NATIONAL RESEARCH
UNIVERSITY

Оптимизация и выполнение декларативных запросов

Б. Новиков
Высшая школа экономики, Санкт-Петербург

2 x 2

Школьная арифметика

- ab + ac = a(b+c)
- Стоимость: 2m+a > m+a
- Вычисление многочленов по схеме Горнера
- При условии, что стоимость умножения и сложения не зависит от значений
- Это не всегда верно: 99 a = 100 a + a

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$$

$$a_o + x(a_1 + x(a_2 + x(a_3 + \ldots)))$$

А теперь ...

Декларативные запросы

- Выразительность
- Эффективное выполнение (оптимизация)
- Манифест 3 поколения БД(1990):

  When the programmer navigates to desired data in this fashion, he is replacing the function of the query optimizer by hand-coded lower level calls.

  It has been clearly demonstrated by history that a well-written, well-tuned, optimizer can almost always do better than a programmer can do by hand.

Что нужно для оптимизации?

- Запросы компилируют в алгебраическое выражение (обычно - дерево)
- Алгебра (не так важно, какая)
- Алгебраические тождеста или правила эквивалентныэх трансформаций планы ов
- Алгоритмы для операций
- Модели стоимости для алгоритмов и для планов
- Целевая функция (функция стоимости)

Тождества в реляционной алгебре: селекция и проекция

$\sigma\,[p]\,\sigma\,[q]\,R = \sigma\,[p \wedge q]\,R$

$\pi\,[A]\,R = \pi\,[A]\,\pi\;[AB]\,R$

$\sigma[p(A)]\,\pi\,[B]\,R = \pi\,[B]\,\sigma\,[p(A)]\,R,\;$ if $A \subseteq B$

$\sigma\,[p \vee q]\,R = (\sigma\,[p]\,R) \cup (\sigma\,[q]\,R)$

$\sigma\,[p \vee \nmid q]\,R = \sigma\,[p]\,R \setminus \sigma\,[q]\,R$

$\sigma\,[p(R)]\,(R \rhd\lhd S) = (\sigma\,[p(R)]\,R) \rhd\lhd S$

$\sigma\,[p]\,(R \cup S) = (\sigma\,[p]\,R) \cup (\sigma\,[p]\,S)$

$R \rhd\lhd S = S \rhd\lhd R$

$R \cup S = S \cup R$

$R \cap S = S \cap R$

$(R \rhd\lhd S) \rhd\lhd T = R \rhd\lhd (S \rhd\lhd T)$

$(R \cup S) \cup T = R \cup (S \cup T)$

$(R \cap S) \cap T = R \cap (S \cap T)$

$R \rhd\lhd (S \cup T) = (R \rhd\lhd S) \cup (R \rhd\lhd T)$

$R \rhd\lhd (S \cap T) = (R \rhd\lhd S) \cap (R \rhd\lhd T)$

Алгоритмы и модели стоимости

- Выборка хранимых данных
  - Full scan : card (*R*)
  - Index + scan card(s(*R*))
  - Index only scan
  - . . .
- Бинарные операции (join, group, eliminate duplicates, …)
  - Nested loops card(R)*card(S), card(R)*card(S) / card(*I*)
  - Sort-merge N log N + M log M + card (output)
  - Hash M + N + card (output)

SPJ запросы

- SPJ = Select Project Join
- Операции sele3ct и project сокращают объем данных , их следует выполняясь как можно раньше
- Если эти операции применяются к промежуточным результатам, их можно выполнить “на лету” при передаче между операциями
- Сложная часть оптимизации: в каком порядке выполнять операции соединения

Стратегия подготовки запросов

- Синтаксический разбор и построение дерева
- Переписывание запроса
- Логическая оптимизацйия ?
- Выбори оптимального плана

Переписывание

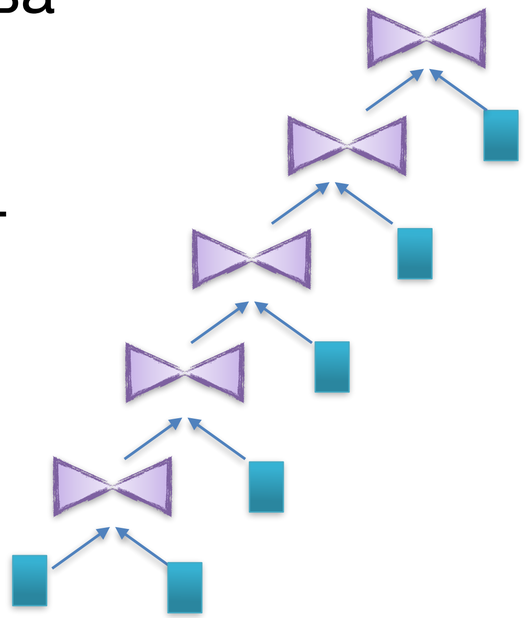
- Упрощение условий фильтрации ?
- Устранение вложенных поздапросов и выражений
- Проталкивание фильтров и проекций к листьям
-

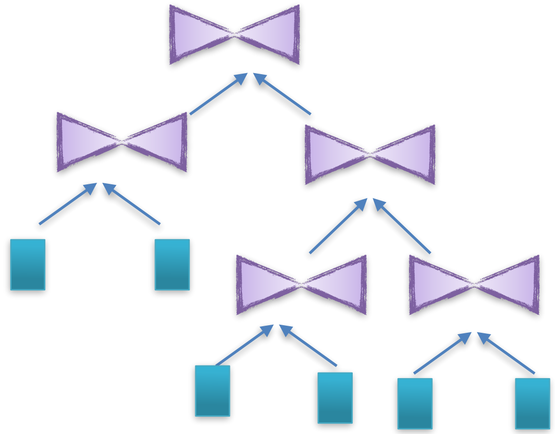
Алгоритмы оптимизацйии

- Снизу вверх, сверху вниз, трансформационные
- Точные или прилближенные

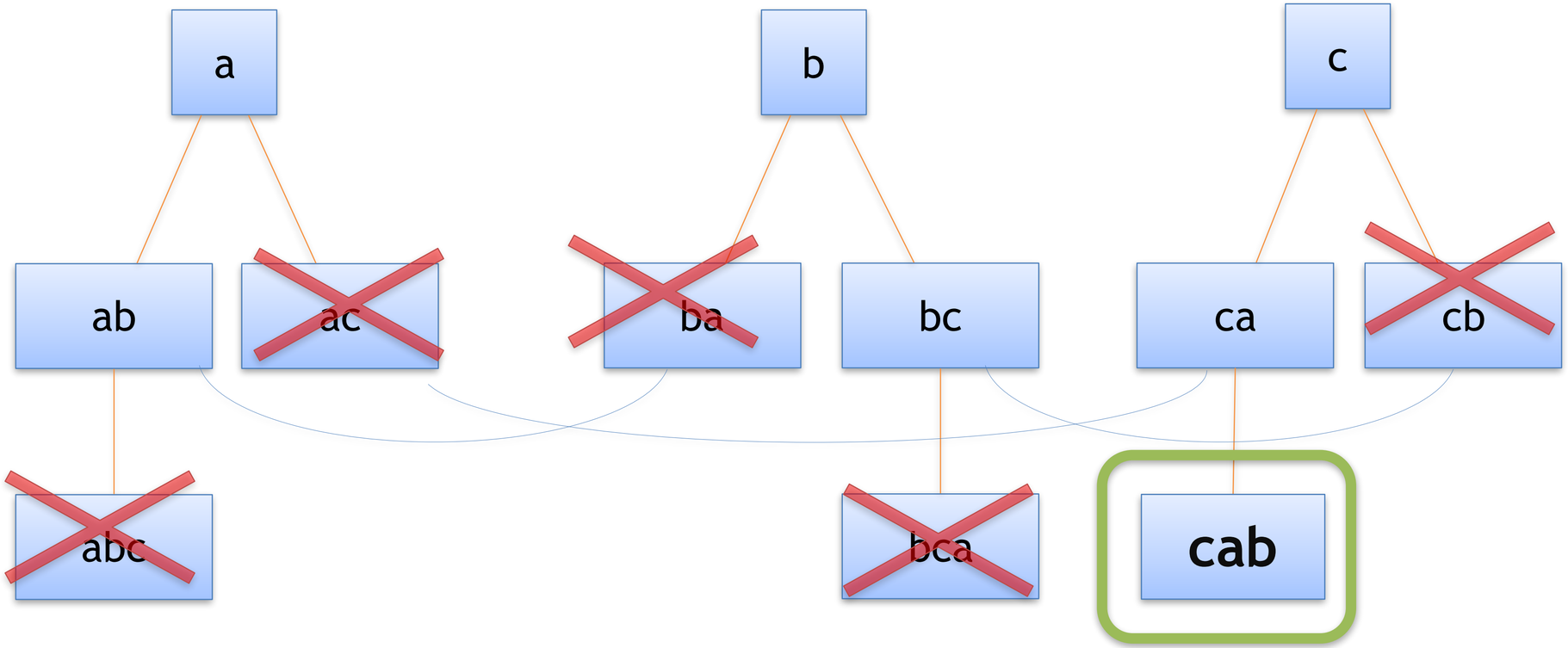
Односторонние или кустистые планы

- Сокраение размеров пространства планов
- Односторонние планы позволяют использжовать индексы
- Во многих случаях оптимальные планы являются кустистыми





Алгорим динамического программирования



Алгоритм динамического программирования

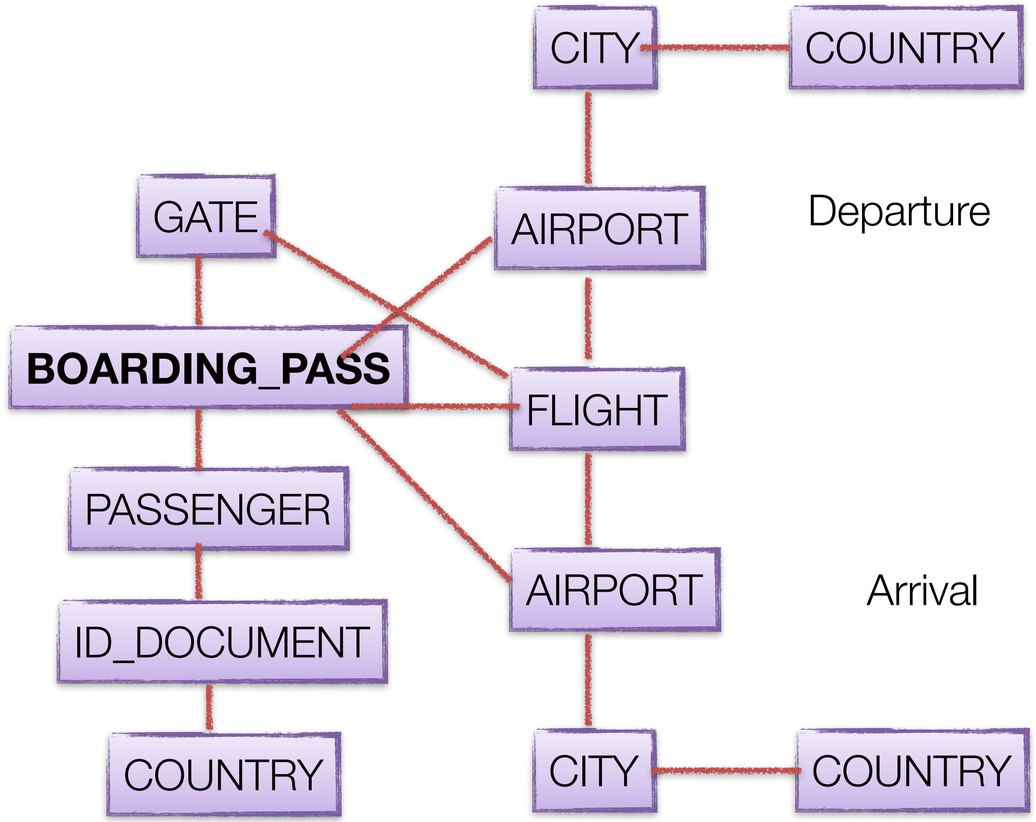
- В общем случае высокая сложность по памяти и по времени
- Сложность зависит от структуры графа запроса, для некоторых классов графов полиномиальная
- Только алгоритм NL -> полиномиальная сложность
- Односторонние или кустистые планы?

Стратегия работы оптимизатора

- Небольшие запросы - точный алгоритм (динамического программирования)
- Большие запросы - зыбкая область
  - Случайные блуждания
  - Генетичевкий алгоритм
  - Отавить, как написано

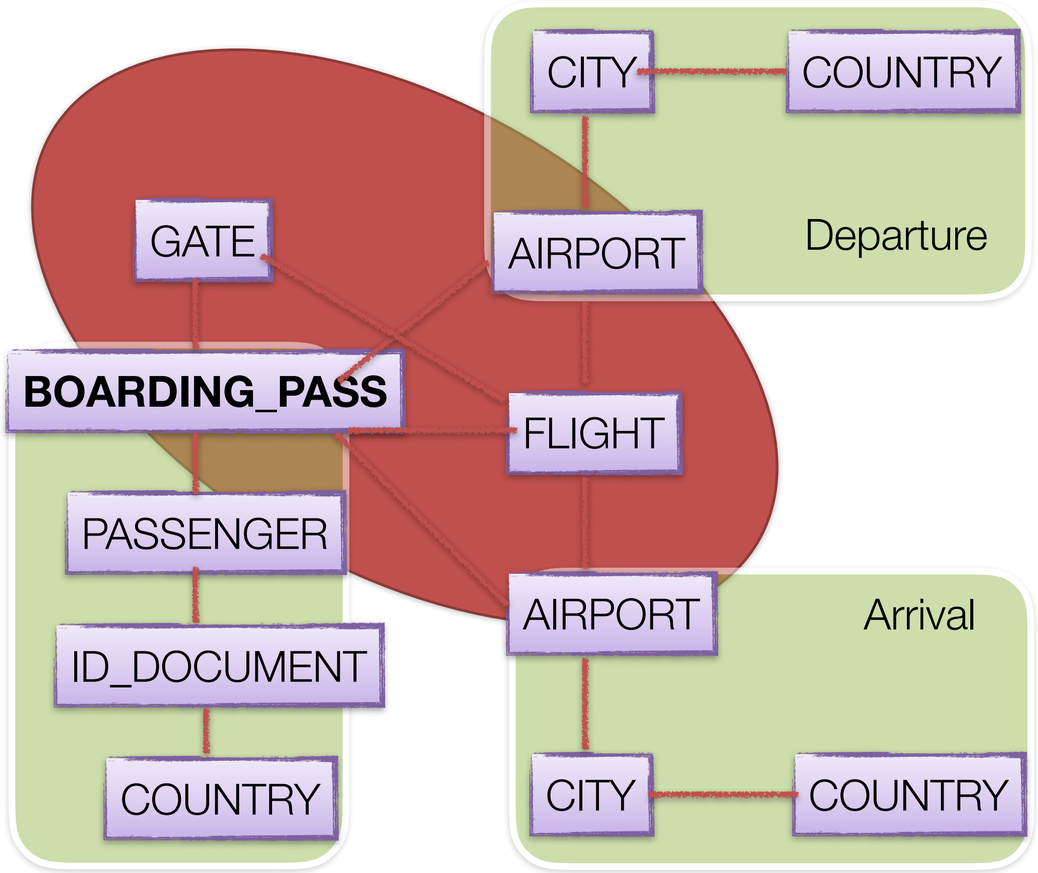
Граф запроса

```
SELECT …

FROM boarding_pass b

JOIN flight f

  ON b.flight_id = f.flight_id

JOIN gate g

  ON g.flight_id=f.flight_id and b.flight_id=g.flight_id

JOIN airport dep_a

  ON dep_a.code=f.dep_code

      AND b.dep_code = dep_a.code

JOIN airport arr_a

  ON arr_a.code=f.arr_code

      AND b.arr_code = arr_a.code

      …
```



Динамическое программирование с учетом графа запросов

- Исключается рассмотрение подпланов, которые не могу появиться в полном плане
- Сложностть алгоритма зависит не от размеров запроса, а от количества клик
- Для линейный частей алгорим полиномиален



Приближенные алгоритмы "снизу вверх"

- Жадный алгоритьм:
  - на каждом шаге выбирается один лучший частичный план
- Итеративное динамическое программирование (Коссман, 2000)
  - Чередование динамического программирования и жадного

Планы не всегда оптимальны

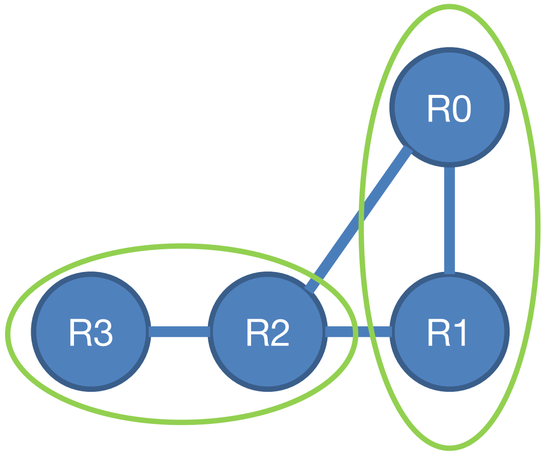
- Минимизируется оценка стоимости, но не фактическая стоимость
- Высокая сложность делает невозможной точную оптимизацию
- Модели стоимости не могут быть точными
- Оценки статистических характеристик (размеров) промежуточных результатов неточны

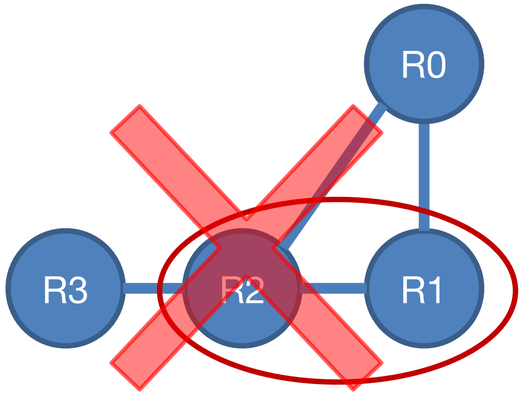
Новые идеи

Перечисление планов «сверху вниз»

G. Moerkotte (2014)

- Гиперграф запроса:
  - Вершины – таблицы
  - Дуги – условия соединения
- несимметричность операций (outer join)





Оценки кардинальности

VLDB 2016 Статья группы из Мюнхена

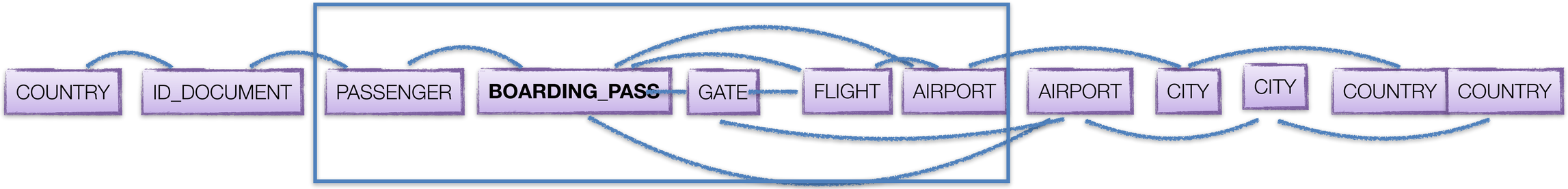
- Анализ качества оптимизаторов промышенных систем
- Никакие статистики, кроме кардинальности, для оценки промежуточных результатов не используются

Основные результаты

- Все оптимизаторы ошибаются в оценках, часто на порядки
- Внедрение точных оценок кардинальности существенно улучшает качество планов
- Качество функции стоимости не так сущсственно влияет на качество планов

Приближенный алгорим динамического программирования: линолеум еаризацйия

1. Линеаризация графа запроса
   - Линеаризация - это односторонний план без выбора алгорима (полиномиально)
2. Динамическое программирование с кустистыми планами, только подпланах-отрезках
   - Квадратичная сложность по числу операций соединения
   - Применим для запросов, содержащих до 100 соединений



Адаптивное динамическое программирование (SIGMOD 2018 та же группа из Мюнхена)

1. Оценка сложности запроса

2. Для малых запросов используется ДП на графах

3. Средние запросы (до 100) - линеаризайция

4. Большие запросы: итеративное ДП с линеаризацией

Возможна обработка запросов, содержащих тысячи операций соединения

memSQL

- Перечисление сверху вниз
- Переписывание подзапросов
- Раздельная оптимизация подзапросов
- Эвристики для получения отсекающих оценок при исчерпывающем поиске

Переписывание подзапросов

- Проталкивание фильтров в подзапросы
- Проталикивание join, если это не ухудшает стоимость

```
Select …
(select …)
From (select …) …
Where  … in (select ...) …
```

Эвристики для отсечений при исчерпывающем поиске

- На основе эвристик генерируется несколько планов хорошего качества, не обязательно оптимальных
- Оценки стоимости этих планов используются для отсечений при исчерпывающей оптимизации

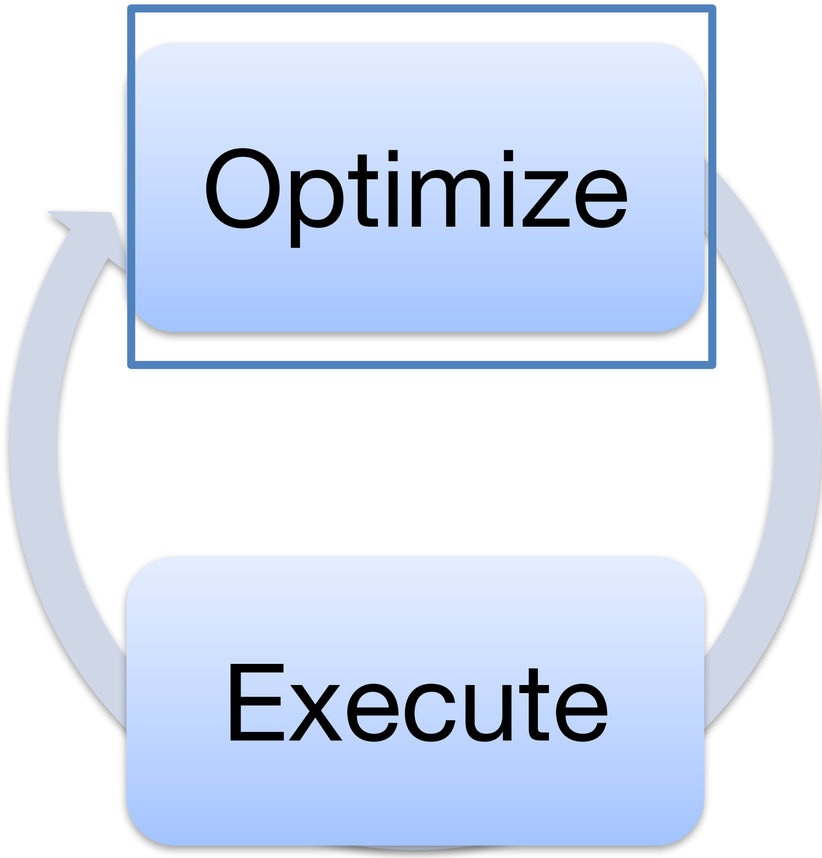
Целочисленное линейное программирование

- Применение обычных оптимизационных пакетов (исслед. Доперацій)
- Переменные (0/1) для операций соединения
- 6 видов ограничений для условий, порядка выполнения операций и др.
- Линейная функция стоимости
- Оптимизируется примерно 50 соединений за несколько минут

Адаптивная оптимизация

Решение: адаптивная оптимизация?

- Повторная оптимизация во время выполнения, когда имеются уточненные оценки селективности и кардинальности
  - Маршрутизация для операций без состояния
  - Точки материализации для планов общего вида
- Почти не спользуется в промышленных системах



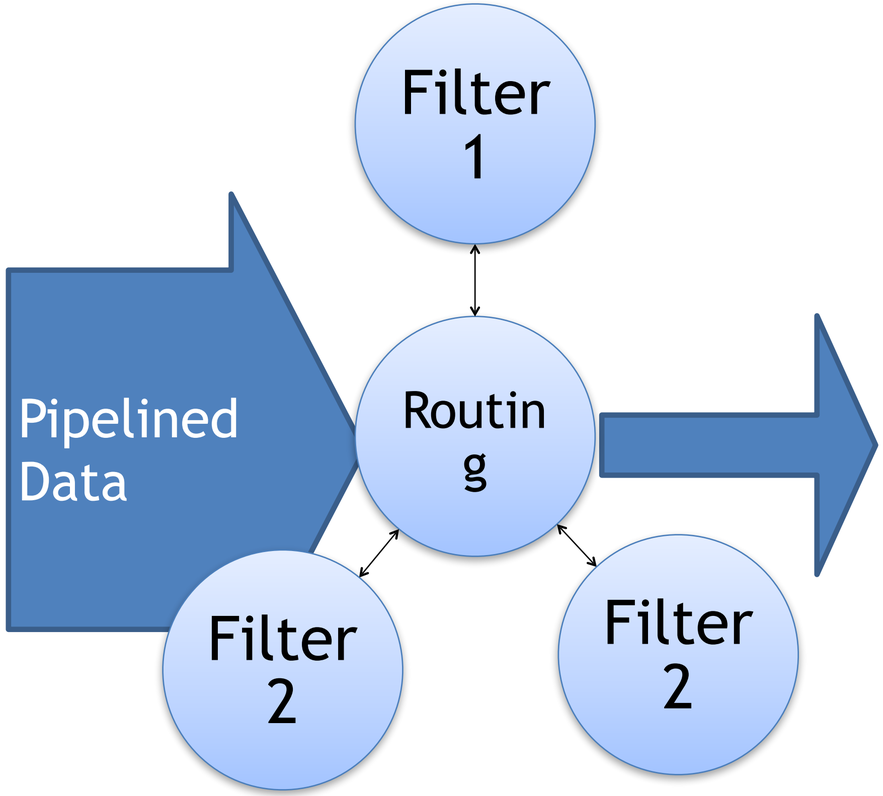
Свойства алгоритмов операций

	Без состояния	С состоянием
Неблокирующие	Filter Project	Nested Loops Symmetric Hach
Блокирующие		Hash join Sort Aggregate

Маршрутизация неблокирующих операций

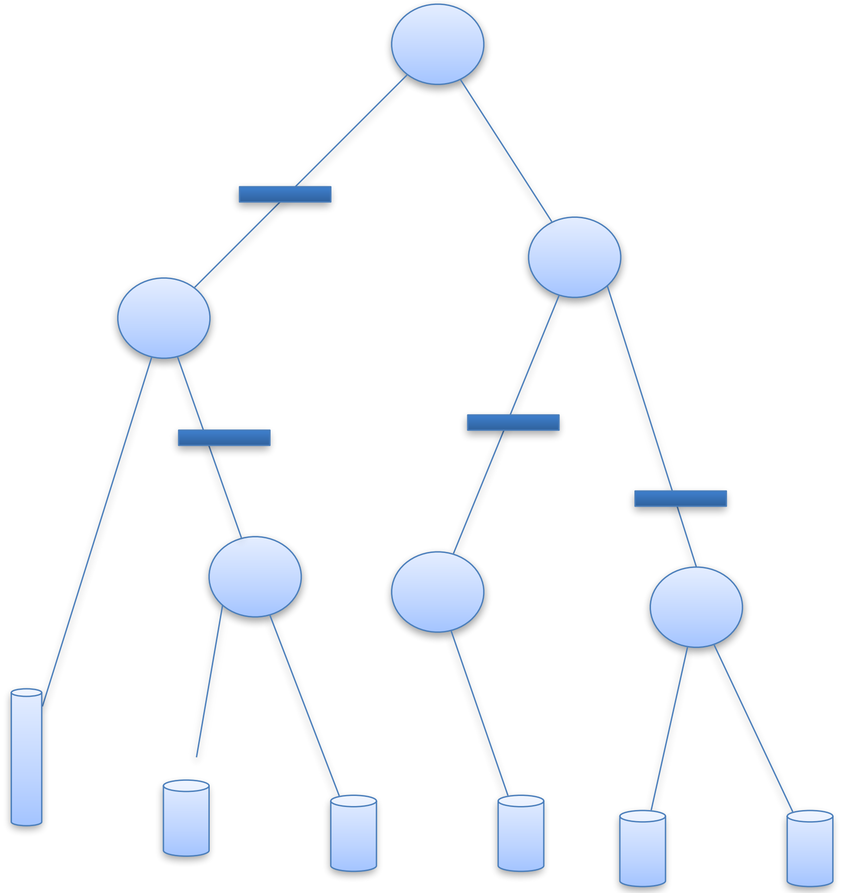
- Каждый объект направляется на еще не пройденный фильтр с лучшей селективностью
- Оценки селективности динамически перевычисляются

```
Select *
from a_table
Where attr1=value1
and attr2=valeu2
and attr3 >= value3
 …
```



Точки материализации

- Работает для операций с состояниями
- Сохранение промежуточных результатов в точке материализации
- Повторная оптимизация оставшейчя части запроса
- Часть уж евыполненной работы может оказаться ненужной



Оптимизация без SQL

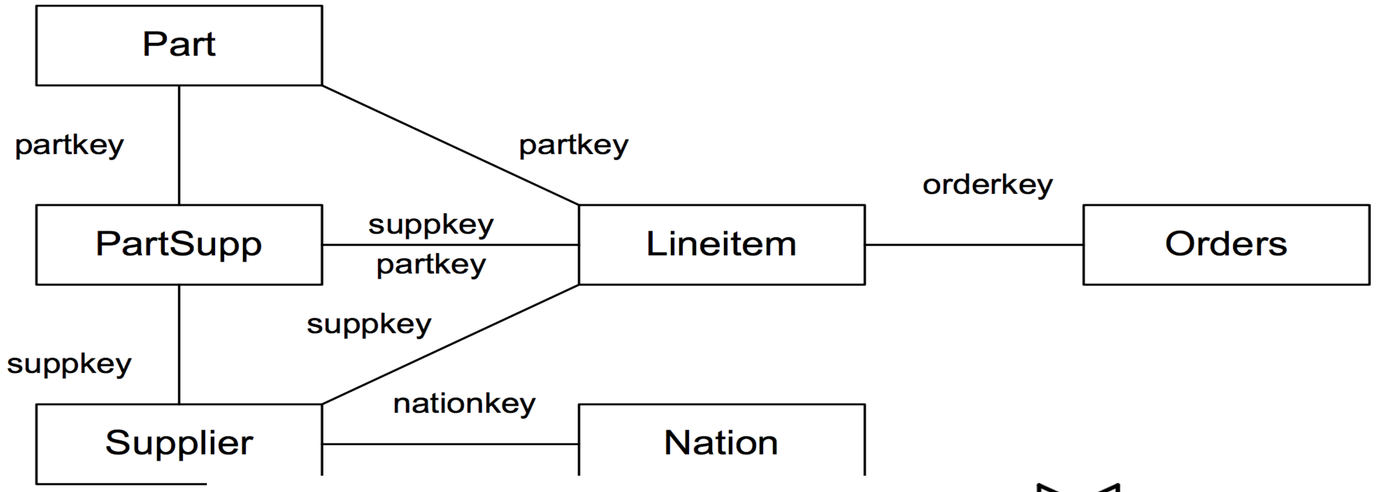
Оптимизация для MapReduce

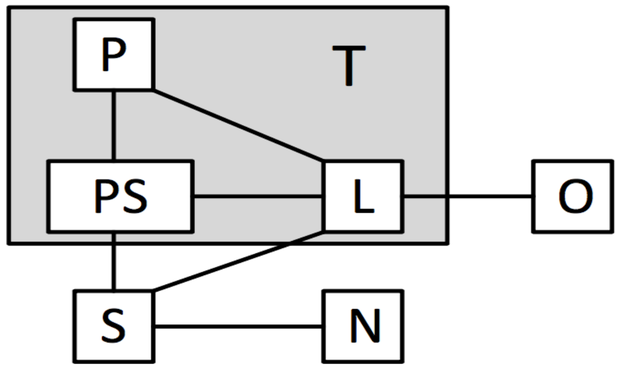
- Задача в терминах баз данных:
  - Map = filter + unnest
  - Reduce = GROUP BY
- Оптимизация внутри пользовательских функций map и reduce невозможна
- Реструктуризация задач
  - Объединение фаз, обрабатывающих однотиный вход

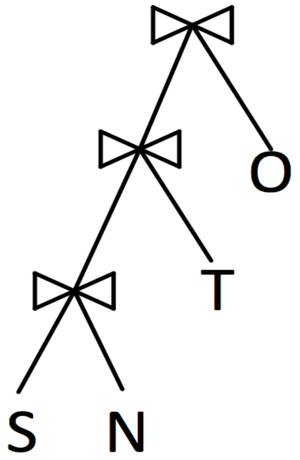
Алгоритмы для операции соединения

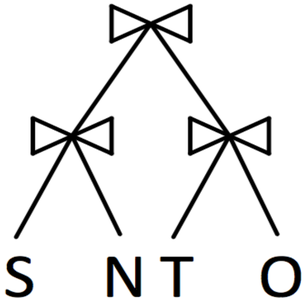
- Map side
- Reduce side
- Both map and reducs
- Multi-way joins

Оптимизатор AQUA









Оптимизатор Stubby

- Аннотации
  - Данные (разбиение, упорядочение, файлы)
  - Опенации (схема, фильтры, …)
  - Run-Time (статисника, оценки стоимости)

- Трансформации задач на основе аннотаций
  - Совмещение map или reduce при совпадении схемы
  - Совмедение map и reduce
  - …

Поддержка декларативных средств в масштабируемых системах

- SCOPE, Asterix, Spark, …
- Развитый набор операций
  - Обработка (фильтры, соединения и т.д.)
  - Рассылка
- Альтернативные методы передачи данных между операциями
- Потенциал для оптимизации
- Модели стоимости могут быть проблемой

Заключение: может ли SQL выжить?

- Языки запросов почти не используются для приложений класса OLTP
- Пока SQL используется для OLAP, но:
- Data Science не знает ни о чем, кроме CSV