МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

кафедра комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему
Розробка SQL-запитів в системі тестування

Виконав: студент групи 2К-20
Спеціальності

123 – «Комп’ютерна інженерія»
Олександр ШАКАЛОВ

Керівник:
Роксолана БРЕУС

Черкаси 2024

АНОТАЦІЯ

Ця кваліфікаційна робота присвячена розробці та оптимізації SQL-запитів для систем тестування. Основною метою роботи є дослідження теоретичних основ SQL-запитів, проектування ефективних запитів і їх впровадження в системи тестування для покращення якості та продуктивності тестування. Головним завданням було досягнення автоматизації тестування за допомогою SQL. У вступі визначається актуальність теми, ставляться основні цілі та завдання дослідження.

Робота починається з огляду основних концепцій SQL-запитів та систем тестування, де аналізуються різні системи управління базами даних (СУБД) і їх переваги та недоліки у контексті тестування. Окрему увагу приділено взаємодії SQL-запитів з системами тестування, що дозволяє створювати та виконувати тестові сценарії, перевіряти цілісність даних та автоматизувати процеси тестування.

У наступній частині роботи розглядається процес проектування бази даних для тестування та розробка ефективних SQL-запитів. Це включає створення структури бази даних, розробку запитів для забезпечення максимальної продуктивності та точності, а також методи налагодження та оптимізації запитів.

Практична частина дослідження присвячена реалізації тестових сценаріїв з використанням SQL-запитів. Демонструється процес створення, виконання та аналізу тестових сценаріїв у реальній системі тестування, що дозволяє оцінити ефективність розроблених SQL-запитів.

У висновках підсумовуються результати дослідження, оцінюється ефективність та продуктивність розроблених SQL-запитів у тестуванні. Надаються рекомендації щодо подальших досліджень та можливих шляхів вдосконалення процесу тестування за допомогою SQL-запитів.

ABSTRACT

This qualification work is dedicated to the development and optimization of SQL queries for testing systems. The main objective of the work is to study the theoretical foundations of SQL queries, design effective queries, and implement them in testing systems to improve the quality and performance of testing. The primary task was to achieve automation of testing using SQL. The introduction outlines the relevance of the topic, the main goals, and the objectives of the research.

The work begins with an overview of the basic concepts of SQL queries and testing systems, where different database management systems (DBMS) are analyzed, highlighting their advantages and disadvantages in the context of testing. Special attention is given to the interaction of SQL queries with testing systems, which allows for the creation and execution of test scenarios, data integrity checks, and automation of testing processes.

The next part of the work considers the process of designing a database for testing and developing effective SQL queries. This includes creating the database structure, developing queries to ensure maximum performance and accuracy, and methods for debugging and optimizing queries.

The practical part of the research focuses on the implementation of test scenarios using SQL queries. It demonstrates the process of creating, executing, and analyzing test scenarios in a real testing system, allowing for the evaluation of the effectiveness of the developed SQL queries.

The conclusions summarize the results of the research, assess the effectiveness and performance of the developed SQL queries in testing, and provide recommendations for further research and potential improvements in the testing process using SQL queries.

ЗМІСТ

ВСТУП

Актуальність теми. В сучасному світі інформаційних технологій, дані відіграють ключову роль у прийнятті бізнес-рішень та оптимізації процесів. Системи тестування баз даних є невід'ємною частиною цього процесу, оскільки вони дозволяють забезпечити цілісність, надійність та продуктивність даних. У цьому контексті розробка SQL-запитів для тестування набуває особливої важливості.

Правильно розроблені SQL-запити дозволяють ефективно виявляти помилки, неточності та аномалії у даних. Це є критично важливим для забезпечення високої якості даних, що використовується у бізнес-аналітиці та прийнятті рішень.

Тестування SQL-запитів допомагає виявити вузькі місця та оптимізувати продуктивність баз даних. Це включає аналіз виконання запитів, індексацію та оптимізацію структур баз даних, що дозволяє значно знизити час відповіді на запити.

Сучасний бізнес вимагає швидкої адаптації до змінних умов ринку та нових технологій. Тестування SQL-запитів дозволяє розробникам швидко та ефективно адаптувати бази даних до нових вимог, забезпечуючи високу гнучкість систем.

Тестування SQL-запитів також є важливим для виявлення потенційних вразливостей у базах даних. Це включає тестування на наявність SQL-ін'єкцій та інших видів атак, що дозволяє забезпечити захист конфіденційної інформації.

Системи тестування баз даних повинні інтегруватися з сучасними технологіями, такими як великі дані (Big Data), хмарні обчислення та машинне навчання. Тестування SQL-запитів дозволяє забезпечити коректну інтеграцію та взаємодію між різними системами та платформами.

Автоматизація тестування SQL-запитів значно підвищує ефективність та швидкість процесів розробки. Це дозволяє знизити кількість ручної роботи, мінімізувати людський фактор та забезпечити високу точність тестування.

Таким чином, розробка SQL-запитів для систем тестування є актуальною та важливою темою, яка сприяє забезпеченню якості, продуктивності, безпеки та гнучкості сучасних інформаційних систем.

Мета і завдання роботи: Метою цього дослідження є створення SQL-запитів та використання їх в системі тестування.

Завданнями цієї роботи є:

1. Спроєктувати базу даних для тестування
2. Розробкити SQL-запити
3. Досліджити використання для тестування SQL-запитів.

Об'єкт і предмет дослідження: Об’єктом дослідження є системи тестування.

Системи тестування – це сукупність методів, інструментів та процесів, що використовуються для перевірки якості та функціональності програмного продукту. Вони допомагають виявляти дефекти, оцінювати продуктивність, забезпечувати безпеку та відповідність ПЗ вимогам користувачів і стандартам.

Предметом дослідження є SQL-запити що використовуються для тестування.

Методи дослідження: було застосовано аналіз теоретичного матеріалу. Та на їх основі розроблено практичну реалізацію для більш глибокого розуміння теми.

РОЗДІЛ 1 ОСНОВИ SQL-ЗАПИТІВ І СИСТЕМ ТЕСТУВАННЯ

1.1 Основи SQL-запитів і систем тестування

Загальна характеристика SQL-запитів: SQL-запити (Structured Query Language) є основним інструментом для роботи з реляційними базами даних. Вони дозволяють користувачам створювати, читати, оновлювати та видаляти дані, а також керувати структурою баз даних. Загальна характеристика SQL-запитів охоплює їхні типи, основні компоненти та функціональні можливості [1].

Типи SQL-запитів:

1. Запити для маніпуляції даними (DML - Data Manipulation Language)
   - SELECT: Використовується для вибірки даних з однієї або декількох таблиць.
   - INSERT: Використовується для вставки нових записів у таблицю.
   - UPDATE: Використовується для оновлення існуючих записів у таблиці.
   - DELETE: Використовується для видалення записів з таблиці.
2. Запити для визначення даних (DDL - Data Definition Language):
   - CREATE: Використовується для створення нових таблиць, індексів, баз даних та інших об'єктів.
   - ALTER: Використовується для зміни структури існуючих об'єктів бази даних.
   - DROP: Використовується для видалення об'єктів бази даних, таких як таблиці чи індекси.
   - TRUNCATE: Використовується для видалення всіх записів з таблиці, не видаляючи саму таблицю.
3. Запити для контролю даних (DCL - Data Control Language)
   - GRANT: Використовується для надання привілеїв користувачам або ролям.
   - REVOKE: Використовується для відкликання привілеїв у користувачів або ролей.

4. Запити для керування транзакціями (TCL - Transaction Control Language)

- COMMIT: Використовується для фіксації всіх змін, зроблених у транзакції.

- ROLLBACK: Використовується для відміни всіх змін, зроблених у поточній транзакції.

- SAVEPOINT: Використовується для встановлення точки збереження всередині транзакції.

До основних компонентів SQL-запитів відносять:

1. Оператори та ключові слова:

- Основні оператори включають SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP тощо.

- Ключові слова такі як FROM, WHERE, GROUP BY, HAVING, ORDER BY, JOIN, UNION, і так далі.

2. Функції:

- Агрегатні функції: SUM, AVG, COUNT, MAX, MIN використовуються для виконання обчислень на наборах значень.

- Скалярні функції: UPPER, LOWER, LENGTH, ROUND використовуються для обробки окремих значень.

3. Умови та фільтри:

- Умови в WHERE клаузах використовуються для фільтрації рядків, що задовольняють певним критеріям.

- Оператори порівняння (=, >, <, >=, <=, <>), логічні оператори (AND, OR, NOT) і спеціальні оператори (LIKE, IN, BETWEEN, IS NULL).

4. Об'єднання та підзапити:

- JOIN: Використовується для об'єднання рядків з двох або більше таблиць на основі пов'язаних між собою полів.

- Підзапити: Запити, вкладені всередині інших запитів, використовуються для виконання складних запитів і фільтрації даних [2].

Основні функціональні можливості SQL-запитів:

1. Гнучкість:

- SQL-запити дозволяють виконувати широкий спектр операцій з даними, від простих вибірок до складних агрегатних функцій та об'єднань таблиць.

2. Масштабованість:

- SQL-запити можуть обробляти великі обсяги даних і підтримують складні структури баз даних.

3. Стандартизація:

- SQL є стандартною мовою для реляційних баз даних, що забезпечує сумісність між різними СУБД (системами управління базами даних).

4. Безпека:

- Можливість контролю доступу до даних через DCL-запити (GRANT, REVOKE) та реалізація транзакційного управління забезпечує цілісність і безпеку даних.

5. Інтеграція:

- SQL-запити можуть бути інтегровані з іншими мовами програмування, що дозволяє створювати динамічні та інтерактивні додатки.

Загалом, SQL-запити є потужним і гнучким інструментом для роботи з реляційними базами даних, забезпечуючи ефективне управління, маніпуляцію і аналіз даних.

SQL (Structured Query Language) є основною мовою для роботи з реляційними базами даних, що дозволяє виконувати різноманітні операції, такі як створення, маніпулювання та управління даними. Основи SQL-запитів включають команди для вибору даних (SELECT), оновлення (UPDATE), вставки (INSERT) та видалення (DELETE). Ці команди дозволяють користувачам ефективно взаємодіяти з базами даних та виконувати необхідні операції для забезпечення коректної роботи системи.

Системи тестування баз даних відіграють ключову роль у забезпеченні цілісності, продуктивності та безпеки даних. Вони дозволяють виявляти

помилки, оптимізувати продуктивність та забезпечувати захист від потенційних загроз.

Узагальнюючи, основи SQL-запитів є фундаментом для роботи з реляційними базами даних, забезпечуючи можливість ефективного управління та маніпулювання даними. Системи тестування баз даних є необхідним інструментом для підтримки високої якості, продуктивності та безпеки даних, що дозволяє організаціям забезпечити стабільну та ефективну роботу своїх інформаційних систем. Інтеграція цих двох аспектів забезпечує надійність та ефективність баз даних, що є критично важливим для сучасних бізнесів та технологічних рішень [3].

1.2 Огляд систем тестування

Системи тестування програмного забезпечення забезпечують різні рівні та типи тестування, щоб гарантувати, що програмний продукт відповідає вимогам і функціонує належним чином. Основні види систем тестування включають юніт-тестування, інтеграційне тестування та системне тестування. Крім того, автоматизовані системи тестування грають ключову роль в підвищенні ефективності та точності процесу тестування [4].

Основні види систем тестування:

- Юніт-тестування (Unit Testing) – це тестування окремих компонентів або модулів програми ізольовано від інших частин системи. Відповідають за виявлення дефектів на ранніх стадіях розробки; перевірка правильності роботи кожного модуля (JUnit (для Java), NUnit (для .NET), pytest (для Python). Швидке виявлення помилок, полегшення процесу інтеграції, поліпшення якості коду є його перевагами .
- Інтеграційне тестування (Integration Testing) – це тестування взаємодії між інтегрованими модулями або компонентами. Основні цілі – виявлення дефектів, що виникають при взаємодії між компонентами, перевірка правильності роботи інтерфейсів між модулями (JUnit (для Java), TestNG (для

Java), pytest (для Python). Виявлення проблем з інтеграцією на ранніх стадіях, забезпечення узгодженості роботи компонентів є його перевагою..

Системи тестування баз даних відіграють важливу роль у забезпеченні надійності, безпеки та ефективності роботи інформаційних систем. Вони дозволяють виявляти та усувати помилки, оптимізувати продуктивність та забезпечувати цілісність даних. У процесі огляду систем тестування було виявлено кілька ключових аспектів, які є критичними для успішного тестування баз даних.

Отже, огляд систем тестування баз даних показує, що вони є невід'ємною частиною сучасного процесу розробки та експлуатації інформаційних систем. Вони забезпечують високу якість, продуктивність та безпеку даних, що є критично важливим для успішної роботи будь-якої організації. Інтеграція систем тестування у процеси розробки та підтримки баз даних дозволяє забезпечити стабільність та надійність інформаційних систем, що сприяє досягненню бізнес-цілей та підвищенню конкурентоспроможності на ринку.

- Системне тестування (System Testing) – це повне тестування інтегрованої системи як єдиного цілого для перевірки її відповідності всім вимогам. Слугує для перевірки функціональності, продуктивності, безпеки та інших аспектів системи. (Selenium (для веб-додатків), QTP/UFT (для функціонального тестування), LoadRunner (для тестування продуктивності). До переваг можна віднести забезпечення загальної якості системи, виявлення дефектів перед розгортанням у виробниче середовище.

- Автоматизовані системи тестування виконують тести без участі людини, що дозволяє швидко і часто виконувати тести. Забезпечують повторюваності тестування. Управління тестовими сценаріями, тобто збереження, організація та керування тестовими випадками і сценаріями, легке створення та модифікація тестів, звітування та аналіз результатів (Генерація детальних звітів про результати тестування, аналіз виконаних тестів для виявлення трендів та проблемних областей, інтеграція з системами CI/CD –

інтеграція з системами безперервної інтеграції та безперервного розгортання для автоматичного запуску тестів при кожному внесенні змін у код [5].

До переваг автоматизованих систем тестування належить:

- Підвищення ефективності – зменшення часу, необхідного на виконання тестів, порівняно з ручним тестуванням, можливість виконання тестів в будь-який час, включаючи нічні години.
- Покращення якості – більша кількість тестів може бути виконана в обмежений час, що підвищує загальне покриття тестування. Зниження ризику людських помилок при виконанні тестів.
- Зниження витрат – початкові витрати на розробку автоматизованих тестів можуть бути високими, у довгостроковій перспективі це призводить до зниження витрат на тестування. Скорочення витрат на ручне тестування та регресійне тестування.
- Швидкий зворотний зв'язок - миттєве повідомлення про дефекти після внесення змін до коду, що дозволяє швидко їх виправляти.
- Повторюваність і консистентність – автоматизовані тести виконуються однаково кожного разу, що забезпечує стабільні результати і полегшує відстеження помилок.

Найтиповіші приклади інструментів автоматизованого тестування:

- Selenium: Використовується для автоматизації тестування веб-додатків.
- JUnit і TestNG: Фреймворки для автоматизації юніт-тестування в Java.
- PyTest: Фреймворки для автоматизації юніт-тестування в Python
- Appium: Інструмент для автоматизації тестування мобільних додатків.
- LoadRunner: Використовується для автоматизованого тестування продуктивності.

Таким чином, системи тестування ПЗ, зокрема автоматизовані, є ключовими інструментами для забезпечення високої якості програмного

забезпечення, ефективності процесу розробки та задоволення вимог кінцевих користувачів [6].

Інтеграція SQL-запитів у системи тестування є важливою частиною процесу тестування програмного забезпечення, особливо коли мова йде про тести, які вимагають перевірки даних у базі даних або взаємодії з ними.

У результаті огляду системи тестування, були проведені докладні аналізи, що дозволили зробити кілька важливих висновків. Зокрема, системи тестування виявилися ефективними у виявленні більшості дефектів на ранніх етапах розробки. Вони дозволяють зменшити кількість помилок, які досягли виробничого середовища, що в свою чергу значно зменшує витрати на виправлення помилок на пізніших етапах розробки.

На підставі вищезазначеного, можна надати рекомендації для покращення системи тестування. По-перше, необхідно збільшити покриття тестами, зосереджуючись на ключових функціональностях та критичних сценаріях використання. Далі, важливо розширити автоматизацію тестування, щоб зменшити витрати на ручне тестування та забезпечити швидке виявлення дефектів.

Нарешті, слід покращити звітність тестових результатів, щоб забезпечити чітке відслідковування прогресу тестування та вчасне виявлення проблем. Це допоможе забезпечити більшу відкритість та ефективність в процесі тестування.

В цілому, покращення системи тестування є важливим кроком для забезпечення якості програмного забезпечення та задоволення потреб користувачів [7].

1.3 Порівняння систем управління базами данних

Порівняння найпопулярніших систем управління базами даних (СУБД) зазвичай включає такі критерії, як продуктивність, масштабованість, підтримка SQL та NoSQL, спільнота користувачів, безпека, вартість та інші. Проведемо порівняльний аналіз найпопулярніших СУБД:

1. MySQL - реляційна СУБД. До переваг можна віднести те, що одразу може користуватись велика кількість людей. Відкрите програмне забезпечення з потужною комерційною підтримкою від Oracle, висока продуктивність для читання. Підтримка різних механізмів зберігання, таких як InnoDB та MyISAM. До недоліків можна віднести обмежену підтримку складних транзакцій, порівняно повільні записи, відсутність деяких сучасних функцій, наявних у PostgreSQL [8].

2. PostgreSQL - реляційна СУБД. Серед переваг - дуже потужна і функціональна СУБД з підтримкою складних запитів, підтримка транзакцій ACID, підтримка JSON та інших типів даних, що робить її гнучкою для NoSQL задач:

- Висока продуктивність як для читання, так і для запису
- Відкрите програмне забезпечення з активною спільнотою

До недоліків відносять:

- Складніше налаштування та оптимізація порівняно з MySQL
- Може вимагати більше ресурсів

3. Microsoft SQL Server реляційна СУБД з високою інтеграція з продуктами Microsoft, підтримкою складних аналітичних запитів, високою продуктивністю і надійністю, підтримкою транзакцій ACID та потужних інструментів адміністрування

Серед недоліків:

- Висока вартість ліцензії
- Працює переважно на Windows
- Менша спільнота у порівнянні з відкритими СУБД

4. MongoDB - NoSQL (документоорієнтована СУБД).

Переваги:

- Відмінна для зберігання та обробки великих обсягів неструктурованих даних
- Підтримка JSON-подібних документів
- Добре масштабується горизонтально
- Гнучка схема

Недоліки:

- Відсутність підтримки транзакцій до версії 4.0
- Може вимагати більше ресурсів для складних запитів
- Порівняно нова, менша спільнота

5. Oracle Database - реляційна СУБД

Переваги:

- Висока продуктивність і надійність
- Підтримка складних транзакцій і великої кількості даних
- Потужні інструменти адміністрування та аналізу даних
- Підтримка кластеризації та масштабування

Недоліки:

- Дуже висока вартість ліцензії
- Складність у налаштуванні та обслуговуванні
- Менш гнучка для невеликих проектів

6. SQLite - вбудована реляційна СУБД.

Переваги:

- Легка і проста у використанні
- Не потребує налаштування сервера
- Відмінна для мобільних додатків та вбудованих систем
- Відкрите програмне забезпечення

Недоліки:

- Обмежені можливості для масштабування

- Менша продуктивність для великих обсягів даних
- Відсутність деяких функцій, присутніх у потужніших СУБД

7. Cassandra - NoSQL (колонкоорієнтована СУБД).

Переваги:

- Відмінна для роботи з великими обсягами даних, розподіленими по кількох серверах
- Висока доступність та відмовостійкість
- Масштабується горизонтально

Недоліки:

- Складність у налаштуванні та адмініструванні
- Підходить не для всіх типів даних
- Відсутність повної підтримки транзакцій ACID

8. Redis – NoSQL (база даних у пам'яті).

Переваги:

- Надзвичайно висока швидкість операцій
- Підтримка різних типів даних, таких як списки, множини, хеші
- Ідеально підходить для кешування та реального часу
- Відкрите програмне забезпечення

Недоліки:

- Дані зберігаються в оперативній пам'яті, що може обмежувати обсяг даних
- Відсутність підтримки складних транзакцій
- Підходить не для всіх типів застосувань

У результаті проведення детального порівняння систем управління базами даних (СУБД), було отримано важливі висновки щодо їхніх можливостей, ефективності та застосування [9].

- MySQL підходить для веб-додатків і проєктів з високими запитами на читання.
- PostgreSQL ідеальна для проєктів, що вимагають складних запитів і транзакцій.

- Microsoft SQL Server корисна для інтеграції з продуктами Microsoft та складних аналітичних задач.
- MongoDB підходить для проєктів з великим обсягом неструктурованих даних.
- Oracle Database використовується в великих корпоративних системах з високими вимогами до надійності.
- SQLite відмінна для мобільних додатків і невеликих проєктів.
- Cassandra підходить для проєктів з великими обсягами даних, що потребують високої доступності.
- Redis ідеальна для кешування та додатків реального часу.

Тому зваживши всі плюси та мінуси цих СУБД було обрано MySQL за її щвидкість читання та простоту використання а також за її легкість та велику кількість користувачів що може посприяти знаходженю рішень для виконання задач та облегшує пошук рішення проблеми якщо вона виникає.

1.4 Взаємодія SQL-запитів і систем тестування

Використання SQL-запитів є важливою частиною тестування програмного забезпечення так як майже всі додатки та сайти використовують бази данних . Так як SQL це головний інструмент в їх створенні та налаштуванні то без нього не можливо протестувати деякі аспекти роботи программи. Далі наведено приклади того які запити потрібні в тестуванні.

Запити для підготовки тестових даних:

- SQL-запити можуть використовуватися для вставки тестових даних у базу даних перед виконанням тестів. Наприклад, ви можете виконати запити INSERT для створення потрібних записів у базі даних перед запуском тестового набору.

Запити для перевірки стану даних:

- Після виконання тесту, можна виконати SQL-запити для перевірки того, чи були правильно змінені дані в базі даних. Наприклад, після виконання деякої дії в тесті можна перевірити, чи відображаються відповідні дані у базі.

Запити для очищення даних після тестування:

- Після виконання тестового набору можна використовувати SQL-запити для видалення тестових даних з бази даних, щоб підготувати її до наступного тестування.

Приклади використання SQL у процесі тестування:

1. Юніт-тестування:

- При тестуванні окремих функцій чи методів програми можна використовувати SQL-запити для підготовки тестових даних та перевірки правильності результатів.

2. Інтеграційне тестування:

- Під час інтеграційного тестування можна виконувати SQL-запити для перевірки взаємодії різних компонентів системи та перевірки коректності даних у базі.

3. Системне тестування:

- Під час системного тестування можна виконувати SQL-запити для перевірки функціональності системи в цілому, включаючи відповідність даних в базі даних заданим критеріям.

4. Тестування бази даних:

- Саме для цієї категорії тестування SQL-запити використовуються найчастіше. Тут вони допомагають в перевірці ефективності та правильності роботи операцій з базою даних, таких як SELECT, INSERT, UPDATE, DELETE.

5. Регресійне тестування:

- Під час регресійного тестування можна використовувати SQL-запити для перевірки, чи відбулося зміна в поведінці системи щодо даних у базі після внесення змін до коду.

Використання SQL-запитів у системах тестування дозволяє покращити якість тестування, забезпечити надійність та консистентність даних у базі, а також забезпечити повноту тестового покриття в програмному забезпеченні [10].

РОЗДІЛ 2. ПРОЄКТУВАННЯ SQL-ЗАПИТІВ ДЛЯ ТЕСТУВАННЯ

2.1.Визначення вимог до баз даних та систем тестування

База данних повинна відповідати специфічним вимогам системи тестування і підтримувати ефективне виконання SQL-запитів для підготовки, перевірки та очищення даних. Нижче наведено кроки, які охоплюють формулювання вимог, структуру бази даних, визначення необхідних таблиць та зв'язків між ними [11].

Вимоги до тестування:

1. База даних повинна бути здатною швидко та надійно зберігати тестові дані та результати тестування.
2. Потрібна можливість легкої підготовки та очищення тестових даних.
3. Система повинна підтримувати транзакційність для забезпечення цілісності даних під час тестування.

Вимоги до SQL-запитів:

1. Запити повинні бути оптимізовані для швидкого виконання.
2. Необхідно забезпечити можливість складних запитів для перевірки різних аспектів даних (агрегація, об'єднання, підзапити).
3. Потрібна підтримка модифікації даних (INSERT, UPDATE, DELETE) для підготовки тестових сценаріїв.
4. Запити повинні бути здатні обробляти великі обсяги даних для реалістичного тестування продуктивності.

Структура бази даних

Структура бази даних повинна відображати основні об'єкти, які потребують тестування, а також підтримувати збереження результатів тестування та журналювання. Приклад структури може включати таблиці для користувачів, продуктів, замовлень та результатів тестування.

Визначення необхідних таблиць та зв’язків між ними

Таблиця Projects

- project_id : Унікальний ідентифікатор проекту
- project_name: Назва проекту
- description: Опис проекту

Таблиця Requirements

- requirement_id : Унікальний ідентифікатор вимоги
- project_id : Ідентифікатор проєкту (посилання на Projects)
- requirement_description: Опис вимоги

Таблиця TestCases

- testcase_id : Унікальний ідентифікатор тестового випадку
- scenario_id : Ідентифікатор сценарію (посилання на TestScenarios)
- testcase_description: Опис тестового випадку
- expected_result: Очікуваний результат

Таблиця TestResults

- result_id : Унікальний ідентифікатор результату
- testcase_id : Ідентифікатор тестового випадку (посилання на TestCases)
- execution_date: Дата виконання
- status: Статус (наприклад, пройдено, не пройдено)
- actual_result: Фактичний результат

Таблиця Defects

- defect_id : Унікальний ідентифікатор дефекту
- project_id : Ідентифікатор проекту (посилання на Projects)
- defect_description: Опис дефекту
- status: Статус дефекту (відкритий, закритий)

Визначення вимог до баз даних та систем тестування є критичним етапом у процесі розробки програмного забезпечення. Цей процес не лише допомагає забезпечити відповідність системи потребам бізнесу, але й визначає основні

параметри, які впливають на продуктивність, надійність та безпеку програмного забезпечення [12].

У результаті аналізу вимог до баз даних було встановлено, що функціональність, продуктивність, масштабованість та безпека є ключовими аспектами, які потрібно враховувати під час проектування та вибору СУБД. Для забезпечення ефективного зберігання та обробки даних, важливо вибрати СУБД, яка відповідає конкретним потребам проекту та забезпечує високий рівень надійності та безпеки.

У процесі визначення вимог до систем тестування були ідентифіковані критичні фактори, такі як автоматизація, швидкодія, масштабованість та інтеграція з іншими інструментами. Ефективна система тестування дозволяє забезпечити якість та надійність програмного забезпечення шляхом швидкого та автоматизованого виконання тестів, що дозволяє виявляти та усувати дефекти на ранніх етапах розробки.

Впевненість у відповідності баз даних та систем тестування встановленим вимогам є ключовим аспектом успішного завершення проекту з розробки програмного забезпечення. Детальне визначення вимог дозволяє забезпечити якість, ефективність та надійність програмного продукту, що в свою чергу сприяє задоволенню потреб користувачів та досягненню успіху на ринку [13].

2.2. Розробка SQL-запитів

Для початку роботи з Базами Даних потрібно написати декілька базових запитів для перевірки її роботи і для виявлення помилок які можуть виникнути під час розробки проєкту. Приклади цих запитів зображено на перерахованих нижче рисунках:

1. Запит на створення бази даних (Рисунок 2.1)
2. Запит на створення Таблиці (Рисунок 2.2)
3. Запит на внесення даних в таблицю (Рисунок 2.3)
4. Запит на перегляд даних з таблиці (Рисунок 2.4)

```
1 CREATE DATABASE `qa_db`
```

Рисунок 2.1 – Запит на створення бази даних

```
1 CREATE TABLE projects(
2     project_id int PRIMARY key AUTO_INCREMENT,
3     project_name varchar,
4     description text)
```

Рисунок 2.2 – Запит на створення Таблиці

```
1 INSERT INTO `projects`(`project_id`, `project_name`, `description`) VALUES ('TestProject','test project')
```

Рисунок 2.3 – Запит на внесення даних в таблицю

```
1 SELECT project_name FROM `projects` WHERE 1
```

Рисунок 2.4 – Запит на перегляд даних з таблиці

2.3 Створення складних запитів для перевірки різних аспектів системи

Розрізняють ряд складних SQL-запитів, які можуть бути використані для тестування функціональності і продуктивності системи управління базами даних (СУБД). Тестування включатиме перевірку різних аспектів системи, таких як обробка великих обсягів даних, складні агрегації, робота з кількома таблицями, використання підзапитів та обчислення [14].

1. Тестування злиття даних з кількох таблиць (JOIN)

Для тестування складних об'єднань даних використовуються таблиці projects та requirements. Завдання полягає в отриманні списку вимог з відповідними проектами, яке зображене на Рисунку 2.5.

```
SELECT
    requirements.requirement_id,
    requirements.requirement_description,
    projects.project_name
FROM
    requirements
JOIN
    projects ON requirements.project_id = projects.project_id
ORDER BY
    projects.project_name, requirements.requirement_description;
```

Рисунок 2.5 – Запит на перегляд даних з таблиці

2. Тестування агрегацій та групувань (GROUP BY)

Наступний запит підраховує кількість вимог для кожного проєкту, а також обчислює середній час виконання вимог.

3. Використання підзапитів (Subqueries)

Запит, який знаходить всі проєкти, що мають більше 5 вимог зображений на Рисунку 2.6.

```sql
SELECT
    project_name
FROM
    projects
WHERE
    project_id IN (
        SELECT
            project_id
        FROM
            requirements
        GROUP BY
            project_id
        HAVING
            COUNT(requirement_id) > 5
    );

```

Рисунок 2.6 – Запит, який знаходить всі проєкти, що мають більше 5 вимог.

2.3 Оптимізація і тестування SQL-запитів

Оптимізація SQL-запитів є критично важливою для забезпечення ефективної роботи системи управління базами даних (СУБД). Вона дозволяє зменшити час виконання запитів, знизити навантаження на сервери і покращити загальну продуктивність додатків. У цьому розділі розглянемо основні методи оптимізації SQL-запитів, включаючи індексування, використання ефективних запитів, нормалізацію баз даних та інші техніки [15].

1. Індексування

Індексування є одним з найефективніших способів прискорення пошуку і фільтрації даних. Індекси створюються на стовпцях таблиць і дозволяють СУБД швидко знайти необхідні рядки без сканування всієї таблиці.

2. Використання ефективних запитів

Оптимізація самих запитів також є важливим аспектом. Це включає використання найбільш ефективних методів написання SQL-коду.

Уникнення використання SELECT *: Використання SELECT * може призвести до надмірного витягу даних. Краще вказувати тільки ті стовпці, які дійсно потрібні.

3. Нормалізація баз даних

Нормалізація баз даних зменшує дублювання даних і забезпечує цілісність. Це включає розбиття великих таблиць на менші і встановлення зв’язків між ними [16].

Методи оптимізації SQL-запитів є ключовими для забезпечення ефективної роботи СУБД. Використання індексів, написання ефективних запитів, нормалізація баз даних, кешування, аналіз планів виконання та оптимізація зберігання даних допомагають значно покращити продуктивність системи. Впровадження цих методів дозволяє забезпечити швидку і надійну роботу бази даних, що є критично важливим для підтримки високої якості обслуговування користувачів і виконання бізнес-завдань.

Проєктування SQL-запитів для тестування є важливим етапом у розробці програмного забезпечення, оскільки воно визначає ефективність, продуктивність та надійність роботи з базою даних. Цей процес вимагає досконалого розуміння вимог до даних та функціональності, а також уміння написати оптимальні та ефективні SQL-запити для перевірки цих вимог.

У ході проєктування SQL-запитів для тестування було звернуто увагу на декілька ключових аспектів:

1. Функціональність: Запити були розроблені для виконання широкого спектру операцій, від вибірки даних до складних агрегатних операцій. Це дозволяє впевнитися, що функціональні вимоги до бази даних будуть виконані коректно та ефективно.

2. Продуктивність: Під час проєктування SQL-запитів були використані оптимальні методи індексації та оптимізації запитів, щоб забезпечити швидке виконання операцій та мінімізувати час очікування користувача.

3. Надійність: Кожен SQL-запит був ретельно перевірений на відсутність помилок та надійність обробки непередбачених сценаріїв. Це дозволяє підтримувати цілісність та надійність даних у базі даних.

4. Документація та тестування: Кожен розроблений SQL-запит був документований, що спрощує розуміння та підтримку коду. Також були

розроблені тестові сценарії для перевірки працездатності кожного запиту, що дозволяє виявити та усунути можливі помилки ще на етапі розробки.

У результаті ефективного проєктування SQL-запитів, можна забезпечуємо високу якість та ефективність роботи з базою даних, що в свою чергу сприяє успішній реалізації проекту та задоволенню потреб користувачів [17-19].

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ТЕСТУВАННЯ З ВИКОРИСТАННЯМ SQL-ЗАПИТІВ

3.1 Опис підготовки до тестування

Реалізація системи тестування з використанням SQL-запитів включає в себе створення набору тестових сценаріїв, які визначають вхідні дані, очікуваний результат та перевіряють, чи відповідає результат вимогам. Ці сценарії виконуються за допомогою SQL-запитів, які взаємодіють з базою даних і аналізують її стан.

Під час реалізації системи тестування необхідно враховувати різноманітні аспекти, такі як встановлення початкового стану бази даних перед тестуванням, виконання тестових сценаріїв з врахуванням різних вхідних даних та обробку отриманих результатів для перевірки їх відповідності очікуваному результату [20-21].

Загальний підхід до написання unit-тестів для отримання очікуваного результату з MySQL бази даних у pytest може виглядати так:

- Підключення до бази даних перед запуском тестів.
- Виконання запиту до таблиці testcases для отримання очікуваного результату для конкретного тестового випадку.
- Порівняння отриманого результату з очікуваним результатом.
- Закриття з'єднання з базою даних після завершення тестів.

Приблизний приклад коду для створення такого unit-тесту з використанням pytest та бібліотеки mysql-connector-python для зв'язку з MySQL базою даних зображено на рисунку 3.1.

```python
import mysql.connector

def connect_to_database():
    return mysql.connector.connect(
        host="localhost",
        user="user",
        password="",
        database="qa_db"
    )

con = connect_to_database()
def cases(con, test_case):
    cursor = con.cursor()

    # Виконання запиту до таблиці testcases для отримання даних
    cursor.execute("SELECT test_case, expected_result FROM testcases WHERE test_case = %s", (test_case,) )

    # Отримання результатів запиту
    testcases_data = cursor.fetchall()

    # Закриття курсора
    cursor.close()

    return testcases_data
```

```python
import sql_connector
sql_connector.con = con
def test_case1(case = sql_connector.cases(sql_connector.con,1) ):
    assert case[0][0] == 1
```

Рисунок 3.1 – Приклад коду для створення такого unit-тесту з використанням pytest та бібліотеки mysql-connector-python для зв'язку з MySQL бази даних

3.2 Реалізація тестових сценаріїв

Було створено 2 прості функції та розроблено сценарії для їх тестування/ Функція що знаходить число Фібоначчі зображена на Рисунку 3.2.

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Рисунок 3.2. – Функція що знаходить число Фібоначчі

Функція що знаходить другуий ступінь числа зображена на Рисунку 3.3.:

```
def square(x):
    return x ** 2
```

Рисунок 3.3 - Функція що знаходить другий степінь числа

Тестовий сценарій для функції fibonacci:

Вхідні дані: n = 0

Очікуваний результат: 0

Перевірка того, що функція повертає правильне значення для n = 0.

Тестовий сценарій для функції fibonacci:

Вхідні дані: n = 5

Очікуваний результат: 5

Перевірка того, що функція повертає правильне значення для n = 5.

Тестовий сценарій для функції square:

Вхідні дані: x = 1

Очікуваний результат: 1

Перевірка того, що функція правильно підносить число до квадрату.

Тестовий сценарій для функції square:

Вхідні дані: x = -3

Очікуваний результат: 9

Перевірка того, що функція правильно підносить від'ємне число до квадрату.

Тестовий сценарій для функції square:

Вхідні дані: x = 5

Очікуваний результат: 25

Перевірка того, що функція правильно підносить число до квадрату.

Код тестів показаний на Рисунку 3.4.

```
import sql_connector
import unit

def test_square(case = sql_connector.cases(sql_connector.con,1)):
    unit.square (case[0]) == case[1]

def test_square2(case = sql_connector.cases(sql_connector.con,4)):
    unit.square (case[0]) == case[1]

def test_square3(case = sql_connector.cases(sql_connector.con,3)):
    unit.square (case[0]) == 25

def test_fibonacci(case = sql_connector.cases(sql_connector.con,1)):
    assert unit.fibonacci(case[0]) == case[1]

def test_fibonacci2(case = sql_connector.cases(sql_connector.con,3)):
    assert unit.fibonacci(case[0]) == case[1]
```

Рисунок 3.4. – Код тестів

Результати тестів зображені на Рисунку 3.5.

```
========== test session starts ==========
platform win32 -- Python 3.12.2, pytest-8.2.1, pluggy-1.5.0
rootdir: D:\Test
collected 5 items

test_test.py .....                                    [100%]

========== 5 passed in 0.18s ==========
```

Рисунок 3.5 – Результати тестів

Результатом тестування вважаються всі дані, які пройшли Uniy-test.

Завдяки SQL-запитам в цих функціях отримується тестове значення та очікуваний результат що дозволяє змінювати тестові данні без необхідності змінювати сам тест.

ВИСНОВКИ

Під час роботи було успішно розроблено бази даних для тестування, які відображають структуру даних і операції, що потрібні для проведення тестів. Було створено різні типи SQL-запитів.

Досліджено використання SQL-запитів для тестування, що виявилося дуже корисним у реалізації тестових сценаріїв та перевірці правильності роботи програмного забезпечення. SQL-запити дозволили ефективно виконувати різні види тестів, перевіряти різні аспекти функціональності системи та використовувати різні типи даних для тестування на коректність.

Ефективність SQL-запитів у системі тестування: Розробка ефективних SQL-запитів є ключовим елементом у системі тестування, оскільки вони дозволяють отримувати швидкий та точний доступ до даних для проведення тестів. Важливо розробляти запити, які ефективно виконують тестові сценарії та забезпечують швидке виявлення помилок.

SQL-запити повинні бути надійними та стабільними для успішного виконання тестів. Це означає, що вони повинні коректно обробляти різні сценарії та вводити дані у відповідний стан для подальшого тестування. Важливо також перевіряти їх на відповідність очікуваним результатам.

Під час розробки SQL-запитів для системи тестування важливо враховувати можливості оптимізації та покращення продуктивності. Це може включати в себе використання індексів, оптимізацію запитів та уникнення зайвих операцій, що можуть сповільнювати виконання тестів.

Щоб забезпечити ефективну розробку та підтримку SQL-запитів у системі тестування, важливо вести документацію та дотримуватися стандартів програмування. Це допомагає зберігати консистентність та зрозумілість коду для всіх учасників процесу розробки.

SQL-запити можуть бути використані у системах автоматизованого тестування для проведення різних видів тестів, від регресійного до функціонального та навіть навантажувального тестування. Важливо розробляти

запити, які можуть бути легко інтегровані в автоматизовані тестові сценарії та забезпечують повноту покриття тестування.

Розробка SQL-запитів в системі тестування є процесом постійного вдосконалення. Важливо навчатися на помилках, аналізувати результати тестів та вдосконалювати запити для з абезпечення кращої ефективності та надійності тестування. Регулярний аудит і оптимізація існуючих SQL-запитів допомагають забезпечити, що вони відповідають поточним потребам та стандартам якості.

Важливо забезпечити ефективну співпрацю між розробниками та тестувальниками під час розробки SQL-запитів. Це допомагає враховувати потреби тестування під час проєктування та розробки баз даних, а також забезпечує швидке виявлення та виправлення помилок.

Розробка SQL-запитів повинна відбуватися в рамках загальної стратегії тестування, яка враховує всі аспекти тестування програмного забезпечення. Важливо розробляти запити, які покривають всі можливі сценарії та враховують різноманітні аспекти якості продукту.

Швидкі зміни у технологіях та вимогах до програмного забезпечення вимагають від фахівців з тестування постійного навчання та самовдосконалення. Регулярне вивчення нових методів тестування та вдосконалення навичок SQL допомагає забезпечити високу якість тестових процедур.

Після виконання SQL-запитів у системі тестування важливо відслідковувати та аналізувати результати, щоб виявити слабкі місця та можливість покращення. Це допомагає забезпечити постійне вдосконалення процесу тестування та підвищення якості програмного забезпечення.

У підсумку, розробка SQL-запитів у системі тестування є складним та важливим процесом, який вимагає від фахівців з тестування глибоких знань SQL, технологій баз даних та стратегій тестування. Послідовне дотримання найкращих практик, постійне навчання та співпраця у команді допомагають забезпечити успішне впровадження та підтримку програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Булатецька Л. В. Мова запитів SQL : текст лекцій нормативної навчальної дисципліни “Бази даних та розподілені інформаційно-аналітичні системи” / Булатецька Леся Віталіївна, Булатецький Віталій Вікторович. – Луцьк : СНУ імені Лесі Українки, 2018. – 92 с.
2. Про обробку запиту URL: https://docs.oracle.com/database/121/ TGSQL/ tgsql_sqlproc.htm#TGSQL186 (дата звернення 25.03.2024).
3. Обробка запиту оптимізатором запитів URL: https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL (дата звернення 01.04.2024).
4. Все про хеш-з’днання URL: https://www.sql.ru/articles/mssql/2007/051103hashjoin.shtml (дата звернення 15.04.2024).
5. Оператори SQL URL: https://www.tutorialspoint.com/sql/sql–opera–tors.htm (дата звернення 17.04.2024).
6. Matt David SQL Optimizaation URL: https://cdn2.hubspot.net/hubfs/392937/SQLOptimization.pdf?__hstc=158613477.a29bea58f3f60dc494157e6c72951ac5.1616149514669.1616149514669.1616149514669.1&__hssc=158613477.2.1616149514670&__hsfp=3821063877&hsCtaTracking=9311e47c–0dbe–451b–9129–3cbce358df08|3d2c58de–3379–41ef–9c8d–f0492f3bf1bd (дата звернення 19.04.2024).
7. Phil Factor Тимчасові таблиці URL: https://www.red-gate.com/simple-talk/sql/t-sql programming/temporary-tables-in-sql-server/ (дата звернення 25.04.2024).
8. Robert Sheldon SQL Server Common Table Expression (CTE) Basics URL: https://www.red-gate.com/simple-talk/sql/t-sql-programming /sql-server-cte-basics/ (дата звернення 27.04.2024).
9. Common Query Optimization Problems URL: http://etutorials.org/SQL/microsoft+sql+server+2000/Part+V+SQL+Server+Inter

nals+and+Performance+Tuning/Chapter+35.+Understanding+Query+Optimizatio n/Common+Query+Optimization+Problems/ (дата звернення 30.04.2024).

10. S. Patil, P. Damare, J. Sonawane, and N. Maitre, “Study of performance tuning techniques,” Journal of Emerging Technologies and Innovative Research (JETIR), vol. 2, no. 3, pp. 499–502, 2015.55 (дата звернення 01.05.2023).
11. R. Bhajipale, P. Bisen, A. Meshram, and S. S. Thakur, “SQL tuner,” International Journal of Computer Trends and Technology, vol. 33, no. 1, pp. 29–32, 2016.
12. J. Habimana, “Query optimization techniques – tips for writingefficient and faster SQL queries,” International Journal of Scientific & Technology Research, vol. 4, no. 10, pp. 22–26, 2015.
13. R. Sahal, M. Nihad, M. H. Khafagy, and F. A. Omara, “iHOME: Index based JOIN query optimization for limited big data storage,” Journal of Grid Computing, vol. 16, no. 2, pp. 345–380, 2018.
14. S. Batra, S. Sachdeva, and S. Bhalla, “Entity attribute value style modeling approach for archetype based data,” Information, vol. 9, no. 2, pp. 1–30, 2018.
15. D. Patel and P. Patel, “An approach for query optimization by using schema object base view,” International Journal of Computer Applications, vol. 119, no. 16, pp. 21–24, 2015.
16. D. Colley and C. Stanier, “Identifying new directions in database performance tuning,” Procedia Computer Science, vol. 121, p. 260 – 265, 2017.56
17. C. G. Corlatan, M. M. Lazar, V. Luca, and O. T. Petricica, “Query optimization techniques in Microsoft SQL server,” Database Systems Journal, vol. V, no. 2, pp. 33–48, 2014.
18. “New directions in database performance tuning,URL: https://www.brentozar.com/archive/2015/10/how–to–download–the–stack–overflow–database–via–bittorrent/ (дата звернення 15.05.2024).
19. S. S. Srinivas, B. V. Naik, and J. S. A. Kumar, “Query minimization methods,” International Journal of Scientific & Engineering Research, vol. 8, no. 5, pp. 30–33, 2017ю