МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ОПТИМІЗАЦІЯ ЗАПИТІВ ДО БАЗИ ДАНИХ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторної роботи
з дисципліни «База даних»

для студентів-бакалаврів галузі знань 12 Інформаційні технології

Дніпро

2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ ЕЛЕКТРОЕНЕРГЕТИКИ
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

ОПТИМІЗАЦІЯ ЗАПИТІВ ДО БАЗИ ДАНИХ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторної роботи
з дисципліни «База даних»

для студентів-бакалаврів галузі знань 12 Інформаційні технології

Дніпро
НТУ «ДП»
2019

Жукова О.А.

Оптимізація запитів до бази даних. Методичні рекомендації до виконання лабораторної роботи для студентів-бакалаврів галузі знань 12 Інформаційні технології / Упоряд.: О.А. Жукова, Ю.А. Мілінчук ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро: НТУ «ДП», 2019. – 13 с.

Упорядники:
О.А. Жукова, доцент;
Ю.А. Мілінчук, асистент.

Затверджено методичною комісією зі спеціальності 125 Кібербезпека (протокол № 7 від 07.03.2019) за поданням кафедри безпеки інформації та телекомунікацій (протокол № 7 від 07.03.2019).

Подано методичні рекомендації до виконання лабораторної роботи «Оптимізація запитів до бази даних» для студентів-бакалаврів галузі знань 12 Інформаційні технології.

Відповідальний за випуск зав. кафедри БІТ В.І. Корнієнко, д-р техн. наук, проф.

Мета роботи

Отримати навички проведення оптимізації структури бази даних та часто використовуваних SQL-запросів до бази даних.

Хід роботи

1. Перед початком роботи ознайомтеся з методичними вказівками.

2. З операторів *SELECT* найчастіше використовуватимуться ті оператори, які лежать в основі представлень даних. Виберете 10 *SQL*-запитань, вважаючи, що користувачі їх застосовуватимуть найчастіше. Помістите відомості про найчастіше використовувані *SQL*-запитання в таблицю з трьох колонок: номер по порядку, формулювання запиту, оператор *SELECT*.

3. Виберете ще 10 *SQL*-запитань, вважаючи, що користувачі їх застосовуватимуть найчастіше. Помістите відомості про найчастіше використовувані *SQL*-запитання в таблицю з трьох колонок: номер по порядку, формулювання запиту, оператор *SELECT*.

4. Для всіх часто використовуваних *SQL*-запитань, використовуючи методичні вказівки, встановите порядок проходження умов в пропозиції *WHERE* оператора *SELECT*, так, щоб мінімізувати кількість використовуваних індексів для прискорення доступу до даних. Якщо в часто використовуваних *SQL*-запитаннях присутні пропозиції *ORDER BY*, то необхідно проглянути порядок проходження стовпців в цих фразах на предмет мінімізації кількості індексів для сортування даних.

5. Проаналізуйте, які індекси, що забезпечують посилальну цілісність, можна використовувати для прискорення доступу до даним. Визначите, які індекси необхідно додати в базу даних, щоби забезпечити прискорення доступу до даних. Використовуючи оператор *CREATE INDEX*, додайте ці індекси в *SCRIPT*-файл.

6. Результати оптимізації запитів до бази даних розмістіть в таблиці, в якій будуть наступні колонки: номер по порядку, порядок проходження умов в пропозиції *WHERE* і (або) *ORDER BY* до оптимізації, те ж, що і в попередній колонці після оптимізації, ім'я індексу, прискорюючого доступ до даних для

поточного оператора *SELECT*. Номери по порядку для операторів *SELECT* повинні співпадати з тими, які вказані в таблиці, побудованій в пункті 4.

7. Вкажіть список індексів, які підтримують посилальну цілісність у Вашій базі даних і не беруть участь в прискоренні доступу до даним.

МЕТОДИЧНІ ВКАЗІВКИ

Індексування таблиць бази даних.

Індекс *– це структура даних, що дозволяє скоротити час доступу до окремих рядків таблиць*, незалежно від їхнього фізичного розміщення. Він аналогічний предметному покажчику, що наведений наприкінці книги. Це структура, яка пов'язана з файлом і призначена для пошуку інформації за тим принципом, що й предметний покажчик у книзі. Індекс дозволяє уникнути проведення послідовного або покрокового сканування файлу у пошуках потрібних даних. При його використанні об'єктом пошуку може бути один або кілька записів файлу. Як і предметний покажчик книги, індекс упорядкований і кожний його елемент містить назву об'єкту пошуку, а також один або кілька покажчиків (ідентифікаторів записів), що мають посилання на місце його розташування.

В InterBase розділені поняття ключів та індексів на логічному рівні [5, 6]. Первинний (*PRIMARY KEY*) і зовнішній (*FOREIGN KEY*) ключі будуються для забезпечення посилальної цілісності реляційно-пов'язаних таблиць. Первинний ключ, крім цього, виконує функції підтримки унікальності своїх значень, що обумовлене його основним призначенням – однозначно характеризувати запис у таблиці БД. Для таких же цілей може використовуватися й просто унікальний ключ (*UNIQUE*). Однак вони використовуються операторами *SELECT* для прискорення доступу до даних. «Звичайні» індекси на відміну від ключів, служать лише для оптимізації доступу до даних. З фізичної точки зору, те, що логічно при створенні таблиць поділялося на ключі та індекси, на фізичному рівні перетвориться на індекси. Однак, якщо «звичайним» індексам можна привласнити ім'я, то фізичні індекси, реалізовані на основі визначень ключів, будуються й іменуються системою автоматично.

Більшість діалектів, у тому числі InterBase, підтримують наступний оператор, що дозволяє створювати індекси для таблиць БД:

CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]

***INDEX** Ім'я **ON** Ім'я таблиці (атрибут [, атрибут ...]).*

Тут обов'язково необхідно вказати Ім'я індексу, Ім'я таблиці й перерахувати імена атрибутів, по яких необхідно побудувати індекс. Нагадаємо, що формат оператора *CREATE INDEX* дозволяє виключити дублювання значень у ключових атрибутах, вказавши пропозицію *UNIQUE*. У цьому випадку унікальність значень потенційного ключа буде автоматично підтримуватися системою. За замовчуванням сортування значень індексних полів ведеться по зростанню. Це можна вказати явно, використовуючи пропозицію *ASC[ENDING]*. Якщо необхідно забезпечити зворотний порядок сортування – по зменшенню, тоді використовують пропозицію *DESC[ENDING]*.

Створення індексів можливо в будь-який момент, навіть для вже заповненої даними таблиці. Однак можуть виникнути проблеми, що пов'язані з дублюванням значень у різних рядках, якщо в операторі використовується пропозиція *UNIQUE*. Тому має сенс створювати унікальні індекси безпосередньо при створенні таблиці, перед тим як у неї почали заносити інформацію. У результаті система відразу ж візьме на себе контроль над унікальністю значень у відповідних атрибутах.

Для вилучення індексу використовується оператор

DROP INDEX Ім'я;

Їм не можна вилучити індекс, створений в операторові *CREATE TABLE* для визначення первинного й зовнішнього ключів, а також унікальності значень атрибута або групи атрибутів, що утворюють потенційний ключ (*PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE*). Для цієї мети слід застосовувати оператор *ALTER TABLE*. Не можна також вилучити індекс, що використовується та може мати місце при виконанні іншими користувачами запитів до БД. Крім цього, для вилучення індексу потрібно мати відповідні привілеї доступу до БД.

Оптимізація структури індексів. Загальні рекомендації. Від структури індексів у величезному ступені залежить ефективність виконання запитів [2, 3]. При виконанні запитів СУБД спочатку переглядає список індексів, які

визначені для таблиць, що беруть участь у запиті. Потім вибирається одна із двох схем виконання запиту – використовувати наявні індекси або послідовно переглянути таблиці. Оптимізатор СУБД прагне виконати запит з максимальною швидкодією й з мінімальними накладними витратами. Перед першим викликом СУБД завжди оптимізує виконання запиту, ґрунтуючись на поточному стані БД. Повторно запити, у яких змінюються тільки значення параметрів, не оптимізуються. Відбувається лише попереднє зв'язування формальних та фактичних параметрів, після чого запит виконується.

Повністю передбачити за якою схемою оптимізатор СУБД буде виконувати запит неможливо, тому що стан БД може змінюватися. Але існують загальні положення, які слід враховувати при проектуванні запитів і структури індексів. Індекси необхідно створювати у випадку, коли по стовпцю або групі стовпців часто:

1) проводиться пошук у БД: атрибут або група атрибутів часто перелічуються в пропозиції *WHERE* оператора *SELECT*;

2) будуються об'єднання таблиць;

3) проводиться сортування результатів у відношеннях, що вертаються в якості, запитів до БД: атрибут або група атрибутів часто використовуються в пропозиції *ORDER BY* оператора *SELECT*.

Не рекомендується будувати індекси по стовпцях або групах стовпців, які:

1) рідко використовуються для пошуку, об'єднання й сортування результатів запитів;

2) часто міняють значення, що приводить до необхідності частого оновлення індексів, що здатне суттєво сповільнити швидкість роботи із БД;

3) містять невелике число варіантів значення.

У випадку, коли при виконанні запитів використовується сортування по тих самих стовпцях, необхідно пам'ятати наступне: створення єдиного індексу здатне прискорити виконання запитів. Якщо в умові пошуку або в пропозиції *ORDER BY* використовуються не всі стовпці із цього індексу, то для оптимізації запиту застосовують тільки безперервну послідовність стовпців.

Часткове використання складеного індексу дозволяє суттєво скоротити витрати ресурсів СУБД [1, 2, 4]. Для цього складений індекс слід будувати так, щоб стовпці, по яких найбільш часто ведеться пошук, перебували на початку списку. Тоді для пошуку може бути використана частина індексу. Наприклад, нехай часто виконуються запити

```
SELECT   *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB;
SELECT *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB AND C = : ParamC;
SELECT *
        FROM SOMETABLE
        WHERE A = : ParamA AND B = :ParamB AND
                C = :ParamC AND D = :ParamD.
```

Тоді, побудувавши складений індекс по стовпцях *A, B, C, D*, ми можемо затверджувати, що даний індекс буде використаний при оптимізації всіх трьох запитів. У першому випадку буде використана підмножина індексу, тобто значення *A, B*; у другому - значення *A, B, C*, у третьому - *A, B, C, D* (тобто всі значення індексу).

Порядок проходження умов по стовпцях у пропозиції *WHERE* оператора *SELECT* не важливий, якщо умови об'єднані за допомогою *AND*. Наприклад, для виконання наступних запитів може використовуватися індекс, що *використовувався в попередньому випадку:*

```
SELECT   *
        FROM SOMETABLE
        WHERE A = 100 AND B = 200;
SELECT *
        FROM SOMETABLE
        WHERE B = 200 AND A = 100.
```

Для пропозиції *ORDER BY* порядок перерахування стовпців у складеному індексі має значення. Наприклад, для оптимізації виконання наступних запитів необхідно використовувати інший індекс:

***SELECT** **

***FROM** SOMETABLE*

***ORDER BY** A, B, C;*

***SELECT** **

***FROM** SOMETABLE*

***ORDER BY** A, C, B.*

Слід пам'ятати, що в пропозиції *ORDER BY* можна використовувати тільки безперервну послідовність значень. Наприклад, попередній індекс не *може використовуватися для виконання запитів.*

***SELECT** **

***FROM** SOMETABLE*

***ORDER BY** A, C;*

***SELECT** **

***FROM** SOMETABLE*

***ORDER BY** B, D.*

Але він буде корисним для запиту виду:

*SELECT *FROM SOMETABLE ORDER BY A, B.*

Багатопоточність пошуку по OR та IN. При частому використанні в умовах пошуку пропозиції *WHERE* декількох стовпців, які пов'язані між собою операцією «або» (*OR*) замість індексу по стовпцях *A, B, C* створюють кілька індексів, що побудовані по кожному стовпцю окремо, оскільки інакше буде здійснений послідовний перегляд усієї таблиці. Це пов'язане з тим, що індексно-послідовний доступ для індексу по атрибутах *A, B, C* може бути здійснений тільки для стовпця *A*; значення стовпців *B* и *C* у цьому випадку розкидані по індексу. Важливо пам'ятати, що при використанні оператора *OR* кожна частина умов пошуку спричиняє окреме сканування таблиць, що беруть участь у запиті. Так, наприклад, при виконанні оператора *SELECT*

***SELECT** **

***FROM** SOMETABLE*

***WHERE** A = 100 **OR** B = 200 **OR** C = 300*

буде здійснено три окремі сканування таблиці для пошуку значень, що задовольняють умовам *A = 100; B = 200; C = 300*.

Окремий потік пошуку породжує й кожний елемент у списку *IN*. Наприклад, *WHERE A IN (100, 200, 300)* інтерпретується як *WHERE A = 100 OR A = 200 OR A = 300*. Однак при вказівці діапазону *BETWEEN WHERE A BETWEEN 100 AND 300* цього не відбувається. Тому там, де можливо, слід заміняти *IN* на *BETWEEN*.

Зменшення загальної кількості індексів здійснюється за рахунок раціонального використання складених індексів, а також забезпечення посилальної цілісності за допомогою тригерів. При великій кількості індексів знижується швидкість додавання, зміни й вилучення записів у таблицях БД. Нам відомо, що в БД визначається два види індексів: одні для прискорення доступу до даних, інші – для забезпечення посилальної цілісності. Якщо останні не беруть участь в оптимізації доступу до даних, тоді їх доцільно вилучати, а посилальну цілісність забезпечувати за допомогою тригерів.

Поліпшення продуктивності роботи індексу

Розбалансування індексу приводить до того, що його глибина (depth) зростає понад критичне значення (2). **Глибина індексу** – параметр, що показує максимальне число операцій, необхідних для знаходження шуканого значення в таблиці БД з використанням даного індексу. Індекси таблиці можуть бути розбалансовані після багаторазового внесення змін. У випадку розбалансування цінність індексу при виконанні запитів знижується через збільшення часу, затрачуваного на вибірку даних**.** Тому час від часу необхідно робити одну з перерахованих нижче дій:

1) виконувати перебудову індексу;

2) перерахувати показник «вибираємості» індексу;

3) знищити індекс і заново його створити.

Перебудова індексу полягає в перестворенні й збалансуванні індексу, що наступає після деактивізації індексу оператором

ALTER INDEX Ім'я INACTIVATE

і наступної його активізації оператором

ALTER INDEX Ім'я ACTIVATE.

Деактивізація індексу корисна також у тому випадку, коли має місце вставка в таблицю БД великої кількості записів. У звичайному режимі додавання записів при активному індексі зміни вносяться в індекс у міру додавання записів у таблицю, що може його розбалансувати. Слід пам'ятати, що:

1) не можна перебудувати індекс, використовуваний у цей момент іншими користувачами запитів до БД;

2) не можна перебудувати індекс, створений у результаті визначення первинного й зовнішнього ключів, а також унікальності значень стовпця або групи стовпців (*PRIMARY KEY*, *FOREIGN KEY*, *UNIQUE*). Для цієї мети слід застосовувати оператор *ALTER TABLE*;

3) для виконання оператора *ALTER INDEX* потрібно мати відповідні привілеї доступу до БД.

Показник «корисності» індексу заснований на відомостях про число повторюваних рядків індексу. Він використовується СУБД при доступі до таблиці для вироблення оптимального плану виконання запиту. Ефективність використання індексу при пошуку інформації в таблиці БД сильно залежить від того, чи побудований індекс за унікальними значенням і, якщо ні, наскільки відрізняються дані, за якими він побудований.

Показник корисності індексу розраховується при його створенні як число значень, що різняться, індексних полів усередині індексу, яке віднесене до середньої кількості записів. Після внесення змін у таблицю змінюється ступінь відмінності значень стовпців, по яких побудований індекс. Тому розрахований показник корисності може не відображати реального стану індексу й значення показника

рекомендується примусово перераховувати: час від часу – при внесенні невеликих змін і завжди – при внесенні істотних змін. Перерахування реалізується оператором

SET STATISTICS INDEX *Ім'я.*

Для участі у виконанні запиту вибираються індекси з максимальним показником корисності. Такі індекси забезпечують більш швидкий пошук. Максимальний показник корисності мають унікальні індекси.

Середня кількість записів - показник, який розраховується щораз при оптимізації запиту як кількість сторінок БД, зайнятих цієї таблицею, поділене на максимальне число записів на сторінці. Зменшення числа сторінок, зайнятих БД, і знищення на них «дірок» ведуть до зменшення показника середнього числа записів і, як наслідок, до підвищення показника корисності індексів. Це ще один аргумент на користь періодичної оптимізації БД шляхом створення резервної копії та її оновлення.

Перегляд і складання плану виконання запитів

Утиліті *WISQL* можна встановити режим показу плану виконання запиту (вибравши елемент меню *Session* | *Basic Settings* і відзначивши режим *Display Query Plan*). Тоді при виконанні запитів буде виводитися разом з результатом план їх виконання. Під планом виконання запиту розуміється перелік індексів, які використовуються СУБД для вибірки даних з таблиць.

Для примусового виконання запиту по тому або іншому плану, необхідно в операторі *SELECT* указати наступну пропозицію:

PLAN *<план виконання запиту>, де <план виконання запиту> =*

[JOIN | [SORT] MERGE] *({елемент плану | план виконання запиту}*

[, {елемент плану | план виконання запиту} ...]), де

<елемент плану> = {таблиця | псевдонім}

NATURAL | INDEX *(<індекс> [, <індекс> ...])* ***| ORDER*** *<індекс>.*

Синтаксис пропозиції *<план виконання запиту>* дозволяє з'єднати між собою кілька таблиць (пропозиція *JOIN*). У випадку відсутності індексів, по яких записи таблиці можуть бути впорядковані, застосовують примусове сортування (пропозиція [*SORT*] *MERGE*). У пропозиції *<елемент плану>* –

вказують таблицю, у якій проводиться пошук даних. Якщо таблиця кілька раз бере участь у запиті, для скорочення тексту плану корисно застосовувати її псевдонім. Пропозиція *NATURAL*, вказує, що для пошуку записів застосовується послідовний доступ. Це єдиний спосіб пошуку в тому випадку, якщо немає відповідних індексів. Можна вказати один або кілька індексів, які повинні використовуватися для пошуку записів, що задовольняють умові запиту (пропозиція *INDEX*). А пропозиція *ORDER* дозволяє визначити індекс, що необхідний для сортування таблиці. Приклади використання пропозиції *PLAN* рекомендується подивитися у відповідній документації до СУБД.

Звіт повинен містити:

1. Номер, найменування і мета лабораторної роботи.
2. Таблицю з відомостями про найбільш використовувані SQL-запитання.
3. Таблицю з результатами оптимізації запитів згідно пункту 7 розділу «Хід роботи».
4. Список індексів, використовуваних для підтримки посилальної цілісності, які не беруть участь в прискоренні доступу до даним.
5. Оператори *CREATE INDEX*, які створюють індекси, що беруть участь в оптимізації доступу до даним.
6. *SCRIPT*-файл.

Перелік літератури

1. Карпова Т.С. Базы данных: модели, разработка, реализация [Текст] / Т.С. Карпова - СПб: Питер, 2002. - 303с.

2. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. [Текст]: учебник/ Т. Коннолли , К. Бегг ;-2-е изд., испр. и доп. - М.: Изд. дом "Вильямс": 2001. - 1111с.

Дейт К. Дж. Введение в системы баз данных [Текст] / 6-е издание: Пер. с англ. - К.; М.; СПб.: Издательский дом "Вильямс", 1999 – 848 с.

3. Мещеряков Е.В. Публикация баз данных в Интернете. [Текст]: Учебное пособие / Е.В. Мещеряков, А.Д. Хоменко - М.: БХВ-Петербург, 2001. - VIII, 552 с.

Упорядники:

Жукова Олена Андріївна

Мілінчук Юлія Анатоліївна

ОПТИМІЗАЦІЯ ЗАПИТІВ ДО БАЗИ ДАНИХ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторної роботи
з дисципліни «База даних»

для студентів-бакалаврів галузі знань 12 Інформаційні технології

Видано в редакції упорядників

Комп'ютерний дизайн, верстка та обробка – О.А. Жукова

Підписано до друку 25.04.2019. Формат 30х42/4.
Папір офсетний. Ризографія. Ум. друк. арк. 0,7.
Обл.-вид. арк. 0,7. Тираж 6 пр. Зам. №

Національний технічний університет «Дніпровська політехніка»
49005, м. Дніпро, просп. Д. Яворницького, 19