Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

«Сравнение скорости CRUD операций (MySQL c SQLite)»

тема

Руководитель от университета \_\_\_\_\_\_\_\_\_ А.Н. Пупков

подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 \_\_\_\_\_\_\_\_\_ Е.В. Железкин

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2020

Содержание

[Содержание 2](#_Toc50136312)

[ЗАДАНИЕ 3](#_Toc50136313)

[1 Теоретическая часть 5](#_Toc50136314)

[1.1 CRUD операции 5](#_Toc50136315)

[1.2 СУБД MySQL 5](#_Toc50136316)

[1.3 СУБД SQLite 8](#_Toc50136317)

[2 Экспериментальная часть 10](#_Toc50136318)

[2.1 Выбор языка для написания скриптов 10](#_Toc50136319)

[2.2 Настройка СУБД 10](#_Toc50136320)

[2.3 Характеристики машины 12](#_Toc50136321)

[2.4 Сравнение результатов тестов СУБД 13](#_Toc50136322)

[Список использованных источников 20](#_Toc50136323)

ЗАДАНИЕ

Теоретическая часть:

1. Описать что такое CRUD операции;

2. Описать как работают хранилища данных, ссылаясь на

соответствующую документацию

3. Найти информацию о том, как и почему скорость CRUD операций

хранилищ отличается, провести сравнительный анализ для каждой операции с

детальным и обоснованным объяснением (со ссылками на источники);

4. 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.

Экспериментальная часть:

- Установить docker toolbox (или более свежее решение);

- Скачать контейнеры с соответствующими базами данных;

- Написать два простых скрипта выполняющих CRUD операции для

каждой из пары баз данных и измеряющих время выполнения;

- Каждый эксперимент провести несколько раз, при этом:

- Нужно указать параметры (виртуальной) машины, на которой

проводились исследования (кол-во RAM, CPU, потоков);

- Указать количество итераций для каждого эксперимента;

- Привести значения математического ожидания и дисперсии для каждого

результата;

- Сделать графики с пояснениями;

- 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.

В качестве основы для сравнения использовались СУБД *MySQL* и *SQLite.*

# Теоретическая часть

## CRUD операции

CRUD — акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete). Введён Джеймсом Мартином (англ. James Martin) в 1983 году как стандартная классификация функций по манипуляции данными.

В SQL этим функциям операциям соответствуют операторы Insert (создание записей), Select (чтение записей), Update (редактирование записей), Delete (удаление записей). В некоторых CASE-средствах использовались специализированные CRUD-матрицы или CRUD-диаграммы, в которых для каждой сущности указывалось, какие базовые функции с этой сущностью выполняет тот или иной процесс или та или иная роль. В системах, реализующих доступ к базе данных через API в стиле REST, эти функции реализуются зачастую (но не обязательно) через HTTP-методы POST, GET, PUT и DELETE соответственно. [1][2]

## СУБД MySQL

MySQL —свободная реляционная *колоночная* система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Сферы применения MySQL

Сам факт того, что MySQL является самой популярной СУБД на рынке, говорит об ее универсальности и предпочтительности в большинстве ситуаций. В частности, ее используют в следующих ситуациях:

* при распределённых операциях, когда функционала SQLite (другая популярная система) не хватает;
* когда требуется обеспечить высокий уровень безопасности, что MySQL делает с успехом;
* для работы с интернет-страницами и веб-приложениями, поскольку MySQL является наиболее удобной СУБД для этой сферы применения;
* при работе со специфическим проектом, где функционал MySQL дает оптимальный результат.
* Тем не менее, существуют ситуации, когда от MySQL всё же следует отказаться в пользу других СУБД. Например, она не годится, если:
* нужно соответствовать стандарту SQL, который данная система поддерживает лишь частично;
* проект предусматривает многопоточность данных, поскольку при осуществлении параллельных операций чтения/записи у MySQL могут возникать проблемы;
* имеющийся функционал MySQL не способен обеспечить весь набор возможностей работы с базой данных.[3][4]

Преимущества MySQL

* Помимо универсальности и распространенности СУБД MySQL обладает целым комплексом важных преимуществ перед другими системами. В частности, следует отметить такие качества как:
* Простота в использовании. MySQL достаточно легко инсталлируется, а наличие множества плагинов и вспомогательных приложений упрощает работу с базами данных.
* Обширный функционал. Система MySQL обладает практически всем необходимым инструментарием, который может понадобиться в реализации практически любого проекта.
* Безопасность. Система изначально создана таким образом, что множество встроенных функций безопасности в ней работают по умолчанию.
* Масштабируемость. Являясь весьма универсальной СУБД, MySQL в равной степени легко может быть использована для работы и с малыми, и с большими объемами данных.
* Скорость. Высокая производительность системы обеспечивается за счет упрощения некоторых используемых в ней стандартов.

Недостатки MySQL

* Как и любой программный продукт, система MySQL имеет определенные ограничения в своем функционале, что не позволяет использовать ее для работы с приложениями, имеющими некоторые специфические требования. К недостаткам этой СУБД относятся:
* Недостаточная надежность. В вопросах надежности некоторых процессов по работе с данными (например, связь, транзакции, аудит) MySQL уступает некоторым другим СУБД.
* Низкая скорость разработки. Как и многим другим программным продуктам с открытым кодом, MySQL не достает некоторого технического совершенства, что порой сказывается на эффективности процессов разработки.[5][6]

## СУБД SQLite

SQLite — компактная встраиваемая *колоночная* СУБД. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O’Reilly Open Source Awards. [7]

Преимущества SQLite:

* SQLite не требует отдельного серверного процесса или системы для работы (без сервера).
* SQLite поставляется с нулевой конфигурацией, что означает отсутствие необходимости в настройке или администрировании.
* Полная база данных SQLite хранится в одном межплатформенном файле на диске.
* SQLite очень маленький и легкий, менее 400 КБ полностью сконфигурирован или менее 250 КБ без дополнительных функций.
* SQLite является автономным, что означает отсутствие внешних зависимостей.
* Транзакции SQLite полностью совместимы с ACID, что обеспечивает безопасный доступ из нескольких процессов или потоков.
* SQLite поддерживает большинство функций языка запросов, представленных в стандарте SQL92 (SQL2).
* SQLite написан на ANSI-C и предоставляет простой и удобный API.
* SQLite доступен в UNIX (Linux, Mac OS-X, Android, iOS) и Windows (Win32, WinCE, WinRT).

SQLite Ограничения:

* Право-наружное соединение - реализовано только LEFT OUTER JOIN.
* Полное наружное соединение - реализовано только LEFT OUTER JOIN.
* ALTER TABLE - поддерживаются варианты RENAME TABLE и ADD COLUMN команды ALTER TABLE. DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT не поддерживаются.
* Поддержка триггеров - триггеры FOR EACH ROW поддерживаются, но не триггеры FOR EACH STATEMENT.
* Просмотры - представления в SQLite доступны только для чтения. Вы не можете выполнять операторы DELETE, INSERT или UPDATE в представлении.[8]

# Экспериментальная часть

## Выбор языка для написания скриптов

Из-за обширного выбора библиотек и относительной простоты их интеграции в проект, для написания скриптов был выбран язык программирования Python.

Для взаимодействия скриптов с СУБД MySQL использовалась библиотека *pymysql* и *sqlite3* для СУБД SQLite соответственно.

Библиотека для измерения времени выполнения скриптов – *datetime.*

## Настройка СУБД

Для СУБД MySQL использовалась серверная обёртка Docker.[9]

Для начала необходимо было скачать и установить Docker Desktop (toolbox) (Рисунок 1).

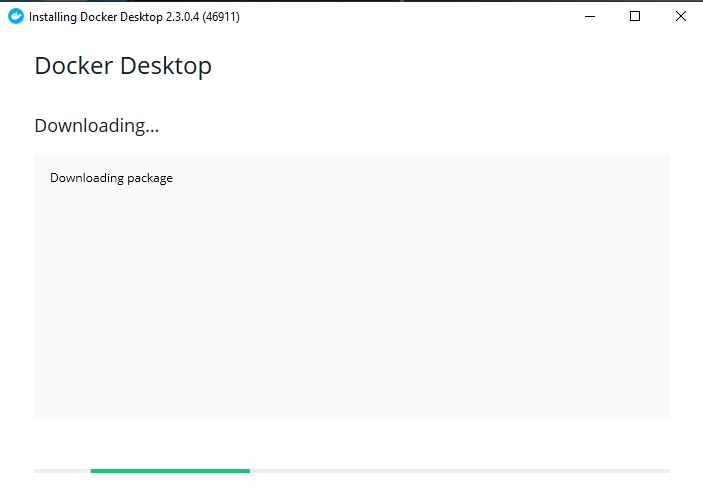


Рисунок 1 – Установка Docker Desktop

Далее скачать и настроить контейнер с MySQL для Docker (рисунок 2, 3).

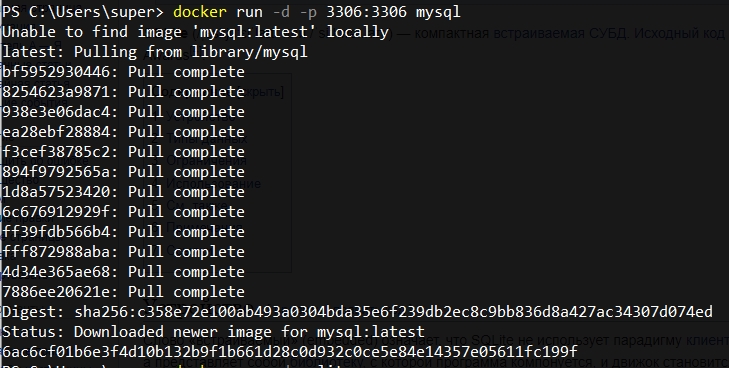


Рисунок 2 – Установка контейнера с MySQL

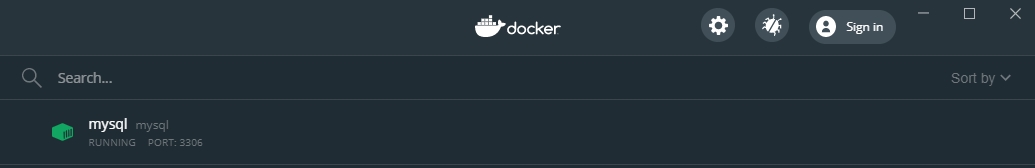


Рисунок 3 – Запущенный сервер MySQL внутри Docker Desktop

СУБД SQLite не является серверной, поэтому оборачивать её в Docker не было необходимости. Для взаимодействия с локальной БД с помощью СУБД SQLite использовался интегрированный в PyCharm Professional плагин (Рисунок 4).

Так же предварительно была создана таблица, с которой выполнялись все взаимодействия (Рисунок 4).

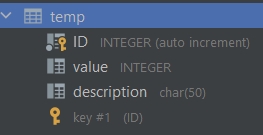


Рисунок 4 – Поля(столбцы) таблицы внутри БД

## Характеристики машины

Все тесты проводились на стационарном компьютере под операционной системой Windows 10 Pro версии 1909.

Компоненты (Рисунок 5):

* ЦП – Intel(R) Core(TM) i5-6400 CPU, разогнанным до частоты 4ГГц
* RAM – 8x2GB с частотой 3200МГц
* Жёсткий диск - WDC WD10EZEX-08WN4A0 с частотой вращения 7200 RPM

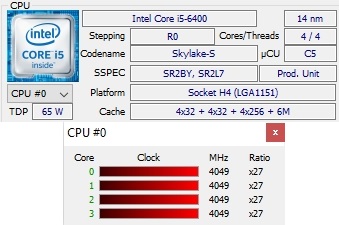


Рисунок 5 – Подробные характеристики процессора

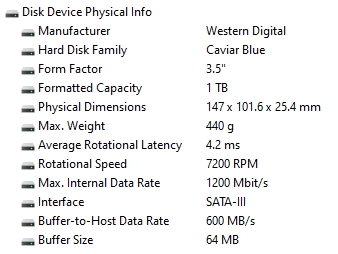


Рисунок 6 – Подробные характеристики диска

## Сравнение результатов тестов СУБД

Чтобы не загружать в данном отчёте область сравнения СУБД, все листинги со скриптами, выполняющими запросы к СУБД, будут приведены в конце работы (Приложение А).

Количество запросов и итераций, направленным к СУБД, было идентично, а именно 10 итераций по 5000 запросов каждого вида.

Как уже было сказано ранее, на рисунке 4 приведена структура таблицы в БД.

Перед работой с таблицей каждой базы данных, таблица каждый раз очищалась (пересоздавалась) с помощью запросов

* *DROP TABLE IF EXISTS* temp;
* *CREATE TABLE* temp (

ID int *NOT NULL AUTO\_INCREMENT*,

value int, description char(50),

*PRIMARY KEY* (ID));

В работе сравнивались все операции типа CRUD. Так как обе СУБД поддерживают язык запросов SQL, то и запросы были практически идентичны:

* Для сравнения скорости запросов типа *CREATE* использовался запрос вида

*INSERT INTO temp (value, description) VALUES* ({value}, {description});

где первый столбец заполняется автоматически (автоинкремент), а два других вставляются непосредственно запросом. В данном случае значение {value} равнялось значению цикла, а значение {description} всегда равнялось «test».

* Для сравнения скорости запросов типа *READ* использовался запрос вида

*SELECT \* FROM temp;*

* Для сравнения скорости запросов типа *UPDATE* использовался запрос вида

*UPDATE temp SET description = ‘new description’;*

соответственно в этом тесте ранее вставленное в столбец *description* значение «test» заменялось на «new description».

* Для сравнения скорости запросов типа *DELETE* использовался запрос вида

*DELETE FROM temp WHERE ID =* {value};

где значение {value} равнялось значению счётчику цикла + 1, чтобы поочерёдно соответствовать каждому значению столбца *ID* в БД.

Результат выполнения скриптов представлен на рисунках 7, 8.

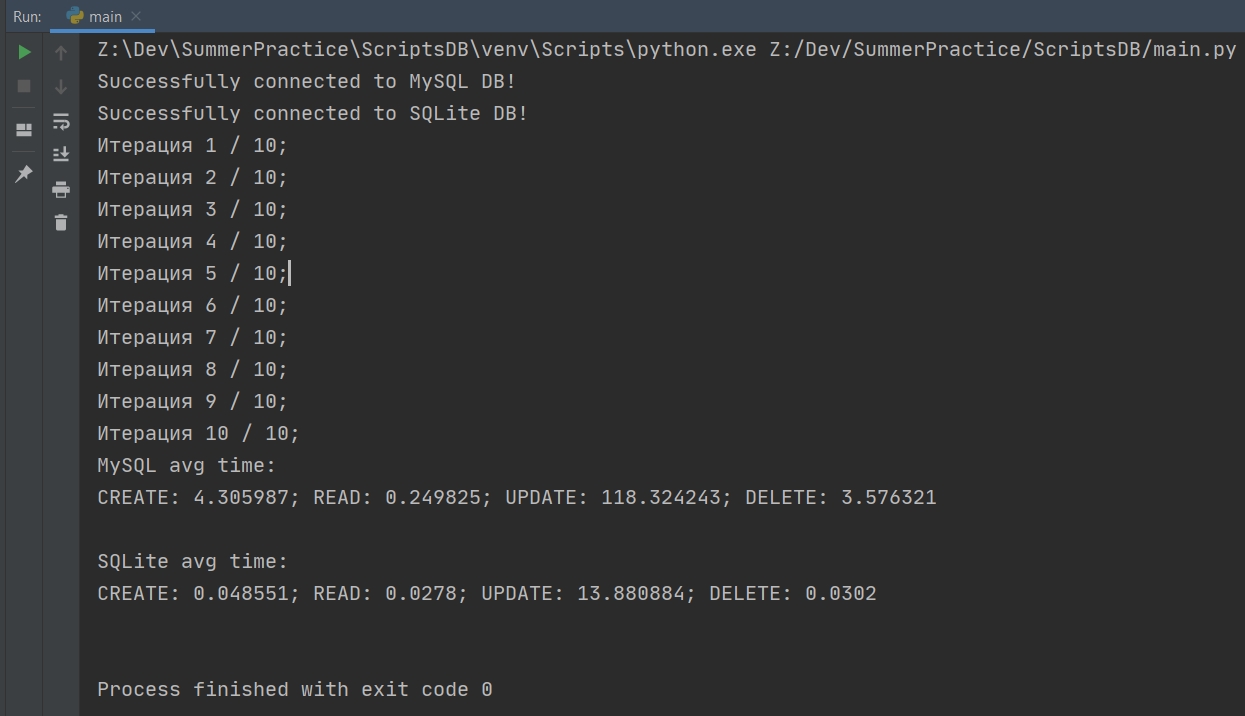


Рисунок 7 – Результат выполнения скриптов в консоли

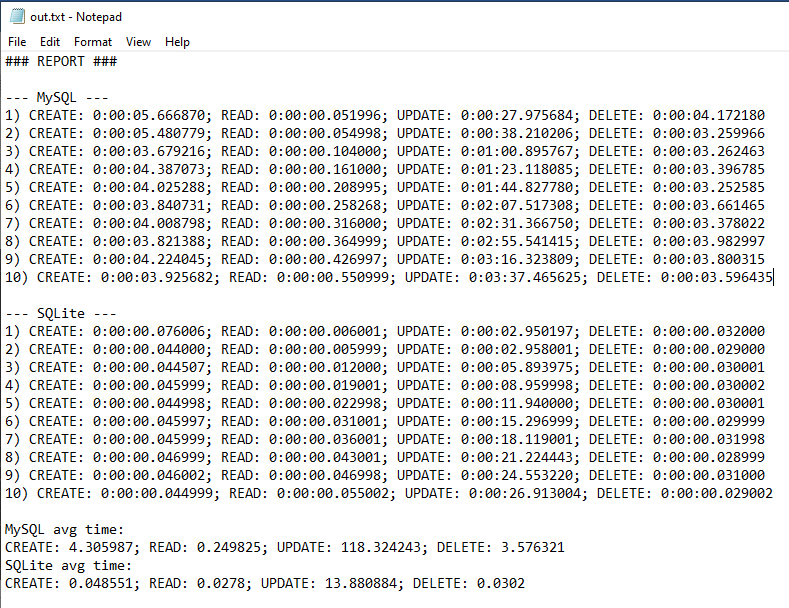


Рисунок 8 – Более подробный лог выполнения скриптов

Наглядное представление статистики (таблица 1, 2; диаграмма 1, 2, 3):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Цикл | *CREATE*, с | *READ*, с | *UPDATE*, с | *DELETE*, с |
| 1 | 5.66687 | 0.051996 | 27.975684 | 4.17218 |
| 2 | 5.480779 | 0.054998 | 38.210206 | 3.259966 |
| 3 | 3.679216 | 0.104 | 60.895767 | 3.262463 |
| 4 | 4.387073 | 0.161 | 83.118085 | 3.396785 |
| 5 | 4.025288 | 0.208995 | 104.82778 | 3.252585 |
| 6 | 3.840731 | 0.258268 | 127.517308 | 3.661465 |
| 7 | 4.008798 | 0.316 | 151.36675 | 3.378022 |
| 8 | 3.821388 | 0.364999 | 175.541415 | 3.982997 |
| 9 | 4.224045 | 0.426997 | 196.323809 | 3.800315 |
| 10 | 3.925682 | 0.550999 | 217.465625 | 3.596435 |
| avg | 4.305987 | 0.249825 | 118.324243 | 3.576321 |

*Таблица 1 – СУБД MySQL, размер 1 цикла – 5000 запросов каждого вида*

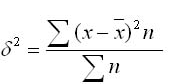
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Цикл | *CREATE*, с | *READ*, с | *UPDATE*, с | *DELETE*, с |
| 1 | 0.076006 | 0.006001 | 2.950197 | 0.032 |
| 2 | 0.044 | 0.005999 | 2.958001 | 0.029 |
| 3 | 0.044507 | 0.012 | 5.893975 | 0.030001 |
| 4 | 0.045999 | 0.019001 | 8.959998 | 0.030002 |
| 5 | 0.044998 | 0.022998 | 11.94 | 0.030001 |
| 6 | 0.045997 | 0.031001 | 15.296999 | 0.029999 |
| 7 | 0.045999 | 0.036001 | 18.119001 | 0.031998 |
| 8 | 0.046999 | 0.043001 | 21.224443 | 0.028999 |
| 9 | 0.046002 | 0.046998 | 24.55322 | 0.031 |
| 10 | 0.044999 | 0.055002 | 26.913004 | 0.029002 |
| avg | 0.048551 | 0.0278 | 13.880884 | 0.0302 |

*Таблица 2 – СУБД SQLite, размер 1 цикла – 5000 запросов каждого вида*

*Диаграмма 1 – Время выполнения CRD операций СУБД MySQL*

*Диаграмма 2 – Время выполнения CRD операций СУБД SQLite*

*Диаграмма 3 – Время выполнения UPDATE операций СУБД MySQL и SQLite*

Вычислим математическое ожидание для статистических данных MySQL по формуле , где – вероятность. Так как в моём случае все исходы равновероятны, то всегда будет равно 1/10 = 0.1 и мат. Ожидание будет равно среднему значению.

Формула для нахождения дисперсии:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *CREATE*, с | *READ*, с | *UPDATE*, с | *DELETE*, с |
| Мат. ожидание | 4.305987 | 0.249825 | 118.324243 | 3.576321 |
| Дисперсия | 0.209794 | 0.049772 | 19.915785 | 0.097632 |

*Таблица 1 – СУБД MySQL, мат. ожидание и дисперсия*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *CREATE*, с | *READ*, с | *UPDATE*, с | *DELETE*, с |
| Мат. ожидание | 0.048551 | 0.0278 | 13.880884 | 0.0302 |
| Дисперсия | 0.002906 | 0.005213 | 2.623341 | 0.00034 |

*Таблица 2 – СУБД SQLite, мат. ожидание и дисперсия*

**ЗАКЛЮЧЕНИЕ**

Подводя итог данной практической работы, можно сделать вывод, что СУБД MySQL и SQLite кардинально отличаются друг от друга как функционально, так и структурно.

MySQL является самой популярной СУБД, потому что сочетает в себе множество всех необходимых функций СУБД. Она поддерживает такие технологии, как авторизация, параллельное чтение и запись, гибкую системную надстройку для оптимизации, различные технологии работы с таблицами (индексы, уровни доступа и прочее). Так же эта СУБД является серверной, что открывает как свои преимущества, так и недостатки. Она является крайне универсальной, за счёт чего и получила свою популярность.

SQLite является немного менее популярной СУБД. Она решает совершенно другие задачи. SQLite обладает рядом своих достоинств, самое главное из которых - это быстродействие. Это самая быстрая и лёгкая open-source СУБД в мире. Она является встраиваемой и, соответственно, она не нуждается в развертывании, что опять же открывает как свои плюсы, так и минусы. Она отлично подходит для однопользовательских офлайн-приложений, так как обеспечивает наилучшее быстродействие для изолированного (в рамках приложения) пользователя.

Поэтому я думаю, что не совсем корректно сравнивать между собой эти СУБД с точки зрения производительности, так как они выполняют совершенно разные задачи, однако исходя из статистических данных, полученных в работе, очевидно, что SQLite выполняет CRUD-операции существенно быстрее, чем MySQL.

Список использованных источников

1. CRUD [Электронный ресурс] – Режим доступа: <https://en.wikipedia.org/wiki/Crud> (Дата обращения: 31.08.2020).
2. CRUD [Электронный ресурс] – Режим доступа: <https://en.wikipedia.org/wiki/CRUD> (Дата обращения: 31.08.2020).
3. MySQL Documentation [Электронный ресурс] – Режим доступа: <https://dev.mysql.com/doc/> (Дата обращения: 31.08.2020).
4. MySQL - Википедия [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/MySQL> (Дата обращения: 31.08.2020).
5. Система управления базами данных MySQL [Электронный ресурс] – Режим доступа: <https://depix.ru/articles/sistema_upravleniya_bazami_dannyh_mysql> (Дата обращения: 31.08.2020).
6. Oracle MySQL Database Service [Электронный ресурс] – Режим доступа: <https://www.oracle.com/ru/mysql/#close> (Дата обращения: 31.08.2020).
7. SQLite - Википедия [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/SQLite> (Дата обращения: 31.08.2020).
8. SQLite — Краткое руководство [Электронный ресурс] – Режим доступа: <https://coderlessons.com/tutorials/bazy-dannykh/vyuchit-sqlite/sqlite-kratkoe-rukovodstvo> (Дата обращения: 31.08.2020).
9. Docker - Википедия [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Docker> (Дата обращения: 31.08.2020).

**ПРИЛОЖЕНИЕ А**

Листинг 1 - main.py

from MySQL\_CRUD import MySQL\_CRUD  
from SQLite\_CRUD import SQLite\_CRUD  
  
  
*# Функция для сбора информации и вывода в файл*def collect\_report(count, dbms\_name, \*args):  
 report = str(**'--- '** + dbms\_name + **' ---**\n**'** + str(count) + **') CREATE: '** + str(args[0]) +  
 **'; READ: '** + str(args[1]) +  
 **'; UPDATE: '** + str(args[2]) +  
 **'; DELETE: '** + str(args[3])) + **'**\n**'** *# print(report)* file = open(**'out.txt'**, **'a'**)  
 file.write(report)  
 file.close()  
  
  
*# Основная функция*def main():  
  
 *# Размер итерации (количество запросов)* count = 5000  
  
 *# Очистка старого отчёта* file = open(**'out.txt'**, **'w'**)  
 file.write(**'### REPORT ###**\n\n**'**)  
 file.close()  
  
 *# Инициализация классов + подключение к локальным базам данных* MySQL\_CRUD()  
 SQLite\_CRUD()  
  
 *# Тестирование баз данных; 10 итераций, 5000 экземпляров каждого запроса* for i in range(0, 10):  
 print(**'Итерация '** + str(i + 1) + **' / 10;'**)  
  
 *# Тестирование MySQL* collect\_report(i + 1, **"MySQL"**, MySQL\_CRUD.Crud(count), MySQL\_CRUD.cRud(count),  
 MySQL\_CRUD.crUd(count), MySQL\_CRUD.cruD(count))  
  
 *# Тестирование SQLite* collect\_report(i + 1, **"SQLite"**, SQLite\_CRUD.Crud(count), SQLite\_CRUD.cRud(count),  
 SQLite\_CRUD.crUd(count), SQLite\_CRUD.cruD(count))  
  
 *# print('\n')* my\_sql\_report = MySQL\_CRUD.get\_avg\_time()  
 sq\_lite\_report = SQLite\_CRUD.get\_avg\_time()  
  
 print(my\_sql\_report)  
 print(sq\_lite\_report)  
  
 file = open(**'out.txt'**, **'a'**)  
 file.write(**'**\n**'**)  
 file.write(my\_sql\_report)  
 file.write(sq\_lite\_report)  
 file.close()  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

Листинг 2 - MySQL\_CRUD.py

import pymysql  
import datetime  
  
  
*# Класс, взаимодействующий с СУБД MySQL*class MySQL\_CRUD:  
 cursor = None  
 connect = None  
  
 timings = {**'count'**: 0, **'CREATE'**: 0, **'READ'**: 0, **'UPDATE'**: 0, **'DELETE'**: 0}  
  
 *# Подключение к БД* @classmethod  
 def \_\_init\_\_(cls):  
 try:  
 cls.connect = pymysql.connect(host=**'127.0.0.1'**, port=3306,  
 user=**'root'**, password=**'12345'**, db=**'mysql'**)  
 cls.cursor = cls.connect.cursor()  
 print(**'Successfully connected to MySQL DB!'**)  
 except Exception as ex:  
 print(ex)  
  
 cls.cursor.execute(**"DROP TABLE IF EXISTS temp;"**)  
 cls.cursor.execute(  
 **"CREATE TABLE temp (ID int NOT NULL AUTO\_INCREMENT, value int, description char(50), PRIMARY KEY (ID));"**)  
  
 *# Вычисление среднего значения времени выполнения* @classmethod  
 def get\_avg\_time(cls):  
 count = cls.timings[**'count'**] / 4  
 report = **'MySQL avg time:**\n**CREATE: '** + str(round(cls.timings[**'CREATE'**] / count, 6)) + \  
 **'; READ: '** + str(round(cls.timings[**'READ'**] / count, 6)) + \  
 **'; UPDATE: '** + str(round(cls.timings[**'UPDATE'**] / count, 6)) + \  
 **'; DELETE: '** + str(round(cls.timings[**'DELETE'**] / count, 6)) + **'**\n**'** return report  
  
 *# Тестирование запроса CREATE* @classmethod  
 def Crud(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"INSERT INTO temp (value, description) VALUES ("** + str(i \* i) + **", 'test');"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'CREATE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса READ* @classmethod  
 def cRud(cls, count):  
 timer = datetime.datetime.now()  
 cls.cursor.execute(**"SELECT** *\** **FROM temp;"**)  
 cls.cursor.fetchall()  
 time = datetime.datetime.now() - timer  
 cls.timings[**'READ'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса UPDATE* @classmethod  
 def crUd(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"UPDATE temp SET description = 'new description';"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'UPDATE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса DELETE* @classmethod  
 def cruD(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"DELETE FROM temp WHERE ID = "** + str(i + 1) + **";"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'DELETE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time

Листинг 3 - SQLite\_CRUD.py

import sqlite3  
import datetime  
  
  
*# Класс, взаимодействующий с СУБД SQLite*class SQLite\_CRUD:  
  
 cursor = None  
 connect = None  
  
 timings = {**'count'**: 0, **'CREATE'**: 0, **'READ'**: 0, **'UPDATE'**: 0, **'DELETE'**: 0}  
  
 *# Подключение к БД* @classmethod  
 def \_\_init\_\_(cls):  
 try:  
 cls.connect = sqlite3.connect(database=**'SQLite.db'**)  
 cls.cursor = cls.connect.cursor()  
 print(**'Successfully connected to SQLite DB!'**)  
 except Exception as ex:  
 print(ex)  
  
 cls.cursor.execute(**"DROP TABLE IF EXISTS temp;"**)  
 cls.cursor.execute(**"CREATE TABLE temp (ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, value INTEGER, description char(50));"**)  
  
 *# Вычисление среднего значения времени выполнения* @classmethod  
 def get\_avg\_time(cls):  
 count = cls.timings[**'count'**] / 4  
 report = **'SQLite avg time:**\n**CREATE: '** + str(round(cls.timings[**'CREATE'**] / count, 6)) + \  
 **'; READ: '** + str(round(cls.timings[**'READ'**] / count, 6)) + \  
 **'; UPDATE: '** + str(round(cls.timings[**'UPDATE'**] / count, 6)) + \  
 **'; DELETE: '** + str(round(cls.timings[**'DELETE'**] / count, 6)) + **'**\n**'** return report  
  
 *# Тестирование запроса CREATE* @classmethod  
 def Crud(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"INSERT INTO temp (value, description) VALUES ("** + str(i \* i) + **", 'test');"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'CREATE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса READ* @classmethod  
 def cRud(cls, count):  
 timer = datetime.datetime.now()  
 cls.cursor.execute(**"SELECT \* FROM temp;"**)  
 cls.cursor.fetchall()  
 time = datetime.datetime.now() - timer  
 cls.timings[**'READ'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса UPDATE* @classmethod  
 def crUd(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"UPDATE temp SET description = 'new description';"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'UPDATE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time  
  
 *# Тестирование запроса DELETE* @classmethod  
 def cruD(cls, count):  
 timer = datetime.datetime.now()  
 for i in range(0, count):  
 cls.cursor.execute(**"DELETE FROM temp WHERE ID = "** + str(i + 1) + **";"**)  
 time = datetime.datetime.now() - timer  
 cls.timings[**'DELETE'**] += time.total\_seconds()  
 cls.timings[**'count'**] += 1  
 return time