Міністерство освіти і науки України

Національний університет “Львівська Політехніка”

Кафедра ЕОМ



Курсовий проєкт

З дисципліни «Системне програмне забезпечення»

На тему: «Розробка програмного забезпечення для керування базами даних. Розробка програми, що дозволяє керувати базами даних та виконувати різні запити до них»

Виконала: ст. гр. КІ-307

Коваль К. О.

Перевірив:

викладач

Олексів М. В.

Львів-2024

# Завдання на курсовий проєкт

Розробити програму для керування базами даних з функцією виконання різноманітних запитів до бази даних.

Програма повинна використовувати стандартні бібліотеки класів, які були розглянуті впродовж вивчення курсу «Системне програмне забезпечення».

# Анотація

В курсовій роботі була розроблена програма прямої взаємодії з базами даних, вона забезпечує комфортне керування базами даних, та виконання до них різноманітних запитів.

Програма була розроблена за допомогою Visual Studio 2022, мовою програмування С++ з використанням бібліотеки SQLite3.

Зміст

[Завдання на курсовий проєкт 2](#_Toc167751540)

[Анотація 3](#_Toc167751541)

[Вступ. 5](#_Toc167751542)

[Розділ 1. Аналітичний огляд. 6](#_Toc167751543)

[1.1. Розгляд існуючих методів оптимізації керування базами даних 6](#_Toc167751544)

[1.2. Огляд програм для керування базами даних та методів їх проектування 7](#_Toc167751545)

[1.3. Призначення даних програм 9](#_Toc167751546)

[Розділ 2. Проєктування та аналіз розробки програми для взаємодії з базою даних. 12](#_Toc167751547)

[2.1. Функціональні вимоги: 12](#_Toc167751548)

[2.2. Нефункціональні вимоги: 13](#_Toc167751549)

[2.3. Вибір технологій: 13](#_Toc167751550)

[Розділ 3. Реалізація програмного рішення 18](#_Toc167751551)

[3.1. Детальний опис архітектури 18](#_Toc167751552)

[3.2. Опис взаємодії модулів 19](#_Toc167751553)

[3.3. Переваги модульної архітектури 19](#_Toc167751554)

[Розділ 4. ОПИС ІНТЕРФЕЙСУ ТА ІНСТРУКЦІЇ КОРИСТУВАЧА 22](#_Toc167751555)

[Висновки 26](#_Toc167751556)

[Список використаних джерел 27](#_Toc167751557)

[ДОДАТОК А. Код програми. 28](#_Toc167751558)

# Вступ.

У сучасному світі, де швидкість обробки інформації відіграє ключову роль у продуктивності та ефективності роботи, оптимізація швидкості комп'ютера стає актуальною проблемою. Незалежно від того, чи використовуємо ми комп'ютер для повсякденних завдань, роботи чи розваг, швидкість його роботи напряму впливає на наше задоволення від використання технологій. За допомогою системного програмного забезпечення ми маємо можливість впливати на різноманітні аспекти роботи комп'ютера, включаючи взаємодію комп’ютерних програм з базами даних. Це відкриває перед нами можливості підвищення продуктивності та ефективності роботи наших пристроїв.

Системне програмне забезпечення, реалізоване з використанням мови програмування C++ та технології Sqlite3, надає можливість розробки програм, спрямованих на оптимізацію швидкості роботи комп'ютера. Використання консольного інтерфейсу дозволяє забезпечити зручну взаємодію з користувачем та зробити програму доступною для широкого кола користувачів.

У даній курсовій роботі досліджується комфортне використання баз даних Представлено програму, яка демонструє управління базами даних. Ця програма розроблена з урахуванням сучасних підходів та забезпечує зручний і ефективний інструментарій для кінцевого користувача.

В ході роботи буде розглянуто процес проектування та реалізації програмного забезпечення, а також його функціональні можливості та переваги в контексті оптимізації швидкості роботи комп'ютера, а також розширенням функціоналу, який відповідно надають самі бази даних.

Ця курсова робота спрямована на вивчення методів та інструментів розробки системного програмного забезпечення з метою підвищення продуктивності та ефективності використання комп'ютерних систем.

# Аналітичний огляд.

У цьому розділі буде проведено аналіз відомих методів оптимізації керування базами даних та їхніх особливостей. Перш за все, розглянемо різні підходи до оптимізації та їхні принципи. Далі буде проведений огляд програмного забезпечення, яке вже існує на ринку та спрямоване на керування базами даних та оптимізацію їхньої продуктивності. Розглянемо різноманітні програми, їх призначення та методи проектування.

## Розгляд існуючих методів оптимізації керування базами даних

Оптимізація керування базами даних є важливою проблемою в сфері інформаційних технологій. Існує багато методів та підходів до вирішення цієї проблеми, які можна застосовувати на різних рівнях, починаючи від програмного забезпечення та закінчуючи апаратними засобами. У цьому розділі будуть розглянуті різні підходи до оптимізації керування базами даних з детальним описом кожного з них.

**Оптимізація запитів:**

Першим методом оптимізації є оптимізація запитів. Це включає в себе пошук більш ефективних способів виконання запитів до баз даних, використання індексів, оптимізацію структури таблиць та інші методи, спрямовані на зменшення часу виконання запитів та оптимізацію використання ресурсів бази даних.

**Переваги:**

* Зменшення часу виконання запитів.
* Оптимізація використання ресурсів бази даних.

**Недоліки:**

* Може вимагати значних зусиль на етапі розробки.
* Потребує постійного моніторингу та оновлення при зміні даних.

**Використання кешування та пам'яті:**

Ще одним важливим методом є використання кешування та оптимізація використання пам'яті. Це дозволяє зберігати проміжні результати запитів у пам'яті для швидкого доступу до них, а також використовувати кешування для збереження результатів попередніх запитів та уникнення повторного обчислення.

**Переваги:**

* Зменшення навантаження на базу даних.
* Швидкий доступ до часто використовуваних даних.

**Недоліки:**

* Потребує додаткової пам'яті.
* Можливість виникнення застарілих даних у кеші.

**Оптимізація структури баз даних:**

Не менш важливим методом є оптимізація структури баз даних. Це може включати в себе нормалізацію бази даних, використання індексів, оптимізацію запитів та інші методи, спрямовані на покращення ефективності зберігання та доступу до даних.

Переваги:

* Підвищення продуктивності бази даних.
* Зменшення займаного простору.

Недоліки:

* Може вимагати значних змін в існуючих системах.
* Потребує високої кваліфікації розробників.

## Огляд програм для керування базами даних та методів їх проектування

1.2.1. Види програм для керування базами даних

Програми для керування базами даних можна розділити на декілька категорій залежно від їх функціональності та призначення:

**Універсальні системи управління базами даних (СУБД):**

Ці програми надають широкий спектр функціональності для роботи з різноманітними типами баз даних. Деякі з популярних універсальних СУБД включають:

– MySQL

– PostgreSQL

– Microsoft SQL Server

– Oracle Database

– SQLite

**Переваги:**

* Широкий спектр функцій.
* Підтримка різних типів даних.

**Недоліки:**

* Високі вимоги до ресурсів.
* Можлива складність у налаштуванні та використанні.

**Спеціалізовані програми для керування конкретними типами баз даних:**

Ці програми призначені для оптимізованого управління певними типами баз даних, такими як реляційні, NoSQL, а також для розподілених систем керування базами даних.

**Переваги:**

* Оптимізація під конкретні завдання.
* Вища продуктивність для специфічних типів даних.

**Недоліки:**

* Обмежена функціональність.
* Можливість використання лише для певних типів даних.

1.2.2. Методи проектування програм для керування базами даних

Методи проектування програм для керування базами даних залежать від їх функціональності та цілей. Деякі з поширених методів проектування включають:

* Аналіз вимог користувача: Визначення потреб користувачів та функціональних вимог до програми для керування базами даних.
* Проектування для користувача. Тому перед використанням будь-якої програми для оптимізації, важливо ретельно дослідити її характеристики, відгуки користувачів та рекомендації експертів.

## Призначення даних програм

1.3.1. Задачі, які вирішують програми для керування базами даних

Програми для керування базами даних можуть вирішувати широкий спектр задач, пов'язаних з керуванням та оптимізацією баз даних. До основних задач, які вирішують ці програми, належать:

• Збереження та організація даних: Програми для керування базами даних дозволяють зберігати дані у структурованому форматі та ефективно організовувати їх для швидкого та зручного доступу.

• Оптимізація швидкості запитів: Ці програми можуть автоматично оптимізувати запити до баз даних для забезпечення швидкого виконання та ефективного використання ресурсів.

• Забезпечення безпеки даних: Програми для керування базами даних надають інструменти для захисту даних від несанкціонованого доступу, забезпечуючи конфіденційність, цілісність та доступність інформації.

• Резервне копіювання та відновлення даних: Ці програми дозволяють автоматично створювати резервні копії баз даних та відновлювати дані в разі втрати або пошкодження.

1.3.2. Сфери застосування програм для керування базами даних

Програми для керування базами даних можуть бути корисними для різних галузей та видів діяльності, включаючи:

• Бізнес та фінанси: Компанії використовують бази даних для зберігання та обробки бізнес-даних, таких як клієнтська інформація, фінансові транзакції та інвентар.

• Освіту: Установи освіти використовують бази даних для зберігання студентських записів, розкладів та результатів тестів.

• Медицину: Медичні організації використовують бази даних для зберігання медичної історії пацієнтів, результатів лабораторних досліджень та інших медичних даних.

• Науку та дослідження: Дослідницькі установи використовують бази даних для зберігання наукових даних, результатів експериментів та документації.

1.3.3. Потенційні переваги програм для керування базами даних

Використання програм для керування базами даних може призвести до наступних переваг:

• Підвищення ефективності роботи: Ці програми дозволяють ефективно організовувати та обробляти великі обсяги даних, що підвищує продуктивність та швидкість роботи.

• Зменшення витрат часу та ресурсів: Автоматизація процесів керування базами даних дозволяє зменшити витрати часу та ресурсів, що пов'язані з ручним введенням та обробкою даних.

• Покращення точності та достовірності даних: Збереження даних у структурованому та систематизованому форматі допомагає уникнути помилок та забезпечити точність та достовірність інформації.

• Забезпечення безпеки даних: Програми для керування базами даних надають інструменти для захисту конфіденційної та важливої інформації від несанкціонованого доступу та втрати.

Аналіз існуючих методів оптимізації керування базами даних показує, що кожен метод має свої переваги та недоліки. Оптимізація запитів, використання кешування та пам'яті, а також оптимізація структури баз даних є ключовими підходами, що сприяють підвищенню продуктивності та ефективності використання ресурсів баз даних. Огляд програм для керування базами даних демонструє різноманіття інструментів, які можуть бути використані для збереження, обробки та захисту даних у різних галузях. Враховуючи ці аспекти, у даній роботі буде розроблено систему, яка забезпечить ефективне керування даними, підвищення продуктивності та забезпечення безпеки даних, з урахуванням переваг та недоліків існуючих методів та інструментів.

# Проєктування та аналіз розробки програми для взаємодії з базою даних.

Аналіз задачі розробки програми для керування базами даних може бути корисним для визначення стратегії та технологій, які найкраще підходять для досягнення поставлених цілей. Ось деякі важливі аспекти цього аналізу:

## Функціональні вимоги:

-Меню програми: Програма повинна надавати користувачу меню для зручного користування програмою.

-Додавання даних: Програма повинна дозволяти користувачу додавати дані до існуючих таблиць, тричі за одне використання поданої функції.

-Створення таблиць: Програма повинна дозволяти користувачу створювати нові таблиці з трьома стовпцями.

-Вивід інформації: Програма повинна дозволяти користувачу виводити інформації з обраної таблиці, до 100 записів.

-Очищення даних: Програма повинна дозволяти користувачу очищати обрану таблицю у базі даних.

-Сортування: Програма повинна дозволяти користувачу сортувати у таблиці дані у зростаючому або спадаючому порядку відносно певного стовпця даних, після чого виводити ці дані.

-Автоматичне створення таблиці: Програма повинна дозволяти користувачу автоматично створювати певну таблицю з вже наперед вказаною назвою.

-Автоматичне заповнення таблиці: Програма повинна дозволяти користувачу автоматично заповнювати автоматично створену таблицю наперед заданими даними.

-Вихід: Програма повинна дозволяти користувачу завершувати роботу програми, обравши один з пунктів у меню.

-Заповнення даних правильного типу: Програма повинна видавати помилку у випадку, якщо користувач захоче ввести дані певного типу у стовпець таблиці з іншим типом даних.

-Створення бази даних: У випадку відсутнього файлу бази даних, програма повинна створити новий файл бази даних. Якщо у директорії програми вже є файл, програма повинна використовувати вже існуючий.

## Нефункціональні вимоги:

- Продуктивність: Програма повинна працювати ефективно навіть з великими обсягами даних та складними запитами, забезпечуючи швидкий доступ до інформації.

- Безпека: Забезпечення безпеки даних повинно бути високим пріоритетом, включаючи захист від несанкціонованого доступу та захист від SQL-ін'єкцій.

- Надійність: Програма повинна бути стійкою до відмов та збоїв, а також має мати механізми резервного копіювання даних для запобігання втратам.

-Зручність використання: Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та легким у використанні навіть для користувачів без глибоких знань SQL.

## Вибір технологій:

- Мова програмування:

Для реалізації програми необхідно ретельно обрати мову програмування, враховуючи різні фактори, такі як продуктивність, легкість у використанні, наявність бібліотек та фреймворків, підтримка спільнотою тощо. Розглянемо кілька популярних мов програмування:

* Python: Відомий своєю простотою та великою кількістю бібліотек для різноманітних задач. Має вбудовані модулі для роботи з базами даних, але може поступатися C++ в продуктивності для високонавантажених систем.
* Java: Відмінний вибір для створення великих корпоративних систем, має потужні інструменти для роботи з базами даних. Проте, Java програми можуть мати більше навантаження на пам'ять порівняно з C++.
* C#: Мова, яка активно використовується для розробки програм під Windows, має чудову інтеграцію з Microsoft SQL Server. Тим не менш, C# залежить від екосистеми Microsoft.
* Go: Розроблена компанією Google, мова має гарну продуктивність і простоту написання коду, проте менш популярна у порівнянні з C++ і Java.

-Системи управління базами даних:

Вибір СУБД повинен залежати від конкретних потреб та обмежень проекту. Розглянемо кілька популярних систем управління базами даних:

* MySQL: Відкрита СУБД, широко використовувана в багатьох веб-додатках. Відома своєю надійністю і хорошою продуктивністю, але може потребувати додаткової оптимізації для високонавантажених систем.
* PostgreSQL: Потужна і розширювана СУБД з відкритим кодом, яка підтримує широкий спектр типів даних і має вбудовані механізми для високої надійності. Вимагає більше налаштувань у порівнянні з іншими СУБД.
* SQLite: Легка, вбудована СУБД, яка ідеально підходить для невеликих і середніх додатків. Не потребує встановлення сервера і дуже проста у використанні, але не підходить для високонавантажених систем з великою кількістю паралельних запитів.
* Microsoft SQL Server: Потужна комерційна СУБД від Microsoft, яка має високу продуктивність і хорошу інтеграцію з екосистемою Microsoft. Проте, це комерційний продукт, що може бути дорогим для малих проектів.

Після розгляду різних СУБД, ми вибираємо SQLite. Ця СУБД ідеально підходить для нашого проекту завдяки своїй легкості, простоті у використанні і відсутності потреби у додаткових налаштуваннях. SQLite забезпечує швидке впровадження функціональності і є оптимальним вибором для невеликих та середніх проектів, де немає потреби у високій паралельності запитів.

-Фреймворки і бібліотеки:

Використання бібліотек для взаємодії з базами даних може значно спростити розробку та забезпечити швидке впровадження функціональності. Для взаємодії з базами даних у C++ можна використовувати такі бібліотеки, як SQLite3, що забезпечує простоту використання і високу продуктивність.

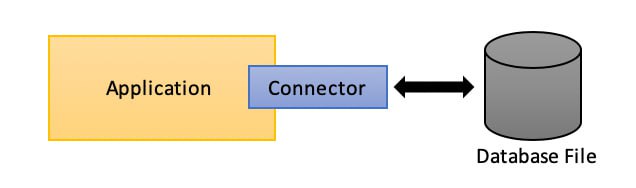


Рис. 2.3.1. Використання бази даних у програмі.

Нижче наведено структурну схему роботи програми та зв’язки між функціями та вибором користувача.

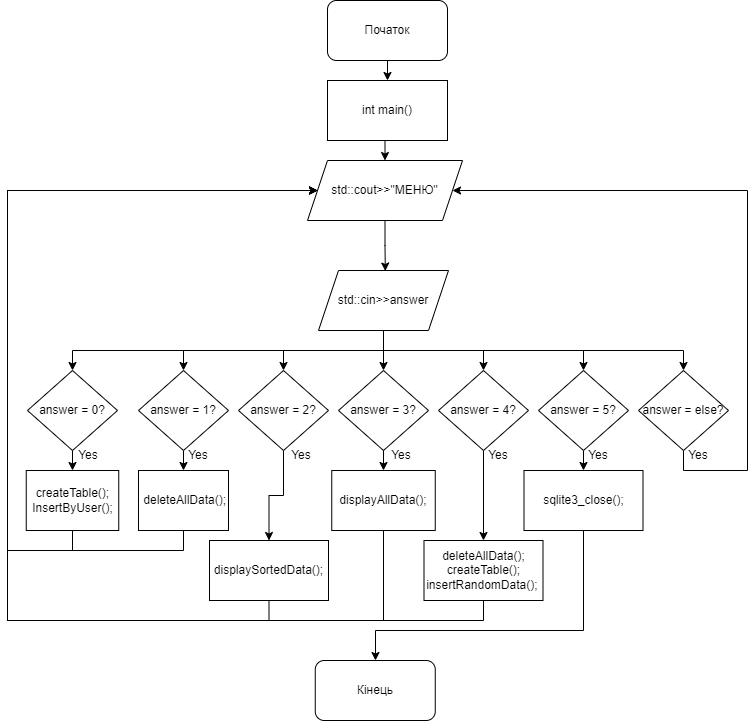


Рис. 2.3.2. Структурна схема.

Як висновок, після порівняння різних мов програмування, ми обираємо C++, яка найкраще підходить для розробки програмного забезпечення завдяки своїй високій продуктивності та ефективному використанню ресурсів. Для управління базами даних ми вибираємо SQLite, яка забезпечує простоту використання, не потребує складних налаштувань та ідеально підходить для нашого проекту. Використання бібліотек, таких як SQLite3, значно спростить процес розробки та впровадження функціональності. Ці вимоги та вибір технологій визначають основні кроки для реалізації програми для керування базами даних та виконання запитів до них.

# Реалізація програмного рішення

## Детальний опис архітектури

Програмне забезпечення для керуванням базою даних буде мати модульну архітектуру, що дозволить розбити його на логічні компоненти з чітко визначеними функціями.

Основні модулі:

Модулі вводу/виводу:

* Відповідають за взаємодію з користувачем через консольний інтерфейс.
* Використовують бібліотеку стандартного вводу/виводу iostream для:
* відображення меню та підменю;
* отримання вибору користувача;
* відображення успішності операцій.

Модуль головного меню:

* Керує головним меню.
* Містить такі функції:
* createTable: створює таблицю з ім’ям, заданим користувачем, а також заповнює імена стовбців відповідно до даних від користувача;
* insertRandomData: створює таблицю з автоматичним заповненням, а також виконує автоматичне заповнення таблиці. Виконує роль тестування роботи бази даних;
* deleteAlldata: видалення усіх даних з обраної таблиці;
* displayAllData: вивід усієї інформації з обраної таблиці;
* displaySortedData: сортування інформації за зростанням;
* InsertByUser: заповнення створеної таблиці користувачем вручну.

Модуль підменю:

* Виконання певних функцій відповідно до вибору користувача, а саме
* отримання даних від користувача;
* виведення певної інформації на екран.

## Опис взаємодії модулів

1. Користувач запускає програму.
2. Модуль вводу/виводу відображає головне меню, яке містить пункти меню.
3. Користувач вибирає пункт.
4. Модуль головного меню викликає функцію підменю (нумерація наведена відповідно до коду):
5. Якщо вибрано Меню заповнення бази даних, керування передається відповідній функції заповнення createTable(), а далі InsertByUser().
6. Якщо вибрано Видалити всі дані, керування передається відповідній функції видалення даних deleteAllData().
7. Якщо вибрано Сортувати, керування передається відповідній функції сортування displaySortedData().
8. Якщо вибрано Вивід всіх даних, керування передається відповідній функції виводу даних displayAllData().
9. Якщо вибрано тестування, керування передається відповідній функції тестового заповнення deleteAllData() для видалення тестової таблиці, якщо така вже наявна у базі даних, createTable() для створення тестової таблиці, insertRandomData() для заповнення щойно створеної таблиці даними, які є задані у коді, як тестовий набір.
10. Якщо вибрано вихід, програма завершує своє виконання.
11. Після виконання кожної функції, керування передається модулю головного меню.

## Переваги модульної архітектури

* **Керованість**: Код розділений на логічні блоки, що полегшує його розуміння, зміну і тестування.
* **Зручність**: Легше знаходити та виправляти помилки, оскільки вони обмежені певними модулями.
* **Розширюваність**: Можна легко додавати нові функції, не впливаючи на існуючий код.

Як висновок, у цьому розділі ми детально розглянули архітектуру програмного забезпечення для керування базою даних, яке було спроектоване з використанням модульної архітектури. Такий підхід дозволяє розбити програму на логічні компоненти з чітко визначеними функціями, що значно підвищує керованість, зручність та розширюваність коду.

Основні модулі програмного забезпечення включають:

1. Модуль вводу/виводу: Відповідає за взаємодію з користувачем через консольний інтерфейс, використовуючи бібліотеку `iostream` для відображення меню, отримання вибору користувача та відображення результатів операцій.

2. Модуль головного меню: Керує головним меню і містить функції для створення таблиць, вставки даних, видалення даних, відображення даних та сортування.

3. Модуль підменю: Виконує функції відповідно до вибору користувача, включаючи отримання даних від користувача та виведення інформації на екран.

Взаємодія модулів забезпечується через чітко визначені функції, що виконуються відповідно до вибору користувача в головному меню. Кожна функція відповідає за конкретну операцію, як-от створення таблиці, вставка даних, видалення даних, сортування даних або їх відображення.

Переваги модульної архітектури очевидні:

* Керованість;
* Зручність;
* Легкодоступне розширювання.

Таким чином, модульна архітектура робить програму більш стійкою до змін, зручнішою для підтримки та легшою для розширення, що є важливими аспектами для довготривалого і успішного використання програмного забезпечення.

# ОПИС ІНТЕРФЕЙСУ ТА ІНСТРУКЦІЇ КОРИСТУВАЧА

Програма має консольний інтерфейс та може бути запущена через виконавчий файл(див. Рис. 4.1). Тестування цих функцій, а також загальний опис програми, як її бачить користувач при роботі, наведено нижче.

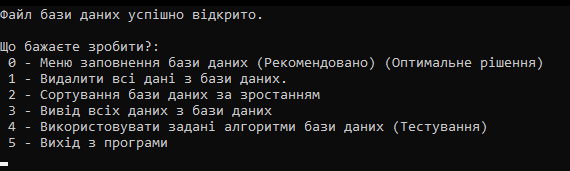


Рис. .. Головне вікно програми

В програмі присутні наступні функції:

* створення бази даних та її заповнення;
* очищення бази даних;
* сортування даних у базі за зростанням певного заданого значення;
* вивід усіх даних з бази;
* автоматичне заповнення бази даних згідно заданого алгоритму;
* вихід з програми.

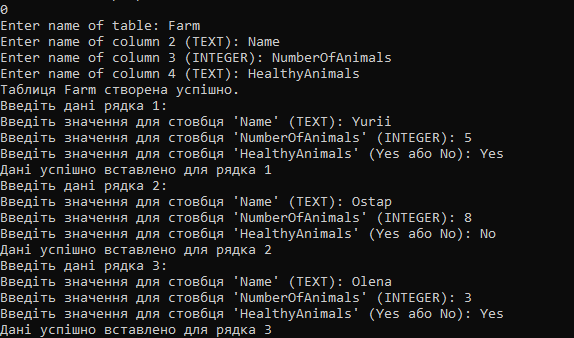


Рис. .2. Заповнення бази даних

Коли користувач вибере заповнення бази даних, він попаде в алгоритм, котрий зчитає дані, обрані користувачем при заповненні, та безпосередньо передасть їх у відповідну базу даних (див. Рис. 4.2).

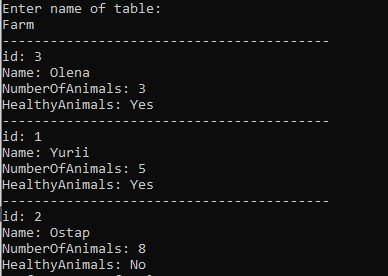


Рис. .3. Сортування бази даних

Відповідно, при виборі пункту сортування бази даних, користувач введе назву таблиці, до якої бажає застосувати цю функцію, та отримає результат (див. Рис. 4.3).

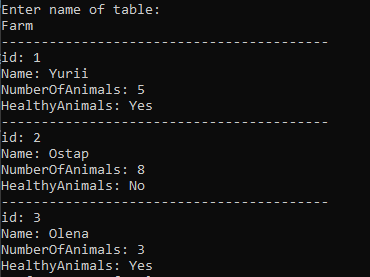


Рис. .4. Вивід даних з бази даних

При виборі пункту 3, користувач теж вводить назву таблиці, дані якої хоче побачити на екрані, та відповідно до цього, отримує результат(див. Рис. 4.4).

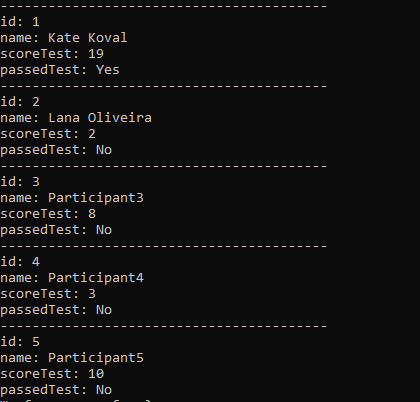


Рис. .5. Тестування бази даних випадковими значеннями

У випадку вибору пункту 4, користувач заповнює базу даних тестовими значеннями, а саме: назвою таблиці бази даних MREO\_Theory, та випадковими значеннями у стовпці балів. Відповідно до цих значень, формується значення третьої таблиці, а саме чи успішно особа склала іспит. Всі дані відповідно виводяться на екран (див. Рис. 4.5). Перед заповненням бази даних цими значеннями, програма очищує таблицю MREO\_Theory, якщо така існувала до виконання дій.



Рис. .6. Видалення даних з бази даних

При обранні пункту 1, користувач вводить таблицю, з якої буде видалено всі дані. У випадку, якщо все пройшло успішно, він може побачити повідомлення про виконання цієї операції на екрані (див. Рис. 4.6).

Приклад взаємодії:

* Користувач запускає програму.
* Модуль вводу/виводу відображає головне меню, яке містить пункти меню.
* Користувач вибирає пункт.
* Модуль меню викликає функцію підменю.
* Модуль підменю просить користувача внести певні дані для виконання відповідної операції.
* Модуль виконує відповідну операцію за допомогою функцій sqlite.
* Після завершення, користувач бачить результат та повертається до головного меню.
* Користувач може вибрати іншу дію або вийти з програми.

Перелічені вище функції дають зрозуміти, що функціонал розроблено відповідно до завдання, та усі поставлені вимоги щодо розробки програми виконано.

Висновки

У даній курсовій роботі було розроблено програмне забезпечення для керування базами даних та виконання до них різноманітних запитів. Програма дозволяє виконувати такі функції:

* створення бази даних та її заповнення;
* очищення бази даних;
* сортування даних у базі за зростанням певного заданого значення;
* вивід усіх даних з бази;
* автоматичне заповнення бази даних згідно заданого алгоритму.

Я навчилася використовувати бібліотеку SQLite3 та інші інструменти для роботи з базами даних у мові програмування С++.

Список використаних джерел

1. Головко В.А. "Системне програмування". - Навчальний посібник. - Львів: Видавництво Національного університету "Львівська політехніка", 2007.
2. Нестеренко О.В. "Програмування для Windows. Системне програмування для Windows у середовищі Microsoft Visual C++". - Навчальний посібник. - К.: Міленіум, 2005.
3. Підгорний О.П. "Програмування для Windows на C++: Об'єктно-орієнтовані технології проектування програмних продуктів". - Навчальний посібник. - Одеса: Наука і техніка, 2009.
4. Архангельський А.Я. "Програмування в C++ Builder та Microsoft Visual C++". - Навчальний посібник. - К.: Видавнича група BHV, 2003.
5. Лазарєва О.С. "Системне програмування: практикум з програмування на мові С++". - Навчальний посібник. - Харків: ХНУРЕ, 2006.
6. Жалдак М.І., Рамський Ю.С. "Системне програмування". - Навчальний посібник. - К.: Вища школа, 2003.
7. Код програми.

#include <iostream>

#include <sqlite3.h>

#include <sstream> // Для std::stringstream

#include <cstdlib> // Для функцій rand() та srand()

#include <ctime> // Для функції time()

#include <Windows.h> // UA LOCALIZATION

void createTable(sqlite3\* db, const std::string& nameOfTable, const std::string& nameOrText, const std::string& scoreOrIntg, const std::string& passedOrText) {

char\* errorMessage = nullptr;

std::string createTableSQL = "CREATE TABLE IF NOT EXISTS " + nameOfTable + " ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

+ nameOrText + " TEXT NOT NULL, "

+ scoreOrIntg + " INTEGER NOT NULL, "

+ passedOrText + " TEXT NOT NULL"

");";

int rc = sqlite3\_exec(db, createTableSQL.c\_str(), nullptr, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при створенні таблиці: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

else {

std::cout << "Таблиця " << nameOfTable << " створена успішно." << std::endl;

}

}

void insertRandomData(sqlite3\* db, const std::string& name) {

int Score = rand() % 20; // Випадковий бал від 0 до 100

std::string passed = (Score >= 18) ? "Yes" : "No";

std::stringstream ss;

ss << Score;

std::string insertSQL = "INSERT INTO MREO\_Theory (name, scoreTest, passedTest) "

"VALUES ('" + name + "', " + ss.str() + ", '" + passed + "');";

char\* errorMessage = nullptr;

int rc = sqlite3\_exec(db, insertSQL.c\_str(), nullptr, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при вставці даних: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

else {

std::cout << "Дані успішно вставлено " << name << std::endl;

}

}

void deleteAllData(sqlite3\* db, const std::string& nameOfTable) {

char\* errorMessage = nullptr;

std::string deleteSQL = "DELETE FROM " + nameOfTable + ";";

int rc = sqlite3\_exec(db, deleteSQL.c\_str(), nullptr, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при видаленні даних: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

else {

std::cout << "Всі дані успішно видалено з таблиці " << nameOfTable << std::endl;

}

}

void displayAllData(sqlite3\* db, const std::string& nameOfTable) {

char\* errorMessage = nullptr;

std::string selectSQL = "SELECT \* FROM " + nameOfTable + ";";

int rc = sqlite3\_exec(db, selectSQL.c\_str(), [](void\* data, int argc, char\*\* argv, char\*\* azColName) -> int {

std::cout << "-----------------------------------------\n";

for (int i = 0; i < argc; i++) {

std::cout << azColName[i] << ": " << (argv[i] ? argv[i] : "NULL") << "\n";

}

return 0;

}, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при виведенні даних: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

}

void displaySortedData(sqlite3\* db, const std::string& nameOfTable, const std::string& orderByColumn) {

char\* errorMessage = nullptr;

std::string selectSQL = "SELECT \* FROM " + nameOfTable + " ORDER BY " + orderByColumn + " ASC;";

int rc = sqlite3\_exec(db, selectSQL.c\_str(), [](void\* data, int argc, char\*\* argv, char\*\* azColName) -> int {

std::cout << "-----------------------------------------\n";

for (int i = 0; i < argc; i++) {

std::cout << azColName[i] << ": " << (argv[i] ? argv[i] : "NULL") << "\n";

}

return 0;

}, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при виведенні даних: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

}

void InsertByUser(sqlite3\* db, const std::string& tableName, const std::string& nameColumn, const std::string& scoreColumn, const std::string& passedColumn) {

std::string name, passed;

int Score;

std::string insertSQL;

for (int i = 0; i < 3; i++) {

std::cout << "Введіть дані рядка " << i + 1 << ":" << std::endl;

std::cout << "Введіть значення для стовбця '" << nameColumn << "' (TEXT): ";

std::cin.ignore(); // Очищення буфера вводу перед викликом getline()

std::getline(std::cin, name);

std::cout << "Введіть значення для стовбця '" << scoreColumn << "' (INTEGER): ";

std::cin >> Score;

std::stringstream ss;

ss << Score;

std::cout << "Введіть значення для стовбця '" << passedColumn << "' (Yes або No): ";

std::cin >> passed;

insertSQL = "INSERT INTO " + tableName + " (" + nameColumn + ", " + scoreColumn + ", " + passedColumn + ") "

"VALUES ('" + name + "', " + ss.str() + ", '" + passed + "');";

char\* errorMessage = nullptr;

int rc = sqlite3\_exec(db, insertSQL.c\_str(), nullptr, nullptr, &errorMessage);

if (rc != SQLITE\_OK) {

std::cout << "Помилка при вставці даних: " << errorMessage << std::endl;

sqlite3\_free(errorMessage);

}

else {

std::cout << "Дані успішно вставлено для рядка " << i + 1 << std::endl;

}

}

}

int main() {

SetConsoleOutputCP(1251); // UA LOCALIZATION

setlocale(LC\_CTYPE, "ukr"); // UA LOCALIZATION

static std::string s;

static std::string n;

static std::string c;

static std::string p;

sqlite3\* db;

char\* errorMessage = nullptr;

int rc;

int answer;

std::string nameOfTableToShow;

rc = sqlite3\_open("DB.db", &db);

if (rc != SQLITE\_OK) {

std::cout << "Помилка! Не вдалося відкрити файл бази даних.\n" << std::endl;

return 1;

}

std::cout << "Файл бази даних успішно відкрито.\n" << std::endl;

do {

do {

std::cout << "Що бажаєте зробити?:\n 0 - Меню заповнення бази даних (Рекомендовано) (Оптимальне рішення) \n 1 - Видалити всі дані з бази даних. \n 2 - Сортування бази даних за зростанням \n 3 - Вивід всіх даних з бази даних \n 4 - Використовувати задані алгоритми бази даних (Тестування) \n 5 - Вихід з програми \n";

std::cin >> answer;

}while(answer != 0 && answer != 1 && answer != 2 && answer != 3 && answer != 4 && answer != 5);

switch (answer) {

case 0:

std::cout << "Enter name of table: ";

std::cin >> s;

std::cout << "Enter name of column 2 (TEXT): ";

std::cin >> n;

std::cout << "Enter name of column 3 (INTEGER): ";

std::cin >> c;

std::cout << "Enter name of column 4 (TEXT): ";

std::cin >> p;

createTable(db, s, n, c, p);

InsertByUser(db, s, n, c, p);

break;

case 1:

std::cout << "Enter name of table: \n";

std::cin >> nameOfTableToShow;

deleteAllData(db, nameOfTableToShow); break;

case 2:

std::cout << "Enter name of table: \n";

std::cin >> nameOfTableToShow;

displaySortedData(db, nameOfTableToShow, c); break;

case 3: std::cout << "Enter name of table: \n";

std::cin >> nameOfTableToShow;

displayAllData(db, nameOfTableToShow); break;

case 4:

srand(time(nullptr)); // Ініціалізуємо генератор випадкових чисел за часом

s = "MREO\_Theory";

n = "name";

c = "scoreTest";

p = "passedTest";

deleteAllData(db, s);

createTable(db, s, n, c, p);

insertRandomData(db, "Kate Koval");

insertRandomData(db, "Lana Oliveira");

insertRandomData(db, "Participant3");

insertRandomData(db, "Participant4");

insertRandomData(db, "Participant5");

displayAllData(db, s);

break;

case 5: sqlite3\_close(db); return 0; break;

}

} while (answer != 5);

sqlite3\_close(db);

}