РОЗРОБКА ЗАСОБУ ПРОЕКТУВАННЯ ВИСОКОНАВАНТАЖЕНИХ РЕЛЯЦІЙНИХ СИСТЕМ ЗБЕРІГАННЯ ДАНИХ: ОПТИМІЗАЦІЯ СТРУКТУРИ ТА ЗАПИТІВ SQL

УДК 004.652.4 : 004.4'22

КОЛЕСНИК Людмила Володимирівна
кандидат технічних наук, доцент, кафедра системотехніки, Харківський національний університет радіоелектроніки
Наукові інтереси: проектування інформаційних систем, управління і моделювання інформаційних ресурсів.
e-mail: liudmyla.kolesnyk@nure.ua, kolesnik_l_v@ukr.net

КИРИЧЕНКО Наталія Андріївна
студентка, кафедра системотехніки, Харківський національний університет радіоелектроніки
Наукові інтереси: проектування систем зберігання даних.
e-mail: nataliia.kyrychenko@nure.ua

КОСТОГЛОТ Ігор Валерійович
студент, кафедра системотехніки, Харківський національний університет радіоелектроніки
Наукові інтереси: інформаційні системи, web-програмування.
e-mail: ihor.kostohlot@nure.ua

ВСТУП

У сучасному світі кожне програмне забезпечення (ПЗ), веб-сайт і в цілому практично кожна система працює з великими об'ємами інформації та даних, які вимагають ресурсів для зберігання, обробки та редагування. Нині виконання цих цілей покладено на спеціально призначені для цього засоби зберігання та обробки даних, що мають усталену назву систем управління базами даних (СУБД), різновидів яких у наш час існує вже кілька десятків. Дана стаття буде присвячена одній конкретній СУБД під назвою MySQL.

АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ПОСТАНОВКА ПРОБЛЕМИ

СУБД MySQL є реляційною СУБД. Це означає те, що дана СУБД заснована на реляційних залежностях, а саме: «відношення», «залежність», «зв'язок». Простіше кажучи, в даній СУБД всі дані представляються не суцільним і безупорядоченним набором інформації, а мають чітку залежність один від одного і від тієї предметної області, для якої була розроблена база даних (БД) в СУБД MySQL. СУБД MySQL має досить великий набір інструментів для створення та управління БД, підтримує велику кількість форматів таблиць БД, як з підтримкою повнотекстового пошуку (MyISAM, InnoDB), так і з підтримкою транзакцій та зовнішніх ключів (InnoDB). Також СУБД MySQL є вільно поширюваним ПЗ, що робить дану СУБД ще більш популярною [1].

Однак багатофункціональність та гнучкість СУБД MySQL також є і мінусом, оскільки найчастіше однозначного рішення у розумінні того, який тип таблиці вибрати, які індекси проставити, як оптимально зв'язати таблиці в БД – немає. Первісне планування структури бази даних здебільшого не враховує великої кількості необхідних таблиць і атрибутів та через це, як наслідок, спроектована база даних має більш громіздкий вигляд і займає більше часу на розробку.

СУБД MySQL, як і багато інших спеціалізованих засобів, має значну кількість налаштувань і особливостей, та якщо на початку розроблена БД працює досить швидко, то буде помилкою вважати, що вона збереже свою швидкість в режимі високої завантаженості та зростанні її об'ємів. Крім налаштувань самої СУБД та структури БД, важливою деталлю є безпосередньо самі SQL запити до бази даних, які також можуть бути оптимальні для однієї структури [2], налаштувань та навантаженості БД і можуть бути абсолютно не оптимальними при внесенні коректувань в структуру, налаштування, а також при підвищенні / зниженні навантаження на БД.

МЕТА І ЗАВДАННЯ ДОСЛІДЖЕННЯ

Метою роботи є розробка програмного засобу для автоматичного проектування / перепроектування високонавантаженої реляційної системи зберігання даних за допомогою оптимізації SQL запитів до БД [3], а також безпосередньо структури БД. Досягнення поставленої мети передбачає вирішення наступних завдань:

– дослідження методів оптимізації структури бази даних;

– дослідження методів оптимізації швидкості виконання SQL запитів;

– розробка програмного засобу для автоматизації процесу оптимізації бази даних і SQL запитів.

ПЕРЕДІСТОРІЯ ВИНИКНЕННЯ ІДЕЇ

Ідеєю для розробки даного програмного засобу виявився власний досвід роботи над веб-проектом, який за кілька років від низьковідвідуваного ресурсу перейшов в стадію високонавантаженого та масштабного. Вихідний код проекту був розроблений самостійно, використовуючи мову PHP в якості серверної частини і JS на фронтенді, в якості СУБД була обрана MySQL. За рахунок оптимально написаного коду проект спочатку мав високу швидкість роботи, час генерації однієї сторінки займав ~ 0.002-0.005 секунд, проте по мірі зростання кількості даних в БД та кількості запитів до БД веб-ресурс почав досить сильно гальмувати. Після проведення нескладних тестів було виявлено, що причина є саме в БД. Запити, які виконувалися за ~ 0.0003 секунди, тепер виконувалися за ~ 0.04 секунди. Першим етапом оптимізації послугувало те, що були переглянуті та поліпшені SQL запити, однак належного ефекту без внесення змін до структури бази даних це не дало. Другим етапом послугувало те, що спільно зі зміною запитів SQL також була змінена і структура БД – великі таблиці були розбиті на більш дрібні, кількість зв'язків у таблиці зросла, проте це хоча і дало бажаний ефект щодо прискорення, але ненадовго. При подальшому зростанні проекту і, відповідно, його БД, потрібна була чергова оптимізація. Цього разу до вищевказаних дій було додано дублювання часто використовуваних даних у кілька таблиць, були застосовані індекси для багатьох стовпців, а також були оптимізовані формати та розмірності атрибутів даних в таблиці. Дані дії дозволили відновити початкову швидкість роботи бази даних. Також, заради експерименту, таблиці в БД були переведені з типу InnoDB до типу MyISAM, що також дозволило виграти додаткову швидкість у обробці SQL запитів, хоча спочатку саме з типом InnoDB запити виконувалися значно швидше, ніж з типом даних MyISAM.

Далі буде детально розглянута послідовність дій для оптимізації бази даних з рекомендаціями, які були отримані на особистому досвіді.

ВИБІР ТИПУ (ENGINE) БАЗИ ДАНИХ

На першому етапі оптимізації БД необхідно визначитися з вибором типу бази даних [5]. На даний момент існує два основних типи – це InnoDB [6] і MyISAM. Не будемо вдаватися в тонкощі відмінностей типів баз даних та відразу викладемо коротку суть кожної з них.

Тип бази даних MyISAM бажано обирати виключно у тому випадку, якщо немає необхідності реалізації транзакцій на рівні бази даних, немає необхідності організації зовнішніх ключів, а також в проекті переважає один з видів доступу – читання або запис, тобто новинний ресурс або, наприклад, логування . Також MyISAM менше займає місця на диску, однак не підтримує мультибайтове кодування.

У всіх інших випадках бажано обирати тип бази даних InnoDB, а також даний тип необхідно вибирати тоді, коли можна пожертвувати швидкістю виконання операції в обмін на підвищену безпеку даних,

наприклад, в додатках фінансового характеру та строгої звітності.

ПРОЕКТУВАННЯ СТРУКТУРИ

Коли визначилися з типом бази даних, далі необхідно безпосередньо спроектувати її структуру. При проектуванні структури необхідно завжди стежити та дотримуватися того, щоб структура відповідала третій нормальній формі [4], однак це не завжди буває доцільним. У деяких випадках можливо зменшити навантаження на базу даних за рахунок перенесення деяких даних з бази в мовні файли ПЗ, що розробляється. Зараз ми докладніше опишемо, що конкретно мається на увазі.

Припустимо, що є база даних у третій нормальній формі і є таблиця з логіном користувача, а також таблиця з коментарями.

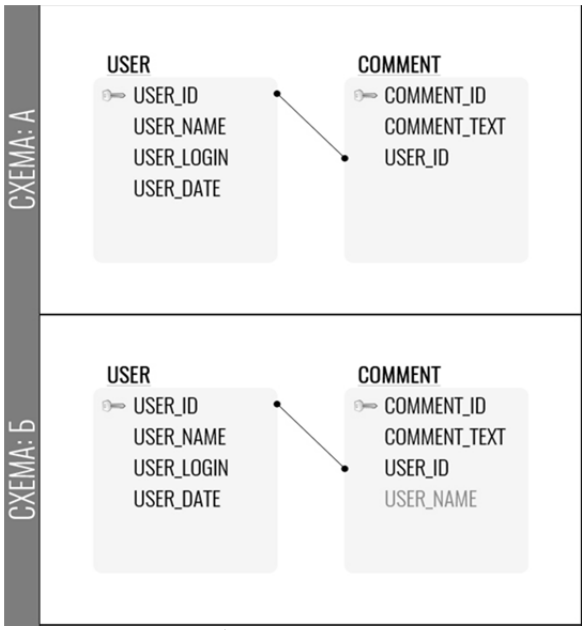

Рис. 1. Схема БД з таблицями користувачів і коментарів

На рисунку 1 зображені два варіанти схеми описаної БД. Припустимо, що спільно з текстом коментарів необхідно також виводити ім'я користувача. Як видно на схемах, зображених на рисунку 1, схема А містить виключно поле user_id, ID користувача, який залишив коментар, в той час, як схема Б, крім поля user_id, містить також і поле user_name, з ім'ям користувача.

У схемі А позначено, що у веб-додатку необхідно буде виконувати 1 складний запит з використанням JOIN або виконувати 2 простих запити – перший запит на вибірку коментарів, другий запит – на вибірку імен авторів коментарів. Даний метод дозволяє зменшити розмір БД за рахунок того, що імена авторів не будуть дублюватися, проте даний варіант становить значно більше навантаження на базу даних.

Схема Б дозволяє за 1 запит зробити вибірку всіх необхідних коментарів відразу ж з іменами користувачів. В даному варіанті отримано незначне зростання розміру бази даних, однак швидкість виконання запитів та роботи бази даних значно виграє перед варіантом, зображеним на схемі А.

Тепер припустимо, що є база даних у третій нормальній формі та є таблиця з інформацією про користувача.

На рисунку 2 зображено дві схеми з варіантами зберігання інформації про стать користувачів. На схемі А зображена таблиця користувачів з полем gender типу TINYINT (1) та додаткова таблиця, в якій буде всього три сутності – «не вказано», «чоловік» і «жінка». На схемі Б зображена одна таблиця, де поле gender представлено типом ENUM з варіантами значень «0», «1», «2». У чому ж відмінності в даних схемах?

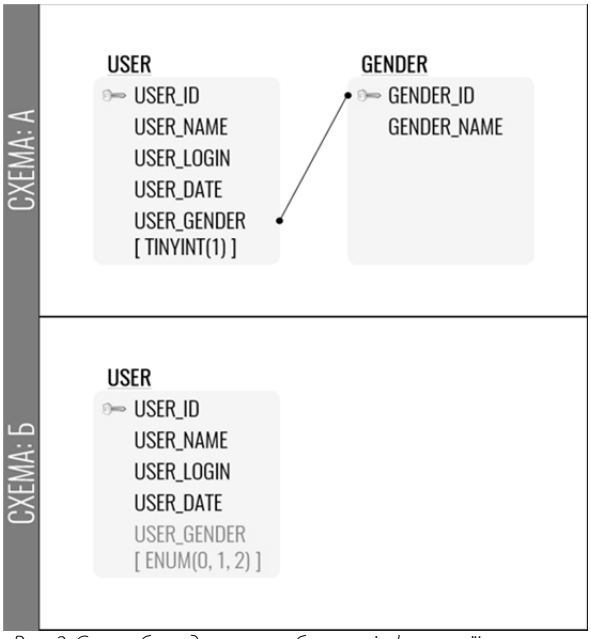

Рис. 2. Схема бази даних з таблицею інформації про користувача

Як і у попередньому випадку, на схемі А позначено, що у веб-додатку необхідно буде виконувати або 1

складний запит з використанням JOIN, або 2 простих – перший запит на вибірку самих користувачів, другий запит на вибірку варіантів значення поля gender користувачів за їх ID. Дана маніпуляція аналогічно дозволяє зменшити розмір бази даних за рахунок того, що значення поля gender не будуть дублюватися, проте у підсумку даний варіант становить значно більше навантаження на базу даних.

У схемі Б буде виконуватися усього один запит до бази даних. У результаті буде отримана вибірка, де у полі gender буде міститися одне зі значень – «0», «1» або «2». У такому випадку вже безпосередньо у ПЗ повинно бути чітко зазначено, що значення «0» відповідає значенню «не вказано», «1» – «чоловік» та «2» – «жінка».

Також можливий варіант, коли дані все ж важливо зберігати спочатку саме в базі даних, в цьому випадку можливе використання схеми А, але з кешуванням результатів вибірки з таблиці gender_table. В даному випадку ми отримуємо при першому запиті до ПЗ роботу за схемою А, а при всіх наступних – за схемою Б, оскільки значення в даному випадку вже будуть взяті не з бази даних, а з кешу.

Одним з важливих параметрів є обладнання, особливо при використанні кешування. Найбільш оптимальним буде використання кешування на серверах та пристроях з FLASH-пам'яттю та / або SSD-дисками.

РОЗБИТТЯ ГРОМІЗДКИХ ТАБЛИЦЬ

Наступний етап полягає у тому, що необхідно врахувати максимально можливий обсяг та розмір таблиці у базі даних [7]. Виключно з особистого досвіду можна зробити висновок, що найбільш оптимальне число записів у одній таблиці не повинно перевищувати 100 000 записів, при цьому обсяг таблиці не повинен перевищувати 100 Mb. Однак дані значення можуть відрізнятися в залежності від потужності обладнання, частоти запитів до таблиці, а також в залежності від того, наскільки важлива висока швидкість виконання запиту. Якщо час відповіді на запит немає великого значення або ж запити до даної таблиці виконуються раз на добу або навіть рідше, тоді даний етап оптимізації для поточної таблиці можна пропустити.

Однак, якщо важливо щоб запит виконувався максимально швидко, а таблиця може мати як 1 млн, так і більш записів або обсяг в кілька гігабайт, то вкрай рекомендовано її розділити. Розділити громіздку таблицю, котра зображена на рисунку 3 (А), можливо на логічні частини, наприклад – за країнами, як зображено на рисунку 3 (Б), за гендерною принадлежністю, за інтересами, або за значеннями індексів «від ... до».

При поділі громіздких таблиць виникає необхідність використовувати додаткову таблицю з покажчиками на таблиці, що, в свою чергу, вимагає виконання додаткових запитів до бази даних, тому не слід розбивати даним чином таблиці, в яких не передбачається значний обсяг даних. Якщо ж в таблиці буде досить громіздкий обсяг даних, то навіть з урахуванням додаткових запитів швидкість отримання підсумкової вибірки буде набагато швидше, ніж у випадку одного запиту до громіздкої бази даних.

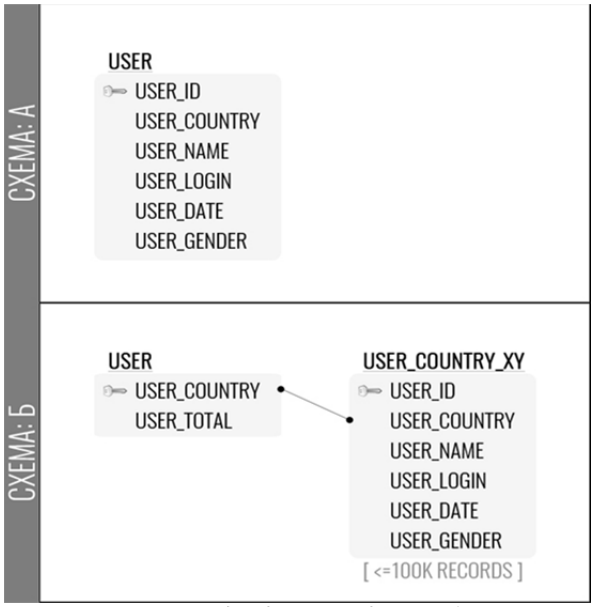

Рис. 3. Приклад поділу громіздких таблиць

ЗАЗНАЧЕННЯ ТИПІВ ТА РОЗМІРНОСТЕЙ ПОЛІВ ДАНИХ (АТРИБУТІВ)

Кожне поле (атрибут) в таблиці бази даних повинен мати свій тип даних та при цьому розмірність поля повинна бути мінімальною. Розглянемо докладніше, що це означає.

Припустимо, що необхідно зберігати у базі даних інформацію про ім'я користувача та дату його реєстрації на сайті в UNIX форматі. Отже, ім'я користувача – це текст, дата в UNIX форматі, логічно, теж могла бути вказана у вигляді тексту. В результаті для імені користувача та дати можливо указати тип даних TEXT, як показано на рисунку 4 (А). Однак це не є оптимальним, оскільки на тип TEXT база даних буде резервувати значно більше місця, ніж необхідно для зберігання даних про ім'я користувача та дату реєстрації. До того ж, ім'я користувача не буде містити значних обсягів тексту, для яких передбачено тип TEXT, а дата реєстрації в UNIX форматі і зовсім не буде перевищувати розмірності в 11 знаків.

Таким чином, можливо припустити, що у полі «ім'я користувача» може бути максимум 18 символів, відповідно для зберігання інформації у даному полі нам цілком буде досить типу VARCHAR з розмірністю у 18 символів. Поле «дата реєстрації», як вже було описано раніше, може містити усього 10-11 знаків. До того ж, дане поле числове та невід'ємне, а також, оскільки дата реєстрації нині не може бути раніше 1 січня 1970 року, то для зберігання дати реєстрації буде досить типу INT з розмірністю в 11 знаків та атрибутом unsigned. Атрибут unsigned означає, що значення поля може бути виключно додатним числом. Таблиця з новими розмірностями полів та атрибутами зображена на рисунку 4 (Б).

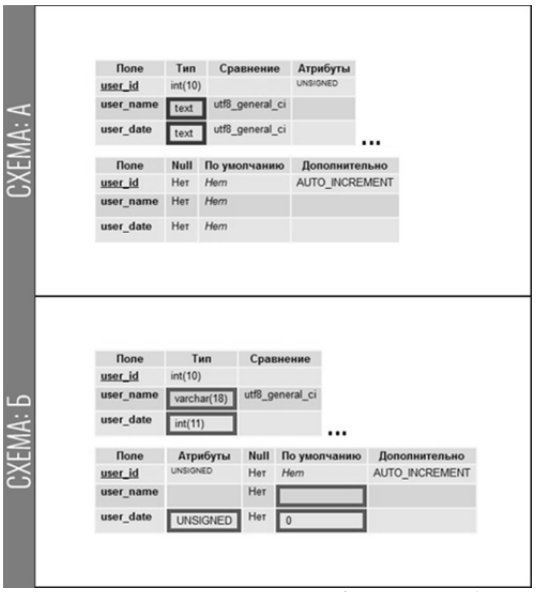

Рис. 4. Типи та розмірності полів даних (атрибутів)

Вибір оптимального типу та розмірності полів даних є дуже важливим у проектуванні структури БД [8]. Чим більш оптимально буде підібраний тип даних та розмірність поля, тим менше буде у підсумку обсяг БД, а відповідно і швидкість роботи БД буде значно вище. У СУБД MySQL, як і в багатьох інших СУБД, існує оптимізація на рівні ядра для роботи з відповідними типами даних. Це означає те, що записати / зчитати числові дані з поля з числовим типом даних СУБД буде простіше, ніж з поля, наприклад, з текстовим типом даних.

ЗАСТОСУВАННЯ ІНДЕКСІВ ДО ПОЛІВ ДАНИХ (АТРИБУТІВ)

Проставлення індексів до полів даних є невід'ємним етапом в оптимізації БД. При створенні будь-якої таблиці в базі даних, незалежно від її призначення, рекомендується використовувати хоча б один індекс – індекс на поле з порядковим номером запису в таблиці, найчастіше це поле ID, який називається PRIMARY_KEY з атрибутом auto_increment [9]. PRIMARY_KEY – це первинний ключ, за яким СУБД буде ідентифікувати запис у таблиці БД, а атрибут auto_increment означає, що запис в це поле буде проводитися автоматично та кожне нове значення буде на одиницю більше попереднього, тобто це буде лічильник записів у таблиці, який буде автоматично заповнюватися.

Крім PRIMARY_KEY існують ще інші типи індексів, такі як UNIQUE та INDEX. Індекс UNIQUE означає, що дані у полі, до якого застосовано даний індекс, не можуть повторюватися, тобто вони завжди унікальні. Індекс INDEX дозволяє індексувати дані по типу бінарного дерева, що, в свою чергу, може дозволити прискорити швидкість виконання запитів більш ніж у два рази.

Однак слід зазначити той факт, що надлишок індексів, як і їх нестача, однаково погано позначаються на швидкості виконання запитів до бази даних. Також не рекомендується застосовувати індекси до полів, дані в яких часто змінюються, оскільки після кожної зміни такого поля СУБД буде необхідно перебудувати індекс спочатку. Перебудувати індекс СУБД буде потрібно і у разі видалення \ додавання нового запису у таблицю. У цьому випадку СУБД потрібно перебудовувати індекс не тільки по одному полю, а по всіх полях, до яких застосовано індекс.

ОПТИМІЗАЦІЯ SQL-ЗАПИТІВ

Після того, як структура бази даних була оптимізована, можна переходити до наступного етапу – на етап оптимізації SQL запитів. Відразу потрібно зазначити кілька моментів, яких бажано уникати при написанні SQL запитів, а саме – використовувати якомога менше:

– вкладених підзапитів;

– операторів JOIN та інших операторів конкатенації;

– функції групування на великих таблицях, наприклад таких, як COUNT();

– повних вибірок рядків, тобто не використовувати оператор *;

– замінників імен стовпців.

Також, для отримання оптимальної швидкості виконання SQL запиту необхідно пам'ятати, що порядок полів у SQL запиті бажано використовувати відповідно до порядку полів у таблиці баз даних [10]. У обов'язковому порядку рекомендується використовувати той порядок полів в SQL запитах, який був при призначенні індексів. На рисунку 5 зображено приклад з неправильним порядком полів у SQL запиті (А) та з правильним порядком полів у запиті, без урахування індексів (Б) та з урахуванням індексів (В).

Якщо все ж таки довелось використати оператор JOIN, слід пам'ятати, що порядок з'єднання таблиць повинен бути таким, щоб пошук відповідного індексу проводився у приєднуваній таблиці, тобто, FROM ГОЛОВНА LEFT JOIN ПРИЄДНУВАНА ON ПРИЄДНУВАНА.ID = ГОЛОВНА.SUBID. Таким чином пошук по ПРИЄНУВАНІЙ таблиці буде проводитися в рази швидше [11].

Рис. 5. Приклади правильних та неправильних SQL запитів

Вибірку та сортування бажано робити за полями індексів, а в найкращому варіанті і зовсім за полем з PRIMARY KEY. Вставку INSERT даних в базу краще робити не поштучно, а об'єднуючи її в невеликі групи до 100 записів за 1 раз [12]. Оновлення даних UPDATE також практичніше робити не за всією кількістю полів, а тільки за тими полями, які оновилися. Наприклад, коли користувач редагує свою анкету та при цьому змінює тільки одне поле з десяти – необхідно на рівні ПО визначити, яке саме поле було змінено та тільки це поле оновити в базі даних.

Якщо після всіх дій запит так і залишається ресурсоємним та довготривалим – рекомендується кешувати результат виконання даного SQL запиту.

ОПИС ПРИНЦИПІВ РОБОТИ ПЗ

Програмний засіб (ПЗ) для оптимізації та побудови нової структури бази даних MySQL функціонує у двох режимах – напівавтоматичний та автоматичний.

У напівавтоматичному режимі під час виконання оптимізації та побудови нової структури ПЗ дозволяє користувачеві підтверджувати або відхиляти в ручному режимі кожне нове оптимальне рішення, а також, в разі кількох варіантів, вибрати найкращий з них на думку користувача.

В автоматичному режимі весь процес оптимізації ПЗ проводить самостійно, без участі користувача.

Незалежно від режиму ПЗ, до початку роботи користувачеві необхідно вказати в якості вхідних параметрів доступи до існуючої бази даних, доступи до бази даних для проведення тестування та доступи до бази даних, в якій буде сформована остаточна структура нової бази даних, список частовиконуваних та проблемних SQL запитів. Також користувачеві необхідно вибрати режим роботи програми та кроки оптимізації структури бази даних, тобто користувач може сам вказати, що треба оптимізувати, а що не треба.

Після завершення роботи ПЗ користувач отримує нову структуру бази даних, оптимізовану спеціально під його запити, отримує виправлені відповідно до нової структури бази даних SQL запити та отримує список рекомендацій, при наявності таких для його бази даних. Також користувачеві відображається безпосередньо результат виконання оптимізації у вигляді звіту про виконану роботу – які таблиці були злиті або розбиті, де

та які індекси були проставлені і т.д. Після кожного кроку оптимізації у звіті додається результат прискорення виконання SQL запитів на новій структурі бази даних в процентному еквіваленті.

Результат прискорення виконання SQL запитів зображено на (1) і вираховується як різниця між сумою часу виконання середнього значення по 100 тестів на новій та на старій структурі бази даних.

$$xTime = \frac{\sum_{n=1}^{100} SQLtOld}{100} - \frac{\sum_{n=1}^{100} SQLtNew}{100}, \quad (1)$$

де $SQLtOld$ – час виконання одного запиту до MySQL зі старою структурою бази даних у секундах;

$SQLtNew$ – час виконання одного запиту до MySQL з новою структурою бази даних у секундах;

$xTime$ – різниця між часом виконання запитів до MySQL зі старою структурою бази даних та новою структурою бази даних у секундах. Якщо $xTime$ більше 0, тоді знайдено більш оптимальній варіант; якщо $xTime$ менше 0, тоді – менш оптимальний варіант запиту; якщо 0, тоді обидва варіанти рівні між собою.

Мова реалізації програмного засобу – PHP. Для оформлення інтерфейсу користувача також були використані такі технології для веб-програмування, як – HTML, CSS, JavaScript (Ajax).

ДЕМОНСТРАЦІЯ ПРОГРАМНОГО ЗАСОБУ ТА РЕЗУЛЬТАТІВ ЙОГО РОБОТИ

Інтерфейс програмного засобу зображено на рисунку 6.

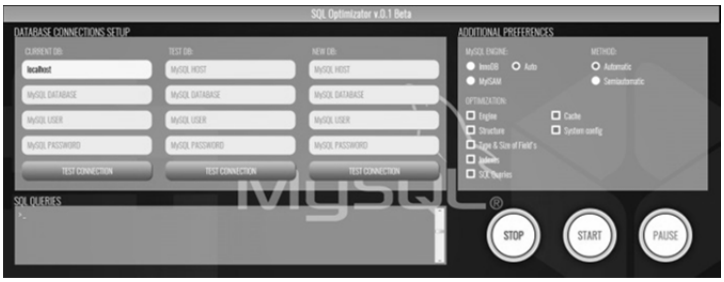

Рис. 6. Інтерфейс програмного засобу

Під час роботи ПЗ у напівавтоматичному режимі користувачеві відображаються проміжні діалогові фрейми, у котрих він може затвердити результат, переглянути інші запропоновані варіанти оптимізації або ж зовсім пропустити цей крок. Також у ході виконання користувачеві відображається вікно з логами роботи ПЗ, на основі яких і буде пізніше надано звіт про виконані дії. Приклад проміжного діалогового фрейму, а також вікна з логами роботи ПЗ зображено на рисунку 7.

Рис. 7. Вікно з проміжним діалоговим фреймом та логами

Після завершення роботи ПЗ користувачеві надаються результати оптимізації структури бази даних та SQL запитів. Також користувач має можливість завантажити на свій ПК: звіт про оптимізацію, структуру нової бази даних у SQL форматі, нові SQL запити, які будуть відповідати новій структурі бази даних.

ВИСНОВКИ

У даній статті були розглянуті основні проблеми СУБД MySQL та організації з її використанням високонавантаженої реляційної системи зберігання даних. Виходячи з особистого досвіду авторів даної статті, безпосередньо по кожному з можливих напрямків оптимізації роботи СУБД MySQL були дані рекомендації. Таким чином, у статті були описані кроки вибору типу (engine) бази даних MySQL, були розглянуті проблеми, пов'язані з проектуванням структури бази даних та надано рекомендації щодо їх усунення, був запропонований варіант вирішення проблеми громіздкості таблиць у базі даних. Також були описані дії та надано рекомендації щодо проставлення типів та розмірностей полів даних (атрибутів), були розглянуті індекси полів бази даних і те, як оптимально їх використовувати. Були дані рекомендації зі складання та використання SQL запитів.

У результаті розгляду вищевказаних проблем та на підставі наданих рекомендацій щодо оптимізації структури та запитів у базах даних MySQL було описано принцип роботи програмного засобу, який в автоматичному режимі буде проводити перепроектування вже існуючої бази даних з метою прискорення її роботи.

У статті були наведені результати експериментів, отриманих дослідним шляхом з використанням бета версії програмного засобу та бази даних діючого високонавантаженого сервісу. За підсумками експериментів були отримані позитивні результати, а саме – в автоматичному режимі програмний засіб надав новий варіант структури бази даних, яка виявилася на 8.43% швидше попереднього варіанту, отриманого шляхом самостійного ручного перепроектування.

ДЖЕРЕЛА:

1. SQL. Zbirny`k receptiv / E. Molinaro. – M. : Vy`d. Sy`mvol-Plyus, 2009. – 672 s.
2. Advanced Data Structures / P. Brass. – C. : Vy`d. Cambridge University Press, 2014. – 474 s.
3. MySQL. Opty`mizaciya produkty`vnosti / B. Shvarcz, P. Zajcev, V. Tkachenko, D. D. Zoodnaj, D. Dzh. Balling, A. Lencz. – M. : Vy`d. Sy`mvol-Plyus, 2010. – 832 s.
4. Zabezpechennya vy`sokoyi dostupnosti sy`stem na osnovi MySQL / Ch. Bell, M. Ky`ndal, L. Talmann. – M. : Vy`d. Russkaya Redakcy`ya, 2012. – 624 s.
5. Effective MySQL Optimizing SQL Statements / R. Bradford. – NY. : Vy`d. McGraw-Hill Education, 2011. – 184 s.
6. Instant InnoDB: short, fast, focused / M. Reid. – B. : Vy`d. Packt Publishing, 2013. – 88 s.
7. Database Systems: A Practical Approach to Design, Implementation, and Management / T. Connolly, C. Begg. – L. : Vy`d. Pearson, 2014. – 1440 s.
8. Bazy` dany`x. Proektuvannya, realizaciya i suprovid. Teoriya ta prakty`ka / T. Konnolly`, K. Begg. – M. : Vy`d. Vy`l`yams, 2017. – 1440 s.
9. MySQL: vy`kory`stannya ta administruvannya / V. Vasvani. – SPb. : Vy`d. Py`ter, 2011. – 368 s.
10. Beginning Database Design: From Novice to Professional / C. Churcher. – NY. : Vy`d. Apress, 2012. – 252 s.
11. Vvedennya do sy`stem baz dany`x / K. Dzh. Dэjt. – M. : Vy`d. Vy`l`yams, 2017. – 1328 s.
12. Relational Database Design and Implementation, Fourth Edition / J. L. Harrington. – B. : Vy`d. Morgan Kaufmann, 2016. – 712 s.

Рецензент: д.т.н., проф. К.Е. Петров
Професор кафедри штучного інтелекту ХНУРЕ