Оптимизация на SQL заявки

Ирена Вълова, Йордан Калмуков

SQL Query Optimization: *This paper focuses on the importance of SQL query optimization for building fast and scalable information systems. It briefly describes the query optimizer embedded to every modern relational database management system and suggests a set of rules for creating computationally efficient SQL queries.*

Key words: *relational databases, query, optimization.*

ВЪВЕДЕНИЕ

Какво означава оптимизация на заявките в базите от данни? Представете си, че имате възможност да изберете и посетите 20 града в Европа, като единственото ограничение, което ви е наложено, е времето за обхождане на тези градове. Бихте ли тръгнали да ги обхождате в произволен ред? Едва ли. Човек, изправен пред подобна задача, обикновено би разпределил градовете в групи, на база на разстоянието между тях. Обхождането започва от най-близката група, след което продължава със следващата най-близко разположена група и т.н. При това е възможно обхождането на градовете да се организира толкова добре, че дори времето, като ограничение, да загуби своето практическо значение.

Процесът на оптимизация на заявките работи по подобен начин. Възможни са множество различни начини за написване на една заявка, а резултатите от изпълнението им са едни и същи. Системата за управление на бази от данни (СУБД) се стреми да обработи заявката по най-оптималния начин, така че ***да минимизира времето, необходимо за нейното изпълнение***. Стремежът винаги е идеалният вариант: да се намери най-добрия план на изпълнение на заявката. На практика много често е приемлив и реалният вариант: да се избегне най-неблагоприятния план на изпълнение на заявката.

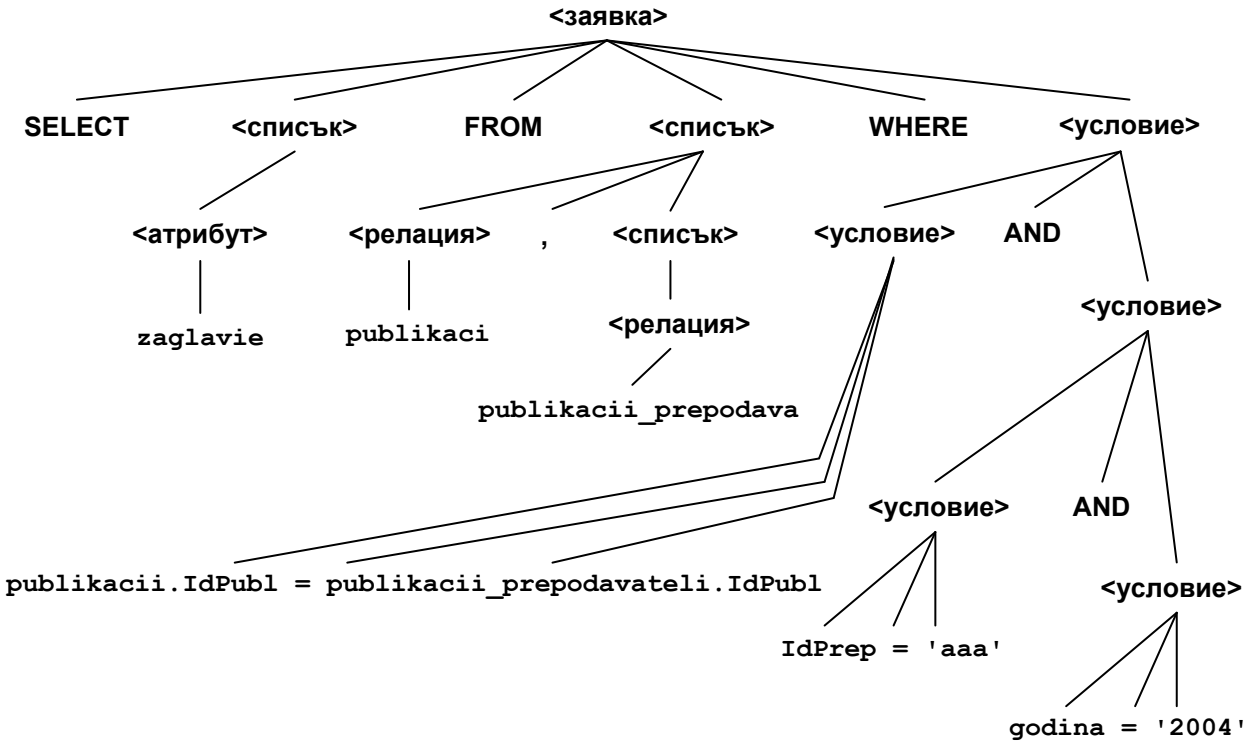
ОБРАБОТВАНЕ НА ЗАЯВКИТЕ В СИСТЕМИТЕ ЗА УПРАВЛЕНИЕ НА БАЗИ ОТ ДАННИ

Като се започне от най-старата СУБД - System-R, повечето от комерсиалните СУБД използват оптимизатори на база разхода на време. Обработката на всяка заявка преминава през три важни стъпки:

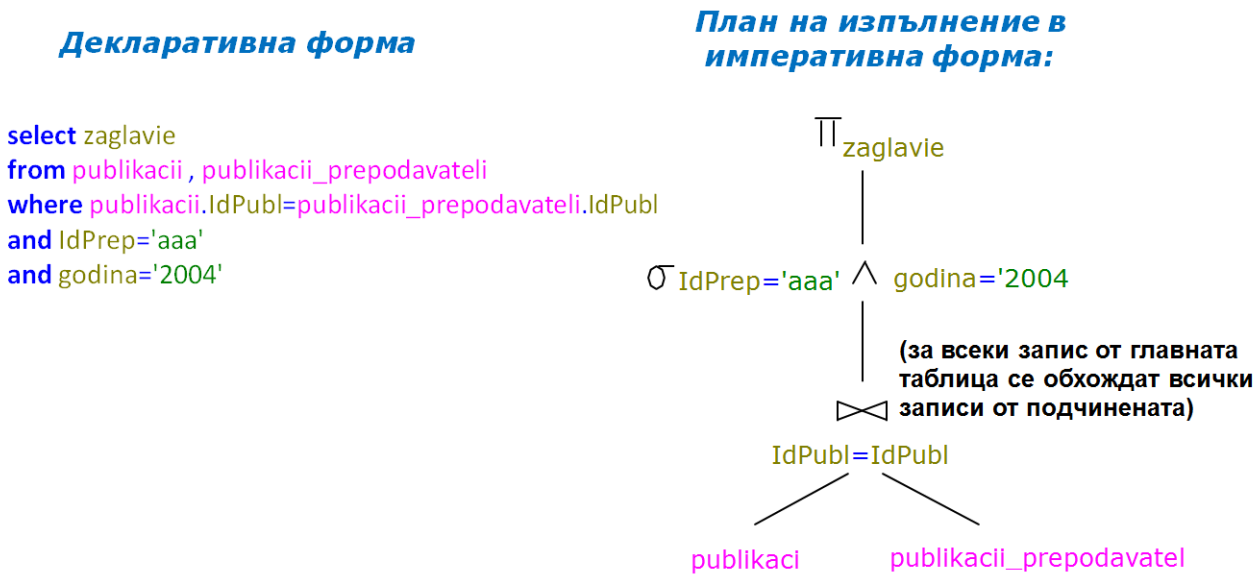
Написаната на езика SQL заявка се анализира синтактично и семантично, и се трансформира от декларативна форма в дърво [1,2] (фигура 1). Ако в заявката се използват изгледи, всеки изглед се заменя със съответното му дървовидно представяне, което се взема от дефиницията на изгледа. Извършва се и семантична проверка, защото дори и да е синтактично вярна дадена заявка, това не гарантира, че не нарушава някое от семантичните правила. Проверяват се използваните релации от FROM клаузата. Всички атрибути от SELECT и WHERE клаузите, освен за принадлежност към релациите се проверяват и дали са пълно дефинирани. Не на последно място се отчитат типовете на атрибутите и дали са съвместими с начина им на използване в заявката.

Полученото на предходния етап синтактично дърво се преобразува, в термините на релационна алгебра, в императивно дърво. Последното представлява логически план на заявката (фигура 2).

```
select zaglavie
from publikacii, publikacii_prepodavateli
where publikacii.IdPubl=publikacii_prepodavateli.IdPubl
and IdPrep='aaa' and godina='2004'
```



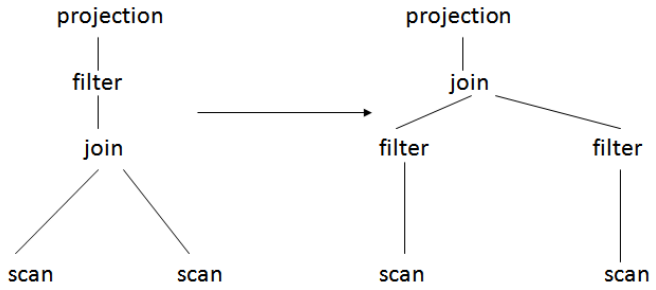
Фигура 1. SQL заявка и представянето ѝ във вид на дърво



Фигура 2. Анализ на SQL заявки и трансформиране от декларативна в императивна форма.

Логическият план може да се преобразува във физически план на изпълнение на заявката, след като се анализират операциите, които ще се извършат, редът на тяхното изпълнение, използваните алгоритми на всяка стъпка, начините на представяне и съхранение на данните, и начините за предаване на данните между отделните операции. В резултат на този анализ се избира оптималният логически план. На фигура 3 са показани два различни варианта на представяне на заявката от фигура 1. На база на законите за преобразуване на еквивалентни изрази в релационната алгебра могат да се получат различни логически представяния и планове за изпълнение на една и съща заявка.

Тези закони се разделят в няколко големи групи:




```
select zaglavie
from publikacii , publikacii_prepodavateli
where publikacii.IdPubl = publikacii_prepodavateli.IdPubl
and IdPrep=' aaa'
and godina='2004'
```

Фигура 3. Два логически плана на изпълнение на една и съща заявка

- ***Закони, които касаят операцията селекция***:

Селекцията е особено важна операция що се касае до оптимизацията на заявките, тъй като тя в голяма степен определя броя на записите, с които се оперира в заявката и тези, които ще бъдат върнати като краен резултат. За да се оптимизират заявките е желателно селекциите да се преместят колкото е възможно по-ниско в дървото на заявката, без това да се отрази на релационния израз. Основните начини за преместване на селекциите са:

- Декомпозиране с цел опростяване на сложните логически условия - разделяне на условието на неговите подусловия. Идеята е, че определена част от тези подусловия касаят отделни атрибути и селекцията за тях може да бъде извършена на много ранен етап от изпълнението на заявката. По този начин ще бъдат редуцираните записите, над които ще се изпълняват селекции на следващите нива в дървото.
- Напоследък обаче се говори и за оптимизация на заявките чрез преместване на селекциите по-високо в дървото, като за целта се използват изгледи. По този начин може част от условието на селекцията да се реализира като изглед. В заявката да се използва опростен логически израз в селекцията и вместо изходните таблици да се използват част от тях и този изглед.

Не може да се каже, че има универсален подход и да се препоръча точно използването на един от двата метода с по-добър успех.

- ***Закони, които касаят операцията проекция***:

Преместването на операцията проекция в логическия план на изпълнение на заявката не оказва толкова съществено влияние върху изпълнението на заявката, както операцията селекция. Принципно тази операция може да бъде преместване на произволно ниво в дървото, ако тя касае само атрибути, които не се използват от операции, намиращи се по-високо в дървото и не присъстват в резултата.

- ***Закони, които касаят операциите съединение и декартово произведение***:

Съединението е декартово произведение с последваща селекция. В зависимост от условието на селекцията се определя и типът на съединението. В SQL заявките селекциите се изпълняват от дясно на ляво по реда на срещането им в условието. След въвеждане на съответния стандарт, всички по-нови СУБД поддържат използването на **`join`** за съединение. Тази клауза е за предпочитане, защото:

- това е коректния стандартен начин за свързване на таблици;
- по този начин се отделят условията на съединението от условията на селекцията;
- в някои СУБД все още няма възможност да бъдат реализирани всички съединения (например ляво свързване) в where клаузата, така че **`join`** е единствен вариант.

- ***Закони, които касаят операциите за елиминиране на дублиращи се елементи***:

Задължително тази операция се премества на възможно най-ниските нива в дървото на заявката. Гаранция, че няма нужда от тази операция е наличието на първичен ключ в отношението-операнд.

- ***Закони, които касаят групиращи и обобщаващи операции***:

Не могат да се формулират общи правила и закони за тази група операции, тъй като изпълнението им зависи от използвания оператор за обобщение. Може да се препоръча премахване на ненужните атрибути от отношенията-операнди още преди да се извърши обобщаване или групиране.

Основен извод, който може да се направи:

Системите за управление на бази от данни имат механизъм за оптимизация на заявките, който се прилага за всяка една заявка, но начинът на изписване и подреждане на операциите в заявките също има значение за времето на тяхното изпълнение. Именно с правилното формулиране на заявките от програмистите могат да бъдат решени много от проблемите при проектиране, реализация и използване на информационните системи.

ПРЕПОРЪКИ ЗА ФОРМУЛИРАНЕ НА ЗАЯВКИТЕ

Какви препоръки могат да бъдат направени:

- Винаги да се използват заявки, които връщат ***точния резултат***, който се изисква, а не да се разчита на допълнителна обработка на върнатите данни в кода на езика от високо ниво. Например: искаме информация за служителите от определен факултет. В този случай можем да постъпим по следния начин:

да вземем от БД данните за всички служители със следната заявка:

```
select * from employees;
```

след което върнатият резултат се филтрира с допълнително разработена функция, за да се отсее ненужното.

или директно да се използва филтриране в самата заявка:

```
select * from employees where IdFaculty='001';
```

Във всички случаи филтрирането в СУБД е по-бързо от филтрирането в език от високо ниво. Но по-важно е, че по-малкият брой записи върнати от СУБД, означава по-малко време за извличане на резултата, а от там и по-бързото му получаване от приложението, което използва СУБД.

- ***Символите за заместване*** са много удобни по време на тестване и проверка на състоянието на базата от данни, но оправдано ли е, за да се напише по-лесно, кратко и бързо една заявка, да се използва *, ако от всичките n на брой колони са необходими данните само от 1, 2 или 3? С използване на * се предизвиква следния ефект:
  - Увеличава се времето за изпълнение на заявката.
  - Извличат се излишни данни.
  - Трансферират се излишни данни, което изисква допълнително време.

Ако са необходими само заглавие и автори на книгите, няма смисъл да се изпълнява заявка: `select * from knigi`

Препоръчително е да се посочат точно нужните атрибути:

```
select zaglavie, avtori from knigi
```

Друг заместващ символ, който трябва да се използва с повишено внимание е символът `%` в **`like`** оператор за сравнение. Особено тежки са заявки, които съдържат този заместващ символ и като начало, и като край за символния низ в

шаблона, защото по този начин се налага сравнение с всички записи, независимо от използването на индекси и сортиране. Най-щадящо е използването му като край на шаблона, така че ако не е наложително, не трябва да се поставя само защото няма да промени крайния резултат от заявката.

- Дали използването на ***клаузата за сортиране*** на резултата от заявката - **order by** – влияе на времето за изпълнение на заявката? Използването ѝ е полезно при тестване с цел по-бързо и лесно сравнение на получени резултати. На фигура 4 е показано времето на изпълнение на една и съща заявка с и без клауза за сортиране:

```
select zaglavie, avtori from publikacii                    (1)
select zaglavie, avtori from publikacii order by zaglavie  (2)
```

```
select zaglavie, avtori from publikacii;
/* 0 rows affected, 13,848 rows found. Duration for 1 query: 0.000 sec. (+ 0.360 sec. network) */
select zaglavie, avtori from publikacii  order by zaglavie;
/* 0 rows affected, 13,848 rows found. Duration for 1 query: 0.141 sec. (+ 0.531 sec. network) */
```

Фигура4. Време за изпълнение на една и съща заявка без и с клауза за сортиране на резултата

Неоспоримо е обаче, че ако така или иначе е необходим сортиран резултат, то сортирането в заявката е по-бързо от който и да е метод за сортиране във външната програма.

- Не се препоръчва ***цикличното извикване на една и съща заявка*** за отделни стойности на променливата на цикъла. Много по-бързо би работил вариантът с еднократна заявка и селекция на определени редове:

```
for i=2000 to 2004
select zaglavie, avtori from knigi
where godina =i                                  (1)

select zaglavie, avtori from knigi
where godina between 2000 and 2004               (2)

select zaglavie, avtori from knigi
where godina >= 2000 and godina <=2004           (3)
```

Варианти (1), (2) и (3) връщат един и същ резултат, но от трите най-оптимален е вариант (2). В (1) се извиква 5 пъти за изпълнение заявката, която връща по 1 запис. В (3) има две сравнения, поради което селекцията ще се изпълни по-бавно. В (2) имаме едно условие на селекцията.

- ***Влагането на заявки*** дава възможност с една заявка да се достигне до максимално конкретен резултат. Дали обаче е за предпочитане използването на вложени заявки във всички клаузи, които допускат това?

Нека имаме следната заявка:

```
SELECT * FROM knigi, knigi_avtori, papers
WHERE knigi.tip=papers.id
and knigi.Id= knigi_avtori.knigaId
and avtori like '%ова%'
and concat(knigi.Id, 'aaa') not in (
           select concat(knigaId, avtorId) from knigi_avtori)
and knigi_avtori.status='1'
order by tip, Avtori;
```

В условието на селекцията (в where клаузата) се съдържа операция **concat**(knigi.Id, 'aaa') за слепване на два символни низа, чийто резултат се сравнява за непринадлежност към множеството на върнатите резултати от подзаявката:

```
select concat(knigaId, avtorId) from knigi_avtori
```

За всеки запис, който отговаря на селекцията във външната заявка ще се изпълнява функцията за слепване и ще се извиква и изпълнява вложената заявка.

Лесно може да се отдели вложената заявка или като изглед, който да се използва във from клаузата, или във вложена заявка във from клаузата, като по този начин подзаявката няма да се изпълнява повече от един път.

Ако все пак се налага използването на подзаявки, то е препоръчително да се минимизира техния брой. Например:

```
select name
from employee
where (salary, age) = (select max(salary), max(age) from
                          employee_details)
and dept = 'electronics';
```

вместо:

```
select name
from employee
where salary = (select max(salary) from employee_details)
and age = (select max(age) from employee_details)
and dept = 'electronics';
```

В последната заявка има две вложени, намиращи се в условието на селекцията на основната. За всеки потенциален резултантен запис на основната заявка ще се изпълняват задължително по две вложени заявки. Поради това първият вариант на изписване е за предпочитане.

- Не на последно място трябва да се отбележи ***правилното използване на филтър*** в where и having клаузите. Не е грешно и заявката ще работи, и връща резултат, ако условията от where клаузата, касаещи селекция, се изместят в having клаузата, когато се използват групиращи функции.

Например следната ситуация:

```
select avtori.ime, avtori.familia, count(knigi_avtori.avtorId) broi
from avtori, knigi_avtori
where avtori.Id=knigi_avtori.avtorId
and (knigi_avtori.status ='1' or knigi_avtori.status='2')
group by avtori.ime, avtori.familia
having avtori.familia like '%A%'
```

В having клаузата имат място само условията, които касаят резултатите от групиращата функция, тъй като те не могат да се използват в where клаузата. Всички останали условия, касаещи селекцията на записи трябва да бъдат винаги в where клаузата:

```
select avtori.ime, avtori.familia, count(knigi_avtori.avtorId) broi
from avtori, knigi_avtori
where avtori.Id=knigi_avtori.avtorId
and (knigi_avtori.status ='1' or knigi_avtori.status='2')
and avtori.familia like '%A%'
group by avtori.ime, avtori.familia
```

За тази примерна заявка има смисъл от having клауза само, ако е необходимо филтриране на групите в крайния резултат по стойностите на групиращата функция:

```
select avtori.ime, avtori.familia, count(knigi_avtori.avtorId) broi
from avtori, knigi_avtori
where avtori.Id=knigi_avtori.avtorId
and (knigi_avtori.status ='1' or knigi_avtori.status='2')
and avtori.familia like '%A%'
group by avtori.ime, avtori.familia
having count(knigi_avtori.avtorId)>10
```

Други, важни за оптимизацията на изпълнението на заявките, препоръки:

- Използването за индекси – за атрибутите, представляващи първични ключове по подразбиране се създава индекс от самата СУБД. Желателно е атрибутите, дефинирани като unique също да бъдат индексирани, ако по тях ще се извършва търсене. Разбира се, трябва да се има предвид, че многото създадени индекси могат значително да забавят изпълнението на заявките за добавяне, редактиране и изтриване на данни.
- Известно е, че по възможност трябва да се избягват сравнения и логически изрази с NOT оператор. По-бързо се изпълняват неотрицателните съпоставяния.
- Различните видове съединения между таблиците от базата от данни също оказват влияние върху изпълнението на заявката.

ЗАКЛЮЧЕНИЕ

Съвременните системи за управление на бази от данни притежават вграден оптимизатор на заявки, чиято основна задача е да минимизира времето за изпълнение на постъпилите заявки. За целта, той динамично променя синтаксиса им и/или мястото на което се изпълняват съответните операции в тях. Въпреки това неговите възможности не са неограничени, а ефективността му може силно да варира в зависимост от начина, по който е формулирана всяка конкретна заявка. Отговорност на програмиста е да напише SQL заявките си така, че да се изпълняват възможно най-бързо, използвайки минимално количество изчислителни ресурси. Като правило, достъпът до данните е многовариантна задача. Възможно е да се формулират множество различни SQL заявки, които да водят до един и същ краен резултат. В този случай е препоръчително да се използват средствата предоставени от SQL и избраната СУБД, и да се изследва поведението на всички алтернативни заявки. В последствие, след анализ на плана на тяхното изпълнение, от всички алтернативни заявки може да се подбере най-оптималната и само тя да бъде интегрирана в разработваното приложение / информационна система.

ЛИТЕРАТУРА

[1] H. Garcia-Molina, Jeffrey Ullman, J. Widom. Database Systems: The Complete Book. Prentice-Hall, Englewood Cliffs, NJ, 2002.

[2] Вълова, И. Бази от данни - Въведение в SQL. Русе, България, Печатна База при Русенски Университет, 2009, ISBN 978-954-712-465-3.

За контакти:

Доц. д-р Ирена Вълова, Катедра “Компютърни системи и технологии”, Русенски университет “Ангел Кънчев”, тел.: 082-888 685, e-mail: Irena@ecs.uni-ruse.bg

ас. инж. Йордан Калмуков, Катедра “Компютърни системи и технологии”, Русенски университет “Ангел Кънчев”, тел.: 082-888 685, e-mail: JKalmukov@ecs.uni-ruse.bg

Докладът е рецензиран.

Оптимизация на SQL заявки

Ирена Вълова, Йордан Калмуков

SQL Query Optimization

Irena Valova, Yordan Kalmukov

Abstract: *This paper focuses on the importance of SQL query optimization for building fast and scalable information systems. It briefly describes the query optimizer embedded to every modern relational database management system and suggests a set of rules for creating computationally efficient SQL queries.*

Key words: *relational databases, query, optimization.*