Лекция 5

Обработка запросов

Обработка запросов

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

Основные шаги обработки запросов

Парсинг и трансляция

Подготовка – установка соединения, выделение ресурсов, аутентификация и прочее.

Парсинг запроса:

- Синтаксический анализ запроса (правильно ли написана команда SQL).
- Семантический анализ (существуют ли указанные в команде объекты (используется системный словарь)
- Трансляция в выражение реляционной алгебры

Выражение реляционной алгебры

Один и тот же запрос может иметь эквивалентные выражения реляционной алгебры:

```
SELECT name, stipend
From Students
WHERE stipend < 7500;
```

1) $\Pi_{name\_stipend}(\sigma_{stipend<7500}(Students))$

2) $\sigma_{stipend<7500}(\Pi_{name\_stipend}(Students))$

Что из этого оптимальнее выполнить?

План запроса

В свою очередь, каждая операция реляционной алгебры может быть вычислена разными алгоритмами:

- Используется индекс на *stipend*, чтобы найти всех студентов со стипендией < 7500,
- Или производится полный скан(обход) отношения, который убирает студентов со стипендией ≥ 7500,
- Или сортировка отношения и отсечение результата?

Связанное выражение, уточняющее детальную стратегию выполнения называется планом вычислений (evaluation plan).

Оптимизация запросов заключается в следующем: из всех эквивалентных планов выполнения выбирается план с наименьшей стоимостью.

- Стоимость вычисляется с использованием статистической информации о БД.

Метаданные

Примеры статистических данных:

- Данные о БД:
  - Наличие таблиц, индексов и других объектов
- Данные о таблицах:
  - Количество кортежей и атрибутов
  - Количество занимаемых страниц во внешней памяти
- Данные об атрибутах (статистика):
  - Количество различных значений атрибута
  - Наибольшие и наименьшие значения атрибута
  - Наиболее часто и редко встречаемые значения атрибута

Меры стоимости запроса

- (Формально) – величина, характеризующая эффективность выполнения запроса.
- Стоимость может быть вычислена на основе
  - **времени ответа**, т.е. общее время ответа на запросы
  - общее **потребление ресурсов**
- Время ответа зависит от загрузки -> непрактично
- Факторы оценки потребления ресурсов:
  - доступ к диску
  - CPU
  - сетевые коммуникации
- Для простоты рассмотрим только чтение с диска, однако:
  - Реальные системы используют стоимость CPU для оценки
  - Стоимость сети рассматривается для параллельных систем

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 T трансферов блоков и S поисков
    $T * t_T + S * t_S$
- 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

Примечания

- 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 (WHERE)

- **Скан (обход) файла**
- **Линейный поиск**. Обходит каждый блок файла и тестирует все записи на условие отбора.
  - Стоимость выполнения = $t_T * b_r + t_S$
    - где b_r определяет число блоков записей в отношении r
  - Если выборка по ключевому атрибуту, то можно остановиться при нахождении, тогда
  - Стоимость выполнения (в среднем) = $t_T * (b_r/2) + t_S$
- Линейный поиск применяется, когда ***файл не отсортирован, индекс отсутствует***
- В случае отсортировонного файла можно применить **бинарный поиск**.
  - Стоимость выполнения = $t_T * \log_2(b_r) + t_S$
- Бинарный поиск применяется, когда ***файл отсортирован по атрибуту, индекс отсутствует***

Выборка при использовании индексов

- **Обход индекса** – алгоритмы поиска, использующие индекс
  - Условие выборки должно содержать ключ поиска в индексе.
- **A1** (**кластерный индекс, равенство по ключу**). Вывод: одна запись, которая подходит по равенству по ключу
  - *Стоимость* = $(h_i + 1) * (t_T + t_S)$
  - h_i – *высота дерева*
- **A2** (**кластерный индекс, равенство не по ключу**). Вывод: несколько записей.
  - Записи должны быть на последовательных блоках
    - Пусть b = число блоков, содержащих совпадающие записи
  - *Стоимость*= $(h_i + 1)* (t_T + t_S) + t_T * (b - 1)$

Выборка при использовании индексов

- **A3** (**вторичный индекс, равенство по ключу**).
  - Нет разницы с случаем A1
    - *Стоимость*= $(h_i + 1) * (t_T + t_S)$
- **A4** (**вторичный индекс, равенство не по ключу**).
  - Каждая из *n* совпадающих записей может быть в разных блоках
  - *Стоимость* = $(h_i + n) * (t_T + t_S)$ *в худшем случае*
    - 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

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.

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 отсортированы 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 = $\frac{\mathrm{b_r}}{b_b} + \frac{b_S}{b_b}$

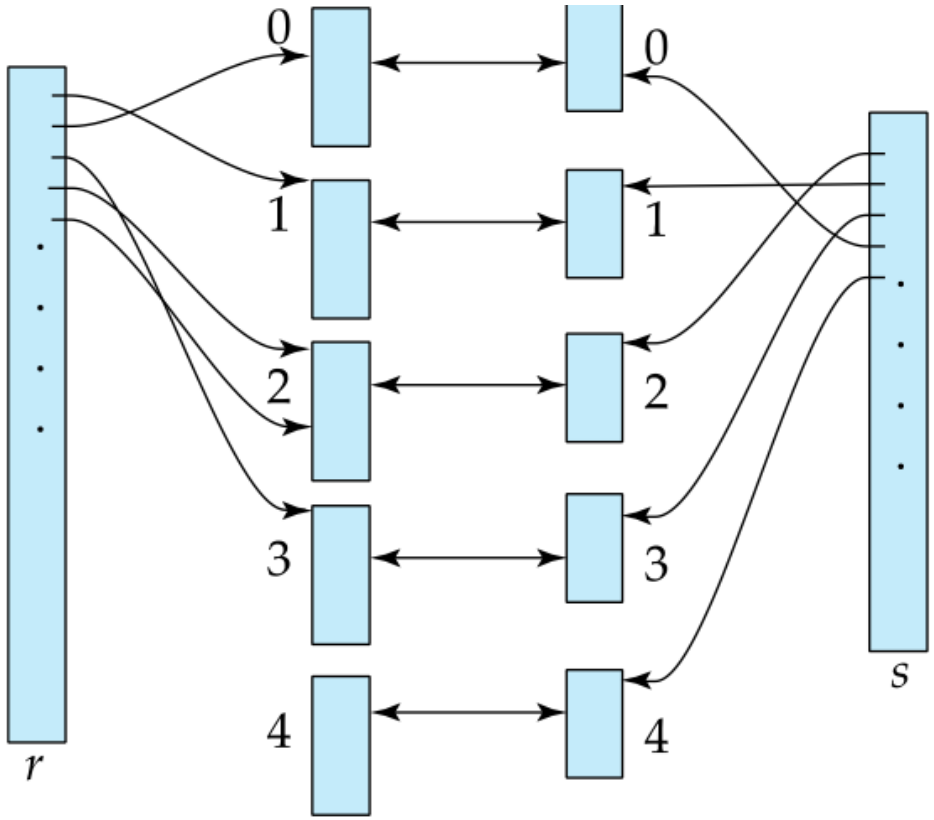
Гибридное соединение слиянием. Если одно из отношений отсортировано, а во втором
есть вторичный индекс B+ дерева по атрибуту соединения

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

Другие операции (убрать дубликаты)

- Забор от дубликатов может быть сделан 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 (убрать дубликаты)

Алгоритм с хешированием:

- Происходит разбиение на разделы:
  - к комбинации атрибутов отношения R применяется хеш-функция h и кортеж записывается в буфер в соответствии с вычисленным значением
  - два кортежа, принадлежащие разным разделам, гарантированно не будут являться дубликатами
- Далее в каждом разделе решается та же задача

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).

  Два этапа:
  - Удаление из отношения ненужных атрибутов.
  - Исключение из результирующей таблицы любых повторяющихся строк, появившихся в результате выполнения предыдущего этапа (см. удаление дубликатов)
    - этот этап требуется, если проекция не содержит ключ отношения

Операции с множествами

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

- 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,…, r_n и s_0,s_1,…, s_n

Операции с множествами

- $r \cup s$. 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

- $r \cap s$. 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

- $r - s$. 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

Внешние соединения

- Для вычисления $r⟕s$ можно вычислить сначала $r \bowtie_\theta s$ и сохранить 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

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.

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»

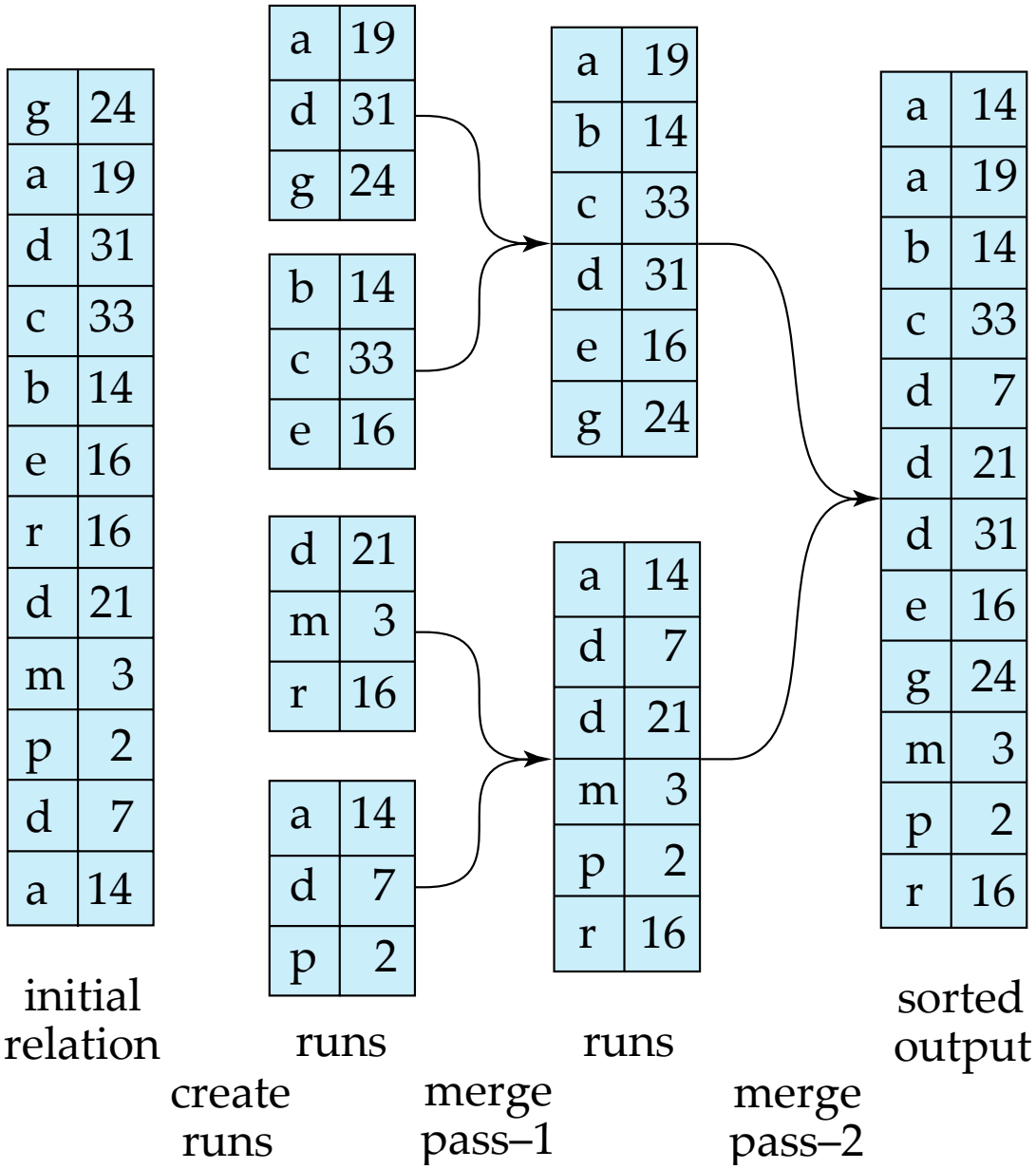
Сортировка

Для отношений, помещающихся в памяти, можно использовать стандартные сортировки, например quicksort. Для не помещающихся **external sort-merge**.

Алгоритм следующий:

- Разделение данных на блоки фиксированного размера
- Сортировка данных в блоках.
- Слияние блоков.

Пример: Sort-Merge



Стоимость сортировки

Пусть b_r - число блоков в отношении 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

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