УДК 004.451.7.031.43

О.В. Скакаліна, канд. техн. наук
ПНТУ ім. Ю.Кондратюка, м. Полтава, Україна

ОРГАНІЗАЦІЯ ЕФЕКТИВНОГО ВИКОНАННЯ ЗАПИТІВ У РЕЛЯЦІЙНИХ БАЗАХ ДАНИХ

Е.В. Скакалина, канд. техн. наук
ПНТУ им. Ю. Кондратюка, г. Полтава, Украина

ОРГАНИЗАЦИЯ ЭФФЕКТИВНОГО ВЫПОЛНЕНИЯ ЗАПРОСОВ В РЕЛЯЦИОННЫХ БАЗАХ ДАННЫХ

O.V. Skakalina, Candidate of Technical Sciences
Poltava National Technical Yuri Kondratyuk University, Poltava, Ukraine

ORGANIZATION OF EFFICIENT PERFORMANSE OF QUERIES IN RELATIONAL DBMS

Проаналізовані питання, пов'язані з ефективною реалізацією виконання запитів у реляційних СКБД, які виконані мовами доступу до даних високого рівня, що вимагають одночасного оброблення декількох реляційних таблиць. Досліджено напрямки щодо визначення коректності таких запитів, розглянуто шляхи визначення оптимальної послідовності оброблення реляційних таблиць.

Ключові слова: *реляційні бази даних, запити, атрибути, оптимальна послідовність.*

Проанализированы вопросы, связанные с эффективностью выполнения запросов в реляционных СУБД, которые выполняются на языках доступа высокого уровня, которые требуют одновременной обработки нескольких реляционных таблиц. Исследованы направления определения корректности таких запросов, рассмотрены пути определения оптимальной последовательности обработки реляционных таблиц.

Ключевые слова: *реляционные базы данных, запросы, атрибуты, оптимальная последовательность.*

The paper deals with questions related to the efficient execution of requests in relational database systems that run on the languages of high level of access, which require simultaneous processing of several relational tables. We consider the direction of determining the correctness of such requests, the ways of determining the optimal sequence of processing of relational tables.

Key words: *relational DBMS, requests, trappings, the optimal sequence.*

Вступ. На думку К.Д. Дейта [1], реляційна модель складається з трьох частин:

- структурної;
- цілісної;
- маніпуляційної.

Структурна частина описує, які об'єкти розглядаються реляційною моделлю. Постулюється, що єдиною структурою даних, що використовується в реляційної моделі, є нормалізовані n-арні відношення.

Цілісна частина описує обмеження спеціального виду, які повинні виконуватися для будь-яких відношень у будь-яких реляційних базах даних. Це цілісність сутностей і цілісність зовнішніх ключів.

Маніпуляційна частина описує два еквівалентних способи маніпулювання реляційними даними – реляційну алгебру і реляційне числення. Перший механізм базується в основному на класичній теорії множин (з деякими уточненнями), а другий – на класичному логічному апараті обчислення предикатів першого порядку.

У цілісній частині реляційної моделі даних фіксуються дві базові вимоги цілісності, які повинні підтримуватися в будь-якій реляційній СКБД. Перша вимога називається вимогою цілісності сутностей. Об'єкту або сутності реального світу в реляційних БД відповідають кортежі відношення. Конкретно вимога полягає в тому, що довільний кортеж будь-якого відношення відрізняється від всіх інших кортежів цього відношення, тобто будь-яке відношення повинно мати один первинний ключ, причому альтернативних ключів може бути багато (або зовсім не бути).

Друга вимога називається вимогою цілісності за посиланнями і є трохи більш складним питанням. Вимога цілісності за посиланням, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, який з'являється у відношенні, повинен знайтися кортеж з таким самим значенням первинного ключа, або значення зовнішнього ключа повинне бути невизначеним (тобто ні на що не вказувати).

Аналіз досліджень і публікацій. У реалізаціях конкретних реляційних СКБД нині не використовується в чистому вигляді ні реляційна алгебра, ні реляційне числення. Фактичним стандартом доступу до реляційних даних стала мова SQL (Structured Query Language). Мова SQL являє собою суміш операторів реляційної алгебри і виразів реляційного числення та використовує синтаксис, близький до фраз англійської мови, і розширений додатковими можливостями, відсутніми в реляційній алгебрі та реляційному численні [2]. Взагалі, мова доступу до даних називається реляційно повною, якщо вона за виразністю не поступається реляційній алгебрі (або, що те ж саме, реляційному численню), тобто будь-який оператор реляційної алгебри може бути виражений засобами цієї мови. Саме такою і є мова SQL.

Оптимізація запитів є найбільш важливим та цікавим напрямком досліджень та розробок по всій області баз даних [3]. Важливість цього напрямку визначається тим, що від розвиненості компонента оптимізації запитів критично залежить загальна ефективність будь-якої SQL-орієнтованої СУБД.

На першій фазі запит, поданий на мові запитів, піддається лексичному і синтаксичному аналізу. При цьому виробляється його внутрішнє подання у структуру запиту і містить інформацію, яка характеризує об’єкти бази даних, що зазначені в запиті (відносини, поля і константи). Інформація про збережених у базі даних об’єктів вибирається з каталогів бази даних. Внутрішнє подання запиту використовується і перетворюється на наступних стадіях оброблення запиту.

На другій фазі запит у своєму внутрішньому поданні піддається логічній оптимізації. При цьому можуть застосовуватися різні перетворення, які "поліпшують" початкове подання запиту. Серед цих перетворень можуть бути еквівалентні перетворення, після проведення яких виходить внутрішнє подання, семантично еквівалентне початковому (наприклад, приведення до запиту деякої канонічної форми). Перетворення можуть бути і семантичними, коли отримане подання не є семантично еквівалентним початковому, але гарантується, що результат виконання перетвореного запиту збігається з результатом запиту в початковій формі при дотриманні обмежень цілісності, існуючих у базі даних. У будь-якому випадку після виконання другої фази оброблення запиту його внутрішнє подання залишається непроцедурним, хоча і є в деякому сенсі більш ефективним, ніж початкове.

Третій етап оброблення запиту полягає у виборі на основі інформації, якою володіє оптимізатор, набору альтернативних процедурних планів виконання цього запиту відповідно до його внутрішнього подання, отриманого на другій фазі. Крім того, для кожного плану оцінюється передбачувана вартість виконання запиту за цим планом. При оцінюваннях використовується статистична інформація про стан бази даних, доступна оптимізатору. З отриманих альтернативних планів вибирається найбільш дешевий, і саме його внутрішнє подання тепер відповідає оброблюваному запиту.

На четвертому етапі за внутрішнім поданням найбільш оптимального плану виконання запиту формується виконання подання плану. Подання плану, яке безпосередньо виконується, може мати вигляд програми в машинних кодах, якщо, як у випадку System R, система орієнтована на компіляцію запитів у машинні коди, або бути машинонезалежним, але більш зручним для інтерпретації, якщо, як у випадку INGRES, система орієнтована на інтерпретацію запитів. Нарешті, на останньому, п’ятому етапі оброблення запиту відбувається його реальне виконання відповідно до виконуваного планом запиту. Це або виконання відповідної підпрограми, або виклик інтерпретатора з передачею йому для інтерпретації виконуваного плану [4].

Формулювання цілей статті. Метою роботи є визначення алгоритму оптимальної послідовності оброблення таблиць при виконанні запитів у реляційних СУБД.

Виклад основного матеріалу статті. Припустімо, що у базі даних зберігаються такі таблиці, що мають такі структури:

1. План_випуску_деталей* (Шифр_деталі*, Вид_виконання, План, Місяць, Рік).
2. Класифікатор_видів_виконання (Шифр, Найменування).
3. Довідник_найменувань (Шифр, Найменування).
4. Звітній місяць (Місяць, Рік).

Створимо запит, що формує виробничу програму випуску деталей у звітному місяці.

SELECT Найменувань. Найменування, Довідник_найменувань. Шифр,
Класифікатор_видів_виконання. Найменування
FROM План_випуску_деталей, Класифікатор_видів_виконання,
Довідник_найменувань, Звітній_місяць
WHERE (План_випуску_деталей. Шифр_деталі=Довідник_найменувань. Шифр AND
План_випуску_деталей. Вид_виконання=Класифікатор_видів_виконання. Шифр AND
План_випуску_деталей. Місяць=Звітній_місць. Місяць AND
План_випуску_деталей. Рік=Звітній_місяць. Рік)

Множині атрибутів, з будь-якої таблиці, що бере участь у запиті, поставимо у відповідність набір вершин T, деякого графа $G = \ll V, E \gg, T \subseteq V$.

Кожна вершина відповідає певному конкретному атрибуту, наприклад, Шифр_деталі з таблиці План_випуску_деталей або Місяць з таблиці Звітний_місяць*.

З'єднаємо ребрами вершини, що належать одній таблиці, і отримаємо таким чином К повних підграфів, вхідного графа G, де К – кількість таблиць, що беруть участь у запиті. Отже, в результаті цього кроку отримаємо К компонент зв'язності для графа G. З'єднаємо ребрами вершини із різних компонент зв'язності, якщо вони відповідають будь-якій елементарній операції порівняння умові пошуку. Причому, якщо така операція порівняння встановлює зв'язок між n таблицями, то це приводить до появи нового підграфа G = <V,E> на n вершинах, де $\hat{V} \subseteq V, \hat{E} \subseteq E$. Таким чином, якщо у запиті у декількох таблицях присутні m різних атрибутів, зв'язаних між собою в умові пошуку, то у відповідному графі G буде m повних підграфів на вершинах, відповідних цим атрибутам (рис.).

Рис. Графічне відображення зв'язків таблиць у запиті

Якщо у результаті наведених перетворень отриманий граф виявився зв'язним, то це вказує на логічний зв'язок усіх таблиць, що беруть участь у запиті, в іншому випадку запит є некоректним і його виконання не має сенсу. Отже, для визначення коректності запиту необхідно визначити зв'язність граф, заданого своїми повними підграфами. Нині відомо багато алгоритмів визначення зв'язності графа [5; 6; 7]. Найшвидші з них ма-

ють складність 0(|E|). Використовуючи техніку прямої адресації (або алгоритм хешування), розроблено алгоритм визначення зв'язності графа, що має обчислювальну складність 0(|V|) і будує основне дерево, на вершинах деякого графа $\vec{G} = < \vec{V}, \vec{E} >$, відповідних повним підграфам G.

Введемо такі позначки:

N –	поточне ім'я підграфа;
NEXT(N) –	Оператор вибору наступного по порядку підграфа. На першому кроку вибір першого підграфа;
V –	Поточне ім'я вершини підграфа;
NEXT(N,V) –	Вибір наступної вершини підграфа N;
M –	Область, у якій зберігаються пари типу <N,V>;
E –	Область, у якій зберігаються імена ребер типу <N, N >;
A=h(x) –	Визначення точної адреси (або хеш-адреси) деякого імені X в області M або E;
□ –	Пусте значення. Є результатом оператора NEXT, якщо він не може вибрати наступне значення.

Алгоритм 1. Визначення зв'язності графа.

1. Встановити значення K. Обнулити M, E, T.
2. N:=NEXT(x). Якщо N=□, то перейти до п. 5. PRED:=Nю
3. V:=NEXT(N, V). Якщо V= □, то перейти до п. 2. A:=h(V)'.
4. Якщо M(A,1)=0, то M(A,2):=V,M(A,1):=PRED.

Інакше A1:=h(<N,M(A,1)>). Якщо E(A1)≠0, то перейти до п. 3.

Інакше E(A1):= <N,M(A,1)>,PRED:=M(A,1),T:=T+1.

Перейти до п. 3.

5. Якщо T=K-1, то граф зв'язний, інакше – граф не зв'язний.

Другою задачею, як було вказано раніше, що виникає під час реалізації запитів дореляційних баз даних, є визначення оптимальної послідовності оброблення таблиць. Її сенс полягає у тому, що після вибору рядка з однієї таблиці стають відомими значення декількох атрибутів, що визначають значення в умовах пошуку для інших таблиць і відповідно зменшують перебір рядків у цій таблиці. Якщо на фізичному рівні як індекс використовується, наприклад, мультисписок, то пошук буде виконуватися за ключовим атрибутом, значення якого менш всього зустрічається в таблиці. Це, в загальному випадку, відповідає атрибуту з максимальною потужністю. Якщо максимальна потужність пошукового атрибуту рівна P_i , то при пошуку по цьому атрибуту потрібно в середньому перебирати n_i/P_i , де n_i – число рядків в і таблиці. Якщо ж як ключ вибирається складений вторинний ключ [7; 8], то значення P_i буде дорівнювати добутку коефіцієнтів R тих атрибутів, що визначені у складеному ключі. Зміна послідовності оброблення таблиць приводить до значних змін у кількості рядків, що обробляються, що в кінцевому підсумку впливає на загальний час реалізації такого запиту. На фізичному рівні зміна кількості рядків, що обробляються, веде до різноманітної кількості блоків зовнішньої пам'яті, до яких іде звернення. Отже, з погляду ефективної реалізації (з погляду часу) будь-якого запиту для реляційних СУБД необхідно визначити таку послідовність оброблення таблиць, що забезпечить мінімальну кількість блоків зовнішньої пам'яті, до яких іде звернення.

Відомо [9; 10], що випадковий розподіл n записів у M областях (блоках) підкоряється закону розподілу Пуассона за параметром $\lambda=n/M$. Отже, кількість блоків, в які не влучить жоден запис, і, які відповідно не будуть прочитані у процесі пошуку інформації, дорівнюватиме $B_{i0} = B_i e^{-n_i/B_i}$.

Якщо ж в оброблення включається k таблиць, то загальна кількість записів, які обробляються, буде дорівнювати

$$Q = B'_{10} + B'_{10} * B'_{20} + B'_{10} * B'_{20} * B'_{30} + \cdots + B'_{10} * B'_{20} * B'_{30} \ldots * B'_{k0} = \\ = B'_{10}\left(1 + B'_{20}\left(1 + B'_{30}\left(1 + \cdots\left(1 + B'_{k0}\right)\ldots\right)\right)\right), \quad (1)$$

де B'_{i0} – наведена кількість блоків у таблиці (кількість блоків, що реально прочитуються при пошуку інформації за умовою).

Якщо у запиті беруть участь k документів, то загальна кількість можливих варіантів їхнього оброблення дорівнює k!. Складність обчислення значення Q за формулою (1) дорівнює O(k-1). Отже, загальна складність визначення оптимального рішення при розгляді усіх варіантів дорівнює $O_{пер}=O\left(k!\left(K-1\right)\right)$. Очевидно, що навіть при не дуже великому значенні k $O_{пер}$ дуже швидко збільшується із зростанням k.

Для розв'язання поставленої задачі пропонується використати модель у вигляді орієнтованого графа G=<A , R>, вершини A якого відповідають усім непустим підмножинам B_i множини таблиць, що беруть участь у запиті, та визначити на них відношення безпосереднього передування R [11]. Якщо підмножині B_i множини таблиць відповідає на графі крапка a і B_j – крапка b, то дуга $<a, b>\epsilon R$, якщо $B_i \subseteq B_j$ та не існує такого B_1, що $B_1 \subseteq B_i \subseteq B_j$. Граф G=<A, R>, що відображає відношення R, є без контурним, і його вершини можна розмістити по рівням відповідно до порядкової функції. На і-му рівні буде знаходитися $C_k^l, i = l, k$ вершин. Кожній вершині графа припишемо два числа: перше Q_i – вага вершини, що характеризує мінімальне число блоків, які прочитуються під час оброблення усіх таблиць, що складають множину цієї вершини; другий N_i – добуток «приведеного» числа блоків для тих же таблиць. $N_i = B'_{i0} \cdot B'_{l0} \cdot \cdots \cdot B'_{k0}\{i, j, \ldots, k\} \in B_i$. Кожній дузі графа припишемо вагу V_{ij}, що характеризує наведену кількість блоків для таблиці, що буде вибиратися після таблиці, якій відповідає вершина, з якої дуга виходить.

$V_{ij} = B_k\left(1 - B_k e^{-\frac{n_i}{B_i}}\right); k \in \left(B_j - B_i\right)$. Наприклад, для дуги, яка виходить з вершини {1,3,4 у вершину*{1,2,3,4 – вага буде характеризувати наведену кількість блоків для другої таблиці. Припишемо вершинам 1-го рівня $Q_j = N_j = N'_j, j = \overline{1,k}$. Вагу вершини на j рівні визначимо за формулою $Q_j = \min\left\{Q_j + N_j V_{ij}\right\}, \{< i, j >\in R\}$.

Твердження 1. Для будь-якої пари таблиць <*i*,*j*>, якщо $Q_{ij} \le Q_{ji}$, то і будь-яка послідовність оброблення таблиць, що починається із <*i*,*j*>, завжди краще, ніж будь-яка послідовність, що починається із <*j*,*i*>.

Доказ. Оскільки обчислення значення Q виконується за формулою (1), то $Q_{ij} = B'_i\left(1 + B'_j\right)$, $Q_{ji} = B''_j\left(1 + B''_i\right)$ і $B'_i\left(1 + B'_j\right) \le B''_j\left(1 + B''_i\right)$. Треба довести, що за цих умов завжди виконується нерівність $B'_i\left(1 + B'_j X\right) \le B''_j\left(1 + B''_i X\right)$, де X – значення, одержуване у внутрішніх дужках формули (1). З (1) виходить, що $B'_i\left(1 + B'_j\right) \le B''_j\left(1 + B''_i\right)$, $B'_i - B''_j \le B''_j B''_i - B'_i B'_j$, а значить і виконується нерівність $B'_i - B''_j \le \left(B''_j B''_i - B'_i B'_j\right) X$, оскільки значення X за своїм змістом є кількістю блоків, що обробляються, при пошуку інформації у 3-й та наступних таблицях і є числом позитивним більшим або рівним 1.

Проілюструємо роботу алгоритму на найпростішому прикладі. Наприклад, у запиті беруть участь чотири взаємопов'язані таблиці з кількістю зовнішніх блоків 120, 10, 40 і 1 відповідно. Значення Q і N для кожної вершини наведені у табл. 1, а ваги дуг V – у табл. 2.

Таблиця 1

Вага та добуток відповідних вершин графа

Вершина	1	2	3	4	21	31	41	23	42	43	213	421	431	423	4213
Q (вага)	120	10	40	1	130	160	121	410	11	41	250	131	161	411	251
N (добуток)	120	10	40	1	120	120	10	400	10	40	120	210	120	400	

Таблиця 2

Ваги дуг графа

Дуга	Вага дуги V	Дуга	Вага дуги V
1-21	1	31-213	1
1-31	1	31-431	1
1-41	1	41-421	1
2-21	12	41-431	1
2-23	10	23-213	3
2-42	1	23-423	1
3-31	3	42-421	12
3-23	40	42-423	40
3-43	1	43-431	3
4-41	120	43-423	10
4-42	10	213-4213	1
4-43	40	421-4213	1
21-213	1	431-4213	1
21-421	1	423-4213	1

Вага вершини на j рівні є мінімальним числом з *j!* можливих значень різноманітних послідовностей оброблення j таблиць, що складають множину цієї вершини. Після відпрацювання всього алгоритму вага останньої вершини дасть мінімальне значення Q.

Оцінимо обчислювальну складність алгоритму, що пропонується, під час оброблення k таблиць. Оскільки граф має k рівнів, і на кожному рівні обробляється C_k^i вершин, то $Q_{\text{алг}} = \sum_{i=1}^{k} i * C_k^i = k * 2^{k-1}$.

Отриману оцінку можна значно покращити, якщо не одержувати оцінку для усіх вершин, а застосовувати техніку засобу «гілок та кордонів» [12; 13]. Для кожної вершини графа визначається нижній кордон $W_i = Q_i + N_i$, кількість читаємих зовнішніх блоків, і подальший пошук розв'язання виконується для вершини, що має мінімальне значення W_i на кожному кроці.

Висновки. Реляційні мови запитів забезпечують високорівневий «декларативний» інтерфейс для доступу до даних, збережених у реляційних базах даних. Ключовими підкомпонентами компонента обчислення запитів у SQL-орієнтованої системи баз даних є оптимізатор запитів і підсистема виконання запитів. Сформульовані необхідні умови коректності запитів до взаємопов'язаних таблиць у реляційних СУБД, побудований алгоритм перевірки відповідності цим умовам, що має лінійну складність. Розроблений алгоритм визначення оптимальної послідовності оброблення таких таблиць, що, в свою чергу, забезпечує мінімізацію кількості зовнішніх звернень до бази даних.

Список використаних джерел

1. *Дейт К. Дж.* Введение в системы баз данных / К. Дж. Дейт. – 8-е издание. – М. : Вильямс, 2005. – 1316 с.
2. *Карпова І. П.* Основи баз даних / І. П. Карпова. – М., 2009. – 132 с.

3. *Matthias Jarke*, Jurgen Koch. Query Optimization in Database Systems. Computing Surveys. – Vol. 16. – No. 2. – June 1984.

4. *Chaudhuri S.*, Shim K. Query Optimization with Aggregate Views. In Proc. of EDBT. – Avignon, 1996.

5. *Ахо А.* Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – М. : Мир, 1978. – 536 с.

6. *Конноли Т.* Базы даннях: проектирование, реализация и сопровождение / Т. Конноли, К. Бегг. – М. : Вильямс, 2008. – С. 1440.

7. *Пападимитриу Х.* Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц. – М. : Мир, 1985. – 512 с.

8. *Кнут Д.* Искусство программирования для ЭВМ / Д. Кнут. – М. : Мир, 1978. – Т. 3. Сортировка и поиск. – 844 с.

9. *Скакалина Е. В.* Метод организации индексных файлов на основе полиномиальных функций адресации / Е. В. Скакалина, А. В. Тарасов. – Полтава, 2001. – С. 27-35.

10. *Тиори Т.* Проектирование структур баз данных / Т. Тиори, Дж. Фрай. – М. : Мир, 1985. – Кн. 2. – 320 с.

11. *Larson P.* Expected worth-case performance of hash files // Comput. Jornal. – 1982, 25. – № 3. – P. 347-352.

12. *Берзтисс А. Т.* Структуры данных / А. Т. Берзтисс. – М. : Статистика, 1974. – 408 с.

13. *Корбут А. А.* Дискретное программирование / А. А. Корбут, Ю. Ю. Филькенштейн. – М. : Наука, 1969. – 368 с.

14. *Рейнгольд Э.* Комбинаторные алгоритмы. Теория и практика / Э. Рейнгольд, Ю. Нивергельт, Н. Део. – М. : Мир, 1980. – 476 с.