

Обработка и исполнение запросов в СУБД (Лекция 1)

Классические системы: общая архитектура, модель volcano, подходы к реализации различных операторов

v7

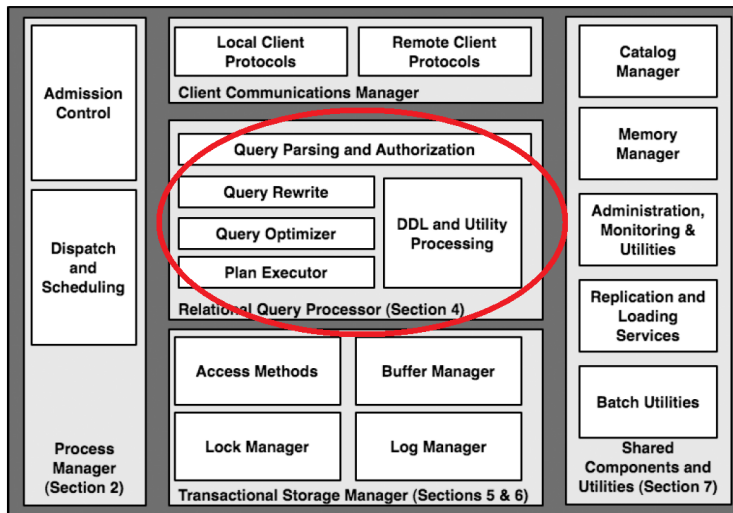
Георгий Чернышев

Высшая Школа Экономики

chernishev@gmail.com

2 сентября 2020 г.

Основные компоненты классической реляционной СУБД



1

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

“SELECT... → ответ. Как?”

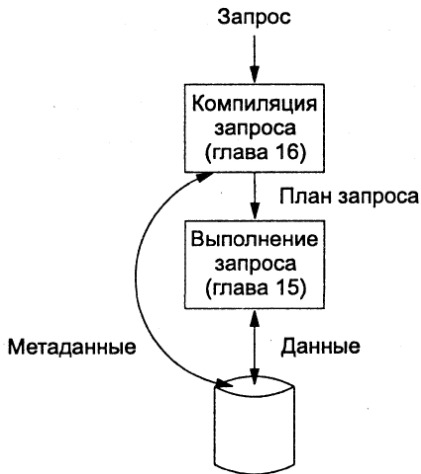


Рис. 15.1. Главные функции процессора запросов

2

²Изображение взято из [Garcia-Molina et al., 2004]

Фаза компиляции запроса

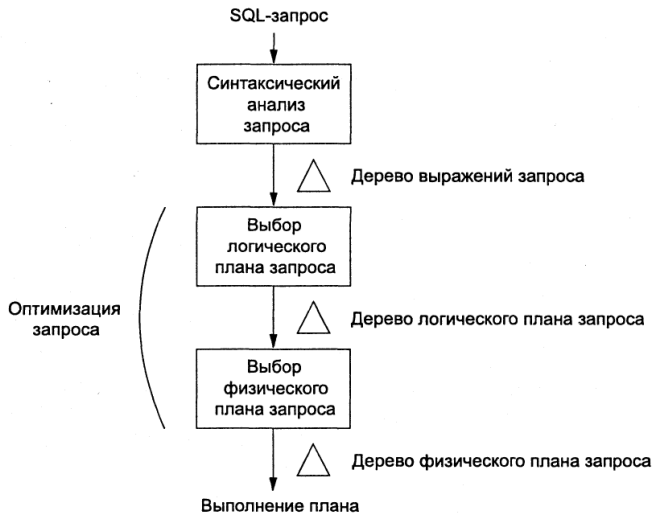


Рис. 15.2. Диаграмма процесса компиляции запроса

3

³ Изображение взято из [Garcia-Molina et al., 2004]

Фаза компиляции запроса

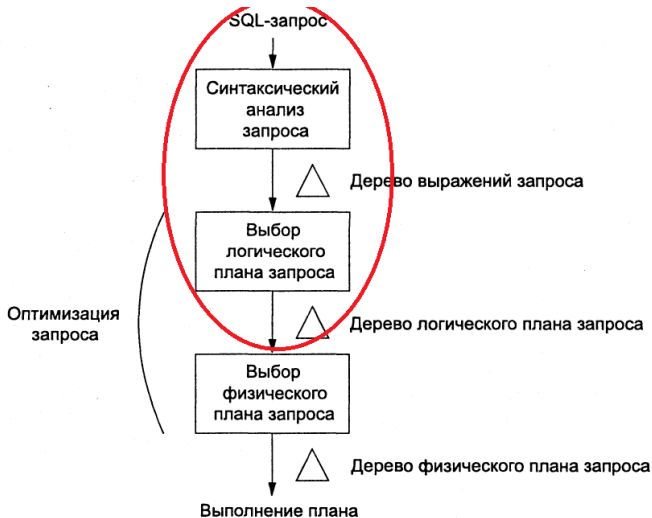


Рис. 15.2. Диаграмма процесса компиляции запроса

4

⁴ Изображение взято из [Garcia-Molina et al., 2004]

От запроса к логическому плану

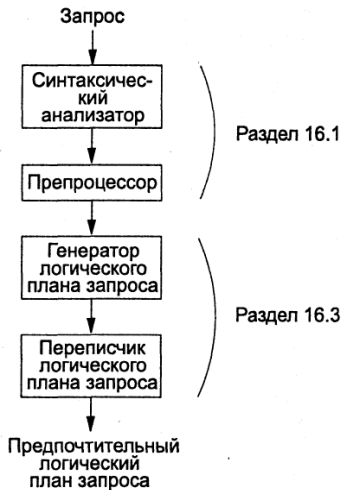


Рис. 16.1. От запроса к логическому плану запроса

5

⁵ Изображение взято из [Garcia-Molina et al., 2004]

Разбор запроса: подробности I

- Синтаксический анализ делается достаточно просто, есть масса готовых средств: Flex, Bison, На выходе получается некое внутреннее представление, дерево запроса;

```
%%      /* Bison grammar rules */
input   : /* empty production to allow an empty input */
        | input line
        ;
line    : term '\n'      { printf("Result is %f\n", $1); }
        ;
term    : term '*' factor { $$ = $1 * $3; }
        | term '/' factor { $$ = $1 / $3; }
        | factor          { $$ = $1; }
        ;
factor  : NUMBER          { $$ = $1; }
        ;
```

6

Разбор запроса: подробности II

- Препроцессор — подстановка дерева выражений для представлений, разрешение сущностей, семантический контроль:
 - Контроль употребления имен отношений;
 - Контроль использования имен атрибутов и их разрешение;
 - Контроль типов.
- Фаза перезаписи запроса: упрощение без потери семантики;
- Запрос “собирается” из набора реляционных операторов, по определенным правилам;
- Реляционная алгебра + теоретико-множественные операции позволяют комбинировать эти операторы;

Итог: логический план запроса.

⁶

Изображение взято из

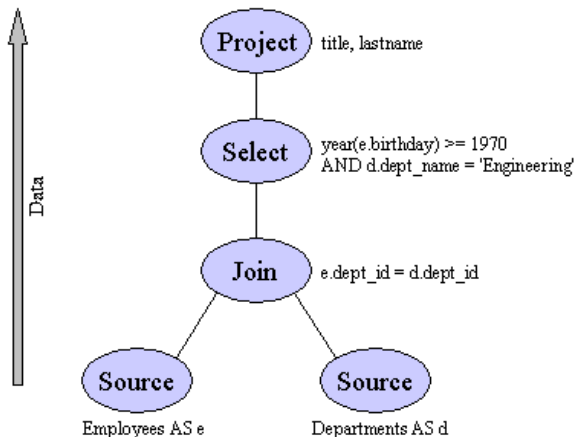
<https://image.slidesharecdn.com/yacclex-140518211555-phpapp02/95/yacc-lex-11-638.jpg>



В нашей реализации, что будет использоваться на практике, мы будем пользоваться Lemon parser generator.

Книга “Flex & Bison” John R. Levine, в 4 главе описывает парсинг для MySQL — содержит довольно значительную грамматику и лексер.

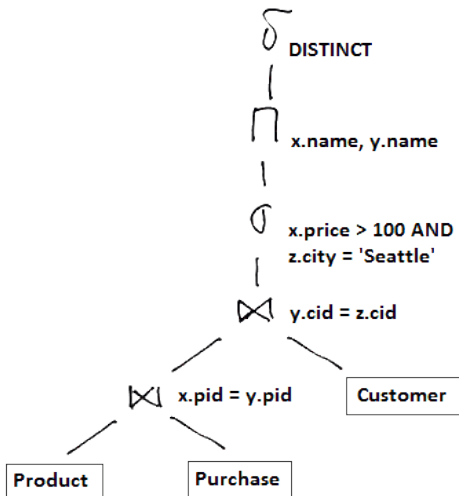
Примеры плана запросов (1)



7

⁷ Изображение взято с https://docs.jboss.org/teiid/6.0/reference/en-US/html/federated_planning.html

Примеры плана запросов (2)



8

8

Изображение взято с http://0agr.ru/wiki/index.php/Logical_Query_Plan_Optimization

Проблемы:

- Планы могут быть очень разные по качеству!
- Планов может быть очень много!
- Планы редко когда можно переиспользовать.
- ...

Вычисление оптимального плана *NP*-трудная задача. Поэтому ищут просто хороший.

Как искать? (1)

Стоимостная модель + эвристический метод:

- Генетические алгоритмы;
- Метод симуляции отжига;
- Метод восхождения к вершине;
- Метод муравьиной оптимизации;
- ...

На самом деле чаще всего имеется какая-то database-specific процедура перебора планов.

Как искать? (2): иллюстрация

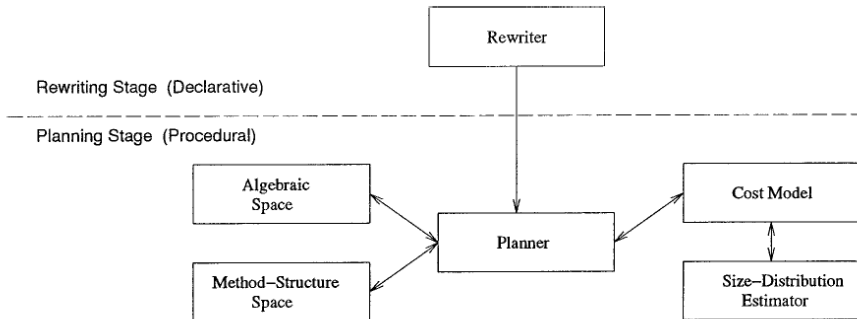


Figure 1. Query optimizer architecture.

9

Итераторная модель Volcano

Пусть у нас есть план, надо его запустить на выполнение.

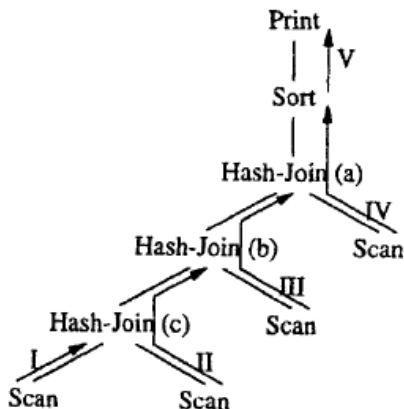


Figure 1. Left-deep query plan with plan phases

10

¹⁰ Изображение взято из [Graefe, 1996]

Итератор

```
1 class AbstractNode{
2     private:
3         AbstractNode* LChild;
4         AbstractNode* RChild;
5         void* ... // internal state
6     public:
7         int Open();
8         int Close();
9         void* GetNext();
10 };
```

- Идея: строим дерево из таких итераторов;
- Его можно запускать, вызывая у верхнего оператора GetNext();
- Обычно, между операторами ходит не одна, а много записей (блочная модель Volcano).

Материализация и пайплайнинг

Сколько записей должно быть в блоке?

- Крайний случай: все — полная материализация;
- Крайний случай: одна — пайплайнинг в стиле начала нулевых;
- Что-то посередине.

Как выбирать?

- Зависит от аппаратных параметров: e.g. размера кеша на машине;
- Зависит от программных параметров: e.g. сколько памяти отдается на запрос;
- Зависит от нагрузки: сколько и каких запросов предполагается обрабатывать одновременно;
- Наконец, зависит от типа системы.

Примеры простых итераторов

Table I. Simplified iterator methods

Iterator	<i>Open</i>	<i>Next</i>	<i>Close</i>	Local state
Print	<i>open</i> input	call <i>next</i> on input; format the item on screen	<i>close</i> input	
Scan	open file	read next item	close file	open file descriptor
Select	<i>open</i> input	call <i>next</i> on input until an item qualifies	<i>close</i> input	
Hash join (without overflow resolution)	allocate hash directory; <i>open</i> left 'build' input; build hash table calling <i>next</i> on build input; <i>close</i> build input; <i>open</i> right 'probe' input	call <i>next</i> on probe input until a match is found	<i>close</i> probe input; deallocate hash directory	hash directory
Merge-join (without duplicates)	<i>open</i> both inputs	get <i>next</i> item from input with smaller key until a match is found	<i>close</i> both inputs	
Sort	<i>open</i> input; build all initial run files calling <i>next</i> on input; <i>close</i> input; merge run files until only one merge step is left	determine next output item; read new item from the correct run file	destroy remaining run files	merge heap; open file descriptors for run files

11

Нужен минимальный набор:

- (σ) Выборка — то, что содержится во WHERE: $T1.X > 255$;
- (\bowtie) Соединение — бывает:
 - во WHERE: $T1.X = T2.Y$ и,
 - в FROM: `FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID`
- (Π) Проекция — остается то, что содержится в SELECT: `SELECT Orders.OrderID, Customers.CustomerName`

Такой класс запросов называется SPJ запросы (Select, Project, Join).

Это самый простой класс, есть еще операторы: агрегация, DISTINCT, TOP N, множественные операции (UNION, MINUS, ...), подзапросы, ...

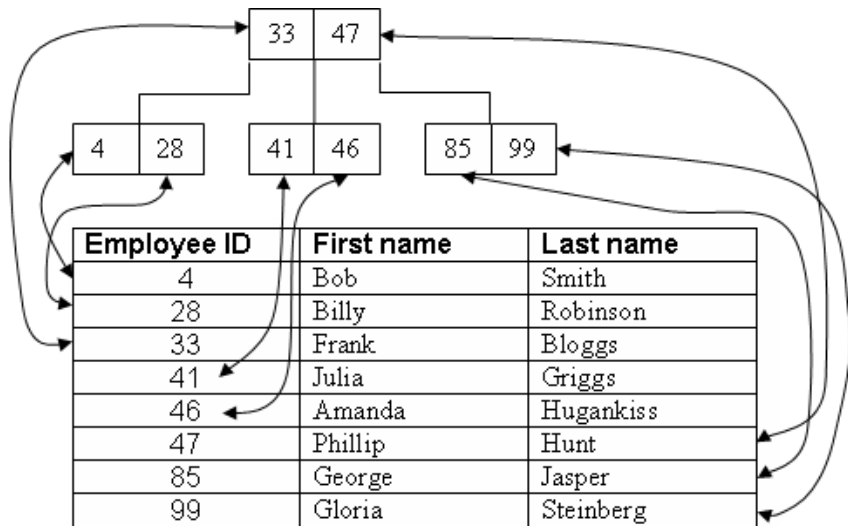
Реализация выборки = методы доступа (access methods).

Основные:

- Полный просмотр — медленный, но не требует ничего дополнительно, данные-то всяко есть;
- Просмотр по кластеризованному индексу — быстрый, но можно только по одной комбинации атрибутов;
- Просмотр с использованием индекса — быстрый, можно по любому количеству комбинаций атрибутов, но надо строить.

Сложные: например, по нескольким индексам, пересечение и вычитка.

Напоминание про индекс на B-tree



Реализация реляционной операции соединение

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
2	303	30	3
3	302	24	3

Таблица: rooms

Основные методы: Nested Loop, Sort-Merge, Hash-Join.

В следующих слайдах обращайте внимание на отсортированность атрибутов!

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3

Nested Loop

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3, 5-3

Sort-Merge

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
4	vasya	60	php	2
5	sasha	70	java	2
3	slava	30	java	3

Таблица: wages

id	name	screen	floor
1	401	14	4
2	302	24	3
3	303	30	3

Таблица: rooms

Выдача: 1-1

Sort-Merge

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
4	vasya	60	php	2
5	sasha	70	java	2
3	slava	30	java	3

Таблица: wages

id	name	screen	floor
1	401	14	4
2	302	24	3
3	303	30	3

Таблица: rooms

Выдача: 1-1, 2-1

Sort-Merge

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
4	vasya	60	php	2
5	sasha	70	java	2
3	slava	30	java	3

Таблица: wages

id	name	screen	floor
1	401	14	4
2	302	24	3
3	303	30	3

Таблица: rooms



Выдача: 1-1, 2-1, 3-2

Sort-Merge

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
4	vasya	60	php	2
5	sasha	70	java	2
3	slava	30	java	3

Таблица: wages

id	name	screen	floor
1	401	14	4
2	302	24	3
3	303	30	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3

Sort-Merge

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
4	vasya	60	php	2
5	sasha	70	java	2
3	slava	30	java	3

Таблица: wages

id	name	screen	floor
1	401	14	4
2	302	24	3
3	303	30	3

Таблица: rooms

Выдача: 1-1, 2-1, 3-2, 4-3, 5-3

Фаза хеширования

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш: 1

Выдача:

Фаза хеширования

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш: 1, 3

Выдача:

Фаза хеширования

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш:

1, 2, 3

Выдача:

Фаза пробинга

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш:

1, 2, 3

Выдача: 1-1

Фаза пробинга

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

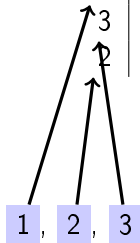
id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш:

1, 2, 3

Выдача: 1-1, 2-1



Фаза пробинга

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш: 1, 2, 3
Выдача: 1-1, 2-1, 3-2



Фаза пробинга

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш: 1, 2, 3
Выдача: 1-1, 2-1, 3-2, 4-3



Фаза пробинга

id	name	wage	skill	roomid
1	ivan	80	c++	1
2	petr	50	c++	1
3	slava	30	java	3
4	vasya	60	php	2
5	sasha	70	java	2

Таблица: wages

id	name	screen	floor
1	401	14	4
3	303	30	3
2	302	24	3

Таблица: rooms

Хеш:

1, 2, 3

Выдача: 1-1, 2-1, 3-2, 4-3, 5-3

- Обычно стараются делать сразу, как только возможно;
- Часто прячут внутри других операторов;

id	name	wage	skill	exp
1	ivan	80	c++	5
2	petr	50	c++	3
3	slava	30	java	1
4	vasya	60	php	1
5	sasha	70	java	3
6	dasha	65	c++	3
7	katya	50	java	1
8	glasha	30	bash	2
9	oleg	20	php	3
10	boris	50	java	3

Таблица: wages

```
SELECT skill, avg(wage), exp FROM wages GROUP BY skill, exp
```

Агрегация: как?

- Влоб: завести набор групп, сверяться каждый раз со всеми;
- Отсортировать по GROUP BY, а потом аккуратно пройти один раз.

Упражнения:

- Что выдаст запрос?
- Расписать как будет происходить вычисление.

Замечание о порядках сортировки

- Они очень сильно помогают;
- Каждый раз при запросе пересортировывать — долго;
- Сделать пресортировку один раз, по этому атрибуту. Однако, на диске таблицу держать в отсортированном виде можно только по одному атрибуту.

Замечание про операторы

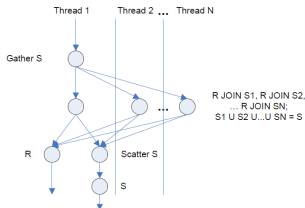
Операторы бывают:

- Блокирующие, например hash-join или сканирование с сортировкой;
- Не блокирующие.

А еще они могут ломать сортировку или нет:

- Sort-merge join сохраняет сортировку по обоим атрибутам;
- Hash join испортит по одному атрибуту.

Операторы можно параллелить, как по-отдельности, так и внутри:



Более подробно смотри [Taniar et al., 2008].

Современность и альтернативы: компиляция запросов I

То, что рассматривалось выше (Volcano) называется **интерпретацией** запросов. А бывает еще и **компиляция** запросов в код и она очень популярна сейчас, фактически баззворд.

- Пытались так делать с 80х, но дело не пошло: долго компилируется, да и сейчас заметные затраты;
- Сейчас (2010+) компиляция есть очень во многих индустриальных системах, в 10х был бум исследований;
- Кажется что для дисковых систем подход не особо важен;
- Из разговоров разработчиками крупных систем:
 - очень тяжело разрабатывать и мейнтейнить такие системы;
 - а выгода-то не такая уж и заметная, на отдельных запросах до 30%

Итог: кажется что овчинка стоит выделки только при проверке предикатов. Но и тут у подхода есть конкуренты (е.g. стековые машины).


```

select      *
from        R1,R3,
            (select R2.z,count(*)
from        R2
where       R2.y=3
group by   R2.z) R2
where       R1.x=7 and R1.a=R3.b and R2.z=R3.c

```

Figure 2: Example Query

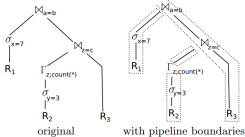


Figure 3: Example Execution Plan for Figure 2

```

initialize memory of  $\mathbb{M}_{a=b}$ ,  $\mathbb{M}_{c=z}$ , and  $\Gamma_z$ 
for each tuple  $t$  in  $R_1$ 
  if  $t.x = 7$ 
    materialize  $t$  in hash table of  $\mathbb{M}_{a=b}$ 
for each tuple  $t$  in  $R_2$ 
  if  $t.y = 3$ 
    aggregate  $t$  in hash table of  $\Gamma_z$ 
for each tuple  $t$  in  $\Gamma_z$ 
  materialize  $t$  in hash table of  $\mathbb{M}_{z=c}$ 
for each tuple  $t_3$  in  $R_3$ 
  for each match  $t_2$  in  $\mathbb{M}_{z=c}[t_3.c]$ 
    for each match  $t_1$  in  $\mathbb{M}_{a=b}[t_3.b]$ 
      output  $t_1 \circ t_2 \circ t_3$ 

```

Figure 4: Compiled query for Figure 3

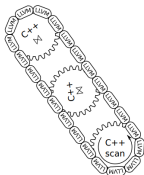


Figure 6: Interaction of LLVM and C++

Современность и альтернативы: компиляция запросов III

Как генерировать код? Авторы попробовали генерировать C++ код но это оказалось долго и неудобно: сложный запрос компилировался несколько секунд.

Пришли к связке прекомпилированных C++ кусочков (реализация сложных компонентов) и генерация в LLVM ассемблер, который JIT компилировался и вызывал C++ кусочки.

```
define internal void @scanConsumer(%8* %executionState, %Fragment_R2* %data) {
body:
    ...
    %columnPtr = getelementptr inbounds %Fragment_R2* %data, i32 0, i32 0
    %column = load i32** %columnPtr, align 8
    %columnPtr2 = getelementptr inbounds %Fragment_R2* %data, i32 0, i32 1
    %column2 = load i32** %columnPtr2, align 8
    ... (loop over tuples, currently at %id, contains label %cont17)
    %yPtr = getelementptr i32* %column, i64 %id
    %y = load i32* %yPtr, align 4
    %cond = icmp eq i32 %y, 3
    br i1 %cond, label %then, label %cont17
then:
    %zPtr = getelementptr i32* %column2, i64 %id
    %z = load i32* %zPtr, align 4
    %hash = urem i32 %z, %hashTableSize
    %hashSlot = getelementptr %"HashGroupify::Entry"* %hashTable, i32 %hash
    %hashIter = load %"HashGroupify::Entry"* %hashSlot, align 8
    %cond2 = icmp eq %"HashGroupify::Entry"* %hashIter, null
    br i1 %cond, label %loop20, label %else26
    ... (check if the group already exists, starts with label %loop20)
else26:
    %cond3 = icmp le i32 %spaceRemaining, i32 8
    br i1 %cond, label %then28, label %else47
    ... (create a new group, starts with label %then28)
else47:
    %ptr = call i8 @ZN12HashGroupify15storeInputTupleEmj
    (%"HashGroupify"* %1, i32 hash, i32 8)
    ... (more loop logic)
}
```

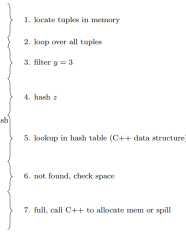


Figure 7: LLVM fragment for the first steps of the query $\Gamma_{\text{count}(\ast)}(\sigma_{y=3}(R_2))$

14

14

Все изображение на этом слайде взяты из [Т. Neumann, 2011]

Современность и альтернативы: push-based модели

Сейчас есть два понимания push-based моделей: интерпретационные и компиляционные. Мы говорим сейчас о первых!

- Идея: оператор живет в своем потоке и уведомляет родителя о том, что результат готов;
- Тяжелее в реализации, требуют большого количества ядер;
- Иногда без них не обойтись: могут решить проблему ромба в потоках данных;
- Не знаю промышленных систем где бы использовался → сейчас представляют только исследовательский интерес:
 - Переиспользование промежуточных результатов: QPipe [Harizopoulos et al., 2005], [Psaroudakis et al, 2014] и другие системы [Makreshanski et al, 2018], [Arumugam et al, 2010];
 - Адаптивность (изменение плана или способа выполнения на ходу): Eddies [Avnur and Hellerstein, 2000]

Современность и альтернативы: как, что устроено |

Название	год	тип	pull/push	t/b/op-at-a-time	source
PostgreSQL	1996	D	pull	tuple	link ¹⁵
SQLite	2000	M	pull	tuple	link ¹⁶
MariaDB	pre-2019	D	none	?	link ¹⁷
MariaDB	2019	D	pull	?	link ¹⁸
CockroachDB	2015	?	pull	block	link ¹⁹
Vertica	2005	D	оба?	?	link ²⁰
SYBASE Adaptive Server Enterprise 15.5	1987	D	pull	tuple	link ²¹
SAP HANA	2010	M	?	block	link ²²
Snowflake	2014	D?	push	block	link ²³
Microsoft SQL Server Hekaton Engine	2014	M	компиляция (pull?)	tuple?	link ²⁴
HyPer	≈ 2006	M	компиляция (push)	?	link ²⁵ , link ²⁶ , link ²⁷

Современность и альтернативы: как, что устроено II

¹⁵<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.357.4302&rep=rep1&type=pdf>

¹⁶<https://dbdb.io/db/sqlite>

¹⁷winand.at/newsletter/2019-12/partiql-microsoft-licenses-volcano-model

¹⁸<https://dev.mysql.com/worklog/task/?id=11785#tabs-11785-4>

¹⁹<https://dbdb.io/db/cockroachdb>

²⁰<https://www.vertica.com/kb/Reading-Query-Plans/Content/BestPractices/Reading-Query-Plans.htm>

²¹<http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc00743.1502/html/queryprocessing/queryprocessing1.htm>


²²<https://dbdb.io/db/sap-hana>

²³<https://dbdb.io/db/snowflake>

²⁴Compilation in the Microsoft SQL Server Hekaton Engine

²⁵Franz Faerber et al. (2017), "Main Memory Database Systems Foundations and Trends® in Database

²⁶<https://15721.courses.cs.cmu.edu/spring2018/notes/03-compilation.pdf>

²⁷Building Efficient Query Engines in a High-Level Language 



Гектор Гарсиа-Молина, Джеффри Д. Ульман, Дженнифер Уидом. Системы баз данных. Полный курс. ISBN 5-8459-0384-X; 2004 г.



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



Yannis E. Ioannidis. 1996. Query optimization. ACM Comput. Surv. 28, 1 (March 1996), 121–123. DOI=<http://dx.doi.org/10.1145/234313.234367>



Kirill Smirnov and George Chernishev. Benchmarking Inter and Intra Operator Parallelism on Contemporary Desktop Hardware. In Proc. of SYRCoDIS 2011, p. 62–67, 2011.



Goetz Graefe. 1996. Iterators, schedulers, and distributed-memory parallelism. Softw. Pract. Exper. 26, 4 (April 1996), 427–452.
DOI=[http://dx.doi.org/10.1002/\(SICI\)1097-024X\(199604\)26:4<427::AID-SPE20>3.3.CO;2-8](http://dx.doi.org/10.1002/(SICI)1097-024X(199604)26:4<427::AID-SPE20>3.3.CO;2-8)



David Taniar, Clement H. C. Leung, Wenny Rahayu, and Sushant Goel. 2008. High Performance Parallel Database Processing and Grid Databases. Wiley Publishing.



Raghu Ramakrishnan and Johannes Gehrke. 2000. Database Management Systems (2nd ed.). Osborne/McGraw-Hill, Berkeley, CA, USA.



Avnur, R., Hellerstein, J.M.: Eddies: continuously adaptive query processing. Proc. SIGMOD 2000, 261–272 (2000)



Stavros Harizopoulos, Vladislav Shkapenyuk, and Anastassia Ailamaki. QPipe: a simultaneously pipelined relational query engine. SIGMOD '05



Iraklis Psaroudakis, Manos Athanassoulis, Matthaïos Olma, and Anastasia Ailamaki. 2014. Reactive and proactive sharing across concurrent analytical queries. SIGMOD '14



Darko Makreshanski, Georgios Giannikis, Gustavo Alonso, and Donald Kossmann. 2018. Many-query join: efficient shared execution of relational joins on modern hardware. The VLDB Journal 27, 5 (October 2018), 669–692.



Subi Arumugam et al. 2010. The DataPath system: a data-centric analytic processing engine for large data warehouses. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). Association for Computing Machinery, New York, NY, USA, 519–530.



Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB, 4(9):539–550, 2011.