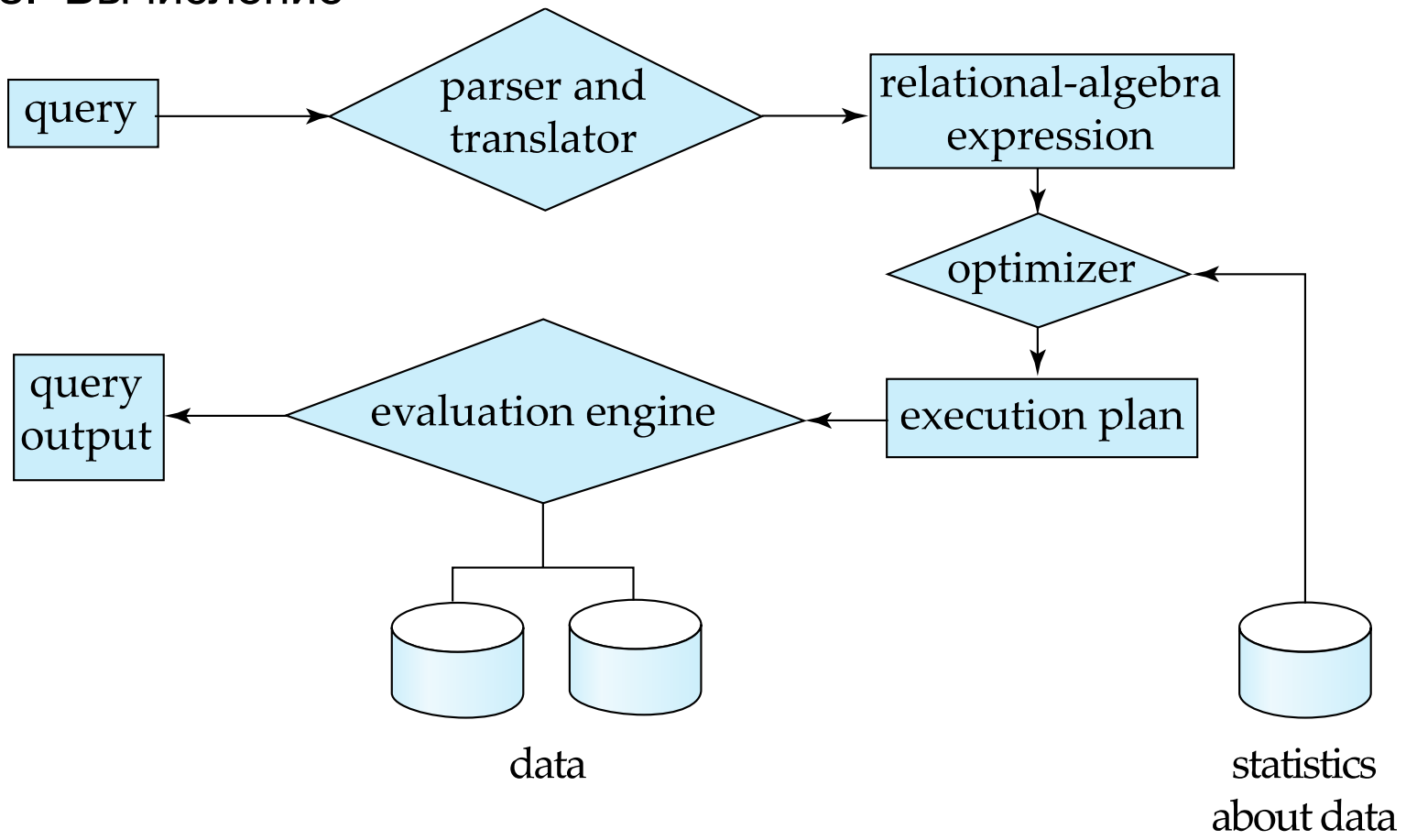
Обработка запросов

Обработка запросов

- Обзор
- Меры стоимости запросов
- Операция выборки
- Сортировка
- Операция соединения
- Другие операции
- Вычисления выражений

Основные шаги обработки запросов

1. Парсинг и трансляция
2. Оптимизация
3. Вычисление



Основные шаги обработки запросов

- Парсинг и трансляция
  - Осуществляется трансляция запроса в его внутреннюю форму. Затем осуществляется трансляция в реляционную алгебру.
  - Парсер осуществляет проверку синтаксиса и проверку отношений
- Вычисление
  - Механизм выполнения запросов принимает на вход план запросов, выполняет данный план, и возвращает ответ для запроса.

Основные шаги обработки запросов: Оптимизация

- Выражения реляционной алгебры могу иметь эквивалентные выражения

```
select student_salary
  from students
 where student_salary < 75000
```

$$\Pi_{student\_salary}(\sigma_{student\_salary<75000}(students))$$

$$\sigma_{student\_salary<75000}(\Pi_{student\_salary}(students))$$

- Каждая операция реляционной алгебры может быть вычислена разными алгоритмами
- Связанное выражение, уточняющее детальную стратегию выполнения называется планом вычислений (evaluation-plan)
  - Используется индекс на *salary*, чтобы найти всех преподавателей с зп < 75000,
  - Или производится полный скан(обход) отношения и убирает инструкторов с зп ≥ 75000

Основные шаги обработки запросов: Оптимизация

- **Оптимизация запросов**: Из всех эквивалентных планов выполнения выбирается с меньшей стоимостью.
  - Стоимость вычисляется с использованием статистической информации из каталога базы данных.
    - Например, число кортежей в каждом отношении, размер кортежей и т.д.
- Далее мы изучим
  - Как измерить стоимость запроса
  - Алгоритмы для оценки операций реляционной алгебры
  - Как объединять алгоритмы для базовых операций в общее выражение
  - Как оптимизировать запросы и находить план выполнения с минимальной стоимостью

Меры стоимости запроса

- Многие факторы участвуют при оценке времени
  - *Доступ к диску, CPU*, и сетевые коммуникации
- Стоимость может быть вычислена на основе
  - **времени ответа**, т.е. общее время ответа на запросы
  - общее **потребление ресурсов**
- Будет рассматривать метрику по потреблению ресурсов
  - Время ответа сложнее вычислить, уменьшение потребления ресурсов – хорошая метрика в общей (shared) БД
- Мы игнорируем стоимость CPU для простоты
  - Реальные системы используют стоимость CPU для оценки
  - Стоимость сети рассматривается для параллельных систем
- Опишем, как вычислить стоимость для каждой операции
  - Стоимость записи на диск игнорируем для простоты

Меры стоимости запроса

- Стоимость доступа на диск может быть рассчитана как:
  - Число поисков * среднее время поиска
  - Число блоков чтения * среднее время чтения блока
  - Число блоков записи * среднее время записи блока
- Для простоты учитываем **число трансфера блока** *с диска* *и* **число поисков** как меры стоимости
  - t_T – время для трансфера одного блока
    - Для простоты предполагаем, что стоимость записи соответствует стоимости чтения
  - t_S – время для одного поиска
  - Стоимость для b трансферов блоков и S поисков
    $b * t_T + S * t_S$
- t_S и t_T зависят от того, где хранятся данные; с 4 KB блоками:
  - HDD: t_S = 4 msec и t_T =0.1 msec
  - SSD: : t_S = 20-90 microsec и t_T = 2-10 microsec для 4KB

Меры стоимости запроса

- Требуемые данные могут быть уже в буфере и, следовательно, не потребуется I/O
  - Но это сложно учитывать при оценке
- Часть алгоритмов может уменьшать дисковое IO, если есть дополнительное место в буфере
  - Количество реально доступной памяти в буфере зависит от параллельных запросов и процессов ОС в момент выполнения
- Худший вариант предполагает, что изначально нет данных в буфере и только минимальное количество памяти может быть выделено для операции
  - Более оптимистичные сценарии обычно предполагают наличие места

Операция выборки

- **Скан (обход) файла**
- Алгоритм **A1** (**линейный поиск**). Обходит каждый блок файла и тестирует все записи на условие отбора.
  - Стоимость выполнения= b_r трансферов блока + 1 поиск
    - b_r определяет число блоков записей в отношении r
  - Если выборка по ключевом атрибуте, то можно остановиться при нахождении
    - Стоимость = (b_r /2) трансферов блока + 1 поиск
  - Линейный поиск может применяться независимо от
    - Условия выборки
    - Сортировки записей в файле
    - Доступности индексов
- Замечание: для бинарного поиск обычно нет смысла, так как данные не хранятся последовательно
  - И бинарный поиск требует большее число поисков

Выборка при использовании индексов

- **Обход индекса** – алгоритмы поиска, использующие индекс
  - Условие выборки должно содержать ключ поиска в индексе.
- **A2** (**кластерный индекс, равенство по ключу**). Вывод: одна запись, которая подходит по равенству по ключу
  - *Стоимость* = $(h_i + 1) * (t_T + t_S)$
- **A3** (**кластерный индекс, равенство не по ключу**). Вывод: несколько записей.
  - Записи должны быть на последовательных блоках
    - Пусть b = число блоков, содержащих совпадающие записи
  - *Стоимость*= $h_i * (t_T + t_S) + t_S + t_T * b$

Выборка при использовании индексов

- **A4** (**вторичный индекс, равенство (не) по ключу**).
  - Возвращает одну запись, если ключ поиска уникален
    - *Стоимость*= $(h_i + 1) * (t_T + t_S)$
  - Возвращает несколько записей, если ключ поиска м.б. не уникален
    - Каждая из *n* совпадающих записей может быть в разных блоках
    - Стоимость = $(h_i + n) * (t_T + t_S)$
      - Может быть очень дорогой операцией

Выборки, включающие сравнения

- Можно разработать выборки формы $\sigma_{A \le V}(r)$ или $\sigma_{A \ge V}(r)$ через
  - линейный обход файла,
  - использование индексов следующими способами
- **A5** (**кластерный индекс, сравнение**). (Отношение отсортировано по A)
  - Для $\sigma_{A \ge V}(r)$ используется индекс для нахождения первого кортежа $\ge v$ и далее последовательный обход отношения
  - Для $\sigma_{A \le V}(r)$ просто последовательный обход отношения до момента, как кортеж> *v;*

Выборки, включающие сравнения

- Можно разработать выборки формы $\sigma_{A \le V}(r)$ или $\sigma_{A \ge V}(r)$ через
  - линейный обход файла,
  - использование индексов следующими способами
- **A6** (**кластерный индекс, сравнение**).
  - Для $\sigma_{A \ge V}(r)$ используется индекс для нахождения первой записи $\ge v$ и осуществлять последовательный поиск для нахождения указателя на записи
  - Для $\sigma_{A \le V}(r)$ обходить корневые страницы до условия $> v$
  - Если условие не соответствует порядку A, то может потребоваться одно чтение на запись

Реализация сложных выборок

- **Конъюнкция:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$
- **A7** (**конъюнктивная выборка с использованием одиночного индекса**).
  - Все условия по индексу проверяем
  - Остальные проверки в буфере памяти
- **A8** (**конъктивная выборка с использованием сложного индекса**.
  - Используется подходящий сложный составной индекс.
- **A9** (**конъюнктивная выборка с пересечением идентификаторов***).*
  - Пересекаем указатели, подходящие для индексов
  - Остальные проверки в памяти

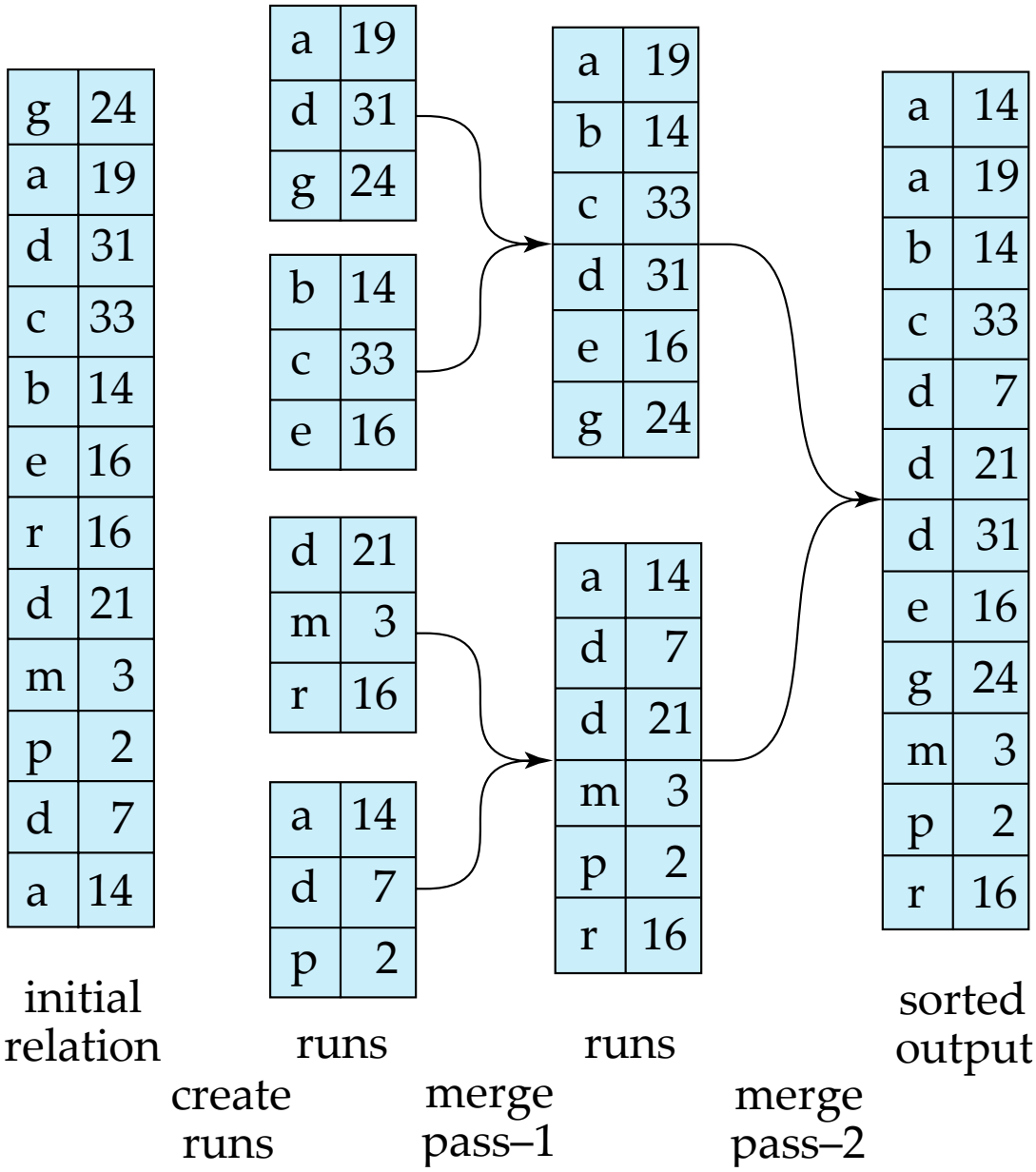
Алгоритмы для сложных выборок

- **Дизъюнкция:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.
- **A10** (**дизъюнктивная выборка по объединению идентификаторов**).
  - Применима, если все условия имеют индексы
    - В другом случае линейный поиск
- **Отрицание:** $\sigma_{\neg\theta}(r)$
  - Линейный поиск по файлу
  - Если не много записей $\neg\theta$, и индекс применим по θ
    - Найти подходящие записи по индексу

Сортировка

- Можно построить индекс по отношению, и использовать индекс для чтения в том же порядке. Для каждого кортежа может потребоваться только одно чтение.
- Для отношений, помещающихся в памяти, можно использовать стандартные сортировки, например quicksort. Для не помещающихся **external sort-merge**.

Пример: Sort-Merge



Стоимость сортировки

Пусть b_r - число блоков в отношение r. Первая стадия – чтение каждого блока отношения и его перезапись - $2b_r$. Изначальное число пробегов - $\frac{b_r}{M}$

Для эффективного слияния требуется b_b буферных блоков для каждого входа пробега и его выхода.

Поэтому $\frac{M}{b_b} - 1$ пробегов сливаются в каждый момент слияния.

Следовательно, общая величина для слияний - $\log_{\frac{M}{b_b - 1}}(\frac{b_r}{M})$

В каждом таком слиянии происходит одно чтение каждого блока и одна запись каждого блока (кроме последнего, запись не нужна)

В итоге общее число переносов будет

$$b_r(2[\log_{\frac{M}{b_b}-1}\left(\frac{b_r}{M}\right) + 1)$$

Стоимость сортировки

Для подсчета числа поисков используется следующее предположение
Каждому слиянию потребуется $\frac{b_r}{b_b}$ поисков, аналогично для записи, кроме последнего слияния
В итоге получаем

$$\frac{2b_r}{M} + (b_r/b_b)\,(2[\log_{\frac{M}{b_b}-1}\left(\frac{b_r}{M}\right) - 1)$$

Соединения

Базовый алгоритм для соединения $r \bowtie_\theta s$:

for каждого кортежа t_r в r:
    for каждого кортежа t_s в s:
        сравнить пару (t_r, t_s) на θ
            если true, то добавить $t_r \cdot t_s$ в результат

R – внешнее отношение
S – внутреннее отношение

Вложенные циклы (оценка)

Для применения данного алгоритма не требуются никакие дополнительные структуры.
Общее число пар - $n_r \cdot n_s$,

где n – число кортежей

В худшем случае, если буфер может содержать только 1 блок каждого отношения, то потребуется $n_r * b_s + b_r$ переносов блоков. В лучшем, $b_s + b_r$

Если одно из отношений помещается в память, то его лучше использовать как внутренне отношение, так как в таком случае чтение будет только одно. В таком случае будет $b_s + b_r$ переносов и всего 2 поиска.

Вложенные циклы (блочный алгоритм)

В случае, когда буфер слишком мал, чтобы содержать полностью отношение, можно ориентироваться не на кортежи, а на блоки.

Алгоритм

for каждый блок B_r отношения r:
    for каждый блок B_s отношения s:
        for каждый кортеж t_r в B_r:
            for каждый блок t_s в B_s:
                проверить пару (t_r, t_s) на θ
                    если $true$, то добавить (t_r, t_s) в результат

Оценка (блочный вариант)

В худшем случае будет $b_r * b_s + b_r$ переносов. Каждый обход внутреннего отношения – 1 поиск, и поиск внешнего отношения – 1 поиск на блок. Поэтому всего $2 * b_r$ поисков. Более эффективно использовать меньшее отношение как внешнее отношение, если они оба не помещаются в память.

Блочный вариант (оптимизация)

В блочном варианте для внешнего отношения можно использовать наибольший объем, который есть в памяти.

Если в памяти M блоков, то M-2 блоков считывается для внешнего отношения.

Уменьшается количество поисков с b_r до $\frac{b_r}{M-2}$.

Общее число переносов блоков = $\frac{b_r}{M-2} * b_s + b_r$

Общее число поисков блоков = $2 * \frac{b_r}{M-2}$

Вложенные циклы (индекс)

Если задан индекс для внутреннего отношения, то работа с таблицей может быть заменена на работу с индексом. Проверка соответствия соединения проверяется на уровне индекса, а не таблицы.

Стоимость $b_r(t_T + t_S) + n_r * c$
c – стоимость работы с индексом и нахождения всех кортежей s для одного кортежа r

Соединение слиянием (Merge-Join)

Предполагается, что оба отношения отсортированные по пересечению их атрибутов соединения.

pr := адрес первого кортежа r
ps := адрес первого кортежа s
while (ps != null and pr != null)
  t_s := кортеж указателя ps
  S_s := $\{t_s\}$
  ps := следующий кортеж а s
  $done := false$
  while $(not\ done\ and\ ps\ != null)$
    t'_s := кортеж указателя ps
    if (t'_s[JoinAttrs] = $t_s[JoinAttrs]$
      $S_s := S_S \cup \{t'_s\}$
      ps := следующий кортеж в s
    else
      $done := true$
   t_r := кортеж указателя pr
  while $(pr\ != null\ and\ t_r[JoinAttr\ ] < t_s[JoinAttrs])$
    pr := следующий кортеж в r
    t_r := кортеж указателя pr
  while $(pr\ != null\ and\ t_r[JoinAttr\ ] = t_s[JoinAttrs])$
    pr := следующий кортеж в r
    $for\ each\ t_s$ в S_s
      Добавить $t_s \bowtie t_s$ в результат.
    pr := следующий кортеж в r
    t_r := кортеж указателя pr

Merge Join

Каждый блок прочитан будет только однажды при условии, что все кортежи помещаются в памяти

Стоимость переноса = $\mathrm{b_r} + b_S$

Стоимость поиска = $\frac{\mathrm{b_r}}{b_b} + \frac{b_S}{b_b}$

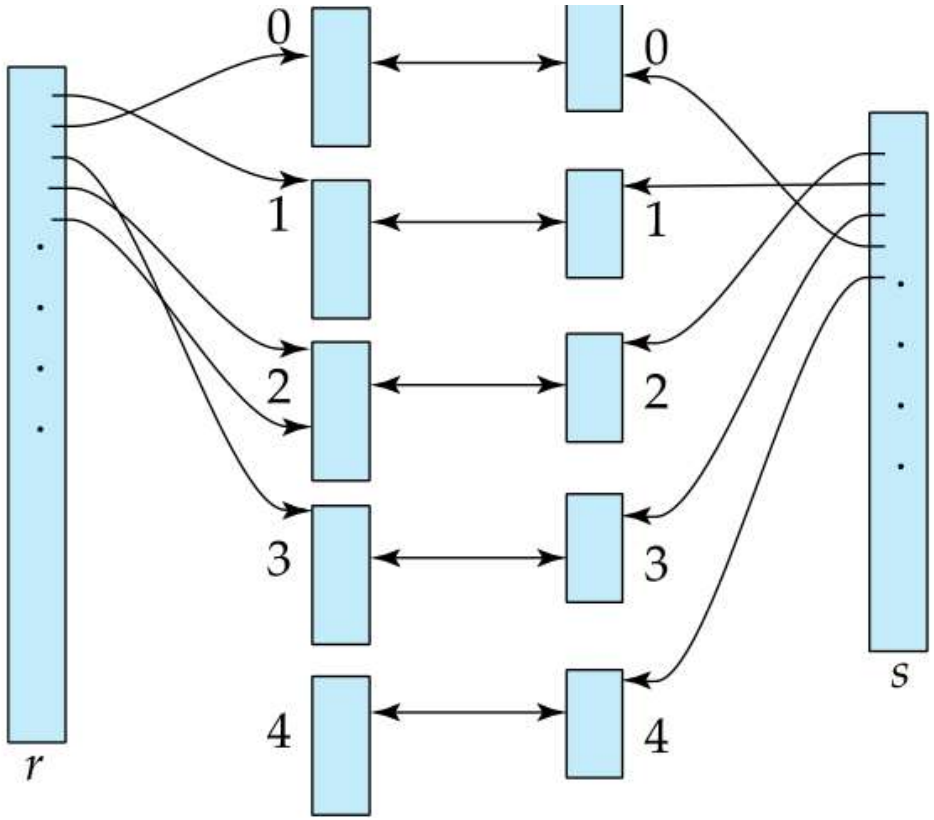
Гибридное соединение слиянием. Если одно из отношений отсортировано, а во втором
есть вторичный индекс B+ дерева по атрибуту соединения

Hash Join

h – функция отображения JoinAttrs на пространство секций

$r_0, r_1, \ldots, r_n$ – секции r

$s_0, s_1, \ldots, s_n$ – секции s



Hash Join алгоритм

1. Разбить отношение s на секции
2. Разбить отношение r на секции
3. Для каждого i:
    1. Загрузить s_i в память и построить хеш-индекс в памяти
    2. Считывать кортежи в r_i по одному. Для каждого кортежа t_r найти совпадающий кортеж t_s в s. Вывести конкатенацию

В случае, если число секций больше числа блоков в памяти, то появляется рекурсивное партиционирование.

Hash Join Стоимость

Требуется $3(b_r + b_s) + 4n_h$ переносов

Требуется $2\left(\frac{b_r}{b_b} + \frac{b_s}{b_b}\right)$ поисков

В случае рекурсивного партиционирования потребуются дополнительные расходы

Обработка запросов (продолжение)

Другие операции (убрать дубликаты)

- Забор от дубликатов может быть сделано с использованием сортировки. Дубликаты находятся последовательно при сортировки, что позволит их быстро удалить
- Если выполняется внешняя сортировка слиянием, то дубликаты можно удалить на стадиях внутренних слияний
- Также можно сделать удаление дубликатов через хеширование. В момент построения внутреннего хеш-индекса кортеж добавляется только, если его до этого не было.

Другие операции (Проекция)

- Проекция выполняется применением проекции для каждого кортежа, а затем отбросом дубликатов по необходимости (если в списке есть ключ отношения, то выполнять отброс дубликатов не требуется).

Другие операции (Операции с множествами)

- При операции с множествами (UNION, INTERSECT, EXCEPT), можно сначала отсортировать оба множества, а затем применять совместный обход двух отношений.

Другие операции (Операции с множествами)

- Хеширование предоставляет другой способ реализации операций со множествами. Первый шаг в любом случае это секционирование двух отношений с использованием одной и той же хеш-функции, что приводит к созданию секций $r_0, r_1, \ldots, r_n$ и $s_0, s_1, \ldots, s_n$
- $r \cup s$. Для каждой партиции $i = 0, 1, \ldots, n$:
  - Построить хеш индекс по r_i
  - Для каждого кортежа в s_i, добавить его в хеш-индекс, если его нет в хеш-индексе
  - Добавить кортежи из хеш-индекса в результат

Другие операции (Операции с множествами)

- Хеширование предоставляет другой способ реализации операций со множествами. Первый шаг в любом случае это секционирование двух отношений с использованием одной и той же хеш-функции, что приводит к созданию секций $r_0, r_1, \ldots, r_n$ и $s_0, s_1, \ldots, s_n$
- $r \cap s$. Для каждой партиции $i = 0, 1, \ldots, n$:
  - Построить хеш индекс по r_i
  - Для каждого кортежа в s_i , добавить его в результат, если он есть в хеш-индексе

Другие операции (Операции с множествами)

- Хеширование предоставляет другой способ реализации операций со множествами. Первый шаг в любом случае это секционирование двух отношений с использованием одной и той же хеш-функции, что приводит к созданию секций $r_0, r_1, \ldots, r_n$ и $s_0, s_1, \ldots, s_n$
- $r - s$. Для каждой партиции $i = 0, 1, \ldots, n$:
  - Построить хеш индекс по r_i
  - Для каждого кортежа в s_i, удалить их хеш-индекса, если он есть в хеш-индексе
  - Добавить все кортежи из хеш-индекса в результат

Другие операции (Внешние соединения)

- Для вычисления $r⟕s$ можно вычислить сначала $r \bowtie_\theta s$ и сохранит данный результат во временное отношение q. Затем вычислить $r - \Pi_R(q)$ для получения кортежей из r, которые не соединились с s. После этого дополнить кортежи null значения в требуемых полях
- Правое и полное внешнее соединения работают аналогично

Другие операции (Внешние соединения)

- Можно доработать алгоритмы. Можно дополнить алгоритм вложенных циклов правилом дополнения кортежа null – значениями в случае не нахождения соединения во внутренним цикле.
- Соединение слиянием может быть расширено: Если не найдено совпадение с другим отношением, то осуществляется дополнение кортежей с помощью null.

Другие операции (Агрегация)

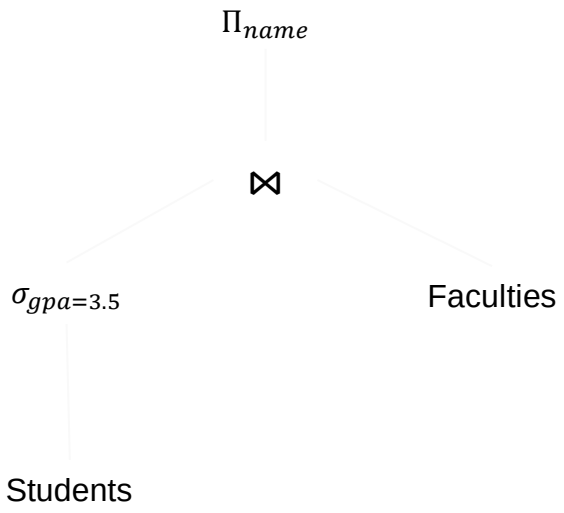
- Агрегация может быть реализована с использованием сортировки или хешированием. Операция производится по полям группировки, а затем агрегация применяется для каждой группы.
- Часть имплементаций предполагает вычисление агрегатов «на лету»

Вычисление выражений

- Для вычисления выражений, содержащих несколько реляционных операций, существует несколько вариантов вычисления.
- Если каждая операция вычисляется отдельно и результатом такого шага является временное отношение, то такая ситуация называется *материализацией.*
- Альтернативным способом является организация *конвейера (pipeline)* выполнения параллельных операций

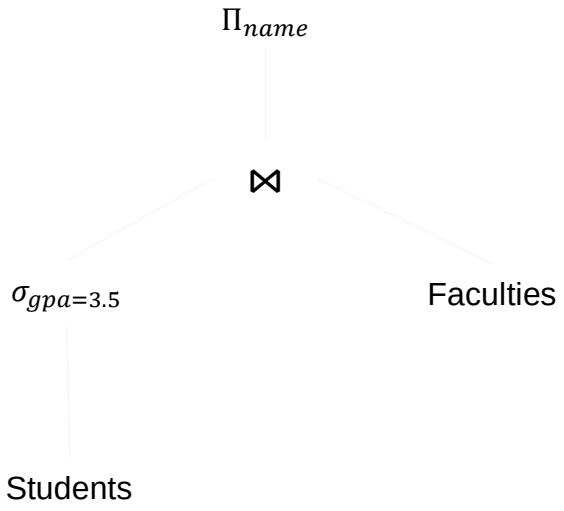
Материализация

Операцию реляционной алгебры можно представить в виде дерева операций: $\Pi_{name}(\sigma_{gpa=3.5}(Students) \bowtie Faculties)$



Материализация

Операцию реляционной алгебры можно представить в виде дерева операций: $\Pi_{name}(\sigma_{gpa=3.5}(Students) \bowtie Faculties)$



На листах дерева находятся отношения в базе.

В данном примере входные элементы для соединения
Являются отношение Faculties из БД и временное отношение, полученное из выборки.

Материализация

Материализованное вычисление представляет собой последовательность вычислений, в которой создается каждая промежуточная операция (материализуется) и используется для вычислений дальнейших шагов.

Для вычисления стоимости требуется добавить стоимость всех операций и стоимость записи временных результатов на диск.

Материализация

Материализованное вычисление представляет собой последовательность вычислений, в которой создается каждая промежуточная операция (материализуется) и используется для вычислений дальнейших шагов.

Для вычисления стоимости требуется добавить стоимость всех операций и стоимость записи временных результатов на диск.

Одним из вариантов оптимизаций является двойная буферизация (один из буферов осуществляет вычисление, другой – запись)

Конвейер

- $\Pi_{a,b}(r \bowtie s)$
- Уменьшение стоимости чтения и записи временных отношений
- Довольно быстрый отклик в передаче результатов запроса, если план запроса позволяет полноценный конвейер.

Реализация конвейера

- Конвейер может быть 2 типов:
- Demand-driven (Ленивое вычисление). Результат запроса операций низкого уровня не передается автоматически на более высокий уровень. Он будет передан, когда будет осуществлен запрос более высокого уровня.

Реализация конвейера

- Конвейер может быть 2 типов:
- Producer-driven (Жадное вычисление). Операции не ждут запрос для вычисления кортежей. Каждая операция в producer-driven конвейере моделируется как отдельный процесс или поток в системе, которая на вход принимает поток кортежей и на выходе также поток кортежей.

Реализация конвейера вычислений

- Каждая операция в ленивом конвейере может быть разработана как итератор, который предоставляет следующие функции: open(), next(), close()
- После вызова open(), каждое обращение к next() возвращает следующий кортеж на выход операции.
- Функция close() сообщает итератору, что больше кортежей не требуется
- Итератор хранит состояние вычисления между вызовами.

Реализация конвейера вычислений

- Конвейеры с жадным вычислением реализованы другим образом
- Для каждой смежной пары операций в конвейере, система создает буфер для фиксации кортежей, передающиеся с одной операции на другую.
- Процессы (или потоки) разных операций работаю параллельно.
- Каждая операция внизу кортежа последовательно вычисляет кортежи для вывода, и передает его в буфер вывода, до момента переполнения буфера.

Реализация конвейера вычислений

- Как только операция использует кортеж из входного буфера, он удаляется из буфера.
- Если буфер вывода полный, то операция ожидает забор данных из буфера более высокоуровневой операцией.
- Существует возможность распараллеливания операций внутри конвейера в случае многопроцессорных систем.

Вычисление алгоритмов для конвейеров

- Планы запросов могут указывать, для каких вершин выполняется конвейеризация (конвейерные вершины). Другие вершины называются блокирующими вершинами (материализованными).
- Множество всех соединенных конвейерных вершин вычисляется параллельно.
- План запроса может быть разделен на поддеревья, вершины между которым являются блокирующими.
- Каждое такое поддерево называется стадией конвейера.

Вычисление алгоритмов для конвейеров

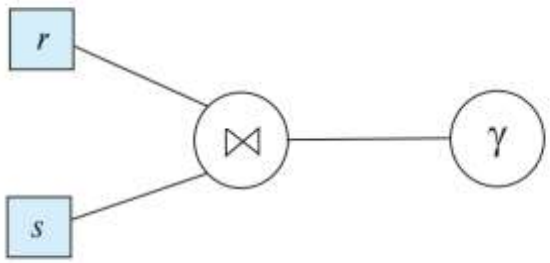
- Каждые операции, такие как сортировка, по существу являются блокирующими, так как они не могут выдавать результат до того, как все кортежи были забраны из входных отношений.
- При этом блокирующие операции могут выдавать выходные результат по мере их появления.
- Пример. Сортировка внешним слиянием. 1) Формирование шагов 2) Слияние. Шаг 1 может быть применен по мере появления кортежей во входном буфере, и поэтому может быть применена конвейеризация. Шаг 2 может посылать выходные результаты по мере их появления, но при этом для начала требуется полное завершение шага 1.

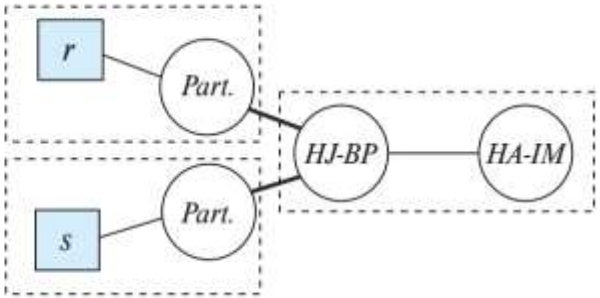
Вычисление алгоритмов конвейеризации

- Другие операции (например, соединение) могут быть не полностью не блокирующими, но при этом специфическое вычисление алгоритмов может приводить к блокировке.
- Например, алгоритм вложенных циклов с использованием индекса может выдавать результирующие кортежи, как только кортеж получен для внешнего отношения. Поэтому данная часть является конвейерной.
- При этом, так как индекс должен быть полностью сконструирован, то правая часть является блокирующей.

Вычисление алгоритмов конвейеризации

- Алгоритм хеш соединения является блокирующей операцией для обоих входов, так как требуется полное получение всех кортежей и секционирование отношения.





Вычисление алгоритмов конвейеризации

- Для каждой материализованной вершины должна быть возможность добавить стоимость записи данных на диск, и стоимость чтения данных на диск.
- Когда материализованная вершина находится между элементами одного оператора (например стадии пробега и слияния), тогда стоимость материализации заранее была добавлена в оценку стоимости и не требует дополнительной оценки.

Вычисление алгоритмов конвейеризации

- В части приложений может быть алгоритм соединения, который является конвейером. Например, если оба входа отсортированы по атрибутам и каждое условие соединения является экви-соединением, может быть применен алгоритм merge-join.
- Существует альтернатива двойного-конвейерного соединения. Алгоритм предполагает, что входные кортежи являются конвейерными.

Вычисление алгоритмов конвейеризации

- Конвейеры для потоковых данных
- Данные, поступающие на непрерывной основе (например IOT), представляют собой потоки данных.
- Запросы могут быть реализованы на потоковых данных по мере их поступления. Такие запросы называются непрерывными.
- Для таких запросов часто выделяют понятие окна. Результатом вычисления в окне может быть выдан в результат, если известно, что новых данных поступать уже не будет.

Обработка запросов в памяти

- Все предыдущие алгоритмы были связаны с минимизацией в I/O
- Для минимизации работы в памяти часто используется оптимизации для кеш уровня.

Алгоритмы оптимизации в кеше

- Разность в скорости между кешем или памяти, и тот факт, что данные передаются между памятью и кешем в элементах под названием кеш-линия (обычно около 64 байт).
- Работа с CPU кешем контролируется алгоритмами, которые сделаны на уровне hardware.
- Для сортировки отношения в памяти используется алгоритм слияния с выбором пробега, который попадает в кеш. Слияние эффективно работает с кешем, так как читени происходит на уровне слов.

Алгоритмы оптимизации в кеше

- Компиляция запросов
- Традиционно обработка запроса представляло собой интерпретатор, который вычислял план запроса. Возникает довольно большая проблема с накладными расходами для работы с метаданными или пользовательскими функциями.
- Современные СУБД компилируют планы запросов в машинный код или байт-код. Например, компилятор может заранее вычислить смещение в странице.