**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc170109122)

[Реализация вспомогательных функций 4](#_Toc170109123)

[Разработка ORM 5](#_Toc170109124)

[Исследования 6](#_Toc170109125)

[ЗАКЛЮЧЕНИЕ 7](#_Toc170109126)

[СПИСОК ЛИТЕРАТУРЫ 8](#_Toc170109127)

[ПРИЛОЖЕНИЕ 9](#_Toc170109128)

**ВВЕДЕНИЕ**

Учебная (научно-исследовательская работа (получение первичных навыков научно-исследовательской работы)) практика проходила на кафедре системного анализа и информационных технологий Института вычислительной математики и информационных технологий КФУ с 09 февраля 2024 по 07 июня 2024 года.

Целью практики является получение навыков проведения исследований и анализа полученных результатов.

Для достижения цели были поставлены следующие задачи:

1. изучить способы генерации случайных данных, методы измерения времени и построения графиков;
2. провести исследования эффективности программ, взаимодействующих с БД, на примере базы данных «Студент»;
3. проанализировать полученные результаты.

**Реализация вспомогательных функций**

Реализация функций/классов:

* функция create\_new\_table создаёт таблицы в MySQL по заданным параметрам, включая названия столбцов и их типы данных, а также опциональные внешние ключи. Она устанавливает соединение с базой данных, формирует SQL-запрос на создание таблицы, выполняет его, фиксирует изменения и закрывает соединение;
* функция generate\_data создаёт данные для студентов, их контактной информации и образования. Она возвращает кортеж из трёх списков: информация о студентах (имя и дата рождения), контактная информация (электронная почта, номер телефона, идентификатор студента) и данные об образовании (курс, квалификация, номер группы, идентификатор студента). Используется библиотека Faker;
* функция create\_sandbox создаёт отдельную тестовую базу данных в MySQL на основе существующей. Она копирует структуру и данные из исходной базы данных в новую, обеспечивая изоляцию для тестирования изменений без воздействия на основную базу данных;
* функция generate\_and\_save\_data создает и сохраняет данные в таблицы Students, Contacts и Education базы данных MySQL. Она генерирует n записей о студентах, вставляет их в таблицу Students, затем использует их идентификаторы для генерации и вставки данных о контактах и образовании в соответствующие таблицы. После успешного завершения операций данные сохраняются с помощью операции commit, или выводится сообщение об ошибке в случае исключения MySQL;
* функция delete\_table\_data удаляет все данные из указанной таблицы MySQL базы данных. Она подключается к базе данных, формирует и выполняет SQL-запрос на удаление данных из table\_name, затем фиксирует изменения. Функция replace\_table\_data заменяет все данные в указанной таблице новыми данными из списка data. В зависимости от table\_name, она может рекурсивно удалять и вставлять данные в связанные таблицы, такие как Contacts и Education, чтобы обновить все связанные записи;
* функции backup\_table и restore\_table предназначены для работы с резервными копиями данных MySQL. backup\_table создает SQL-файл с резервной копией указанной таблицы, используя mysqldump для экспорта данных в файл. restore\_table восстанавливает данные из SQL-файла обратно в указанную таблицу с помощью mysql. Обе функции принимают параметры подключения к серверу MySQL, путь к файлу резервной копии и обрабатывают исключения, если процессы mysqldump или mysql завершаются с ошибками. Функции backup\_all\_tables и restore\_all\_tables обобщают функциональность на несколько таблиц, позволяя резервировать и восстанавливать данные одновременно для списка указанных таблиц;
* функция execute\_time измеряет время выполнения SQL-запроса на MySQL сервере. Она использует модуль timeit для многократного выполнения запроса и измерения времени, что позволяет определить минимальное время выполнения с учетом колебаний производительности. Функция принимает SQL-запрос, параметры подключения к базе данных и опциональное количество повторений запроса для более точных результатов;
* класс PlotBuilder предназначен для создания и сохранения графиков с использованием библиотеки matplotlib. Он инициализируется с заданными заголовком, подписями осей и предопределенными списками цветов, стилей линий и маркеров. Метод plot добавляет на график данные, выбирая следующие значения из списков цветов, стилей линий и маркеров. Для сохранения графика используется метод save\_plot, который настраивает легенду, заголовок и метки осей перед сохранением изображения в указанный файл заданного формата.

Все функции были тщательно протестированы на различных сценариях, включая создание, изменение и удаление данных в таблицах, создание и восстановление бэкапов, а также измерение времени выполнения запросов и построение графиков. Тесты включали проверку на корректность данных, эффективность работы и правильное управление ресурсами базы данных.

**Разработка ORM**

В процессе разработки объектно-реляционного отображения (ORM) были созданы несколько ключевых классов и структур для работы с базой данных MySQL:

1. базовый класс Field: этот класс представляет собой абстракцию для всех типов полей, используемых в моделях базы данных. Он определяет основные атрибуты и методы, такие как тип данных SQL, ограничения и опции первичного ключа;
2. классы полей данных (IntegerField, CharField, ForeignKeyField, ManyToManyField): каждый из этих классов представляет конкретный тип данных, который будет использоваться в структуре таблицы базы данных. Они наследуются от базового класса Field и добавляют специфические для типа данных атрибуты и методы;
3. класс BaseModel: этот класс предоставляет основные операции создания, чтения, обновления и удаления (CRUD) для моделей данных. Включены методы для сохранения экземпляра модели в базу данных, создания таблицы, извлечения всех записей, извлечения записи по заданным критериям и генерации фиктивных данных;
4. класс Database: этот класс управляет соединением с базой данных MySQL и предоставляет методы для создания таблиц из моделей, переданных ему в виде списка;
5. примеры моделей (Students, Contacts, Education): эти классы представляют собой конкретные таблицы в базе данных, каждая со своими уникальными полями и взаимосвязями. Каждый класс моделирует структуру таблицы, определяя поля и их типы, включая внешние ключи и таблицы связей многие ко многим.

В процессе разработки основное внимание уделялось точному отображению структуры базы данных в виде классов, что позволяет легко выполнять операции с данными, обеспечивая высокую степень абстракции и гибкость в работе с моделями данных. Каждый класс был разработан с учетом специфики данных, которые он представляет, и их взаимосвязей, что обеспечивает эффективную работу с базой данных через ясный и структурированный интерфейс ORM. Также была отдельно разработана ORM\_doc, в которой описание столбцов выполняется с помощью документации.

**Исследования**

В своем проекте я использую несколько инструментов. Основу моей базы данных составляет MySQL, популярная реляционная система управления базами данных (СУБД), которая позволяет организовывать, хранить и управлять данными. Визуальное моделирование структуры базы данных я выполняю с помощью MySQL Workbench, инструмента, который предоставляет удобный интерфейс для проектирования схем данных и выполнения SQL-запросов.

На рисунке 1 показана моя реляционная база данных с тремя таблицами.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1 – База данных «Студент»

Для разработки логики и взаимодействия с базой данных я выбрал язык программирования Python. Этот язык программирования известен своей простотой и читаемостью, что делает его идеальным для работы с данными.

В Python я использую несколько ключевых библиотек. Matplotlib.pyplot помогает мне создавать наглядные графики и диаграммы, позволяя визуализировать результаты работы с данными. Библиотека timeit служит для измерения времени выполнения кода, что особенно полезно для оптимизации производительности и проверки эффективности моих алгоритмов. Для взаимодействия с базой данных я применяю мною написанный компонент lib.ORM, который предоставляет возможности объектно-реляционного отображения (ORM). Это позволяет работать с базой данных через объектно-ориентированные методы, упрощая процесс написания и управления SQL-запросами.

Целью исследования времени генерации данных было измерение и сравнение времени, необходимого для генерации различных наборов данных.

Для данного исследования была настроена локальная база данных MySQL с таблицами для студентов (Students), контактов (Contacts) и образования (Education). Для взаимодействия с базой данных использовалась мною написанная ORM (Object-Relational Mapping) система. Класс Database был инициализирован с необходимыми учетными данными для подключения к локальному экземпляру MySQL. Каждый из моделей (Students, Contacts и Education) имел метод generate\_fake\_data, предназначенный для создания указанного количества записей. Время, затраченное на генерацию этих данных, измерялось с использованием модуля timeit из Python. Время выполнения для генерации и вставки данных записывалось для увеличивающихся объемов данных: 100, 1000 и 5000 строк. Записанные времена были отображены на графике с использованием matplotlib для визуализации зависимости между количеством строк и временем, затраченным на каждую операцию. Для обеих осей использовалась логарифмическая шкала, чтобы четко показать тенденции роста при различных размерах данных.

Результаты представлены на рисунке 2.

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Рисунок 2 – График времени генерации данных

Интерпретация результатов на графике:

* students (синие точки и линия): время генерации растет экспоненциально с увеличением количества строк. Это указывает на то, что генерация данных для студентов является более трудоемкой по сравнению с другими наборами данных;
* сontacts (оранжевые квадраты и пунктирная линия): время генерации также увеличивается с увеличением количества строк, но с меньшим наклоном по сравнению с Students. Это указывает на более эффективный процесс генерации данных для контактов;
* education (зеленые треугольники и штрихпунктирная линия): время генерации данных для образования растет медленнее по сравнению с Students и Contacts, что указывает на наименьшее время генерации среди представленных наборов данных.
* combined students and education (красные ромбы и точечная линия): время генерации для комбинированных данных является самым большим среди всех наборов данных, что указывает на высокую сложность и ресурсоемкость генерации такого комбинированного набора данных.

Таким образом, время генерации данных увеличивается с ростом количества строк для всех наборов данных, что ожидаемо и естественно для таких процессов, причем набор Students требует наибольшего времени, а Education — наименьшего. Набор Contacts занимает промежуточное место, тогда как комбинированный набор Students and Education демонстрирует самое высокое время генерации. Это указывает на необходимость оптимизации процессов генерации для более сложных наборов данных, таких как Students и комбинированный набор, чтобы улучшить их эффективность.

Цель второго исследования заключалась в измерении времени выполнения различных типов запросов в зависимости от количества строк в базе данных.

В ходе исследования было измерено время выполнения различных типов SQL-запросов (SELECT, INSERT, DELETE) на трех таблицах (Students, Contacts, Education) в зависимости от количества строк (100, 500, 1000). Для реализации экспериментов сначала было создано подключение к базе данных MySQL с использованием учетной записи и базы данных. Затем были определены модели данных (Students, Contacts, Education) и установлено подключение к базе данных для каждой модели. Таблицы в базе данных были созданы на основе этих моделей. Для выполнения запросов были написаны отдельные функции: для SELECT-запросов (query\_all\_students, query\_students\_by\_exact\_name, query\_students\_by\_name\_prefix), INSERT-запросов (query\_save\_students, query\_save\_contacts, query\_save\_education) и DELETE-запросов (query\_delete\_students, query\_delete\_contacts, query\_delete\_education). Эти функции использовались в цикле для каждого количества строк (100, 500, 1000), и время выполнения каждого типа запросов измерялось с использованием модуля timeit. После выполнения запросов время выполнения сохранялось в соответствующих списках для последующего анализа. Между итерациями таблицы очищались для предотвращения накопления данных. Результаты были визуализированы с помощью библиотеки matplotlib, где для каждого типа запросов строились графики зависимости времени выполнения от количества строк.

Результаты представлены на рисунках 3–7.

Изображение выглядит как текст, линия, График, снимок экрана

Автоматически созданное описание

Рисунок 3 – График времени выполнения запроса SELECT для таблицы Students

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рисунок 4 – График времени выполнения запроса SELECT для таблицы Contacts

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рисунок 5 – График времени выполнения запроса SELECT для таблицы Education

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рисунок 6 – График времени выполнения запроса INSERT для всех таблиц

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рисунок 7 – График времени выполнения запроса DELETE для всех таблиц

Интерпретация результатов на графике:

* время выполнения запросов SELECT \* линейно увеличивается с ростом количества строк. Это связано с тем, что запрос извлекает все данные из таблицы, что приводит к увеличению времени обработки по мере роста объема данных;
* запросы с условием WHERE выполняются значительно быстрее, чем полный обход таблиц. Увеличение времени выполнения происходит более постепенно, так как фильтрация уменьшает количество обрабатываемых строк;
* операции INSERT и DELETE демонстрируют линейную масштабируемость относительно количества обрабатываемых строк, что является желаемым свойством для поддержания предсказуемой производительности. Операции INSERT имеют стабильную производительность для разных таблиц, тогда как операции DELETE показывают вариативность, что может указывать на более сложную индексацию или ограничения, требующие дополнительного времени на обработку при удалении данных.

**ЗАКЛЮЧЕНИЕ**

Исходя из проделанной работы, полученных результатов и всего вышеизложенного, можно сделать следующие выводы:

1. изучены способы генерации случайных данных, методы измерения времени и построения графиков;
2. проведены исследования эффективности программ, взаимодействующих с БД, на примере базы данных «Студент»;
3. проанализированы полученные результаты.

За период практики были приобретены следующие компетенции (таблица 1 – ).

Компетенции

|  |  |  |
| --- | --- | --- |
| Компетенция | Расшифровка компетенции | Описание приобретенных знаний, умений и навыков |
| ПК-4 | Оценка и выбор варианта архитектуры программного средства | - умения и навыки проектирования программных средств проведения исследований |
| ПК-5 | Разработка тестовых случаев, проведение тестирования и исследование результатов | - умения и навыки проведения тестирования |

Таким образом, в ходе этой научно-исследовательской работы я приобрел навыки проведения исследований и анализа полученных данных. Полученные знания и опыт окажутся полезными для моей будущей профессиональной деятельности и дальнейших научных исследований.

**СПИСОК ЛИТЕРАТУРЫ**

1. Python основные понятия и возможности языка [Электронный ресурс]. —2020. — URL: https://danyefimoff.medium.com/python-основные-понятия-и-возможности-языка-гайд-для-начинающих-5139116262bf (дата обращения 20.02.2024).
2. The Python Standard Library [Электронный ресурс]. —2019. — URL: https://docs.python.org/3/library/index.html (дата обращения 19.03.2024).
3. Руководство по MySQL [Электронный ресурс]. —2020. — URL: https://metanit.com/sql/mysql/ (дата обращения 05.04.2024).
4. ORM [Электронный ресурс]. —2016. — URL: https://ru.wikipedia.org/wiki/ORM (дата обращения 05.04.2024).
5. Python: Работа с базой данных, часть 2/2: используем ORM [Электронный ресурс]. —2020. — URL: https://habr.com/ru/articles/322086/ (дата обращения 29.04.2024).

**ПРИЛОЖЕНИЕ**

from faker import Faker

import matplotlib.pyplot as plt

from lib.ORM import \*

from timeit import default\_timer as timer

def measure\_generation\_time(func, \*args):

start\_time = timer()

result = func(\*args)

end\_time = timer()

return end\_time - start\_time, result

db = Database(

host='localhost',

user='root',

password='root',

database='tests'

)

BaseModel.set\_db(db)

db.create\_tables([Students, Contacts, Education])

row\_counts = [100, 1000, 5000]

students\_times = []

contacts\_times = []

education\_times = []

combined\_times = []

for count in row\_counts:

gen\_time, students\_data = measure\_generation\_time(Students.generate\_fake\_data, count)

students\_times.append(gen\_time)

gen\_time, contacts\_data = measure\_generation\_time(Contacts.generate\_fake\_data, count)

contacts\_times.append(gen\_time)

gen\_time, education\_data = measure\_generation\_time(Education.generate\_fake\_data, count)

education\_times.append(gen\_time)

combined\_gen\_time\_students, combined\_students\_data = measure\_generation\_time(Students.generate\_fake\_data, count)

combined\_gen\_time\_education, combined\_education\_data = measure\_generation\_time(Education.generate\_fake\_data, count)

combined\_times.append((combined\_gen\_time\_students + combined\_gen\_time\_education))

def plot\_generation\_times(row\_counts, students\_times, contacts\_times, education\_times, combined\_times):

plt.figure(figsize=(12, 8))

plt.plot(row\_counts, students\_times, label='Students (Generation Time)', marker='o', linestyle='-')

plt.plot(row\_counts, contacts\_times, label='Contacts (Generation Time)', marker='s', linestyle='--')

plt.plot(row\_counts, education\_times, label='Education (Generation Time)', marker='^', linestyle='-.')

plt.plot(row\_counts, combined\_times, label='Combined Students and Education (Generation Time)',

marker='d', linestyle=':')

plt.xlabel('Number of Rows ')

plt.ylabel('Time (seconds)')

plt.title('Data Generation Time Measurement')

plt.legend()

plt.grid(True, which="both", ls="--", linewidth=0.5)

plt.xscale('log')

plt.yscale('log')

plt.savefig('data\_generation.png')

plt.show()

plot\_generation\_times(row\_counts, students\_times, contacts\_times, education\_times, combined\_times)

import timeit

import matplotlib.pyplot as plt

from lib.ORM import \*

db = Database(

host='localhost',

user='root',

password='root',

database='tests1'

)

Students.set\_db(db)

Contacts.set\_db(db)

Education.set\_db(db)

db.create\_tables([Students, Contacts, Education])

# Function to retrieve all students

def query\_all\_students():

return Students.all()

# Function to retrieve students by exact full\_name match

def query\_students\_by\_exact\_name():

return Students.get(full\_name='Hailey Hayden')

# Function to retrieve students by full\_name prefix match

def query\_students\_by\_name\_prefix():

return Students.filter(full\_name\_\_like='Hailey%')

# Number of rows to test

row\_counts = [100, 500, 1000]

# Measure execution times for each query type

all\_students\_times = []

exact\_match\_times = []

prefix\_match\_times = []

for count in row\_counts:

# Generate fake data

students\_data = Students.generate\_fake\_data(count=count)

for student in students\_data:

Students(\*\*student).save()

# Measure query times with timeit

all\_students\_time = timeit.timeit('query\_all\_students()', globals=globals(), number=10)

exact\_match\_time = timeit.timeit('query\_students\_by\_exact\_name()', globals=globals(), number=10)

prefix\_match\_time = timeit.timeit('query\_students\_by\_name\_prefix()', globals=globals(), number=10)

# Convert to milliseconds and store

all\_students\_times.append(all\_students\_time)

exact\_match\_times.append(exact\_match\_time)

prefix\_match\_times.append(prefix\_match\_time)

# Clear database tables for the next iteration (optional, if needed)

Students.delete\_all()

# Plotting the results

plt.figure(figsize=(10, 6))

plt.plot(row\_counts, all\_students\_times, marker='o', label='SELECT \* FROM Students')

plt.plot(row\_counts, exact\_match\_times, marker='s', label="SELECT \* FROM Students WHERE full\_name = 'Hailey Hayden')")

plt.plot(row\_counts, prefix\_match\_times, marker='^', label="SELECT \* FROM Students WHERE full\_name LIKE 'Hailey%'")

plt.xlabel('Number of Rows')

plt.ylabel('Time (seconds)')

plt.title('Measuring the execution time of a SELECT query')

plt.xticks(row\_counts)

plt.grid(True)

plt.legend()

plt.tight\_layout()

plt.savefig('select\_students.png')

plt.show()