Обработка и исполнение запросов в СУБД (Лекция 2)

Классические системы: оптимизация запросов в реляционных СУБД

v6

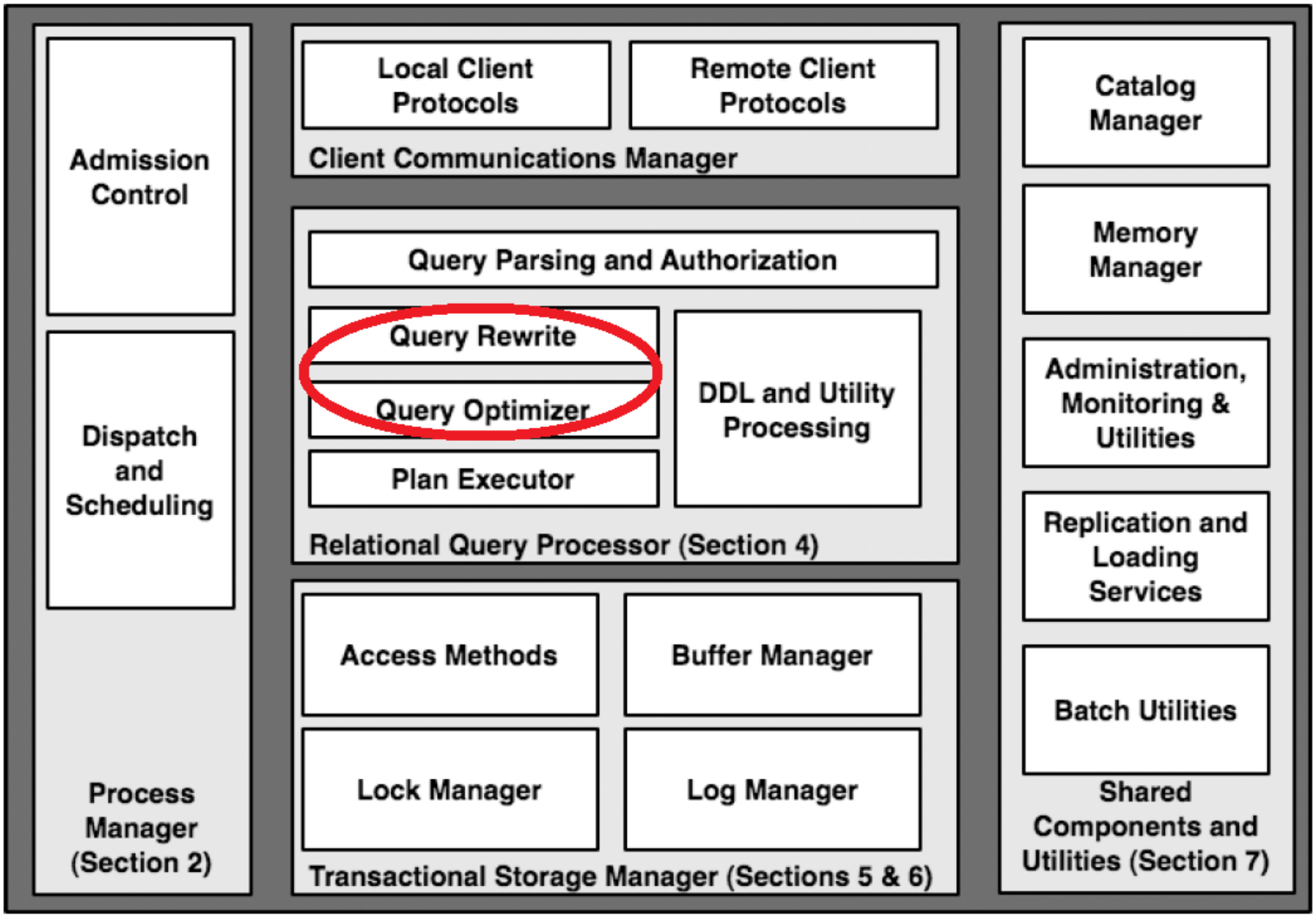
Георгий Чернышев

Высшая Школа Экономики

chernishev@gmail.com

9 сентября 2020 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:
  CREATE VIEW [Brazil Customers] AS SELECT CustomerName, ContactName, Status FROM Customers WHERE Country = 'Brazil';
  ...
  SELECT * FROM [Brazil Customers] WHERE Status = 1;
  $\Longrightarrow$
  SELECT CustomerName, ContactName, Status FROM Customers WHERE Country = 'Brazil' AND Status = 1;

Основные типы преобразований II

- 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
- 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
    $\Longrightarrow$
    {}

Основные типы преобразований III

- распространение предикатов по транзитивности:
  R.x $<$ 10 AND R.x = S.y
  $\Longrightarrow$
  AND S.y $<$ 10

- 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!
    Классический способ решения проблемы поддержки “широких” таблиц (например, в Siebel).

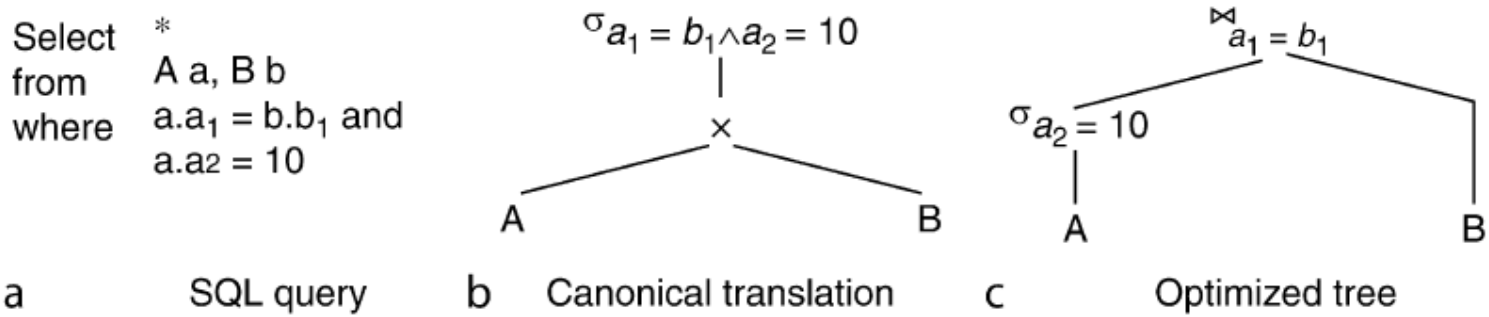
Основные типы преобразований IV

В случае вложенных запросов оптимизатор работает с блоками SELECT-FROM-WHERE, поодиночке. Почему? Чтобы уменьшить сложность работы!

- Что делают при работе с блоками на фазе рерайта:
  - Подготовка блока к оптимизации — нормализация запроса (получение канонической формы, как вариант);
  - Уплощение запроса (раскрытие подзапросов): иногда[2] подзапрос можно переписать в соединение;
  - Межблочный перенос предикатов с использование транзитивности[3];
  - Распараллеливание кореллированных подзапросов[4].

[2] https://db.apache.org/derby/docs/10.6/tuning/ctuntransform36368.html
[3] упражнение: придумать самому
[4] https://en.wikipedia.org/wiki/Correlated_subquery

Оптимизация в первом приближении: I



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

5

Порядок:

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

[5] 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 как основное средство перебора пространства планов;
- Строит план из кусочков, чаще всего 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 соединения — логическая алгебра; реализация оператора внутреннего соединения методом nested loop — физическая алгебра.

Оптимизация запроса I

Цель:

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

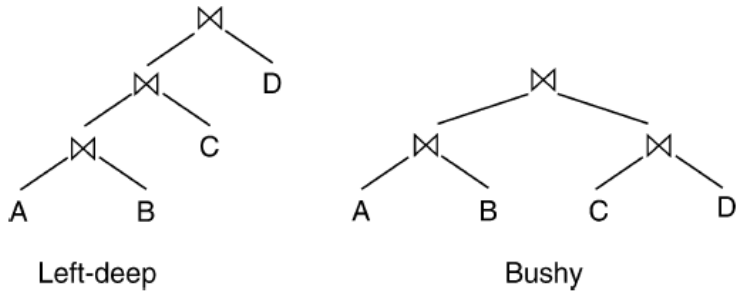
- System R [Selinger et al., 1979]:
  - Первый оптимизатор;
  - Динамическое программирование для выбора порядка соединений;
  - Придумали “interesting orders”, учет отсортированности данных;
- Оптимизатор Starburst [Haas et al., 1989]:
  - Использует правила для комбинирования физических операторов;
  - Подобна System R, сборка снизу вверх;
  - Новое внутреннее представление (Query Graph Model).
- Volcano, Cascades [Graefe and McKenna, 1993], [Graefe, 1995]:
  - Кеширующая сборка дерева сверху вниз;
  - Трансформационный оптимизатор.

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;
  - = задаче поиска оптимального бинарного дерева с n листьями;
  - а количество таких деревьев = число Каталана: $\mathcal{C}(n-1)$;
  - которое растет как $\theta(\frac{4^n}{n^{3/2}})$.

Дерево операторов II



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

6

- 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 — $n!\mathcal{C}(n-1)$
  - ... да скорее всего не окажется + кустистые хорошо параллелятся;

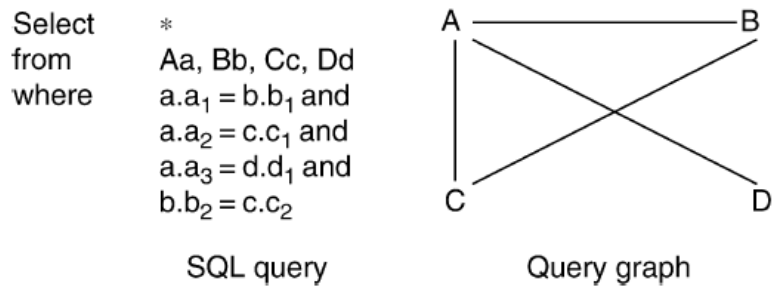
[6] Изображение взято из [Neumann, 2009]

Дерево операторов III

"Объединять" два отношения можно с помощью $\bowtie$ или же $\times$:

- Чаще всего $\bowtie$ эффективнее.
- Однако бывают случаи когда лучше сделать $\times$ (упражнение).

Когда можно делать $\bowtie$?



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

7

Не всегда можно соединять без прямых произведений (e.g. B, C, D). Две крайности:

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

Обычно, реальный случай где-то посередине.

[7] Изображение взято из [Neumann, 2009]

Промежуточный итог

Стандартный подход — строим оптимизационный алгоритм исходя из следующих положений:

- 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($R = \{R_1,...,R_n\}$)

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][$\{R_1,...,R_n\}$]

[8]

Улучшенный алгоритм из [Selinger et al., 1979], в нем мы строим кустистые деревья!

[8] Изображение взято из [Neumann, 2009]

Пример

Демонстрация.

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;

О более production-ready алгоритмах и распараллеливании

Еще информации про несовсем “игрушечные” алгоритмы:
Meister A., Saake G. (2017) Cost-Function Complexity Matters: When Does Parallel Dynamic Programming Pay Off for Join-Order Optimization. ADBIS 2017.

О распараллеливании (изображение оттуда же):

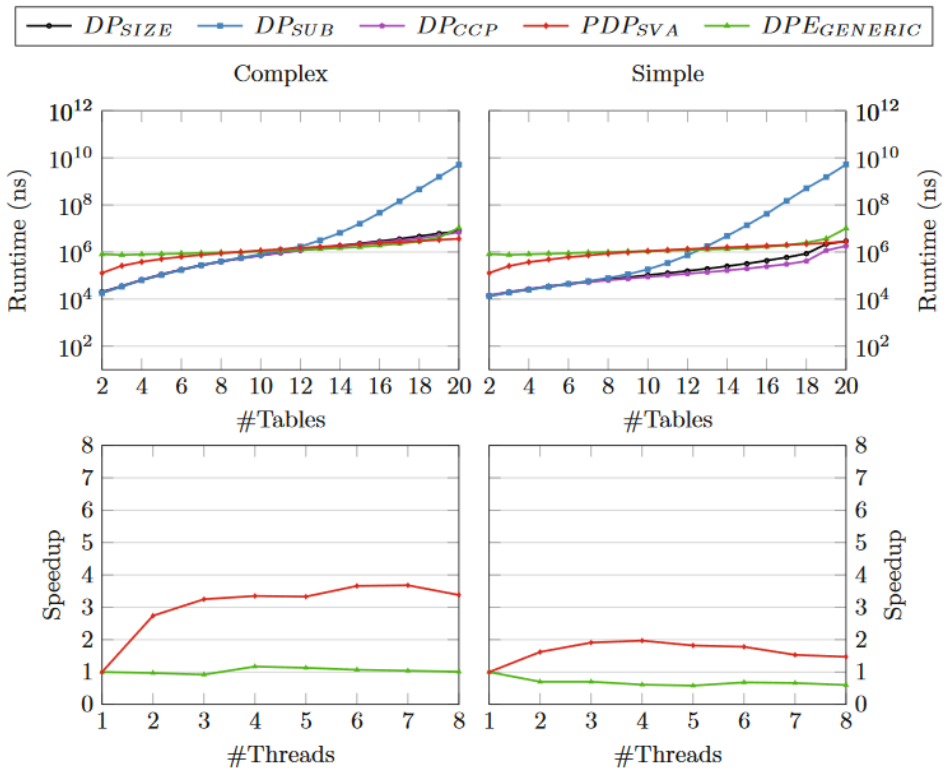


Fig. 2. Runtime and speedup of parallel dynamic programming variants for linear queries. A simple cost function leads to a reduced parallelism compared to a complex cost function.

Замечания по оптимизации сложных запросов

- Говоря соединение выше имелось ввиду внутреннее соединение, в случае внешних[9] всё будет сложнее — оно не коммутативно и не ассоциативно;
- Агрегацию можно бездумно проталкивать вниз, под соединения, если выше находятся соединения 1 к 1 (в некоторых случаях можно делать пре-аггрегацию, не смотря на тип соединения);
- 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.

[9] 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 I

- ученые в современных статьях могут делать до 500 соединений однако...
- ...если много таблиц (6–15+), индустриальные системы часто переходят на эвристические методы...
- ...и виноваты в этом методы сбора статистики!
- появление новых моделей и систем привел к всплеску интереса к тематике, очень много работ на топовых БД-конференциях от компаний

TABLE III
DISTRIBUTION OF NUMBER OF TABLE REFERENCES (A LOWER BOUND ON THE NUMBER OF JOINS) PER QUERY, AND THE PERCENTAGE OF CUSTOMERS WHOSE WORKLOAD CONTAINED QUERIES WITH THAT MANY TABLE REFERENCES. THIS DATA REFLECTS THE SMALL PERCENTAGE OF OUR OVERALL CUSTOMER BASE FOR WHICH WE HAVE DETAILED USAGE DATA.

Tables Per Query	Customers (Percent)
100+	3 %
50-100	7%
25-49	15%
15-25	37%
10-15	52%
9	49%
8	47%
7	58%
6	68%
5	81%
4	90%
3	95%
2	98%
1	100%
Total	100%

[a]Изображение взято из [Tran, 2014]

Как обстоят дела с оптимизацией запросов в индустриальном мире II

Теперь про конкретные индустриальные системы:

СУБД	год	тип	Оптимизатор
PostgreSQL	1996	D	Стоимостной оптимизатор, переупорядочивает соединения, использует генетический алгоритм для обхода[10]
MariaDB	1995	D	До 2012 года не использовала статистику вообще[11], работали на эвристиках. Сейчас умеют переупорядочивать соединения[12], еще список[13] чего умеют.
SQLite	2000	M	Оптимизируют (стоимостно) index-based nested-loop joins[14] (порядок и используемый индекс). В прошлой версии использовалась эвристика “ближайший сосед” на графе соединений, теперь “N ближайших”. Другие интересные оптимизации[15].
Sybase Adaptive Server Enterprise	1987	D	Стоимостной оптимизатор [16]: типы, порядок соединений. Есть выделенная фаза рерайта: трансформации предикатов — транзитивное замыкание, схлопывание и т.д.

Как обстоят дела с оптимизацией запросов в индустриальном мире III

Vertica	2005	D	Стоимостной оптимизатор для Data Warehouse системы: есть рерайтер, выбор порядка соединений, выбор проекций для работы (система колоночная, с репликацией)[17].
Snowflake	2012	D	Облачная DW система, нацелена на убирание "tuning knobs" (нет индексов, partition keys, только кластеризация и автотюнинг — 11 лекция)[18].
Microsoft SQL server PDW	2012?	D	Классический двухшаговый[19] (ждите 5 пятой лекции) распределенный стоимостной оптимизатор. Это не база данных, а еще и железка!
MemSQL	2013	D/M	Классический стоимостной оптимизатор с рерайтером (есть интересные!), есть подробности про распределенную оптимизацию[20], e.g. стоимости решафлинга.
GreenPlum	2005	D	Не оптимизатор, а фреймворк для создания оптимизаторов[21]. Полезно посмотреть для "суммаризации" знаний по предмету.

Как обстоят дела с оптимизацией запросов в индустриальном мире IV

[10] https://postgrespro.com/docs/postgresql/11/geqo-intro2.html

[11] https://mariadb.com/kb/en/differences-between-the-mysql-and-mariadb-query-optimizer/

[12] https://www.oreilly.com/library/view/high-performance-mysql/9780596101718/ch04.html

[13] https://mariadb.com/kb/en/query-optimizations/

[14] https://www.sqlite.org/queryplanner-ng.html

[15] https://www.sqlite.org/optoverview.html

[16] http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc00743.1502/html/queryprocessing/BHCDBECI.htm

[17] The Vertica Query Optimizer: The case for specialized query optimizers

[18] https://www.snowflake.com/blog/automatic-query-optimization-no-tuning/

[19] Query optimization in microsoft SQL server PDW

[20] The MemSQL Query Optimizer: A modern optimizer for real-time analytics in a distributed database

[21] Orca: a modular query optimizer architecture for big data

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.


N. Tran, A. Lamb, L. Shrinivas, S. Bodagala and J. Dave. The Vertica Query Optimizer: The case for specialized query optimizers. 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, 2014, pp. 1108-1119, doi: 10.1109/ICDE.2014.6816727.