УДК 004.657

Кучерявий І.В.
Веренько А.І.
Романюк О.В.

ОПТИМІЗАЦІЯ ЗАПИТІВ ДО БАЗ ДАНИХ

Вінницький національний технічний університет

Анотація

Проаналізовано вимоги до баз даних для оптимізації запитів . Проведено аналіз основних видів запитів. Визначено методи для оптимізації запитів баз даних.

Ключові слова: Запит, БД, SQL, XML, оптимізація запитів, критичні запити, СУБД.

Abstract

Database requirements for query optimization are analyzed. The main types of queries are analyzed. Methods for optimizing database queries are defined.

Keywords: Query, DB, SQL, XML, Query Optimization, Critical Queries, DBMS.

Вступ

Сьогодні найпопулярнішими системами керування базами даних (СКБД) є реляційні [1]. Завдяки нормалізації реляційних відношень можна уникнути надлишковості даних та легко підтримувати їх цілісність, однак наявність великої кількості зв'язків між відношеннями призводить до уповільнення обробки складних запитів до бази даних. В таких випадках необхідно проводити оптимізацію запитів.

Головним завданням оптимізації запитів є системні скорочення ресурсів, необхідних для виконання запиту, що в кінцевому підсумку надає користувачеві результати за менший проміжок часу, що робить додаток комфортнішим для користувача; надає можливість програмному додатку обслуговувати більше запитів за одиницю часу, оскільки кожен запит займає менше часу, ніж неоптимізовані запити; зменшує навантаження на обладнання і дозволяє серверу працювати більш ефективно.

Мета дослідження – аналіз основних підходів до оптимізації запитів до баз даних.

Об'єкт досліджень – процес оптимізації запитів до бази даних.

Предмет дослідження – підходи та засоби оптимізації SQL-запитів до бази даних.

Результати дослідження

SQL – декларативна мова, яка використовується для формування запитів до реляційної бази даних, а також визначення та маніпулювання даними [2]. При розробці БД важливо звертати увагу на оптимізацію коду SQL, оскільки від цього на пряму залежить швидкість і коректність обробки запитів. Для аналізу поточної ситуації проекту спочатку аналізують план виконання запиту. Такий план – послідовність операторів, використання яких забезпечує мінімальні витрати обчислюваних ресурсів. Він формується на принципах реляційної алгебри. Існує багато офіційних, як платних так і безкоштовних, аналізаторів оптимальності коду SQL [3]. Вони мають можливість відображати, яким чином запит звертається до таблиці і як при цьому використовуються ключі доступу до даних. Подібний аналіз наочно демонструє всі присутні недоліки та допомагає визначити їх причину й спосіб оптимізації.

Розглянемо роботу з системою SQL Server Management Studio. Однією з великої кількості її функцій є можливість відображати та керувати планом виконання, який відображає інформацію про перебіг компіляції процедур і про їх виклики з довільної глибини вкладення. Виконуючи інструкцію SELECT можна стежити за тим, що Database Engine звертається не тільки до індексів, а й виконує аналіз всієї таблиці. Перегляд виключно по індексу буде використовуватися, якщо це є доцільним методом отримання даних з таблиці. Зазвичай використовують два основних способи оптимізації запиту.

Перший спосіб полягає в аналізі та перетворенні еквівалентних виразів, для чого необхідно мінімізувати кількість стовпців проміжного і кінцевого процесу запиту.

Другий спосіб передбачає використання різних алгоритмів для кожної операції: в основному ці алгоритми визначають, як буде виконуватись доступ до кортежів з структур даних, в яких вони зберігаються, індексація, хешування, вилучення даних, а тому впливають на кількість звернень до диска.

Таким чином, саме від коректності індексів залежить вибір оптимального плану.

При частому видаленні записів з таблиці виникають «відсутні» індекси. Це сильно впливає на оптимізацію. Їх можна знайти за допомогою такого запиту:

```
SET STATISTICS XML ON
GO
select * from [Library]
where status < GETDATE()
GO
```

Результатом буде виведення статистики в форматі XML, яка міститиме інформацію про відсутні індекси в таблиці «Library».

Для контролю стану можна використати такі функції:

- **SET STATISTICS TIME ON GO** – дозволяє визначити час виконання запиту;
- **SET STATISTICS IO ON GO** – статистика вводу/виводу;
- **SET STATISTICS XML ON GO** – виводить план виконання запиту.

Розглянемо загальноприйняті підходи до оптимізації запитів.

Критичні запити – такі, які через те, що виконуються дуже часто, займають багато загального часу роботи алгоритму. Рішенням є їх групування та хешування. Кеш плану виконання зберігає статистику, звернувшись до якої можна аналізувати критичні запити, використовуючи динамічні адміністративні представлення.

Статистика індексів – аналіз по індексам корисний тільки в тому випадку, якщо їх щільність розподілення не нижча за 90%. Це можна обрахувати за формулою «1/кількість записів». В тому випадку, коли щільність низька, варто використовувати сканування по самій таблиці.

Фільтрація по ключових полях передбачає використання предикату фільтра для індексації частини записів таблиці. Наприклад

```
select * from Product
where Quantity > 9900;
```

Некластеризований індекс – це структура даних, що використовується для підвищення швидкості отримання даних з таблиць. На відміну від кластеризованого індексу, некластеризованний сортує і зберігає дані окремо від рядків даних в таблиці. Це копія обраних стовпців даних з таблиці з посиланнями на пов'язану таблицю. Подібно кластерному індексу, некластерізованний використовує структуру дерева для організації своїх даних. Індекси використовуються для отримання даних з бази даних швидше. Його користувачі не можуть бачити індекси, вони просто використовуються для прискорення пошуку та запитів. Для покращення продуктивності використовується конкретизація.

Конкретизація – до команди CREATE INDEX можна додати додаткові параметри ASC|DESC – побудова індексу по зростанню\спаданню ключів. SORT_IN_TEMPDB – створює індекси в відсортованому форматі, а потім розподіляє їх по таблиці.

У запитах, що мають повністю нормалізовану базу нерідко доводиться з'єднувати таблиці. А кожне з'єднання – операція досить трудомістка. Як наслідок, такі запити займають ресурси сервера і виконуються повільно.

Розрахункові значення в запиті часто повільно виконуються і споживають багато ресурсів запити, особливо при використанні групування і агрегатних функцій. Зазвичай є сенс додати в таблицю 1-2 додаткових стовпчика, що містять найчастіше використовувані розрахункові дані. Наприклад, до таблиці Order можна додати поле «Загальна вартість замовлення» і зберігати його постійно, замість того, щоб кожного разу розраховувати за допомогою запиту.

Бекап – створення окремих таблиць для збереження проміжних результатів [4].

Також досягти підвищення швидкості виконання запиту можливо, замінивши функції при створенні умов конструкцією LIKE (рис.1). Тоді в плані Index Scan зміниться на Index Seek, що є швидшою операцією.

```
select
 OrderNumber
from
  [Order]
where
  LEFT(OrderNumber, 1) = '7';
```

а)

```
select
 OrderNumber
from
  [Order]
where
  OrderNumber Like'7%';
```

б)

Рисунок 1 – Використання функцій при створенні умови а) та конструкції LIKE б)

Вагомим фактором оптимізації запитів є підтримання статистики індексів в актуальному стані. В SQL Server за автоматичне оновлення індексів відповідає спеціальна функція, яку можна увімкнути оператором

ALTER DATABASE dbname SET AUTO_UPDATE_STATISTICS ON.

Перевірити, чи є увімкненою дана функція, можна за допомогою оператора

SELECT DATABASEPROPERTYEX('<dbname>','IsAutoUpdateStatistics')

Усі вище наведені способи досить прості в реалізації. При розробці технічного завдання слід звернути увагу на інтегрування їх до коду проекту.

Висновок

Таким чином, виконання оптимізації запитів дозволяє ефективніше використовувати наявні ресурси, та збільшити швидкість роботи додатку. Серед усіх способів оптимізації запитів найкращі результати надає стратегія, що передбачає коректну роботу з індексами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. DB-Engines Ranking [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://db-engines.com/en/ranking.
2. Beaulieu A. Learning SQL / Alan Beaulieu. – Москва, 2007. – 309 с.
3. Шварц Б. MySQL. Оптимизация производительности / Б. Шварц, П. Зайцев, В. Ткаченко., 2010
4. Оптимизация SQL-запросов [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: http://ts-soft.ru/blog/sql-optimization-2.

Кучерявий Ігор Володимирович студент групи 3ПІ-18б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: kucherjavyj228@gmail.com

Веренько Артем Ігорович студент групи 3ПІ-18б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: artem.verenco@gmail.com

Романюк Оксана Володимирівна, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м.Вінниця, e-mail: romaniukoksanav@gmail.com

Kucheriavyi Ihor, student of group 3PI-18b, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: kucherjavyj228@gmail.com

Verenco Artem , student of group 3PI-18b, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: artem.verenco@gmail.com

Oksana Romaniuk, Associate Professor of the Software Chair, Vinnytsia National Technical University, Vinnytsia, e-mail: romaniukoksanav@gmail.com