Погружение в СУБД. Сезон 2017

Оптимизация выполнения запросов

Дмитрий Барашев

Computer Science Center

Санкт-Петербург 2017

Сверстано в Папирии

papeeria.com

онлайн редактор для LaTeX и Markdown• совместное редактирование в реальном времени • интеграция с Git репозиториями • графики

подсветка синтаксиса • автодополнение • проверка орфографии • предпросмотр математических формул • галерея шаблонов

Оптимизация выполнения запросов

Оптимизация выполнения запросов

- Жизнь запроса внутри СУБД

Оптимизация выполнения запросов

- Что влияет на выбор плана выполнения запроса

Оптимизация выполнения запросов

- Инструменты для анализа выполнения запроса

Оптимизация выполнения запросов

- Индексы и материализованные представления

Жизнь запроса

Жизнь запроса внутри СУБД

- Синтаксичесий разбор, проверка метаинформации, прав, вот это вот всё

Жизнь запроса внутри СУБД

- Построение логического плана, его трансформация, выбор физических операций

Логический план

- Результат синтаксического анализа на каком-то внутреннем языке

Логический план

- Дерево с операциями в узлах, таблицами в листьях

Запрос и план

```
SELECT *
FROM Conference C
JOIN Participant P
  USING(conference_id)
JOIN Researcher R
  USING(researcher_id)
JOIN University U
  USING(university_id)
WHERE U.name='Stanford'
  AND C.name='VLDB''15';
```

Запрос и план

```
SELECT *
FROM Conference C,
     Participant P,
     Researcher R,
     University U
WHERE U.name='Stanford'
  AND C.name='VLDB''15'
  AND C.conference_id=
      P.conference_id
  AND P.researcher_id=
      R.researcher_id
  AND R.university_id=
      U.university_id
```

Проталкивание предикатов

Predicate push-down

$$\sigma_{\Theta_R}(R \bowtie S) \quad = \quad \sigma_{\Theta_R}(R) \bowtie S$$

Проталкивание предикатов

Predicate push-down

$$\sigma_{\Theta_R}(R \bowtie S) = \sigma_{\Theta_R}(R) \bowtie S$$

$$\sigma_{\Theta_R}(R \times S) = \sigma_{\Theta_R}(R) \times S$$

Проталкивание предикатов

Predicate push-down

$$\sigma_{\Theta_R}(R \bowtie S) = \sigma_{\Theta_R}(R) \bowtie S$$

$$\sigma_{\Theta_R}(R \times S) = \sigma_{\Theta_R}(R) \times S$$

$$\sigma_{\Theta_1 \,\mathtt{AND}\, \Theta_2}(R) = \sigma_{\Theta_1}(\sigma_{\Theta_2}(R))$$

Эквивалентный запрос

```
SELECT *
FROM (SELECT * FROM Conference
      WHERE name='VLDB''15') C
JOIN Participant P
  USING(conference_id)
JOIN Researcher R
  USING(researcher_id)
JOIN (SELECT * FROM University
      WHERE name='Stanford') U
  USING(university_id)
```

Ещё пара эквивалентных запросов

```sql
-- Подзапрос превращается...
SELECT *
FROM Researcher
WHERE university_id IN (
  (SELECT university_id FROM University
   WHERE country='RU')
)

-- ... в элегантное соединение
SELECT R.*
FROM Researcher R JOIN University U
     USING(university_id)
WHERE U.country = 'RU'
```

Анализ выполнения запросов

Схема БД

```
-- Конференции, 1000 записей
Conference(conference_id, name, country, budget);

-- Участники, 100000 записей
Participant(conference_id, researcher_id);

-- Исследователи, 20000 записей
Researcher(researcher_id, name, university_id);

-- Университеты, 200 записей
University(university_id, name, country);
```

Представление

```
-- Удобно показывать участников конференции,
-- сгруппированных по университетам
CREATE VIEW ParticipantView AS (
  SELECT C.conference_id,
         C.name AS conf_name,
         R.researcher_id,
         R.name AS res_name,
         R.university_id
    FROM Conference C
    JOIN Participant USING(conference_id)
    JOIN Researcher R USING (researcher_id)
    ORDER BY R.university_id);
```

Запросы

```sql
-- Для каждого университета показать его название
-- и суммарное количество фактов участия его
-- сотрудников в конференциях
WITH Tmp AS (
  SELECT * FROM University ORDER BY university_id
)
SELECT Tmp.name, COUNT(*)
FROM Tmp
JOIN ParticipantView USING(university_id)
GROUP BY Tmp.university_id, Tmp.name;

-- Для заданного университета сосчитать количество
-- фактов участия его сотрудников в конференциях
SELECT COUNT(*) FROM ParticipantView
WHERE university_id = (
  SELECT university_id FROM University
  WHERE name='Stanford'
);
```

```sql
-- Для каждого университета показать его название
-- и суммарное количество фактов участия его
-- сотрудников в конференциях
SELECT name, GetParticipantCount(university_id)
FROM University;

-- Возвращает количество фактов участия сотрудников
-- университета _uni_id в конференциях.
-- Параметры: _uni_id -- идентификатор университета
-- Вовзращает: количество фактов участия
-- TODO: добавить документацию, определяющую факт участия
CREATE OR REPLACE
FUNCTION GetParticipantCount(_uni_id INT)
RETURNS BIGINT AS $$
SELECT COUNT(*) FROM ParticipantView
WHERE university_id = _uni_id;
$$ LANGUAGE SQL;
```

```
SELECT name,
       GetParticipantCount(university_id)
FROM University;
```

Как помочь оптимизатору

- Писать относительно простые «плоские» запросы

Как помочь оптимизатору

- Не заниматься преждевременной оптимизацией

Как помочь оптимизатору

- Отказаться от процедурного мышления

Использование индексов

Что такое индекс

применительно к СУБД

- Концептуально: персистентная избыточная структура данных, прозрачная для запросов и предназначенная для ускорения некоторых операций

Что такое индекс

применительно к СУБД

- Логически: отображение ключ $\rightarrow$ значение

Что такое индекс

применительно к СУБД

- Физически: обычно сбалансированное упорядоченное дерево (B-tree) или хеш-таблица

Ключ и значение

- Пусть у нас есть таблица T в которой есть атрибут a
- Индекс строится для атрибута или группы атрибутов
- Ключами в индексе будут значения проиндексированного атрибута
- Значениями в индексе будут указатели на записи

Ключ и значение

если $I_a(T)$ это индекс для атрибута a таблицы T

k это ключ $\in I_a(T)$

$p(k) \to t$ – значение ключа, указатель на запись $t \in T$

Тогда $V_a(t) = k$

Как помогают индексы

чисто теоретически

Если требуется выполнять запросы $\sigma_{a=\ldots}(T)$, то

- поиск без индекса выполняется Sequential Scan'ом – последовательным сканированием всех страниц таблицы
- если страниц B и распределение значений атрибута в запросах равномерное то матожидание количества прочитанных страниц равно $\frac{B}{2}$

Как помогают индексы

чисто теоретически

Если требуется выполнять запросы $\sigma_{a=\ldots}(T)$, то

- поиск с индексом $I_a(T)$ выполняется Index Scan'ом: поиск в индексе + обращение к страницам таблицы в случае успеха
- сколько страниц читается в индексе?
- к скольки страницам таблицы надо будет обратиться?

Поиск в индексе

в случае B-дерева

- B-дерево – это ветвистое дерево поиска
- Надо пройти от вершины к корню
- Каждый переход – чтение страницы

Поиск в индексе

в случае B-дерева

- B-дерево – это ветвистое дерево поиска
- Надо пройти от вершины к корню
- Каждый переход – чтение страницы
- Число прочитанных страниц равно высоте дерева
- Высота равна $log_b(N)$, где b - степень ветвистости, N - число ключей

Ветвистость B-дерева

- Интервалы разделяются ключами поиска
- Переход в поддерево – указатель на страницу
- Если ключами поиска являются целые числа то на дисковую страницу помещается десятки и сотни интервалов

Ветвистость B-дерева

- Интервалы разделяются ключами поиска
- Переход в поддерево – указатель на страницу
- Если ключами поиска являются целые числа то на дисковую страницу помещается десятки и сотни интервалов

$$\log_{100}(N) = 3 \Rightarrow N = 100^3 = 1000000$$

Селективность запроса

- СУБД собирает статистику о значениях атрибутов
- Гистограмма или просто количество разных значений

Селективность запроса

- СУБД собирает статистику о значениях атрибутов
- Гистограмма или просто количество разных значений
- Статистика используется для оценки селективности запроса: отношение количества строк в результате к количеству строк в таблице
- Если селективность плохая то дешевле выполнить Seq Scan, чем использовать индекс

Составной индекс

```sql
CREATE INDEX idx_budget_country
  ON Conference(budget, country);

-- (500000, 'RU') < (600000, 'RU')
-- (500000, 'RU') > (500000, 'DE')
```

Функциональные индексы

```
CREATE INDEX idx_country
  ON Conference(country);

-- Страны: Ru, US, nz, UK, Ca, De, ...
-- 150 штук
SELECT * FROM Conference
WHERE upper(country) = 'NZ';
```

Когда индекс полезен

- Таблица большая
- Индексируемый атрибут небольшой
- У индексируемого атрибута много разных значений
- Статистика не врёт
- Запросы, использующие проиндексированный атрибут, приходят часто

Индекс может быть вреден

- Индексы надо поддерживать в актуальном состоянии
- Ошибочное использование индекса из-за неверной статистики или ошибки в оптимизаторе может очень сильно замедлить выполнение запросов

Материализация и денормализация

Материализация и денормализация

Дерево ключевых слов

- Иерархия из 10000 ключевых слов
- Хранится при помощи Nested Sets
- Максимальная глубина 18
- Средняя глубина 8

Дерево ключевых слов

- Иерархия из 10000 ключевых слов
- Хранится при помощи Nested Sets
- Максимальная глубина 18
- Средняя глубина 8

- Запросы извлекают дерево глубиной 3 из заданного корня

Не очень хороший запрос

```
SELECT  C.* FROM
Keyword P JOIN Keyword C
  ON C.lft > P.lft AND C.lft < P.rgt
                              -- ищем потомков
WHERE (
    SELECT  COUNT(*) FROM Keyword K
    WHERE C.lft > K.lft  -- у потомка число вершин
      AND C.lft < K.rgt  -- выше него
      AND K.lft > P.lft  -- но ниже родителя
      AND K.lft < P.rgt
) < 3                         -- меньше трех
AND P.keyword_id = 2;     -- а это родитель
```

Уберем коррелирующий подзапрос

```
WITH AncestorCount AS (
  SELECT 0 AS keyword_id, 0 AS ancestor_count
  UNION
  SELECT Child.keyword_id,
         COUNT(Parent.keyword_id) as ancestor_count
  FROM Keyword Child JOIN Keyword Parent
    ON Child.lft > Parent.lft ANd Child.lft < Parent.rgt
  GROUP BY Child.keyword_id
)
SELECT K.* FROM
Keyword K JOIN AncestorCount A USING(keyword_id)
WHERE K.lft > (SELECT lft FROM Keyword WHERE keyword_id=2)
AND K.lft < (SELECT rgt FROM Keyword WHERE keyword_id=2)
AND A.ancestor_count <= (
  SELECT ancestor_count
  FROM AncestorCount
  WHERE keyword_id=2
) + 3;
```

Глубина вершины

- Каждый запрос считает глубины вершин
- Что если они меняются редко?

Избыточная информация

- Давайте хранить глубину вместе с вершиной:

```
CREATE TABLE Keyword(
    keyword_id INT PRIMARY KEY,
    lft INT,
    rgt INT,
    depth INT);
```

Избыточная информация

- Давайте хранить глубину вместе с вершиной:

```
CREATE TABLE Keyword(
    keyword_id INT PRIMARY KEY,
    lft INT,
    rgt INT,
    depth INT);
```

- Плюсы: всё будет летать.
- Минусы: обновления стали ещё сложнее, растёт риск несогласованности данных

Материализованное представление

- Представление можно записать на диск
- Чтения представления будут читать уже посчитанные данные с диска
- Если произошло обновление данных то пересчитать представление можно одной командой

Материализованное представление

```
-- Создадим материализованное представление
CREATE MATERIALIZED VIEW AncestorCount AS
  SELECT 0 AS keyword_id, 0 AS ancestor_count
  UNION
  SELECT Child.keyword_id,
         COUNT(Parent.keyword_id) as ancestor_count
  FROM Keyword Child JOIN Keyword Parent
    ON Child.lft > Parent.lft ANd Child.lft < Parent.rgt
  GROUP BY Child.keyword_id;

-- Будем его обновлять после изменения дерева
REFRESH MATERIALIZED VIEW AncestorCount;
```

Давайте материализуем вообще всё!

- В большой read-only БД с большим количеством сложных запросов, возможно, так и следует сделать. Привет, OLAP!
- Если база не read-only или данных немного или запросов мало или они простые, то разумная достаточность велит воздержаться

KEEP CALM AND KEEP IT SIMPLE STUPID

Что нужно запомнить

об оптимизаторе

- Оптимизатор ваш друг

  а друзьям надо помогать

Что нужно запомнить

об оптимизаторе

- Пишите максимально точные и простые запросы
  - не делайте преждевременную оптимизацию
  - не делайте преждевременную оптимизацию
  - не делайте преждевременную оптимизацию
  - избегайте коррелирующих и скалярных подзапросов
  - не увлекайтесь заворачиванием запросов в функции
  - выбирайте только то, что нужно, сортируйте только тогда, когда нужно
  - не пишите императивный код на SQL
  - используйте CTE

Что нужно запомнить

об оптимизаторе

- Используйте средства профилирования. `EXPLAIN ANALYZE` ваш второй друг

Что нужно запомнить

об индексах

- Индексы повысят производительность, если
  - запросы часто
  - выбирают мало записей
  - из большой таблицы

Что нужно запомнить

об индексах

- Обновление индексов может стоить дорого

Что нужно запомнить

об индексах

- Используйте средства профилирования

Что нужно запомнить

об индексах

- Не делайте преждевременную оптимизацию

Что нужно запомнить

о материализации и избыточности

- Тяжёлые долгие запросы к read-only данным, индексы не помогают? Материализуйте или используйте избыточную информацию

Что нужно запомнить

о материализации и избыточности

- Данные меняются, запросы быстрые или редкие? Скорее всего, справитесь и так