УДК 004.06 **doi: 10.26906/SUNZ.2023.2.151**

С. В. Суліма, О. Д. Єрмолаєв

Національний технічний університет України «КПІ імені Ігоря Сікорського», Київ, Україна

МЕТОД ОПТИМІЗАЦІЇ SQL ЗАПИТІВ СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ

Анотація. Актуальність. Розмір баз даних, який відноситься до обсягу збережених даних, може значно варіюватися. Зрозуміло, що чим більший розмір бази даних, тим більше часу потрібно на пошук необхідної інформації, що призводить до збільшення часу обробки запитів сервером. З одного боку, цю проблему вирішують шляхом підвищення продуктивності комп'ютерів, на яких розташовані системи управління базами даних (СУБД). Однак, просто підвищення продуктивності комп'ютерів недостатньо; часто значно кращі результати можна досягти шляхом зміни алгоритмів обробки SQL-запитів. Таким чином, незважаючи на те, що роботи з оптимізації SQL-запитів ведуться десятиліттями, через зростання темпів накопичення інформації та навантаження на сервери баз даних, така робота стала ще більш актуальною. **Мета.** Основною метою цієї роботи є покращення швидкості виконання вхідних запитів у реляційній базі даних, забезпечуючи високу продуктивність та зручну реалізацію для користувача. Це буде досягнуто шляхом розробки вдосконаленого методу оптимізації, який синтезує складні SQL-запити з великої кількості простіших запитів, тим самим підвищуючи загальну ефективність та зручність використання. **Метод.** У статті представлено метод оптимізації синтезу складних SQL–запитів з безлічі простих, що дозволяє підвищити швидкість виконання вхідного запиту реляційною базою даних при одночасному забезпеченні високої продуктивності та легкості застосування. **Результати.** Розроблено метод оптимізації SQL-запитів спеціально для ситуацій, коли швидкість вибірки даних погіршується з часом. Цей метод включає заміну оператора IN на тимчасову таблицю та використання не кластеризованого індексу. Таким чином, він прискорює процес вибірки даних, зменшуючи логічні звернення. **Висновки.** Основні цілі дослідження були визначені та успішно досягнуті: проаналізовано існуючі підходи до оптимізації SQL-запитів та основних засад роботи оптимізатора запитів як компонента СУБД; розроблено метод оптимізації SQL-запитів; оцінено ефективність запропонованого удосконаленого методу.

Ключові слова: бази даних, оптимізація SQL–запитів, індекси, оператори IN.

Аббревіатури

SQL – Structured Query Language;
БД – база даних;
СУБД – система управління базами даних.

Вступ

Обробка запитів SQL включає в себе синтаксичний аналіз, оптимізацію, генерацію коду та виконання SQL запитів. У сфері автоматичного налаштування SQL існує велика кількість інструментів, які допомагають програмістам баз даних виявляти запити, які призводять до проблем, і пропонують рішення для їх виправлення [1]. Однак налаштування SQL все ще потребує значної уваги людини, незважаючи на наявність таких інструментів.

Перелічимо типові проблеми, що впливають на продуктивність SQL-запитів:

1. Недостатня оптимізація запитів: SQL-запити повинні бути оптимізовані для швидкого виконання. Якщо запити не оптимізовані, вони можуть вимагати велику кількість обчислювальних ресурсів і часу для обробки даних. Це може бути викликано неправильними або неефективними виразами, відсутністю необхідних індексів або недостатньою кількістю фільтрів для зменшення обсягу даних, які потрібно обробити.

2. Відсутність необхідних індексів: Індекси використовуються для швидкого пошуку даних у таблицях. Відсутність необхідних індексів або неправильне використання індексів може призвести до повільних запитів. Наприклад, якщо часто виконуються запити з умовою WHERE на певному стовпці, то варто розглянути створення індексу для цього стовпця.

3. Погана структура бази даних: Якщо база даних не має оптимальної структури, це може призводити до повільних запитів. Наприклад, якщо таблиці мають занадто багато зв'язків або неправильні типи даних, то це може впливати на продуктивність запитів.

4. Великий обсяг даних: Якщо база даних містить великий обсяг даних, це може призвести до повільних запитів, оскільки системі потрібно багато часу для обробки такої кількості інформації. У таких випадках можна розглянути використання підходів, таких як розподілена обробка даних або використання партицій для розділення таблиць на менші сегменти.

5. Недостатня оптимізація сервера бази даних: Неправильна конфігурація сервера бази даних, недостатній розмір буферів, погано налаштовані параметри пам'яті або недостатня кількість ресурсів можуть впливати на продуктивність SQL-запитів. Важливо виконати належне налаштування сервера бази даних для забезпечення оптимальної продуктивності.

6. Несправні апаратні засоби: Якщо сервер бази даних або мережа мають проблеми зі швидкістю або стабільністю, це може призвести до повільних відповідей на SQL-запити. Необхідно перевірити апаратне забезпечення, забезпечити достатню пропускну здатність і знизити відсоток непередбачуваних перебоїв у роботі сервера та мережі.

7. Конфлікти паралельного виконання: При одночасному виконанні багатьох запитів можуть виникати конфлікти, які знижують продуктивність. Це можуть бути блокування, колізії або конкуренція за ресурси. Варто розглянути використання транзакцій, блокування на рівні рядків або ізоляційні рівні для уникнення таких проблем.

Найкращий спосіб підготовки набору запитів з оптимальною продуктивністю - написати запити кількома різними способами та порівняти їх планування та виконання, що називається методом проб і помилок. Для підготовки та тестування оптимізації продуктивності запитів, дослідники пропонують безліч методів та порад, які можна використовувати [1-6]. Існує багато різних способів визначити найкращі можливості написання запитів, проте два найбільш часто використовуваних методи - це аналіз кількості логічних даних, створених запитами, та перегляд графічних планів.

Об'єктом дослідження є SQL-запити до бази даних, з метою вдосконалення їх швидкості та ефективності. **Предметом** вивчення є методи вдосконалення швидкості запитів та ефективності.

Метою роботи є збільшити швидкість виконання SQL-запиту базою даних при одночасній підтримці високої продуктивності та легкості застосування рішення шляхом розробки покращеного методу оптимізації синтезу складних SQL–запитів з простих.

1 Постановка задачі

Зараз реляційні бази даних широко використовуються для зберігання та обробки даних, при цьому розмір баз може коливатися від кілобайтів до терабайтів. Збільшення розміру бази даних призводить до збільшення часу, потрібного для пошуку потрібної інформації та обробки запитів сервером. Бази даних також використовуються для публікації даних в Інтернеті, що може призводити до значного зростання кількості користувачів, що звертаються до бази даних в режимі реального часу. Збільшення продуктивності комп'ютерів, на яких працюють системи управління базами даних, не завжди може вирішити ці проблеми. Іноді зміна алгоритмів обробки SQL-запитів, наприклад, використання однорівневих індексів замість сканування таблиці, може дати значно кращий результат. У випадку використання збалансованих дерев індексів, виграш у продуктивності може бути ще більшим. Однак підвищення швидкості роботи зовнішньої пам'яті призводить до лінійного збільшення продуктивності обробки запитів.

Існує низка критеріїв, за якими можна оптимізувати виконання SQL-запитів, такі як швидкість виконання, завантаження процесора та використання пам'яті. Зазвичай, головною метою при оптимізації є мінімізація часу виконання запиту, проте звернення до диска може бути найдорожчою операцією, тому кількість звернень до диска є одним з основних критеріїв оптимізації. Інший важливий критерій - процесорний час.

Для збільшення швидкості виконання запитів можна використовувати різні методи, такі як використання індексів, хеш-функцій, матеріалізованих і секціонованих уявлень, алгоритмів виконання операцій з'єднання та обмеження, ведення статистики даних та інші. Наприклад:

1. Оптимізація запитів. Перевірити SQL-запити та спробувати їх оптимізувати. Слід використовувати ефективні вирази, уникати зайвих обчислень і функцій, які можуть затримувати виконання запитів.

2. Створення індексів. Відповідно до потреб доцільно створити індекси для стовпців, що часто використовуються в операторах WHERE або JOIN. Індекси дозволяють швидше знаходити та фільтрувати дані, покращуючи продуктивність запиту.

3. Перегляд структури бази даних. Доцільно перевірити чи правильно спроектована структура БД. Таблиці повинні мати належні первинні та зовнішні ключі, і оптимальні зв'язки між таблицями. Також слід використовувати нормалізацію, де це необхідно, щоб забезпечити ефективну організацію даних.

4. Кешування запитів. Механізми кешування дозволяють зберігати результати запитів у пам'яті для швидкого доступу. Це особливо корисно для запитів, які виконуються часто та повторюються.

5. Поділ запиту на менші частини. Якщо запит складний або має багато об'єднань, можливо є сенс розбити його на менші запити або підзапити. Це може знизити навантаження і покращити продуктивність.

6. Налаштування сервера бази даних. Перевірити налаштування сервера бази даних, такі як розмір буферів, параметри пам'яті, кількість одночасних з'єднань і т. д. Правильна конфігурація може вплинути на швидкість виконання SQL-запитів.

7. Використання попереднього завантаження даних. Якщо відомо, що певні дані будуть потрібні для виконання запитів, доцільно завантажити їх заздалегідь у пам'ять або тимчасові таблиці. Це допоможе уникнути зайвих запитів до бази даних і прискорити виконання запитів.

8. Паралельне виконання. Використовувати можливості паралельного виконання запитів, де це можливо. Розглянути розділення великих запитів на менші частини, які можуть бути виконані паралельно.

Ці методи можуть бути використані окремо або в поєднанні один з одним для досягнення кращої продуктивності SQL-запитів системи.

Оптимізатор - спеціальний компонент СУБД, що займається побудовою плану запиту, тобто його компіляцією. Оптимізатор розглядає різні критерії та наявність допоміжних структур, встановлює кілька планів виконання, оцінює вартість виконання кожного з них і вибирає найкращий. Однак, складати всі можливі плани та оцінювати їх було б занадто дорого та часово затратно, тому складається тільки підмножина можливих планів, з якої вибирається найкращий.

Зараз проводяться роботи по розробці нових структур, що дозволяють зменшити кількість звернень до диска та підвищенню ефективності оптимізатора запитів, тобто алгоритму вибору можливих планів виконання.

Незважаючи на тривалість робіт з оптимізації SQL-запитів протягом багатьох років, вони залишаються важливими й сьогодні. Справді, із зростанням обсягу інформації й навантаження на бази даних, такі завдання набули ще більшої актуальності.

2 Огляд літератури

Обробка SQL складається з кількох етапів, які включають синтаксичний розбір, оптимізацію, генерацію рядків джерела та виконання оператора SQL. Рис. 1 показує загальні етапи обробки SQL запиту. Існує широкий асортимент інструментів для автоматичного налаштування SQL, які допомагають розробникам баз даних виявляти проблеми в SQL-операторах. та пропонують їх виправлення [1]. Хоча такі інструменти можуть знизити навантаження на розробників баз даних, налаштування SQL все ж вимагає значної участі людини.

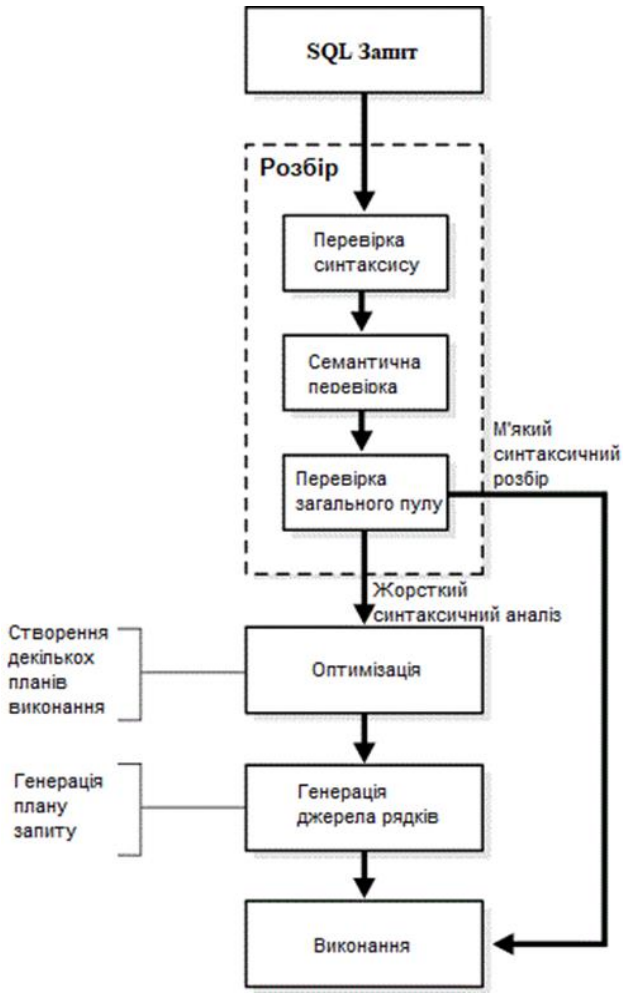

Рис. 1. Загальні етапи обробки SQL запиту

Щоб забезпечити оптимальну продуктивність, найкращий спосіб скласти набір запитів - це написати їх кількома різними способами та порівняти плани планування та виконання, використовуючи метод проб і помилок. Існує безліч методів та порад для підготовки та тестування оптимізації продуктивності запитів [1-6]. Два найбільш часто використовуваних методи - це аналіз кількості логічних даних, створених запитами, та перегляд графічних планів.

В [3] P. Karthik та співавтори висвітлюють важливість налаштування запитів SQL для досягнення високої продуктивності систем управління базами даних. Перш ніж запропонувати методи налаштування SQL-запитів, вони досліджують найкращі підходи до одночасного налаштування кількох операторів SQL. Вони пропонують широкий підхід, який дозволяє паралельно налаштовувати SQL-оператори та може значно зекономити час на налаштування на пізніших етапах. Автори не надають жодних конкретних методів налаштування SQL, але підтверджують, що використання ключового слова DISTINCT у запитах SQL не є доцільним, коли пошук здійснюється на основі атрибуту первинного ключа.

У статті [2] R. Bhajipale та інші провели докладне дослідження необхідності налаштування запитів у великих базах даних з метою підвищення продуктивності. Вони запропонували кілька технік налаштування запитів, що є частиною їх рекомендацій. Дослідники зазначили, що оптимізація запитів - це глибока тема, і їх поради є лише загальними настановами, які охоплюють найважливіші аспекти. Рекомендації, запропоновані у статті, не є вичерпним списком технік або типів складних запитів, тому ще є багато роботи для інших дослідників у цій галузі.

У статті S. S. Srinivas [7] було представлено кілька методів налаштування та оптимізації SQL-запитів. Незважаючи на те, що ці методи є важливими, список їх не є повним і не охоплює всі доступні методи. Це дає можливість дослідникам продовжувати розробку своїх пропозицій з налаштування SQL у майбутньому. У статті не було надано оцінок ефективності будь-якого з запропонованих методів, що є недоліком. Тому не можна гарантувати, що запропоновані у дослідженні методи у реальних ситуаціях будуть ефективними.

У своїй статті D. Patel та P. Patel [8] запропонували метод оптимізації запитів, що ґрунтується на перегляді базового об'єкта Schema для великих баз даних. Вони вважають, що їх схема приводить до скорочення часу виконання запиту та зменшення обсягу запиту, порівняно з іншими методами, такими як Query Graph, Tableaus або Optimization of Queries with Aggregates. Проте, у статті не було представлено жодного методу для покращення синтаксису SQL з метою підвищення продуктивності.

У статті S. Patil та інші [9], було розглянуто 9 різних методів, які можна використовувати для налаштування SQL-запитів, проте, стаття не містить конкретних пропозицій або порад щодо того, як писати ефективні оператори SQL.

Дослідження D. Colley та C. Stanier зосереджені на нових методах, які були запропоновані недавно для збільшення продуктивності баз даних [20]. Вони дослідили три ключові фактори, які впливають на продуктивність запитів до БД, включаючи вплив оператора JOIN на продуктивність, але їхній аналіз не є повним, оскільки вони не розглядають різні способи використання оператора JOIN в операторі SQL.

Натомість, стаття [6] пропонує оптимізацію запитів з використанням перетворень SQL, яка є одним з чотирьох кроків оптимізатора запитів Oracle. Перетворення SQL виконується оптимізатором бази даних, щоб замінити даний оператор SQL на інший, який може бути виконаний більш ефективно, але повертає ті ж результати. Стаття розглядає різні методи перетворення SQL, які реалізовані в Oracle, але

не надає порад щодо написання ефективних SQL-запитів для конкретної ситуації або вимоги.

У роботі [10] C. G. Corlatan та інші досліджували методи оптимізації запитів для сервера Microsoft SQL і з'ясували, що декілька факторів можуть впливати на продуктивність СУБД, такі як відсутність індексів, неточна статистика, погано написані запити, взаємоблокування, використання курсорів в операціях TransactSQL, фрагментація індексів і часта перекомпіляція запитів. Хоча дослідники запропонували кілька порад щодо оптимізації запитів SELECT, вони не були перевірені в реальних експериментах, тому їх можна розглядати лише як загальні рекомендації.

У статтях [11, 12] розглянуто використання підказок при оптимізації вкладених SQL-запитів. Оскільки користувачі не мають прямого доступу до оптимізатора запитів СУБД, підказки, вставлені у вигляді коментарів в оператор SQL, можуть дати оптимізатору вказівку виконати певні операції відповідно до потреб користувача, не вдаючись до автоматичної оптимізації. Дослідники вказують на те, що використання підказок може збільшити продуктивність запитів та що метод є простим у використанні та зрозумілим, через те що підказки пишуться простою мовою. Однак головним недоліком цього методу є те, що підказки синтаксису залежать від певної реалізації СУБД і не можуть бути типізовані, тому що підказки не є частиною SQL-стандартів.

У роботі J. Habimana [4] надав поради до написання ефективних та швидких SQL-запитів. Для дослідження ефективності методів, що пропонуються, було проведено порівняльний аналіз продуктивності налаштованих та не налаштованих запитів. Результати свідчать про те, що налаштовані запити мають деякі переваги. Однак, в роботі було виявлено недоліки, зокрема, всі поради стосуються запитів, які отримують дані з однієї таблиці та не включають в себе вкладені запити. Крім того, відсутня оцінка ефективності запропонованих порад, тому їх ефективність не може бути визначена. Автор не надав жодних атрибутів продуктивності, які можна було б використовувати для перевірки ефективності запропонованих методів іншими дослідниками.

3 Матеріали і методи

Оператор у SQL - це слово або символ, яке застосовується переважно у реченні WHERE для виконання операцій, таких як порівняння та арифметичні операції. Ці оператори допомагають встановлювати умови в запиті та поєднувати декілька умов в одному запиті. Хоча оператори є дуже корисними у SQL, деякі з них можуть вимагати значну кількість ресурсів для виконання.

Далі запропоновано покращений метод [13], що намагається уникати оператор IN в реченні WHERE.

Хід розв'язку:

Зробіть тимчасову таблицю, вставте потрібні дані та виконайте JOIN до основного запиту.

У випадку якщо складений запит містить оператор IN, його слід вилучити, а для створення тимчасової таблиці та додавання її в запит слід здійснити наступне:

1. Визначити атрибути SELECT замість SELECT *.

Якщо таблиця містить багато атрибутів та кортежів, вибірка усіх полів (за допомогою SELECT *) перевикористовує ресурси бази даних для запиту великого об'єму зайвих даних. Одночасно з цим, зазначення стовпців у операторі SELECT буде видавати тільки важливі дані.

2. Обминати SELECT DISTINCT при можливості.

SELECT DISTINCT групує всі поля в запиті, для створення чіткої відповіді. Проте щоб досягти цю ціль треба значний об'єм обчислювальної потужності.

3. Застосовувати WHERE замість HAVING для фільтрації даних.

Згідно з порядком операцій SQL, оператор HAVING виконується після оператору WHERE. Якщо ж треба відфільтрувати запит на базі певних умов, оператор WHERE є ефективнішим.

4. Застосовувати оператор LIMIT для скорочення результатів запиту.

Застосовуючи LIMIT до першого запуску запиту, слід переконатися, що будуть отримані значущі і бажані результати. При цьому якщо у СУБД оператор LIMIT відсутній, то можна використати наступний синтаксис:

SELECT TOP 3 * FROM Table;

5. Змінмти підзапит на оператор з'єднання JOIN.

Підзапити можна замінити з'єднанням, яке буде виконуватися швидше. Наприклад:

```
SELECT authors.id, (SELECT MAX(created)
FROM posts
WHERE author_id = authors.id)
AS latest_post FROM authors
```

Цей запит можливо переписати з використанням JOIN:

```
SELECT authors.id, MAX(posts.created) AS latest_post
FROM authors
INNER JOIN posts
ON (authors.id = posts.author_id)
GROUP BY authors.id
```

6. Прибрати оператор IN, створити таблицю #temp і вставити в неї потрібні дані.
7. Визначити індекс для таблиці #temp.
8. З'єднати #temp з основним відношенням.

4 Експерименти

Перед початком виконання запитів [13] було обрано малу та середню по розміру базу даних з [4], для маніпулювання БД було обрано Microsoft SQL Server Management Studio.

Схема БД представлена на рис. 2, а опис – в табл. 1.

Розмір малої бази даних складає біля 10 ГБ, а середньої 50 ГБ.

Характеристики комп'ютера, на якому проводяться експерименти опичані у табл. 2.

Таблиця 1 – **Опис баз даних, що використовуються**

Назва таблиці	Кількість рядків		Кількість зв'язків
	Маленька БД	Середня БД	
LinkTypes	2	2	1
PostLinks	161519	1421208	2
Posts	3729195	17142169	4
PostTypes	8	8	1
Votes	10143364	52928720	3
VoteTypes	15	15	1
Comments	3875183	24534730	2
Users	299398	2465713	3
Badges	1102019	8042005	1

Таблиця 2 – **Опис комп'ютера, що використовуються**

Характеристика	Значення
Процессор	Intel Pentium N3542 2.16 GHz
Опер. система	Windows 10
Пам'ять	HDD зчитування – 90 MB/s , записування – 85 MB/s
Опер. пам'ять	4 ГБ DDR3–1333 MHz

Оцінку методу було проведено шляхом створення кількох SQL-запитів з оператором IN та використання Temp Tables замість оператора IN, як запропоновано у цьому дослідженні. Процес застосування методу показано на лістінгу 1.

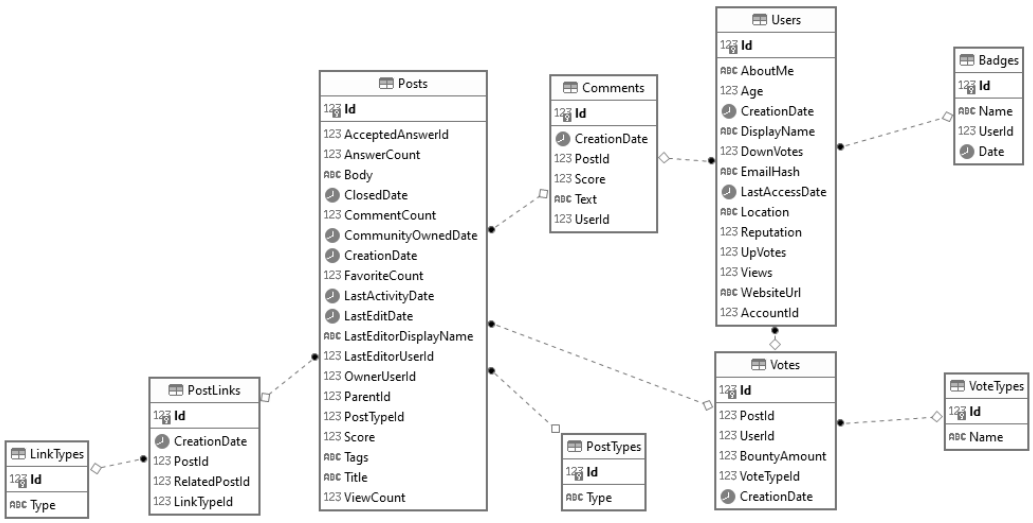

Рис. 2. Схема БД

5 Обговорення

Результати запитів 1 та 2 показані на рис. 3, результати запитів 3 та 4 – на рис. 4, результати запитів 5 та 6 – на рис. 5.

Якщо дивитись на плани виконання запитів, видно, що при використанні запропонованого удосконаленого методу кількість логічних звернень зменшилась, що в результаті дозволило скоротитя час виконання запиту.

Тому, на підставі отриманих результатів виконання, можна зробити висновок, що в малій базі даних запит виконався на 15% швидше, а в середній базі даних запит виконався на 17% швидше.

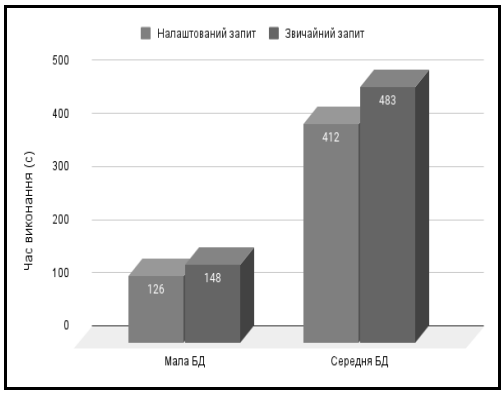

Рис. 4. Час виконання запитів 3 та 4

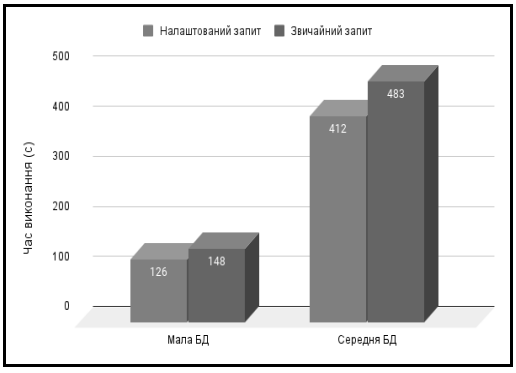

Рис. 3. Час виконання запитів 1 та 2

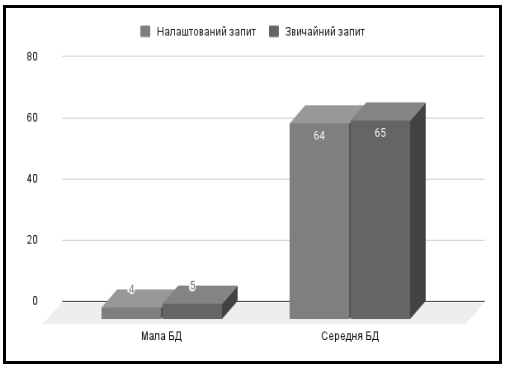

Рис. 5. Час виконання запитів 5 та 6

Запит 1 з використанням IN:

- вибираємо стовпці та обєднуємо такі таблиці як: Users, Posts, PostTypes,Comments,Votes
```
select U.DisplayName,u.AboutMe,p.Body,pt.type,c.text from Users u
join Posts p on u.Id = p.OwnerUserId
join PostTypes pt on p.PostTypeId = pt.Id
join Comments c on c.PostId = p.Id
join Votes v on u.Id = v.UserId
```
- задаємо умову використовуючи оператори between та in;
```
where u.Views between 5000 and 50000
and Location in ('El Cerrito, CA','Corvallis, OR')
and v.UserId in (select UserId from Votes where VoteTypeId in(5,8))
```
- При виконанні запиту індексів додано не було.

Запит 2 з використанням Temp Tables:

- створюємо тимчасову таблицю
```
create table #temptable (UserID int)
```
- вставляємо відповідні дані в тимчасову таблицю
```
insert into #temptable (UserID) select UserId from Votes where VoteTypeId in(5,8)
```
- створення індексу на таблицю #temptable
```
create INDEX INX_temp on #temptable (UserID)
```
- вибираємо стовпці та обєднуємо такі таблиці як: Users, Posts, PostTypes,Comments та #temptable
```
select U.DisplayName,u.AboutMe,p.Body,pt.type,c.text from Users u join Posts p on u.Id = p.OwnerUserId
join PostTypes pt on p.PostTypeId = pt.Id
join Comments c on c.PostId = p.Id
join #temptable t on t.UserID = u.Id
```
- задаємо умову використовуючи оператори between та in;
```
where u.Views between 5000 and 50000
and Location in ('El Cerrito, CA','Corvallis, OR')
```
- видаляємо тимчасову таблицю
```
drop table #temptable
```

Запит 3 з використанням IN:

- -- вибираємо стовпці та об’єднуємо таблиці Users та Badges
```
select DisplayName,Location,B.name from Users U
join Badges B on U.Id = B.UserId
```
- -- задаємо умову використовуючи оператори IN та between
```
where Location in ('Israel','United States','Germany')
and Reputation between 50 and 200
```

Запит 4 з використанням Temp Tables:

- створюємо тимчасову таблицю
```
create table #temptable (UserID int,DisplayName varchar(255),Location varchar(255))
```
- вставляємо відповідні дані в тимчасову таблицю
```
insert into #temptable (UserID,DisplayName,Location) select Id,DisplayName,Location from Users where Location in
('Israel','United States','Germany') and Reputation between 50 and 200
```
- створення індексу на таблицю #temptable
```
create index INDx on #temptable (Location,UserID)
```
- вибираємо стовпці та обєднуємо такі таблиці як:#temptable та Badges
```
select t.DisplayName,t.Location,B.name from #temptable t join Badges B on t.UserID = B.UserId
```
- видаляємо тимчасову таблицю
```
drop table #temptable
```

Запит 5 з використанням IN:

- В наступному запиті було вирішено дослідити метод при використанні під запиту
- вибиранємо стовпці та задаємо умову за домогою IN та WHERE
```
select Votes.PostId,Votes.UserId from Votes
where UserId in (
select Users.Id from Users
where Users.DisplayName in ('Jeff Atwood','Nick','Gaius'))
```

Запит 6 з використанням Temp Tables:

- створюємо тимчасову таблицю вставляємо дані та створюємо индекс
```
create table #temptable (UserID int)
insert into #temptable (UserID) select Users.Id from Users where Users.DisplayName in ('Jeff Atwood','Nick','Gaius')
create index INdx on #temptable(UserID)
```
- вибираємо стовпці та використовуємо temptable в підзапиті
```
select Votes.PostId,Votes.UserId from Votes
where UserId in (select UserId from #temptable)
drop table #temptable
```

Лістінг 1. Процес застосування запропонованого удосконаленого методу

Висновки

Було проведено огляд та аналіз існуючих методів оптимізації SQL-запитів та основних принципів роботи оптимізатора запитів як компонента СУБД. Було обрано напрямок зменшення логічних операцій та вирішено проблему оптимізації SQL-запитів до реляційної бази даних.

Розроблено метод оптимізації SQL запитів у випадках, коли швидкість вибірки даних починала погіршуватись, який полягає у заміні оператора IN на тимчасову таблицю та використанні некластеризованого індексу, що дозволяє швидше вибирати дані за рахунок зменшення логічних операцій порівняно з існуючими методами.

Були виконані запити, що використовували запропонований метод на основі тимчасової таблиці та некластеризованого індексу, що підтверджує життєздатність запропонованого рішення. Була проведена експериментальна оцінка, яка показала, що виведення даних було на 15% швидше при використанні малої бази даних загальним обсягом 10 ГБ і на 17% швидше при використанні середньої бази даних обсягом 50 ГБ.

СПИСОК ЛІТЕРАТУРИ

1. Rupley, M. L. (2008). Introduction to query processing and optimization. Indiana University, South Bend, South Bend, IN, USA, TechReport TR–20080105–1.
2. Bhajipale, R., Bisen, P., Meshram, A., & Thakur, S. S. (2016). SQL tuner. International Journal of Computer Trends and Technology, 33(1), 29–32.
3. Karthik, P., Reddy, G. T., & Vanan, E. K. (2012). Tuning the SQL query in order to reduce time consumption. International Journal of Computer Science Issues, 9(4/3), 418–423.
4. Habimana, J. (2015). Query optimization techniques – tips for writing efficient and faster SQL queries. International Journal of Scientific & Technology Research, 4(10), 22–26.
5. Sahal, R., Nihad, M., Khafagy, M. H., & Omara, F. A. (2018). iHOME: Index based JOIN query optimization for limited big data storage. Journal of Grid Computing, 16(2), 345–380.
6. Sharma, M. (2012). Query optimization using SQL transformations. International Journal of IT, Engineering and Applied Sciences Research, 1(1), 100–104.
7. Srinivas, S. S., Naik, B. V., & Kumar, J. S. A. (2017). Query minimization methods. International Journal of Scientific & Engineering Research, 8(5), 30–33.
8. Patel, D., & Patel, P. (2015). An approach for query optimization by using schema object base view. International Journal of Computer Applications, 119(16), 21–24.
9. Patil, S., Damare, P., Sonawane, J., & Maitre, N. (2015). Study of performance tuning techniques. Journal of Emerging Technologies and Innovative Research (JETIR), 2(3), 499–502.
10. Corlatan, C. G., Lazar, M. M., Luca, V., & Petricica, O. T. (2014). Query optimization techniques in Microsoft SQL server. Database Systems Journal, 5(2), 33–48.
11. Lokhande, A. D., & Shete, R. M. (2012). The use of hints in SQL–Nested query optimization. Journal of Data Mining and Knowledge Discovery, 3(1), 54–57.
12. Brent Ozar Unlimited. (2023). How to Download the Stack Overflow Database via BitTorrent. Retrieved April 3, 2023, from https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent/.
13. Єрмолаєв, О. Д. Метод оптимізації SQL запитів : дипломна робота … бакалавра : 172 Телекомунікації та радіотехніка / Єрмолаєв Олександр Дмитрович. – Київ, 2021. – 56 с.

Received (Надійшла) 05.04.2023
Accepted for publication (Прийнята до друку) 17.05.2023

The method of optimizing sql queries of the database management system

S. Sulima, O. Iermolaiev

Abstract. Context. The size of a database, which refers to the volume of stored data, can vary significantly. It is clear that the larger the size of the database, the more time is required to search for the necessary information, resulting in increased query processing time for the server. On one hand, this problem can be addressed by improving the performance of the computers hosting the database management systems (DBMS). However, simply enhancing computer performance is insufficient; often, much better results can be achieved by changing the algorithms for processing SQL queries. Therefore, despite the fact that optimization of SQL queries has been conducted for decades, due to the rapid accumulation of information and the workload on database servers, such work has become even more relevant. **Objective**. The main objective of this work is to improve the speed of executing incoming queries in a relational database, ensuring high performance and user convenience. This will be accomplished by developing an enhanced optimization method that synthesizes complex SQL queries from a large number of simpler queries, thereby increasing overall efficiency and usability. **Method**. This article presents a method for optimizing the synthesis of complex SQL queries from numerous simple queries, which enhances the speed of executing the incoming query by a relational database while simultaneously ensuring high performance and ease of use. **Results**. A method for optimizing SQL queries specifically for situations where data retrieval speed deteriorates over time has been developed. This method involves replacing the IN operator with a temporary table and utilizing a non-clustered index. Consequently, it accelerates the data retrieval process by reducing logical references. **Conclusions**. The main research objectives were identified and successfully achieved: the analysis of existing approaches to SQL query optimization and the fundamental principles of query optimizers as components of DBMS were conducted, a method for optimizing SQL queries was developed, and the effectiveness of the proposed enhanced method was evaluated.

Keywords: databases, SQL query optimization, indexes, IN operators.