

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра Информатики
кафедра

**ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ
НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

Кафедра “Информатика”
место прохождения практики

Сравнение скорости CRUD операций PostgreSQL с SQLite
тема

Руководитель

подпись, дата

А.Н. Пупков
инициалы, фамилия

Студент

КИ19-17/1Б, 031941855
номер группы, зачетной книжки

подпись, дата

В.А. Жолобов
инициалы, фамилия

Красноярск 2021

СОДЕРЖАНИЕ

Введение.....	3
1 Цель.....	4
2 Календарный план.....	5
3 Описание CRUD операций.....	6
4 Docker.....	7
5 PostgreSQL.....	8
5.1 Описание.....	8
5.2 Установка и запуск.....	8
6 SQLite.....	9
6.1 Описание.....	9
6.2 Установка и запуск.....	9
7 Сравнение скорости SQLite и PostgreSQL от создателей SQLite.....	10
8 Скрипты для измерения времени выполнения CRUD операций.....	11
9 Сравнение скоростей выполнения CRUD операций.....	18
9.1 Предисловие.....	18
9.2 Параметры машины и версии ПО.....	18
9.3 Запуск скриптов.....	18
9.4 Результаты.....	19
9.5 Вывод.....	22
Заключение.....	24
Список использованных источников.....	25

Введение

Для сравнения были выбраны СУБД PostgreSQL и SQLite. Были установлены докер контейнеры с выбранными СУБД, написаны скрипты на Java 11 для сравнения скорости выполнения CRUD операций.

1 Цель

Сравнить скорости выполнения CRUD операций в СУБД PostgreSQL и SQLite.

Для выполнения работы необходимо выполнить следующие задачи:

1. Описать что такое CRUD операции;
2. Описать как работают хранилища данных, ссылаясь на соответствующую документацию;
3. Найти информацию о том, почему скорость CRUD операций хранилищ отличается, провести сравнительный анализ для каждой операции с детальным и обоснованным объяснением (со ссылками на источники);
4. Установить docker toolbox (или более свежее решение);
5. Скачать контейнеры с соответствующими базами данных;
6. Написать два простых скрипта выполняющих CRUD операции для каждой из пары баз данных и измеряющих время выполнения;
7. Каждый эксперимент провести несколько раз;
8. Указать параметры машины, на которой проводились исследования;
9. Указать количество итераций для каждого эксперимента;
10. Привести значения математического ожидания и дисперсии для каждого результата;
11. Сделать графики с пояснениями;
12. Сделать выводы о том, почему в данных хранилищах имеются различия в выполнении CRUD операций, чем это вызвано и как дизайн системы влияет на данный параметр.

2 Календарный план

В Таблице 1 приведен календарный план исследования.

Таблица 1 – Календарный план исследования

Мероприятия	Сроки выполнения
1 Выбор темы исследования	до 27 июня 2021 года
2 Изучение материалов, посвященных выбранной теме	до 1 июля 2021 года
3 Определение цели и объекта исследования	до 1 июля 2021 года
4 Выбор методов, определяющих эффективный способ достижения результата	до 4 июля 2021 года
5 Проведение экспериментов	до 7 июля 2021 года
6 Подготовка и сдача отчета	до 8 июля 2021 года

3 Описание CRUD операций

CRUD – акроним, состоит из первых букв четырех операций, наиболее часто используемых при работе с базами данных:

1. Create – создание;
2. Read – чтение;
3. Update – модификация;
4. Delete – удаление.

4 Docker

Docker – программное обеспечение, которое позволяет упаковать приложение со всем его окружением и зависимостями в контейнер. Благодаря этому можно запускать одну программу на разных машинах с одинаковым результатом.

На Листинге 1 представлена использованная последовательность команд для установки Docker на Ubuntu 20.04.

Листинг 1 – Последовательность команд для установки Docker

```
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -
o /usr/share/keyrings/docker-archive-keyring.gpg

echo \
    "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

5 PostgreSQL

5.1 Описание

PostgreSQL – серверная СУБД. Написана на языке С. Дистрибутив PostgreSQL 13.3 весит 260MB.

PostgreSQL создает для каждой таблицы и индекса отдельный файл [1].

PostgreSQL может обработать неограниченное количество дисковой памяти [2].

PostgreSQL создает отдельный процесс для каждого клиента. Клиенты могут работать параллельно, одновременно внося изменения [3].

PostgreSQL - самая продвинутая СУБД. В PostgreSQL есть множество функций, которых нет у других субд: поддержка JSON и индексируемого JSONB [4], многомерные массивы, создание собственных типов данных. PostgreSQL выполняет 170 из 177 требований для полного соответствия стандарту SQL [5].

При получении запроса, сервер выполняет синтаксическую проверку запроса и создает дерево запроса. Система правил принимает дерево запроса и преобразует его. Планировщик принимает дерево запроса и создает план запроса. Исполнитель рекурсивно проходит по дереву и выполняет инструкции [6].

5.2 Установка и запуск

На Листинге 2 представлена последовательность команд для установки и запуска PostgreSQL в Docker.

Листинг 2 – Установка и запуск PostgreSQL в Docker

```
sudo docker pull postgres:alpine
```

```
sudo docker run --name postgres -e POSTGRES_PASSWORD=password -d -p 0.0.0.0:25565:5432 postgres:alpine
```


6 SQLite

6.1 Описание

SQLite - СУБД, которая осуществляет доступ к базе данных напрямую, без сервера [7]. Написана на языке C.

SQLite – простая СУБД. Ее дистрибутив имеет размер от 500KiB до 10MB. SQLite не требует установки и конфигурации.

SQLite хранит всю базу данных в одном файле [8].

SQLite может обработать до 281TB дисковой памяти, но, часто, файловая система не позволит создать настолько большой файл [9].

SQLite не тратит дополнительных ресурсов на обслуживание клиентов, но клиенты не могут вносить изменения одновременно. SQLite накладывает блокировки на файл, когда один из клиентов вносит изменения [10].

В SQLite запрос попадает в процессор SQL команд, преобразуется в байт-код, который выполняется виртуальной машиной. Результат получает менеджер страниц, который фиксирует результат в базе данных [11].

SQLite может работать в оперативной памяти [12].

SQLite может работать быстрее, чем прямое обращение к файлу через файловую систему [13].

6.2 Установка и запуск

В Docker Hub нет официального изображения SQLite, поэтому было установлено изображение alpine. В alpine была вручную установлена SQLite.

На Листинге 3 представлена последовательность команд для установки и запуска SQLite в Docker.

Листинг 3 – Установка и запуск SQLite в Docker

```
sudo docker pull alpine
sudo docker run -d -it alpine
sudo docker exec -it alpine /bin/sh
apk add sqlite
```

7 Сравнение скорости SQLite и PostgreSQL от создателей SQLite

Было найдено исследование от создателей SQLite, в котором сравниваются скорости CRUD операций в SQLite 2.7.6 и PostgreSQL 7.1.3 на машине с процессором 1.6GHz Athlon и 1GB оперативной памяти на операционной системе RedHat Linux 7.2 [14].

Согласно найденному исследованию, в PostgreSQL 1000 операций вставки выполняются за 4.373 секунды, в синхронизированной версии SQLite за 13.061 секунд, в SQLite с отключенной синхронизацией за 0.223 секунды.

100 операций выбора без индекса в PostgreSQL выполняются за 3.629 секунд, в синхронизированной версии SQLite за 2.494 секунд, в SQLite с отключенной синхронизацией за 2.526 секунд.

1000 операций обновления данных в PostgreSQL выполняются за 1.739 секунд, в синхронизированной версии SQLite за 0.637 секунд, в SQLite с отключенной синхронизацией за 0.638 секунд.

Удаление строк по шаблону %fifty% в PostgreSQL выполняются за 1.509 секунд, в синхронизированной версии SQLite за 4.004 секунд, в SQLite с отключенной синхронизацией за 0.560 секунд.

Во всех тестах самой быстрой оказалась SQLite с отключенной синхронизацией. Это связано с тем, что SQLite записывает данные в файл напрямую, а PostgreSQL через обращение к серверу. SQLite с отключенной синхронизацией также не тратит времени на вызов fsync(). При работе с одним клиентом выигрыш в скорости SQLite очевиден, но, если с базой данных будет работать большое количество клиентов, то преимущество будет на стороне PostgreSQL, так как PostgreSQL позволяет клиентам вносить изменения одновременно [14].

8 Скрипты для измерения времени выполнения CRUD операций

Для измерения времени выполнения CRUD операций был написан скрипты на Java 11 с использованием драйверов JDBC для SQLite и PostgreSQL.

На Листинге 4 представлен код измерения времени операций вставки, на Листинге 5 представлен код измерения времени операций выбора, на Листинге 6 представлен код измерения времени операций модификации, на Листинге 7 представлен код измерения времени операций удаления, на Листинге 8 представлен код запуска всех тестов, на Листинге 9 представлен код подключения к PostgreSQL, на Листинге 10 представлен код подключения к SQLite.

Листинг 4 – Измерение времени операций вставки

```
package test;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.concurrent.TimeUnit;

public final class InsertTest implements Test {
    private final Connection connection;
    private final int rowsNumber;

    public InsertTest(Connection connection, int rowsNumber) {
        this.connection = connection;
        this.rowsNumber = rowsNumber;
    }

    @Override
    public long test() throws Exception {
        try(PreparedStatement insertStatement =
            connection.prepareStatement("INSERT INTO test
VALUES (?, ?, ?)") ) {
            char c = 'a';
            double d = 1.0;
            long startTime = System.nanoTime();
            for(int i = 0; i < rowsNumber; i++) {
```

```

insertStatement.setInt(1, i);
insertStatement.setString(2, String.valueOf(c));

```

Продолжение Листинга 4

```

insertStatement.setDouble(3, d);
insertStatement.executeUpdate();

c++;
d *= 1.3;
}
long endTime = System.nanoTime();

return TimeUnit.MILLISECONDS.convert(endTime - startTime,
TimeUnit.NANOSECONDS);
}
}
}

```

Листинг 5 – Измерение времени операций выбора

```

package test;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.concurrent.TimeUnit;

public final class SelectTest implements Test {
    private final Connection connection;
    private final int rowsNumber;

    public SelectTest(Connection connection, int rowsNumber) {
        this.connection = connection;
        this.rowsNumber = rowsNumber;
    }

    @Override
    public long test() throws Exception {
        try(PreparedStatement selectStatement
            = connection.prepareStatement("SELECT * FROM test WHERE x
= ?")) {
            long startTime = System.nanoTime();
            for(int i = 0; i < rowsNumber; i++) {
                selectStatement.setInt(1, i);
                selectStatement.executeQuery();
            }
        }
    }
}

```

```

    }
    long endTime = System.nanoTime();

```

Продолжение Листинга 5

```

        return      TimeUnit.MILLISECONDS.convert(endTime      -      startTime,
TimeUnit.NANOSECONDS);
    }
}

```

Листинг 6 – Измерение времени операций модификации

```

package test;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.concurrent.TimeUnit;

public final class UpdateTest implements Test {
    private final Connection connection;
    private final int rowsNumber;

    public UpdateTest(Connection connection, int rowsNumber) {
        this.connection = connection;
        this.rowsNumber = rowsNumber;
    }

    @Override
    public long test() throws Exception {
        try(PreparedStatement updateStatement =
            connection.prepareStatement("UPDATE test SET x=? where
x=?")) {
            long startTime = System.nanoTime();
            for(int i = 1; i < rowsNumber; i++) {
                updateStatement.setInt(1, i - 1);
                updateStatement.setInt(2, i);
                updateStatement.executeUpdate();
            }
            long endTime = System.nanoTime();

            return      TimeUnit.MILLISECONDS.convert(endTime      -      startTime,
TimeUnit.NANOSECONDS);
        }
    }
}

```

```
    }  
}
```

Листинг 7 – Измерение времени операций удаления

```
package test;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.util.concurrent.TimeUnit;  
  
public final class DeleteTest implements Test {  
    private final Connection connection;  
    private final int rowsNumber;  
  
    public DeleteTest(Connection connection, int rowsNumber) {  
        this.connection = connection;  
        this.rowsNumber = rowsNumber;  
    }  
  
    @Override  
    public long test() throws Exception {  
        try(PreparedStatement deleteStatement =  
            connection.prepareStatement("DELETE FROM test WHERE x=?"))  
        {  
            long startTime = System.nanoTime();  
            for(int i = 0; i < rowsNumber; i++) {  
                deleteStatement.setInt(1, i);  
                deleteStatement.executeUpdate();  
            }  
            long endTime = System.nanoTime();  
  
            return      TimeUnit.MILLISECONDS.convert(endTime      -      startTime,  
TimeUnit.NANOSECONDS);  
        }  
    }  
}
```

Листинг 8 – Код запуска всех тестов

```
package test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Properties;

public class Tests {
    private final String url;
    private final Properties properties;
    private final int rowsNumber;

    public Tests(String url, Properties properties, int rowsNumber) {
        this.url = url;
        this.properties = properties;
        this.rowsNumber = rowsNumber;
    }

    public void run() throws Exception {
        try(Connection connection = DriverManager.getConnection(url,
properties);
            Statement statement = connection.createStatement()) {
            System.out.println("Connection successfully established");

            statement.executeUpdate("CREATE TABLE test(x integer, y char, z
double precision)");

            System.out.println("Insert test time: " + new
InsertTest(connection, rowsNumber).test() + "ms");
            System.out.println("Select test time: " + new
SelectTest(connection, rowsNumber).test() + "ms");
            System.out.println("Update test time: " + new
UpdateTest(connection, rowsNumber).test() + "ms");
            System.out.println("Delete test time: " + new
DeleteTest(connection, rowsNumber).test() + "ms");

            statement.executeUpdate("DROP TABLE test;");
        }
    }
}
```

Листинг 9 – Подключение к PostgreSQL

```
import test.Tests;

import java.util.Properties;

public class PostgreSQL {
    public static void main(String[] args) throws Exception {
        if(args.length < 6) {
            System.out.println("Insufficient arguments passed. You need to
pass: ");

            System.out.println("1) host");
            System.out.println("2) port");
            System.out.println("3) database");
            System.out.println("4) username");
            System.out.println("5) password");
            System.out.println("6) rows number");
            return;
        }

        String host = args[0];
        String port = args[1];
        String database = args[2];
        String username = args[3];
        String password = args[4];
        int rowsNumber = Integer.parseInt(args[5]);

        Properties properties = new Properties();
        properties.setProperty("user", username);
        properties.setProperty("password", password);

        Tests tests = new Tests("jdbc:postgresql://" + host + ":" + port + "/"
+ database,
            properties,
            rowsNumber);

        tests.run();
    }
}
```


Листинг 10 – Подключение к SQLite

```
import test.Tests;

import java.util.Properties;

public class SQLite {
    public static void main(String[] args) throws Exception {
        if(args.length < 2) {
            System.out.println("Insufficient arguments passed. You need to
pass: ");
            System.out.println("1) path to database");
            System.out.println("2) rows number");
            return;
        }

        String database = args[0];
        int rowsNumber = Integer.parseInt(args[1]);

        Properties properties = new Properties();

        Class.forName("org.sqlite.JDBC");
        Tests tests = new Tests("jdbc:sqlite:" + database,
            properties,
            rowsNumber);
        tests.run();
    }
}
```

Для запуска скрипта для PostgreSQL нужно передать через аргументы командой строки адрес базы данных, порт базы данных, название базы данных, имя пользователя, пароль пользователя и число итераций эксперимента.

Для запуска скрипта для SQLite нужно передать через аргументы командной строки путь до базы данных и число итераций эксперимента.

Время на подключение к базе данных не учитывается.

9 Сравнение скоростей выполнения CRUD операций

9.1 Предисловие

Так как производительность SQLite сильно падает из-за постоянной синхронизации диска было решено сравнивать 2 версии SQLite: с синхронизацией диска и без нее.

При запуске скриптов, указывалось число итераций 5000. Скрипты запускались по 10 раз для каждой СУБД.

9.2 Параметры машины и версии ПО

Сравнение проводилось на Cloud машине с 1ГБ оперативной памяти, процессором AMD EPYC 7551 32-Core Processor с двумя доступными CPU и 2 потоками на ядро. Операционная система – Ubuntu 20.04. Доступ к постоянной памяти машины осуществляется по сети и шифруется.

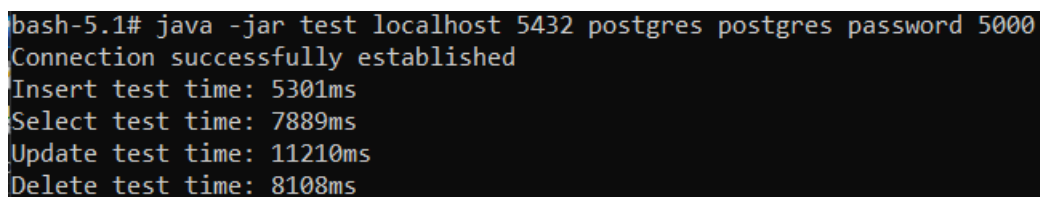
Для запуска контейнеров СУБД был установлен Docker 20.10.7.

Для сравнения СУБД были установлены контейнеры с PostgreSQL 13.3 и SQLite 3.35.5

Для измерения времени выполнения CRUD операций использовалась Java 11.

9.3 Запуск скриптов

На Рисунке 1 показан запуск скрипта для PostgreSQL.



```
bash-5.1# java -jar test localhost 5432 postgres postgres password 5000
Connection successfully established
Insert test time: 5301ms
Select test time: 7889ms
Update test time: 11210ms
Delete test time: 8108ms
```

Рисунок 1 – Запуск скрипта для PostgreSQL

На Рисунке 2 показан запуск скрипта для SQLite с синхронизацией диска.

```

/ # java -jar javatest_sync test 5000
Connection successfully established
Insert test time: 26958ms
Select test time: 1608ms
Update test time: 29523ms
Delete test time: 28640ms

```

Рисунок 2 – Запуск скрипта для SQLite с синхронизацией диска

На Рисунке 3 показан запуск скрипта для SQLite без синхронизации диска.

```

/ # java -jar javatest_nosync test 5000
Connection successfully established
Insert test time: 1415ms
Select test time: 1668ms
Update test time: 4027ms
Delete test time: 2659ms

```

Рисунок 3 – Запуск скрипта для SQLite без синхронизации диска

9.4 Результаты

На Рисунке 4 показан график времени операций вставки в сравниваемых СУБД.

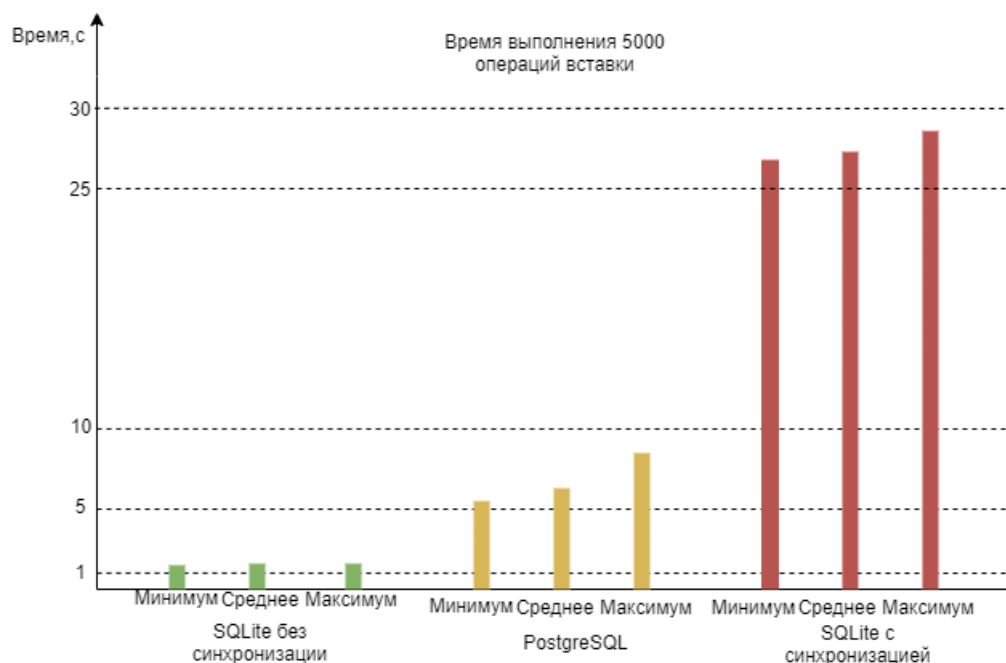


Рисунок 4 – График времени операций вставки

SQLite без синхронизации, ожидаемого, выполнила операции намного быстрее PostgreSQL и SQLite с синхронизацией. Результат объясняется тем, что SQLite записывает данные напрямую в файл и с отключенной синхронизацией не тратит дополнительного времени на обеспечение безопасного доступа к файлу.

SQLite с синхронизацией выполнил операции медленнее, чем PostgreSQL, хотя ожидался другой результат. Это связано с тем, что доступ к постоянной памяти производится по сети и данные шифруются, а синхронизация удаленного диска после каждой операции не могла не сказаться на производительности.

Для SQLite без синхронизации математическое ожидание: 1.4 секунд, дисперсия: 0.03 секунд.

Для PostgreSQL математическое ожидание: 6.673 секунд, дисперсия: 17.436 секунд.

Для SQLite с синхронизацией математическое ожидание: 27.268 секунд, дисперсия: 244.57 секунд.

На Рисунке 5 показан график времени операций выбора.

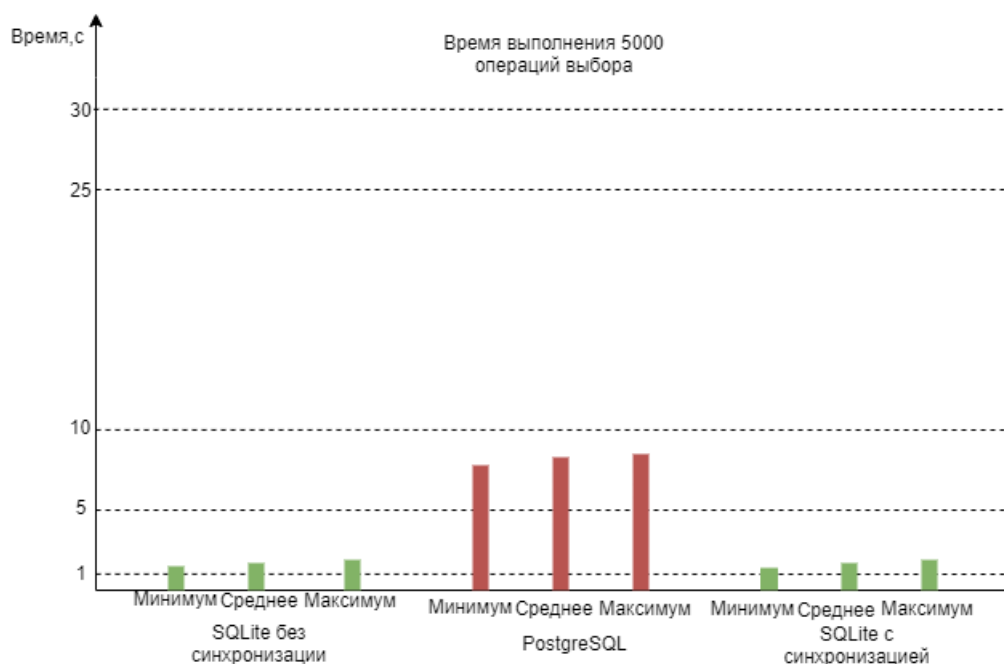


Рисунок 5 – График времени операций выбора

Здесь SQLite с синхронизацией показывает результат такой же, как и без синхронизации потому, что операция выбора не изменяет базу данных и не нужно синхронизировать диск.

Для SQLite без синхронизации математическое ожидание: 1.642 секунд, дисперсия: 0.911 секунд.

Для PostgreSQL математическое ожидание: 8.283 секунд, дисперсия: 19.047 секунд.

Для SQLite с синхронизацией математическое ожидание: 1.623 секунд, дисперсия: 1.013 секунд.

На Рисунке 6 показан график времени операций модификации.

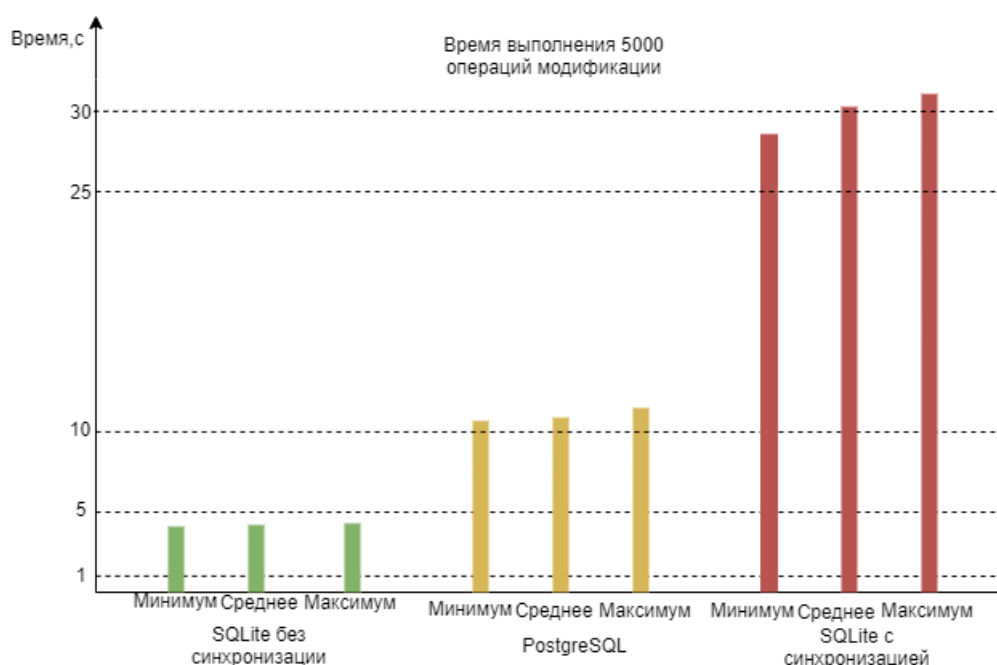


Рисунок 6 – График времени выполнения операций модификации

Здесь SQLite с синхронизацией снова показывает худшее время потому, что нужно синхронизировать диск.

Для SQLite без синхронизации математическое ожидание: 4.059 секунд, дисперсия: 1.228 секунд.

Для PostgreSQL математическое ожидание: 10.941 секунд, дисперсия: 0.868 секунд.

Для SQLite с синхронизацией математическое ожидание: 30.23 секунд, дисперсия: 105.707 секунд.

На Рисунке 7 показан график времени выполнения операций удаления.

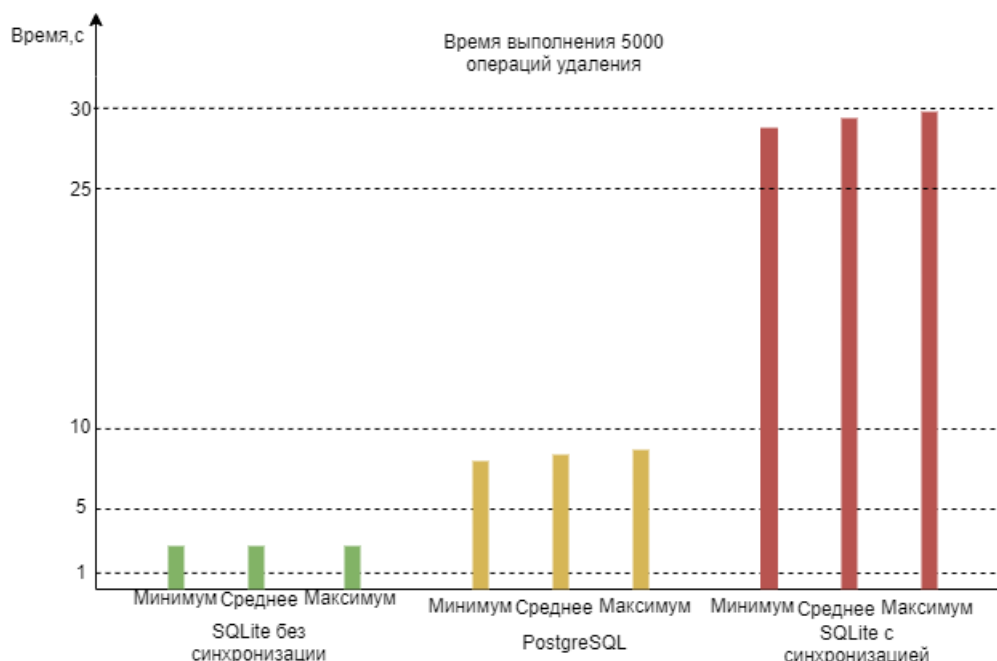


Рисунок 7 – График времени выполнения операций удаления

Ситуаций не изменилась. SQLite с синхронизацией показывает худший результат, когда нужно синхронизировать диск.

Для SQLite без синхронизации математическое ожидание: 2.639 секунд, дисперсия: 0.902 секунд.

Для PostgreSQL математическое ожидание: 8.193 секунд, дисперсия: 67.96 секунд.

Для SQLite с синхронизацией математическое ожидание: 29.215 секунд, дисперсия: 34.607 секунд.

9.5 Вывод

SQLite записывает данные в файл напрямую, а PostgreSQL через обращение к серверу. SQLite с отключенной синхронизацией также не тратит времени на синхронизацию. Поэтому во всех операциях, связанных с изменением базы данных SQLite без синхронизации будет быстрее.

SQLite с синхронизацией уступает PostgreSQL во всех операциях с изменением данных, хотя все найденные источники утверждают, что SQLite намного быстрее при работе с 1 клиентом [15] [16] [17]. Причины расхождения результатов:

1. При сравнении использовался официальный докер контейнер PostgreSQL. Сервер PostgreSQL был настроен создателями для повышения производительности. Официального SQLite докер контейнера не существует, поэтому SQLite была установлена вручную, дополнительная настройка не проводилась.

2. Специфика работы cloud машины, на которой проводилось сравнение [18] [19]. Обращение к постоянной памяти происходит по сети, и каждая операция шифруется. Вызов `fsync()` при работе с удаленным диском тратит больше времени. PostgreSQL не синхронизирует диск, конкурентный доступ регулирует сервер.

Заключение

Были изучены основы работы с ПО Docker, СУБД PostgreSQL и SQLite.

Было проведено сравнение скорости выполнения CRUD операций СУБД PostgreSQL и SQLite. По его результатам PostgreSQL выполнила операции быстрее синхронизированной SQLite, хотя ожидался другой результат. Это вызвано тем, что исследование проводилось на cloud машине, на которой доступ к постоянной памяти производится по сети и шифруется. Полученные результаты нельзя интерпретировать на обычные машины, где постоянная память и вычислительные ресурсы находятся рядом.

Список использованных источников

- 1) Документация PostgreSQL [Электронный ресурс] : файловая система.
– Режим доступа: <https://postgrespro.ru/docs/postgresql/13/storage-file-layout>;
- 2) Документация PostgreSQL [Электронный ресурс] : ограничения –
Режим доступа: <https://postgrespro.ru/docs/postgresql/13/limits>;
- 3) Документация PostgreSQL [Электронный ресурс] : подключение
клиентов – Режим доступа: <https://postgrespro.ru/docs/postgresql/13/connect-estab>;
- 4) Документация PostgreSQL [Электронный ресурс] : тип данных JSON –
Режим доступа: <https://postgrespro.ru/docs/postgrespro/10/datatype-json#JSON-INDEXING>;
- 5) Документация PostgreSQL [Электронный ресурс] : особенности –
Режим доступа: <https://postgrespro.ru/docs/postgresql/13/features>;
- 6) Документация PostgreSQL [Электронный ресурс] : путь запроса –
Режим доступа: <https://postgrespro.ru/docs/postgresql/13/query-path>;
- 7) Документация SQLite [Электронный ресурс] : краткое описание –
Режим доступа: <https://sqlite.org/about.html>;
- 8) Документация SQLite [Электронный ресурс] : особенные черты –
Режим доступа: <https://sqlite.org/different.html>;
- 9) Документация SQLite [Электронный ресурс] : ограничения – Режим
доступа: <https://www.sqlite.org/limits.html>;
- 10) Документация SQLite [Электронный ресурс] : когда стоит
использовать – Режим доступа: <https://www.sqlite.org/whentouse.html>;
- 11) Документация SQLite [Электронный ресурс] : архитектура системы
– Режим доступа: <https://www.sqlite.org/arch.html>;
- 12) Документация SQLite [Электронный ресурс] : запуск SQLite в
оперативной памяти – Режим доступа: <https://www.sqlite.org/inmemorydb.html>;
- 13) Документация SQLite [Электронный ресурс] : быстрее чем файловая
система – Режим доступа: <https://sqlite.org/fasterthanfs.html>;

14) Документация SQLite [Электронный ресурс] : сравнение скоростей СУБД – Режим доступа: <https://sqlite.org/speed.html>;

15) Сборник информации об IT heikkisiltala [Электронный ресурс] : сравнение СУБД PostgreSQL и SQLite – Режим доступа: https://heikkisiltala.net/wiki/en/information_technology/postgresql_vs_sqlite;

16) Сборник информации о базах данных tableplus [Электронный ресурс] : сравнение СУБД PostgreSQL и SQLite – Режим доступа: <https://tableplus.com/blog/2018/08/sqlite-vs-postgresql-which-database-to-use-and-why.html>;

17) Интернет портал hevo [Электронный ресурс] : сравнение СУБД PostgreSQL и SQLite – Режим доступа: <https://hevodata.com/learn/sqlite-vs-postgresql/#factors>;

18) Документация Oracle Cloud [Электронный ресурс] : описание хранилищ облачных машин - Режим доступа: <https://docs.oracle.com/enus/iaas/Content/Block/Concepts/bootvolumes.htm>;

19) Документация Oracle Cloud [Электронный ресурс] : описание шифрования операций - Режим доступа: <https://docs.oracle.com/enus/iaas/Content/Block/Concepts/overview.htm#BlockVolumeEncryption>;

20) Платформа облачных вычислений и машинного обучения logz [Электронный ресурс] : сравнение PostgreSQL и SQLite – Режим доступа: <https://logz.io/blog/relational-database-comparison/>;

21) Портал о компьютерных науках Geeks For Geeks [Электронный ресурс] : различия между PostgreSQL и SQLite – Режим доступа: <https://www.geeksforgeeks.org/difference-between-sqlite-and-postgresql/>;

22) Видеохостинг YouTube [Электронный ресурс] : выступление основателя SQLite: основы SQLite – Режим доступа: https://www.youtube.com/watch?v=rtCgnHdRSk0&t=1s&ab_channel=MyCS;

23) Блог Habr [Электронный ресурс] : чем PostgreSQL лучше других SQL – Режим доступа: <https://habr.com/ru/post/282764/>;

24) Провайдер облачных услуг DigitalOcean [Электронный ресурс] : сравнение PostgreSQL и SQLite – Режим доступа: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>.