4 ядер, 8 потоков
* Оперативная память (RAM): 32 ГБ
* Накопитель: SSD 120GB + HDD 1TB
* Операционная система: Windows 10

Методика эксперимента:

1. Для СУБД PostgreSQL использовался официальный Docker-образ, запущенный в Docker Desktop.
2. Для СУБД SQLite использовалась встроенная библиотека Python.
3. Написан скрипт на языке Python, который последовательно выполняет тесты для обеих СУБД.
4. Каждый тест состоит из выполнения 1000 одинаковых CRUD-операций (1000 вставок, затем 1000 чтений и т.д.).
5. Для обеспечения статистической достоверности полный цикл тестов для каждой СУБД запускался 10 раз (итераций).
6. По результатам 10 итераций для каждой операции (например, для 10 замеров времени Create-операции в PostgreSQL) вычислялось математическое ожидание (среднее арифметическое) и дисперсия.

**4.2 Результаты экспериментов**

В ходе выполнения скрипта были получены данные о времени выполнения каждой серии CRUD-операций. Результаты, усредненные по 10 запускам, сведены в таблицу 4.1.

Таблица 4.1 – Результаты экспериментов

|  |  |  |  |
| --- | --- | --- | --- |
| **База данных** | **Операция** | **Среднее время (сек.)** | **Дисперсия** |
| PostgreSQL | Create | 0.395457 | 0.0009464673 |
| PostgreSQL | Read | 0.401898 | 0.0021879551 |
| PostgreSQL | Update | 0.384333 | 0.0016178417 |
| PostgreSQL | Delete | 0.394882 | 0.0005701159 |
| SQLite | Create | 0.008149 | 0.0000672626 |
| SQLite | Read | 0.071930 | 0.0000795884 |
| SQLite | Update | 0.008117 | 0.0000665067 |
| SQLite | Delete | 0.010989 | 0.0000517893 |

Для наглядной демонстрации полученных результатов были построены диаграммы. На рисунке 4.1 сравнивается среднее время выполнения каждой из четырех операций для PostgreSQL и SQLite.

Изображение выглядит как текст, снимок экрана, График, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1 – Среднее время выполнения

На рисунке 4.2 сравнивается дисперсия каждой из четырех операций для PostgreSQL и SQLite.

Изображение выглядит как текст, снимок экрана, диаграмма, График

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.2 – Дисперсия каждой операции

**4.3 Анализ результатов и выводы**

Результаты эксперимента полностью подтвердили гипотезу, выдвинутую в теоретической части.

1. Абсолютное преимущество SQLite: Как видно из таблицы 1 и рисунка 1, СУБД SQLite продемонстрировала на порядок более высокую производительность по всем четырем типам операций. Среднее время выполнения операций в SQLite было в десятки раз меньше, чем в PostgreSQL.
2. Причины различий: Основной причиной такого разрыва является архитектура СУБД. В PostgreSQL каждая операция — это сетевой запрос к отдельному серверному процессу. Этот путь включает в себя: установку TCP-соединения, передачу данных, аутентификацию, обработку запроса сервером и отправку ответа обратно. В SQLite все эти шаги отсутствуют; операции представляют собой прямые вызовы функций библиотеки, которые работают с локальным файлом, что является значительно более быстрым процессом.
3. Дисперсия: Как и предполагалось, значения дисперсии для SQLite оказались значительно ниже. Это говорит о высокой стабильности и предсказуемости времени выполнения операций, так как отсутствуют внешние факторы, такие как задержки в сети или загрузка фоновых процессов ОС, влияющие на сервер.

Итоговый вывод: В рамках сценария с одним пользователем и последовательным выполнением операций встраиваемая бессерверная СУБД SQLite является безоговорочным лидером по производительности. Клиент-серверная СУБД PostgreSQL спроектирована для решения более сложных задач: обеспечения безопасности, надежности и, что самое главное, эффективной одновременной работы множества пользователей, за что приходится "платить" дополнительными накладными расходами на каждую отдельную операцию. Выбор между этими СУБД должен основываться на конкретных требованиях проекта: для локальных приложений, не требующих многопользовательского доступа, SQLite является оптимальным выбором, тогда как для сетевых, многопользовательских систем PostgreSQL является более подходящим и мощным решением.

**ЗАКЛЮЧЕНИЕ**

В ходе научно-исследовательской работы было проведено теоретическое и практическое сравнение производительности CRUD-операций для СУБД PostgreSQL и SQLite. Были изучены их архитектурные особенности, разработан и реализован программный комплекс для автоматизированного тестирования. Полученные экспериментальные данные позволили количественно оценить разницу в скорости и подтвердить, что для однопользовательских задач SQLite значительно превосходит PostgreSQL благодаря отсутствию сетевых и процессных накладных расходов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Официальная документация PostgreSQL // The PostgreSQL Global Development Group : официальный сайт. – 2025. – URL: <https://www.postgresql.org/docs/> (дата обращения: 29.06.2025).

2 Официальная документация SQLite // SQLite : официальный сайт. – 2025. – URL: <https://www.sqlite.org/docs.html> (дата обращения: 29.06.2025).

3 Официальная документация Docker // Docker, Inc. : официальный сайт. – 2025. – URL: <https://www.docker.com/> (дата обращения: 29.06.2025).

4 Официальная документация Python // Python Software Foundation : официальный сайт. – 2025. – URL: <https://www.python.org/> (дата обращения: 29.06.2025).

5 Официальная документация NumPy // The NumPy community : официальный сайт. – 2025. – URL: <https://numpy.org/> (дата обращения: 29.06.2025).

**ПРИЛОЖЕНИЕ А**

**Листинг скрипта для тестирования**

# main.py

import psycopg2

import sqlite3

import time

import numpy as np

import matplotlib.pyplot as plt # Импортируем библиотеку для построения графиков

# --- НАСТРОЙКИ ЭКСПЕРИМЕНТА ---

NUM\_RECORDS = 1000  # Количество записей для каждой CRUD-операции в одном тесте

NUM\_RUNS = 10       # Сколько раз мы повторяем полный набор тестов

# --- ПАРАМЕТРЫ ПОДКЛЮЧЕНИЯ К POSTGRESQL ---

POSTGRES\_PARAMS = {

    "host": "localhost",

    "database": "postgres",

    "user": "postgres",

    "password": "Bangcock1919", # Убедитесь, что здесь ваш актуальный пароль

    "port": "5432",

    "client\_encoding": 'utf8'

}

# --- ФУНКЦИИ ДЛЯ ПРОВЕДЕНИЯ ТЕСТОВ ---

def run\_single\_test\_cycle(db\_type, connection\_params):

    conn = None

    results = {}

    try:

        if db\_type == "PostgreSQL":

            conn = psycopg2.connect(\*\*connection\_params)

        elif db\_type == "SQLite":

            conn = sqlite3.connect(connection\_params)

        else:

            raise ValueError(f"Неизвестный тип БД: {db\_type}")

        cur = conn.cursor()

        cur.execute("DROP TABLE IF EXISTS test\_table;")

        if db\_type == "PostgreSQL":

            cur.execute("""

                CREATE TABLE test\_table (id SERIAL PRIMARY KEY, name VARCHAR(100), value INTEGER);

            """)

        else:

            cur.execute("""

                CREATE TABLE test\_table (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, value INTEGER);

            """)

        conn.commit()

        results['Create'] = test\_create(conn, db\_type)

**Продолжение приложения А**

        results['Read'] = test\_read(conn, db\_type)

        results['Update'] = test\_update(conn, db\_type)

        results['Delete'] = test\_delete(conn, db\_type)

        cur.close()

    except Exception as e:

        print(f" Произошла ошибка во время тестов {db\_type}: {e}")

        return None

    finally:

        if conn:

            conn.close()

    return results

# --- Тестовые функции (возвращают время) ---

def test\_create(conn, db\_type):

    cur = conn.cursor()

    start\_time = time.time()

    for i in range(NUM\_RECORDS):

        if db\_type == "PostgreSQL":

            cur.execute("INSERT INTO test\_table (name, value) VALUES (%s, %s);", (f"User {i}", i))

        else:

            cur.execute("INSERT INTO test\_table (name, value) VALUES (?, ?);", (f"User {i}", i))

    conn.commit()

    end\_time = time.time()

    cur.close()

    return end\_time - start\_time

def test\_read(conn, db\_type):

    cur = conn.cursor()

    start\_time = time.time()

    for i in range(1, NUM\_RECORDS + 1):

        if db\_type == "PostgreSQL":

             cur.execute("SELECT \* FROM test\_table WHERE id = %s;", (i,))

        else:

             cur.execute("SELECT \* FROM test\_table WHERE id = ?;", (i,))

        cur.fetchone()

    end\_time = time.time()

    cur.close()

    return end\_time - start\_time

def test\_update(conn, db\_type):

    cur = conn.cursor()

    start\_time = time.time()

    for i in range(1, NUM\_RECORDS + 1):

        if db\_type == "PostgreSQL":

            cur.execute("UPDATE test\_table SET value = %s WHERE id = %s;", (i \* 2, i))

        else:

            cur.execute("UPDATE test\_table SET value = ? WHERE

**Продолжение приложения А**

id = ?;", (i \* 2, i))

    conn.commit()

    end\_time = time.time()

    cur.close()

    return end\_time - start\_time

def test\_delete(conn, db\_type):

    cur = conn.cursor()

    start\_time = time.time()

    for i in range(1, NUM\_RECORDS + 1):

        if db\_type == "PostgreSQL":

            cur.execute("DELETE FROM test\_table WHERE id = %s;", (i,))

        else:

            cur.execute("DELETE FROM test\_table WHERE id = ?;", (i,))

    conn.commit()

    end\_time = time.time()

    cur.close()

    return end\_time - start\_time

# --- ФУНКЦИИ ДЛЯ АНАЛИЗА И ВИЗУАЛИЗАЦИИ ---

def analyze\_and\_plot\_results(all\_results):

    """

    Принимает словарь со всеми результатами, вычисляет статистику,

    печатает итоговую таблицу и строит графики.

    """

    print("\n" + "="\*80)

    print("--- ИТОГОВЫЕ РЕЗУЛЬТАТЫ ---")

    print(f"Количество записей в каждом тесте: {NUM\_RECORDS}")

    print(f"Количество запусков: {NUM\_RUNS}\n")

    print(f"{'База данных':<12} | {'Операция':<10} | {'Среднее время (сек)':<22} | {'Дисперсия':<20}")

    print("-" \* 75)

    stats = {} # Словарь для хранения вычисленной статистики для графиков

    for db\_type, operations in all\_results.items():

        stats[db\_type] = {'mean': [], 'variance': [], 'ops': []}

        for op\_name, times in operations.items():

            if not times:

                print(f"{db\_type:<12} | {op\_name:<10} | {'Нет данных':<22} | {'Нет данных':<20}")

                continue

            mean\_time = np.mean(times)

            variance\_time = np.var(times)

            stats[db\_type]['mean'].append(mean\_time)

            stats[db\_type]['variance'].append(variance\_time)

            stats[db\_type]['ops'].append(op\_name)

**Продолжение приложения А**

            print(f"{db\_type:<12} | {op\_name:<10} | {mean\_time:<22.6f} | {variance\_time:<20.10f}")

        print("-" \* 75)

    # Построение графиков

    plot\_charts(stats)

def plot\_charts(stats):

    """Строит и сохраняет гистограммы на основе статистики."""

    labels = stats['PostgreSQL']['ops']

    postgres\_means = stats['PostgreSQL']['mean']

    sqlite\_means = stats['SQLite']['mean']

    postgres\_vars = stats['PostgreSQL']['variance']

    sqlite\_vars = stats['SQLite']['variance']

    x = np.arange(len(labels))  # the label locations

    width = 0.35  # the width of the bars

    # --- График 1: Среднее время ---

    fig1, ax1 = plt.subplots(figsize=(12, 7))

    rects1 = ax1.bar(x - width/2, postgres\_means, width, label='PostgreSQL', color='skyblue')

    rects2 = ax1.bar(x + width/2, sqlite\_means, width, label='SQLite', color='lightcoral')

    ax1.set\_ylabel('Среднее время (секунды)')

    ax1.set\_title('Сравнение среднего времени выполнения CRUD-операций')

    ax1.set\_xticks(x)

    ax1.set\_xticklabels(labels)

    ax1.legend()

    ax1.grid(True, linestyle='--', alpha=0.6)

    fig1.tight\_layout()

    plt.savefig('mean\_time\_comparison.png') # Сохраняем график в файл

    print("\nГрафик среднего времени сохранен в файл 'mean\_time\_comparison.png'")

    # --- График 2: Дисперсия ---

    fig2, ax2 = plt.subplots(figsize=(12, 7))

    rects3 = ax2.bar(x - width/2, postgres\_vars, width, label='PostgreSQL', color='skyblue')

    rects4 = ax2.bar(x + width/2, sqlite\_vars, width, label='SQLite', color='lightcoral')

    ax2.set\_ylabel('Дисперсия (сек^2)')

    ax2.set\_title('Сравнение дисперсии времени выполнения CRUD-операций')

    ax2.set\_xticks(x)

    ax2.set\_xticklabels(labels)

    ax2.legend()

    ax2.grid(True, linestyle='--', alpha=0.6)

    fig2.tight\_layout()

    plt.savefig('variance\_comparison.png') # Сохраняем график в файл

**Продолжение приложения А**

    print("График дисперсии сохранен в файл 'variance\_comparison.png'")

    plt.show() # Показываем оба графика на экране

# --- ОСНОВНАЯ ТОЧКА ВХОДА В ПРОГРАММУ ---

if \_\_name\_\_ == "\_\_main\_\_":

    all\_results = {

        "PostgreSQL": {"Create": [], "Read": [], "Update": [], "Delete": []},

        "SQLite":     {"Create": [], "Read": [], "Update": [], "Delete": []}

    }

    print(f"Начинаем {NUM\_RUNS} тестовых запусков для PostgreSQL...")

    for i in range(NUM\_RUNS):

        print(f"  Запуск {i+1}/{NUM\_RUNS}...")

        single\_run\_results = run\_single\_test\_cycle("PostgreSQL", POSTGRES\_PARAMS)

        if single\_run\_results:

            for op\_name, time\_val in single\_run\_results.items():

                all\_results["PostgreSQL"][op\_name].append(time\_val)

    print(f"\nНачинаем {NUM\_RUNS} тестовых запусков для SQLite...")

    for i in range(NUM\_RUNS):

        print(f"  Запуск {i+1}/{NUM\_RUNS}...")

        single\_run\_results = run\_single\_test\_cycle("SQLite", 'test.db')

        if single\_run\_results:

            for op\_name, time\_val in single\_run\_results.items():

                all\_results["SQLite"][op\_name].append(time\_val)

    analyze\_and\_plot\_results(all\_results)