Обработка и исполнение запросов в СУБД (Лекция 2)

Классические системы: оптимизация запросов в реляционных СУБД

v2

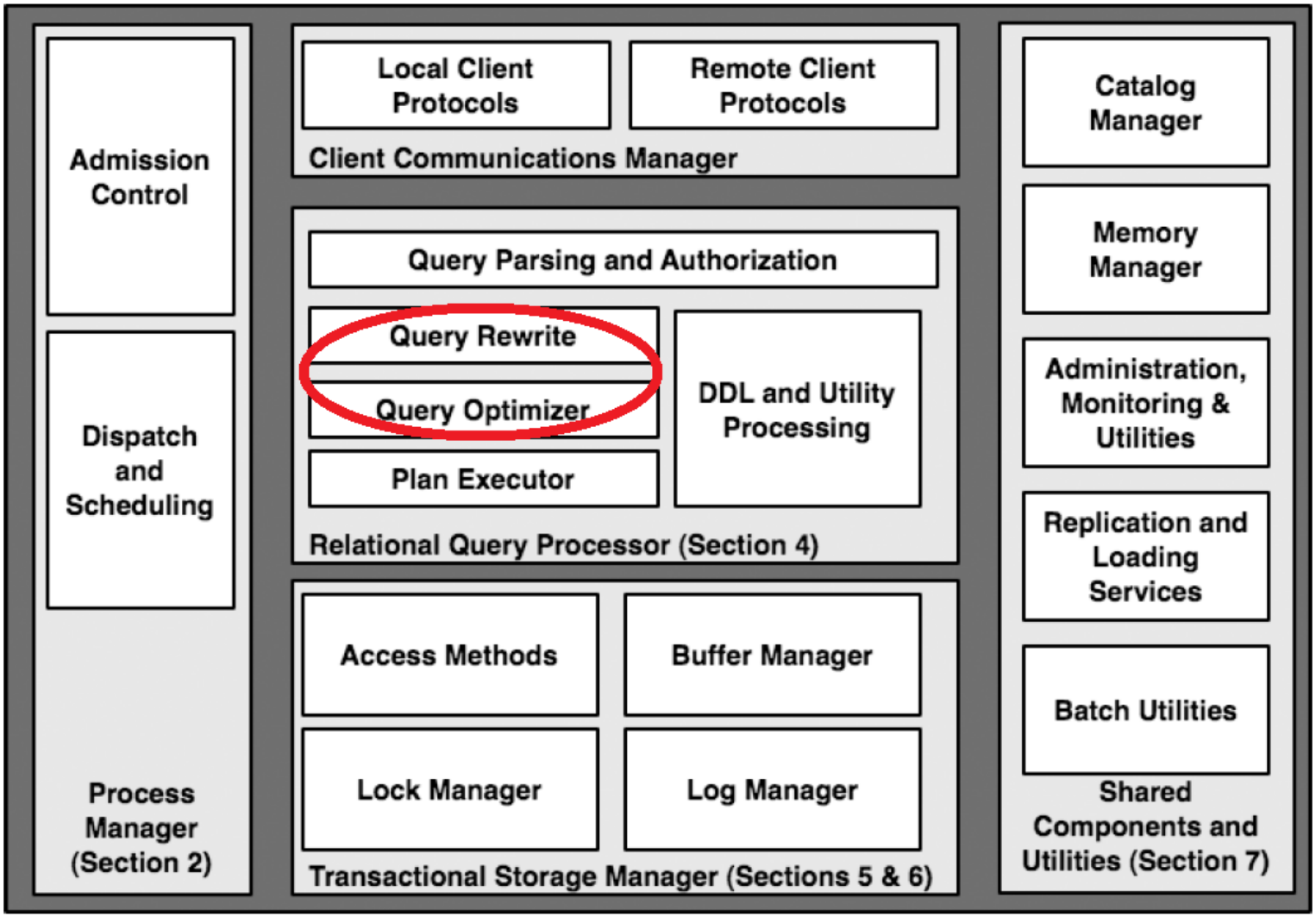
Георгий Чернышев

Академический Университет

chernishev@gmail.com

14 сентября 2016 г.

Основные компоненты классической реляционной СУБД



[1]

[1] Изображение взято из [Hellerstein et al., 2007]

Напоминание: стадии обработки запроса

- Разбор и авторизация;
- Перезапись запроса;
- Оптимизация запроса;
- Выполнение запроса.

Перезапись запроса I

Некоторые преобразования не меняющие семантики запроса, но упрощающие/удешевляющие выполнение.

Идея: пользуемся только запросом и каталогом, на сами данные не смотрим.

Раньше это всегда была отдельная фаза, сейчас не всегда, иногда её вкладывают в оптимизатор.

Основные типы преобразований I

Некоторые преобразования не меняющие семантики запроса, но упрощающие/удешевляющие выполнение. Основные типы [Hellerstein et al., 2007]:

- Перезапись представлений (исторически, основное предназначение.): раскрыть представление, найти исходные таблицы, подставить ссылки на них, перенести предикаты из представления.
- Вычисление константных выражений:
  SELECT 2 + 2 AS result, * FROM R WHERE R.x < 10+2+R.y
  $\Longrightarrow$
  SELECT 4 AS result, * FROM R WHERE R.x < 12+R.y

Основные типы преобразований II

- Перезапись предикатов из WHERE:
  - упрощение для использования подходящего access method:
    NOT Emp.Salary $>$ 1000000
    $\Longrightarrow$
    Emp.Salary $<=$ 1000000
  - упрощение для поиска конфликтующих условий:
    Emp.salary $<$ 75000 AND Emp.salary $>$ 1000000
    $\Longrightarrow$
    $\{\}$
  - распространение предикатов по транзитивности:
    R.x $<$ 10 AND R.x $=$ S.y
    $\Longrightarrow$
    AND S.y $<$ 10

Основные типы преобразований III

- Семантическая оптимизация:
  - Например, учет ограничений целостности для redundant join elimination:
    SELECT Emp.name, Emp.salary FROM Emp, Dept WHERE Emp.deptno = Dept.dno
    Если если есть FK Emp.deptno к Dept, то... $\Longrightarrow$ не надо соединять!
- Уплощение запроса:
  - Нормализация запроса (получение канонической формы)
  - Блочная структура SELECT-FROM-WHERE, раскрытие подзапросов;
  - Межблочный перенос предикатов;
  - Подготовка к распараллеливанию.

Оптимизация запроса I

Цель:

для данного запроса выбрать лучшую стратегию выполнения при условии заданных ресурсных ограничений [Neumann, 2009].

- System R [Selinger et al., 1979]:
  - Первый оптимизатор;
  - Динамическое программирование для выбора порядка соединений;
  - Придумали “interesting orders”, учет отсортированности данных;
- Оптимизатор Starburst [Haas et al., 1989]:
  - Использует правила для комбинирования физических операторов;
  - Подобна System R, сборка снизу вверх;
  - Новое внутреннее представление (Query Graph Model).

QGM: пример I

```
SELECT partno, price, order_qty
 FROM quotations Q1
 WHERE Q1.partno IN
   (SELECT partno
     FROM inventory Q3
     WHERE Q3.onhand_qty < Q1.order_qty
           AND Q3.type = 'cpu')
```

2

QGM: пример II

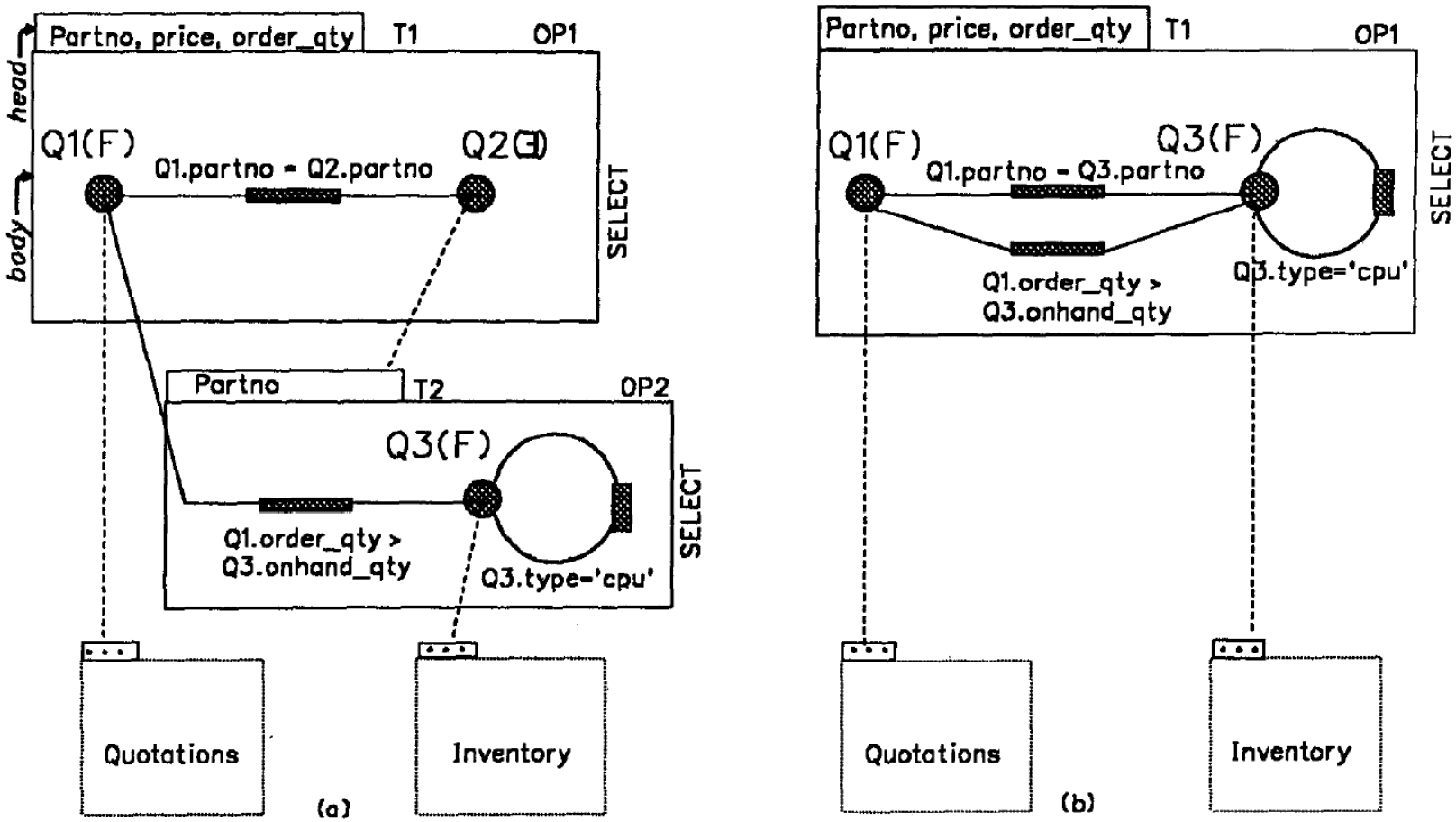


Figure 2: (a) QGM of a query (b) Same QGM after rewrite rules applied

3

[2] Изображение взято из [Haas et al., 1989]

[3] Изображение взято из [Haas et al., 1989]

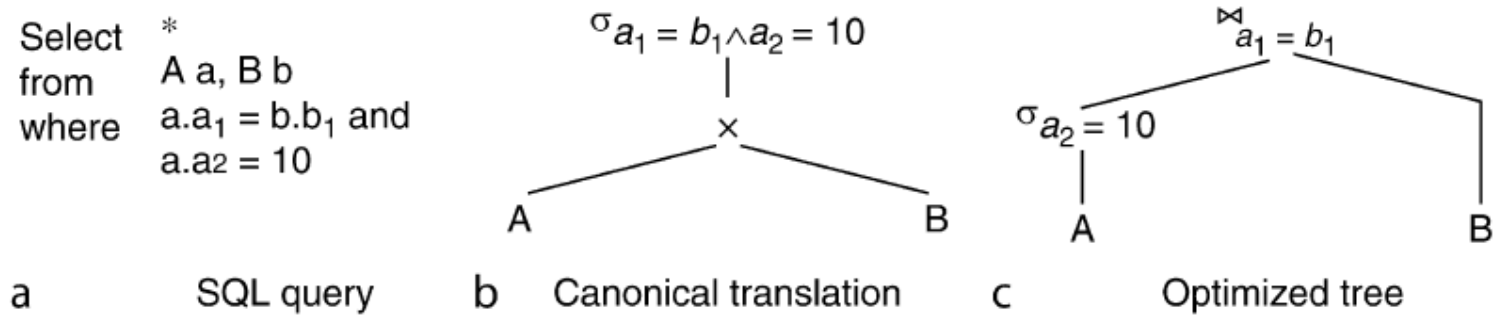
Оптимизация запроса II

Цель:

для данного запроса выбрать лучшую стратегию выполнения при условии заданных ресурсных ограничений [Neumann, 2009].

- Volcano, Cascades [Graefe and McKenna, 1993], [Graefe, 1995]:
  - Кеширующая сборка дерева сверху вниз;
  - Трансформационный оптимизатор.

Оптимизация в первом приближении: I



Query Optimization (in Relational Databases). Figure 1. Translating a SQL query. 4

Порядок:

- Текст запроса;
- Каноническое представление;
- Оптимизированный план.

[4] Изображение взято из [Neumann, 2009]

Трансформационный оптимизатор I

Алгебраические эквивалентности:

$$A \bowtie B \equiv B \bowtie A$$

$$A \bowtie (B \bowtie C) \equiv (A \bowtie B) \bowtie C$$

База для построения трансформационных оптимизаторов:

- Просто (проще) строить;
- Сложно сделать эффективный обход пространства планов $\Rightarrow$ большинство из них это эвристические оптимизаторы.

Конструктивный оптимизатор I

Альтернатива:

- Не применяет эквивалентности, но строит план из кусочков, снизу вверх;
- Может эффективно обходить пространство планов;
- Этим способом сложные эквивалентности трудно находить и применять, поэтому трансформационный шаг нужен:
  - вынесен на rewrite или пост-оптимизационную фазу.

Как строить план из кусочков? Оценивая стоимости.

Конструктивный оптимизатор II: стоимости

- Самая простая идея: количество обработанных записей
  - алгебра (реляционные операции) + статистика + селективность;
  - неточна, на практике работает плохо.
- Сложные модели:
  - вычисляем ожидаемое время выполнения;
  - учитываются шаблоны доступа к диску, стоимости вычисления “дорогих” предикатов и т.д.;
  - обычно стоимостная функция это линейная комбинация стоимостей I/O и CPU;
  - в алгебре таких параметров нет.

Селективность предиката это вероятность того, что запись ему будет удовлетворять. Высокоселективный предикат это предикат возвращающий мало записей из таблицы.

Логическая и физическая алгебра I

- Логическая алгебра:
  - концепции операторов, известных движку базы данных;
  - более абстрактна, позволяет трансформации;
- Физическая алгебра:
  - реализация этих концепций;
  - физическая алгебра “знает” параметры;
  - в этой алгебре представлен результат работы оптимизатора.

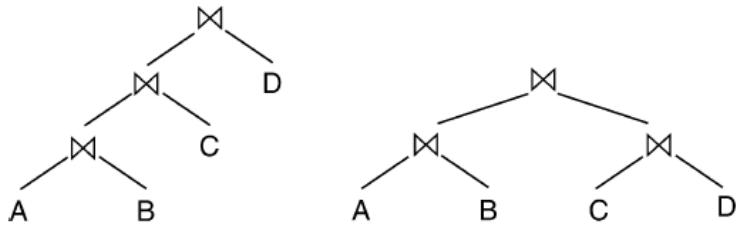
Пример: оператор внутреннего соединия – логическая алгебра; реализация оператора внутреннего соединения методом nested loop – физическая алгебра.

Дерево операторов I

Рассматриваем простой класс запросов: SPJ без подзапросов
Select-Project-Join = SELECT ... FROM ... WHERE ...

- полностью описывается
  - выборка (σ)
  - соединение/прямое произведение ($\bowtie$ / $\times$)
  - проекция ($\prod$)
- задача нахождения оптимального плана для него *NP*-трудна!
- если оставить только соединения и прямые произведения то...
  - и эта задача *NP*-трудна;
  - задача поиска оптимального бинарного дерева с n листьями;
  - количество деревьев = число Каталана: $\mathcal{C}(n-1)$;
  - $\theta(\frac{4^n}{n^{3/2}})$

Дерево операторов II



Query Optimization (in Relational Databases).
Figure 2. Left-deep Versus bushy join trees.

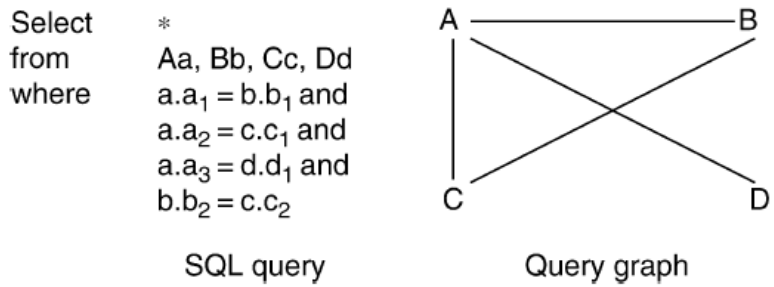
[5]

- Надо упрощать задачу дальше: линейные деревья и лево-линейные деревья;
  - проще выполнять;
  - их “всего” $n!$, проще перебирать при поиске;
  - оптимального может и не оказаться: кустистых больше — $n!\mathcal{C}(n-1)$
  - скорее всего не окажется, кустистые хорошо параллелятся;

Но! Можно избегать некоторых прямых произведений...

[5] Изображение взято из [Neumann, 2009]

Дерево операторов III



Query Optimization (in Relational Databases).
Figure 3. A query and its query graph. [6]

Не всегда можно соединять без прямых произведений. Две крайности:

- Линейный граф, тогда $O(N^3)$;
- Полный граф, тогда опять же *NP*-hard.

Обычно, реальный случай где-то посередине.

[6] Изображение взято из [Neumann, 2009]

Промежуточный итог

Стандартный подход — строим оптимизационный алгоритм исходя из:

- Оптимизируем только порядок соединений;
- Проекции и агрегации добавляем сверху, снизу жадно выборки;
- Используем граф соединений для проверки возможности соединения;
- Строим лево-линейные деревья, правая сторона соединения содержит только одного ребенка.

Алгоритм

DPsize($R = \{R_1, \ldots, R_n\}$)

for each $R_i \in R$
    dpTable[1][$\{R_i\}$] = R_i
for each $1 < s \leq n$ **ascending** // size of plan
    for each $1 \leq s_1 < s$ // size of left subplan
        for each $S_1 \in$dpTable[s_1], $S_2 \in$dpTable[$s - s_1$]
            if $S_1 \cap S_2 \neq \emptyset$ **continue**
            if $\neg(S_1$ connected to $S_2)$ **continue**
            p = dpTable[s_1][S_1]$\bowtie$dpTable[$s - s_1$][S_2]
            if dpTable[s][$S_1 \cup S_2$]=$\emptyset$ V cost(p)<cost(dpTable[s][$S_1 \cup S_2$])
                dpTable[s][$S_1 \cup S_2$] = p
return dpTable[n][$\{R_1, \ldots, R_n\}$]

7

Улучшенный алгоритм из [Selinger et al., 1979], строим кустистые деревья.

[7] Изображение взято из [Neumann, 2009]

Пояснения по алгоритму

- Динамическое программирование, даны n отношений $R_1, ..., R_n$;
- Строим таблицу где будут стоимости, инициализируем стоимостью скана;
- s – строим план по возрастанию, s_1 – левый план;
- Проходимся по парам S_1, S_2 (уже посчитанный промежуточный результат);
- Если пересекаются то нельзя соединить;
- Если нет предиката то тоже пропускаем, прямых произведений не создаем;
- Иначе, если выгодно соединять – соединяем.

Замечания по алгоритму

Алгоритм упрощенный:

- Нет выборок, проекций, но можно добавить в строки 2 и 8 (неоптимально);
- Нет выбора физического оператора соединения, добавляем в 8-10 строку;
- Sort-Merge ведет себя очень по-разному, надо хранить interesting orders;
  - это, в свою очередь, ведет к структуре physical properties — характеристика плана, вляющая на исполнение (в будущем), но не на логическую эквивалентность;
  - цель: пытаемся сохранить упорядоченность для будущих итераций алгоритма;
  - концепция доминирования: план X доминирует план Y, если имеет те же самые physical properties и его стоимость меньше;
  - в dpTable находятся наборы планов, в которых нет доминирующих пар;

Замечания по оптимизации сложных запросов

- Говоря соединение выше имелось ввиду внутреннее соединение, в случае внешних[8] всё будет сложнее – переставлять так просто нельзя;
- Агрегацию можно проталкивать вниз, под соединения, если выше находятся соединения 1 к 1;
- Бывают вложенные запросы:
  SELECT ... WHERE X IN (SELECT ...);
  - Можно бить на блоки и обрабатывать их отдельно, но часто невыгодно;
  - Вместо этого: стараться пропихнуть внешние предикаты внутрь или попытаться уплощить запрос.

[8] http://www.datamartist.com/sql-inner-join-left-outer-join-full-outer-join-examples-with-syntax-for-sc

Как обстоят дела с оптимизацией запросов в индустриальном мире [Hellerstein et al., 2007]

- Не факт что схема из System R хороша: Microsoft и Tandem не пользуются описанным подходом, используется сверху вниз, с целью, как в Cascades [Graefe, 1995];
  - но, проблема: экспоненциальны от количества таблиц по памяти и по времени;
- если много таблиц, индустриальные системы переходят на эвристические методы;
- MySQL (был) — чистая эвристика;
- PostgreSQL (был) оптимизатор.

Ссылки I


Joseph M. Hellerstein, Michael Stonebraker, and James Hamilton. Architecture of a Database System. Found. Trends databases 1, 2 (February 2007), 141–259.


Thomas Neumann. Query Optimization (in Relational Databases). Encyclopedia of Database Systems. Springer US, 2009. 2273–2278.http://dx.doi.org/10.1007/978-0-387-39940-9_293


Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management System. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.


Haas L.M., Freytag J.C., Lohman G.M., and Pirahesh H. Extensible query processing in starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 377–388.


Graefe G. The cascades framework for query optimization. Q. Bull. IEEE TC on Data Engineering, 18(3):19–29, 1995.


Graefe G. and McKenna W.J. The volcano optimizer generator: Extensibility and efficient search. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209–218.

Ссылки II


Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1998, pp. 34–43.


Ioannidis Y. Query optimization. In Handbook of Computer Science, A.B. Tucker (ed.). CRC Press, 1996.


Jarke M. and Koch J. Query optimization in database systems. ACM Comput. Surv., 16(2):111–152, 1984.


Yannis Ioannidis. 2003. The history of histograms (abridged). In Proceedings of the 29th international conference on Very large data bases - Volume 29 (VLDB '03), Johann Christoph Freytag, Peter C. Lockemann, Serge Abiteboul, Michael J. Carey, Patricia G. Selinger, and Andreas Heuer (Eds.), Vol. 29. VLDB Endowment 19-30.