Лекция 7.

Производительность запросов. Индексирование

Как работает SQL-запрос?

Запрос

```
select name, email from users limit 10;
```

Магия

name	email
Devonte Stamm	marianne.krajcik@bartoletti.com
Merritt Grimes	rempel.yvette@kertzmann.com
Dianna Herzog	jarrell.stokes@gmail.com
Jamel Nader	edgar.bayer@kohler.com
Immanuel Bode	jamal68@yahoo.com
Dwight Reichel	camron36@yahoo.com
Holden Lueilwitz	hdaugherty@hotmail.com
Pablo Kuhlman	arden.lemke@leffler.com
Roscoe Rice	katelynn.heathcote@thompson.org
Arlene Harvey	crist.shannon@hansen.com

Результат

SQL - язык программирования?

Программа

```
select name, email from users limit 10;
```

~~Магия~~
Компиляция/
Интерпретация

name	email
Devonte Stamm	marianne.krajcik@bartoletti.com
Merritt Grimes	rempel.yvette@kertzmann.com
Dianna Herzog	jarrell.stokes@gmail.com
Jamel Nader	edgar.bayer@kohler.com
Immanuel Bode	jamal68@yahoo.com
Dwight Reichel	camron36@yahoo.com
Holden Lueilwitz	hdaugherty@hotmail.com
Pablo Kuhlman	arden.lemke@leffler.com
Roscoe Rice	katelynn.heathcote@thompson.org
Arlene Harvey	crist.shannon@hansen.com

Результат

SQL - язык программирования

Для описания ЯП нужно определить:

- **Алфавит** – набор символов, используемый при записи программ.
- **Синтаксис (грамматику)** – правила, задающие внешний вид программы и определяющие допустимые тексты в языке.
- **Семантику** – правила, определяющие действия, которые выполнит компьютер под управлением программы.

Описание правил грамматики ЯП

- **Текстовые формы Бэкуса-Наура (БНФ)**

  - **Терминальные символы** — это отдельные символы или их последовательности, имеющие конкретные известные значения и являющиеся неразрывным целым, не сводимым к другим символам.
  - **Нетерминальные символы (метапеременные)** – элементы языка, не имеющие заранее известного значения (формулы, команды и т.п.), которые раскрываются правилам грамматики.
  - **::=** "есть по определению"
  - **|** "или"
  - **{a}** повтор а 0 или более раз
  - **[a]** а входит 0 или 1 раз

- **Визуальные синтаксические диаграммы**

Описания правил грамматики SQL

- **Текстовые формы Бэкуса-Наура (БНФ)**

  Стандарт SQL-92
  https://jakewheat.github.io/sql-overview/sql-92-grammar.html

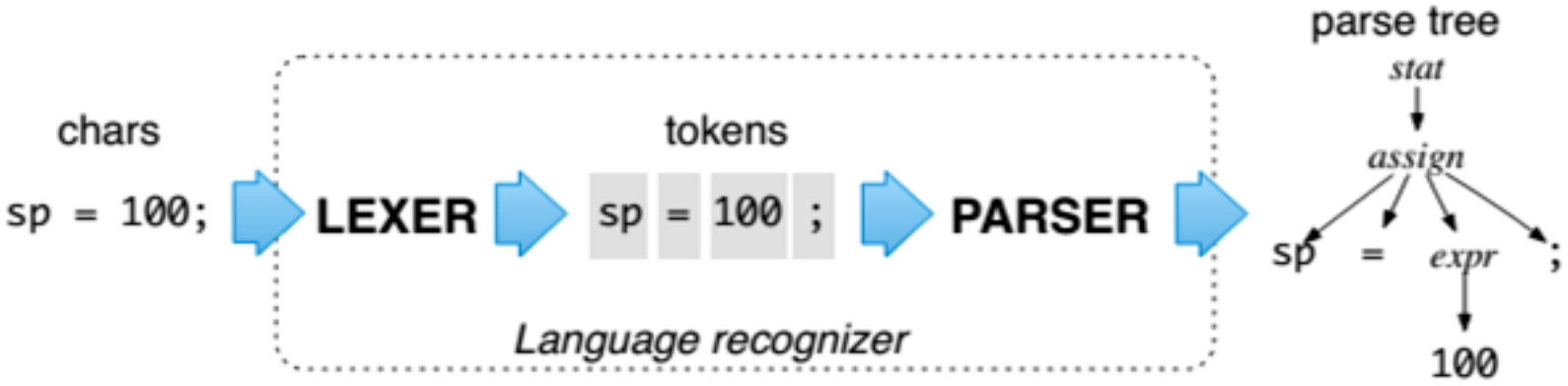
- **Визуальные синтаксические диаграммы**

  Реализация в SQLite
  https://www.sqlite.org/syntaxdiagrams.html

Анализ синтаксиса программы

Лексер – разбирает текст и представляет его в виде массива токенов. **Токен** – совокупность значения лексемы (значимого слова), ее типа и другой метаинформации.

Парсер – группирует по поступающие из лексера токены в дерево разбора, отражающее иерархию элементов программы и связи между ними.



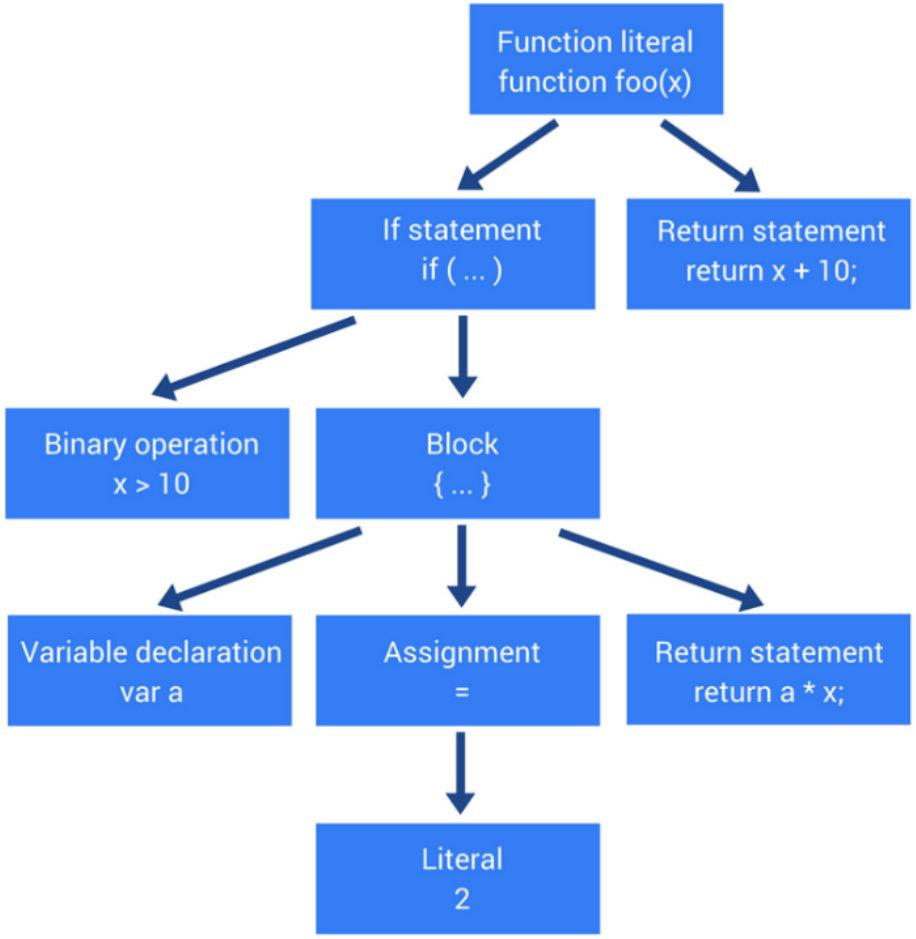
Анализ синтаксиса программы

Абстрактное синтаксическое дерево (AST) — дерево разбора, из которого удалены незначимые токены (скобки, запятые, …).

Это структурное представление исходного кода в виде дерева, где каждая вершина обозначает различные типы конструкций языка (выражение, переменную, оператор и т.п.)

```
function foo(x) {
    if (x > 10) {
        var a = 2;
        return a * x;
    }

    return x + 10;
}
```



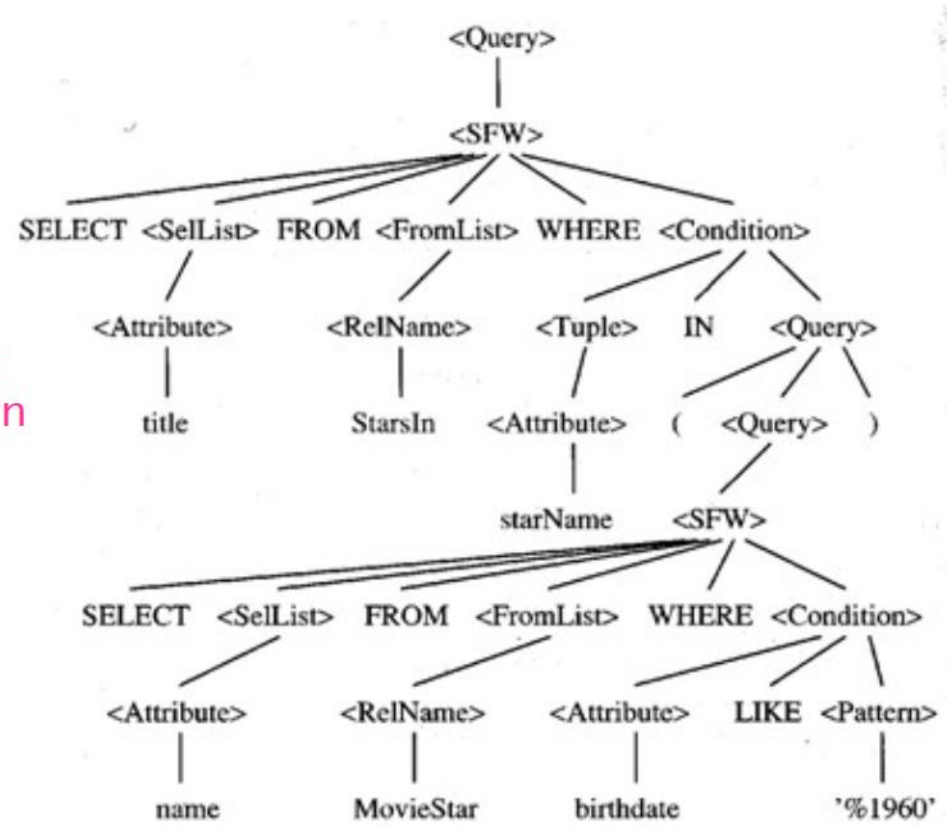
AST для SQL

```
StarsIn(
   title,year,starName
)
MovieStar(
   name,address,gender,bdate
)
```

Query:

Give titles of movies that have at least one star born in 1960

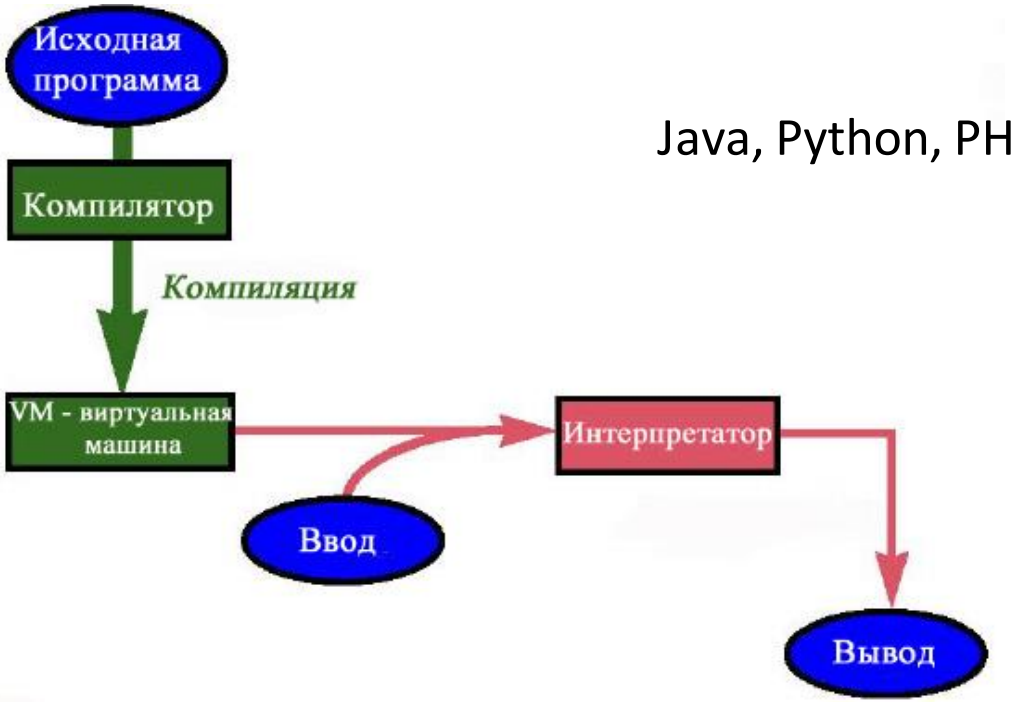
```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE
        birthdate LIKE '%1960%'
);
```



Процесс компиляции

1. **Лексический анализ.** Последовательность символов исходного файла преобразуется в последовательность лексем (токенов).
2. **Синтаксический анализ.** Последовательность лексем преобразуется в дерево разбора.
3. **Семантический анализ.** Дерево разбора обрабатывается для установления его семантики (смысла): привязка идентификаторов к определениям, проверка совместимости, определение типов выражений и т.д. Результат – промежуточное представление/код.
4. **Оптимизация.** Удаление лишних конструкций и упрощение кода с сохранением его смысла.
5. **Генерация машинного кода.**

Комбинирование компиляции и интерпретации



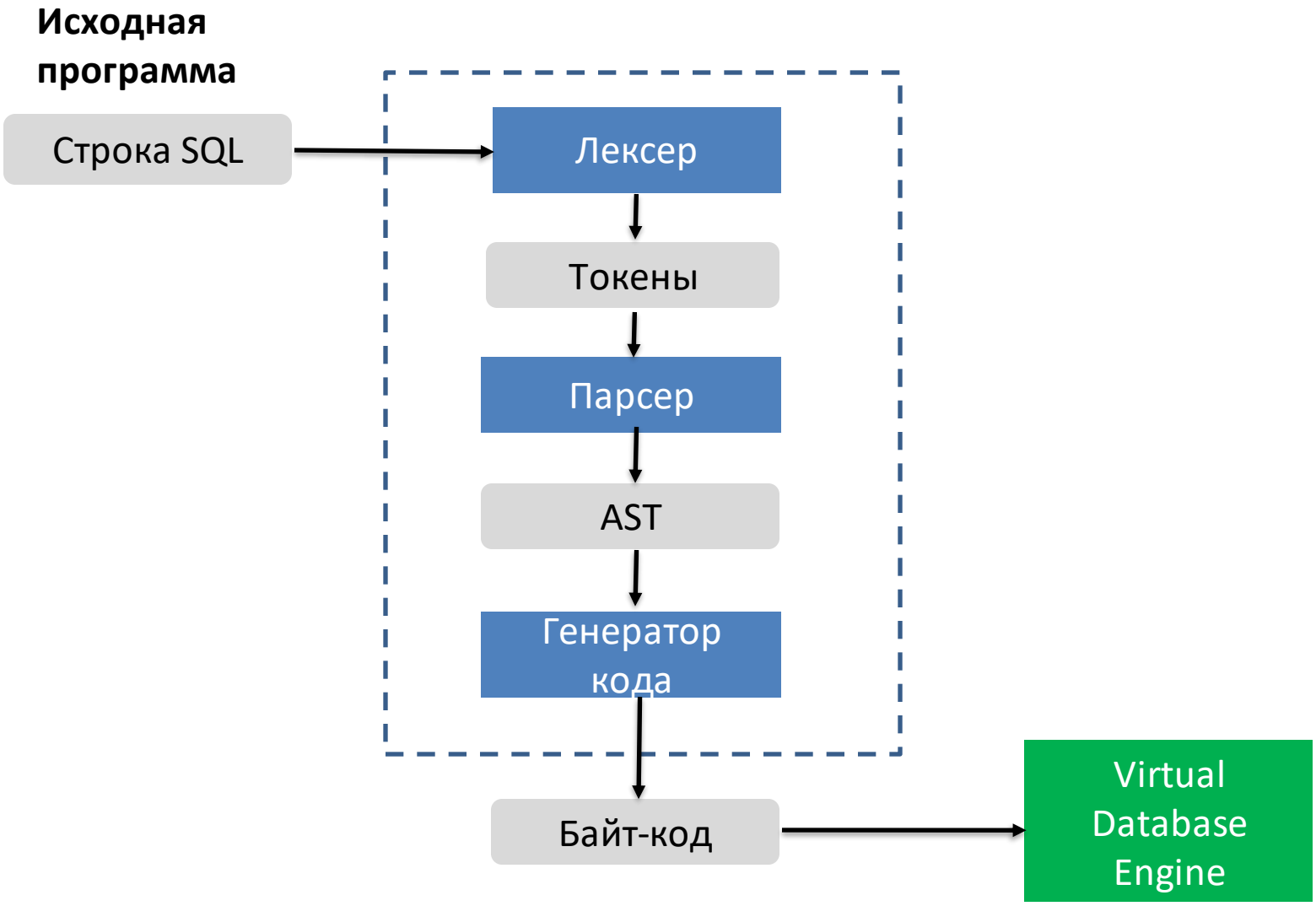
Java, Python, PHP, Ruby

Компилятор создает байт-код на промежуточном языке, понимаемом некоторой виртуальной машиной (интерпретатором байт-кода).

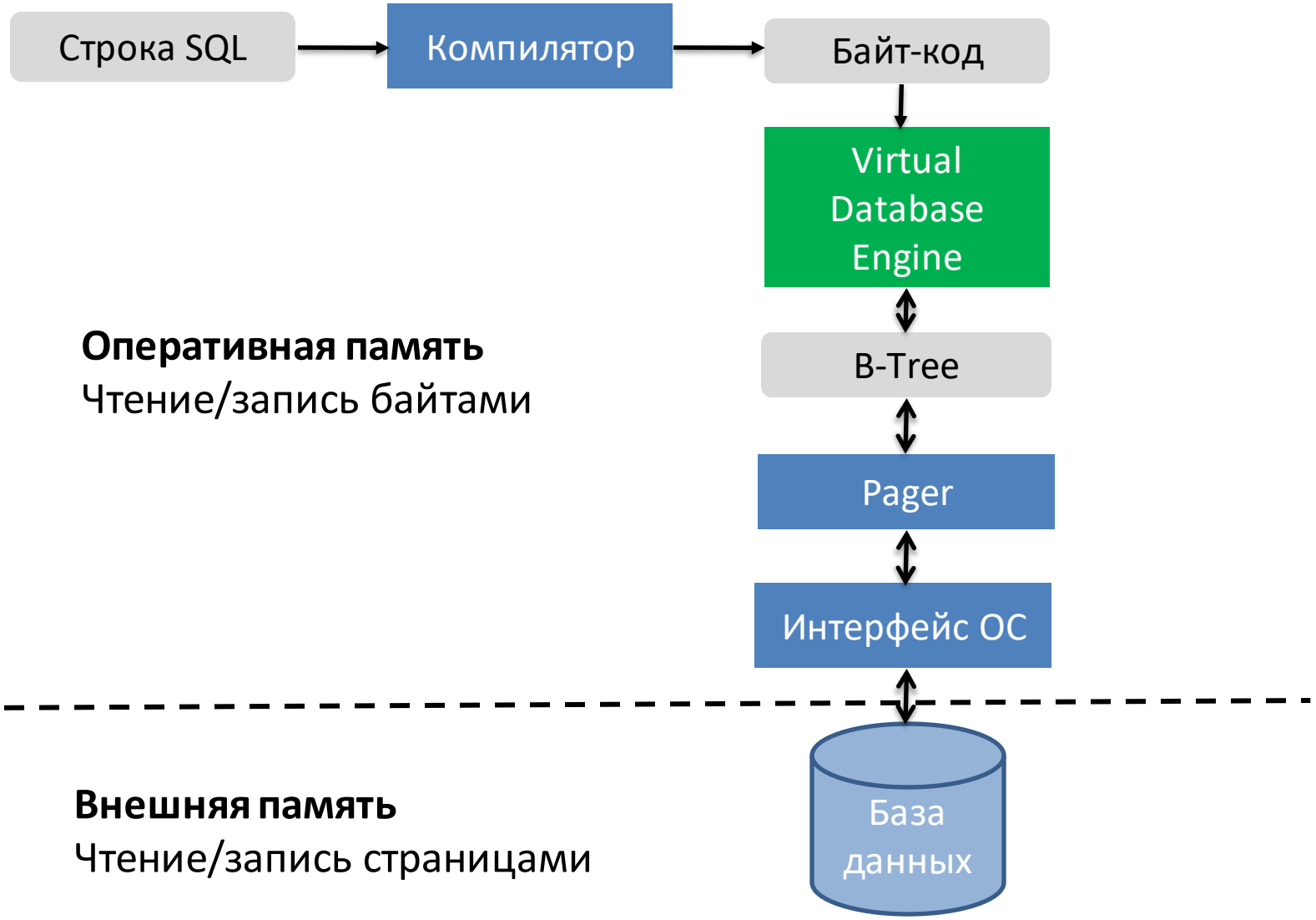
Преимущества:

- **Кроссплатформенность**, так как VM-код не зависит от специфики процессоров;
- **Повышение эффективности**, так как создаваемый промежуточный код легко интерпретируется.

SQL - язык программирования



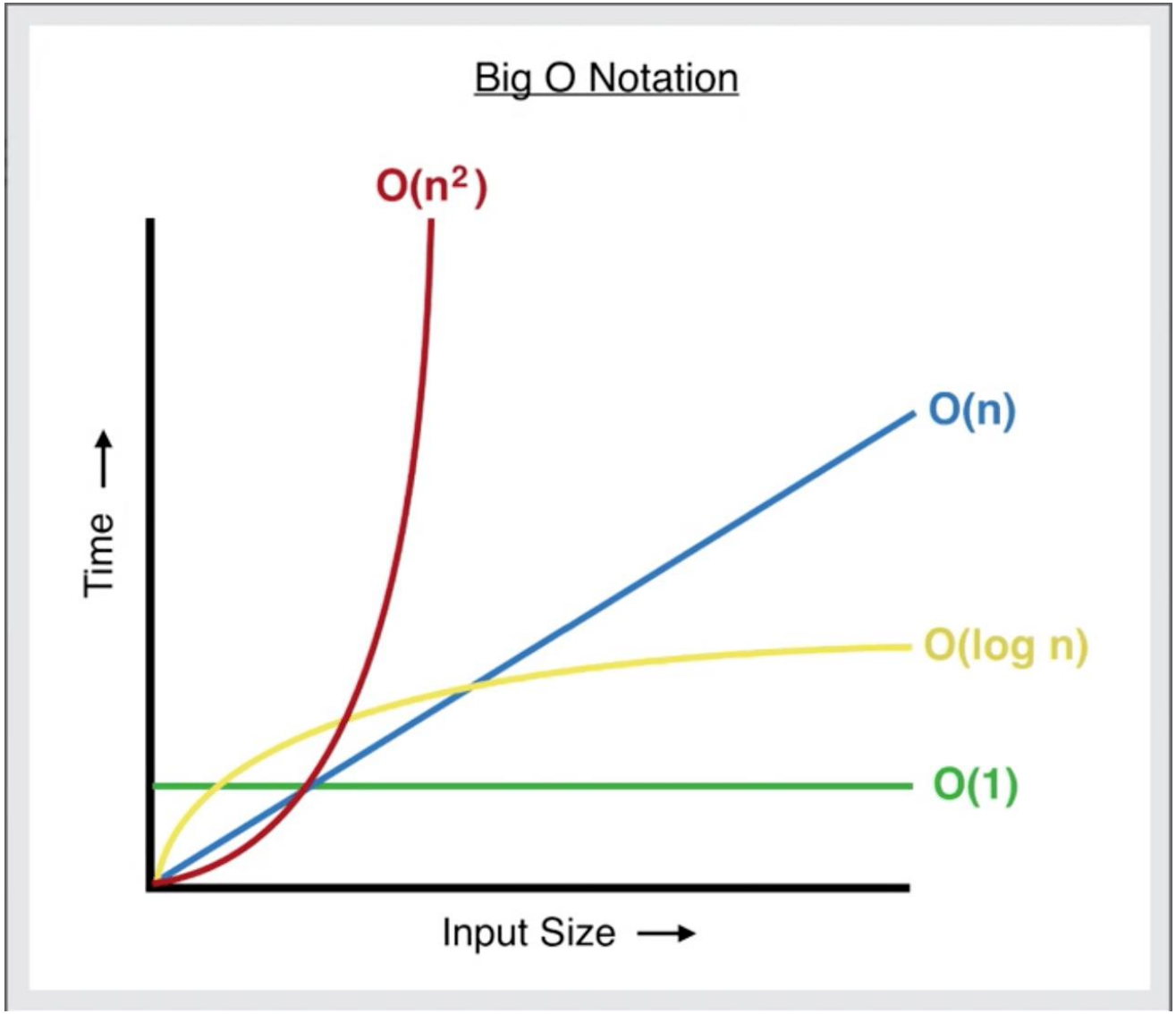
Архитектура SQLite



Скорость типичных операций

Стоимость операции	**нс (ns)**	**мкс (µs)**	**мс (ms)**
Получение значения из L1	0.5		
Ошибка предсказания перехода в CPU	5		
Получение значения из L2	7		
Mutex lock/unlock	25		
Получение значения из RAM	100		
Сжатие 1Кб методом Zippy	3 000	3	
Отправка 1Кб через 1Гбит/сек сеть	10 000	10	
Чтение 4Кб с SSD (случайный доступ)	150 000	150	
Чтение 1Мб из RAM (последовательный доступ)	**250 000**	**250**	
Round trip внутри одного датацентра	500 000	500	
Чтение 1Мб из SSD (последовательный доступ)	**1 000 000**	**1 000**	**1**
Позиционирование HDD	10 000 000	10 000	10
Чтение 1Мб из HDD (последовательный доступ)	**20 000 000**	**20 000**	**20**
Round trip между США и Нидерландами	150 000 000	150 000	150

Алгоритмическая сложность, время обработки



Алгоритмическая сложность, время обработки

Если компьютер выполняет миллиард операций в секунду ...

O()	Миллиард элементов	Триллион элементов
O(n)	1 секунда	16 минут
$O(\log_2 n)$	30 наносекунд	40 наносекунд
$O(n*\log_2 n)$	30 секунд	11 часов
$O(n^2)$	32 года	32 миллиона лет

Индексирование

Основные данные

Id	Фамилия	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
5	Иванова	Ольга	19
7	Антонов	Борис	43
11	Ливанов	Антон	25
15	Козлов	Сергей	26

Поиск по фамилии

Сколько записей нужно прочитать?

Индексирование

Основные данные

Id	Фамилия	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
5	Иванова	Ольга	19
7	Антонов	Борис	43
11	Ливанов	Антон	25
15	Козлов	Сергей	26

Поиск по фамилии

O(N) операций

Индексирование

= Логическое упорядочение физических записей

Основные данные

Id	Фамилия	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
5	Иванова	Ольга	19
7	Антонов	Борис	43
11	Ливанов	Антон	25
15	Козлов	Сергей	26

Поиск по фамилии

O(N) операций

Индекс

ID	Фамилия
7	Антонов
5	Иванова
15	Козлов
11	Ливанов
1	Петров
2	Сидоров

Поиск по фамилии

Сколько записей нужно прочитать?

Индексирование

= Логическое упорядочение физических записей

Основные данные

Id	Фамилия	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
5	Иванова	Ольга	19
7	Антонов	Борис	43
11	Ливанов	Антон	25
15	Козлов	Сергей	26

Поиск по фамилии

O(N) операций

Индекс

ID	Фамилия
7	Антонов
5	Иванова
15	Козлов
11	Ливанов
1	Петров
2	Сидоров

Поиск по фамилии

$O(\log_2 N)$ операций

Индексирование

= Логическое упорядочение физических записей

Основные данные

Id	Фамилия	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
5	Иванова	Ольга	19
7	Антонов	Борис	43
11	Ливанов	Антон	25
15	Козлов	Сергей	26

Индекс

ID	Фамилия
7	Антонов
5	Иванова
15	Козлов
11	Ливанов
1	Петров
2	Сидоров

Поиск в индексированной таблице:

1. Дихотомический поиск в индексе
2. Определение Id записи в основной таблице
3. Дихотомический поиск по Id в основной таблице

Индексирование

= Логическое упорядочение физических записей

Основные данные

N	Фамилия (PK)	Имя	Возраст
1	Петров	Иван	20
2	Сидоров	Андрей	34
3	Иванова	Ольга	19
4	Антонов	Борис	43
5	Ливанов	Антон	25
6	Козлов	Сергей	26

Индекс

Фамилия (FK)	Номер записи
Антонов	4
Иванова	3
Козлов	6
Ливанов	5
Петров	1
Сидоров	2

Поиск в индексированной таблице:

1. Дихотомический поиск в индексе
2. Определение номера записи в основной таблице
3. Считывание нужной записи в основной таблице

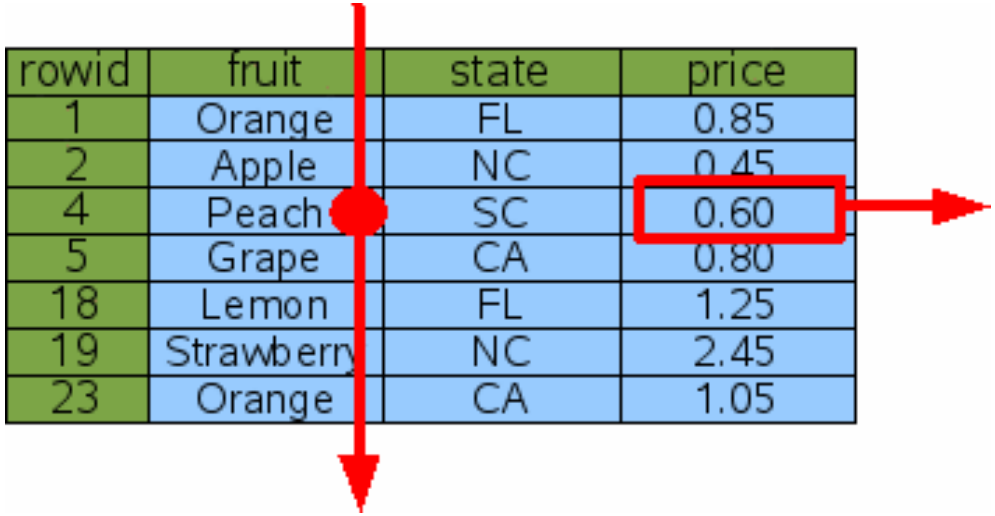
Индексированная таблица

Важно: в идеале индексный файл имеет малый размер и полностью считывается в оперативную память за один раз.

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Логическое представление таблицы fruits

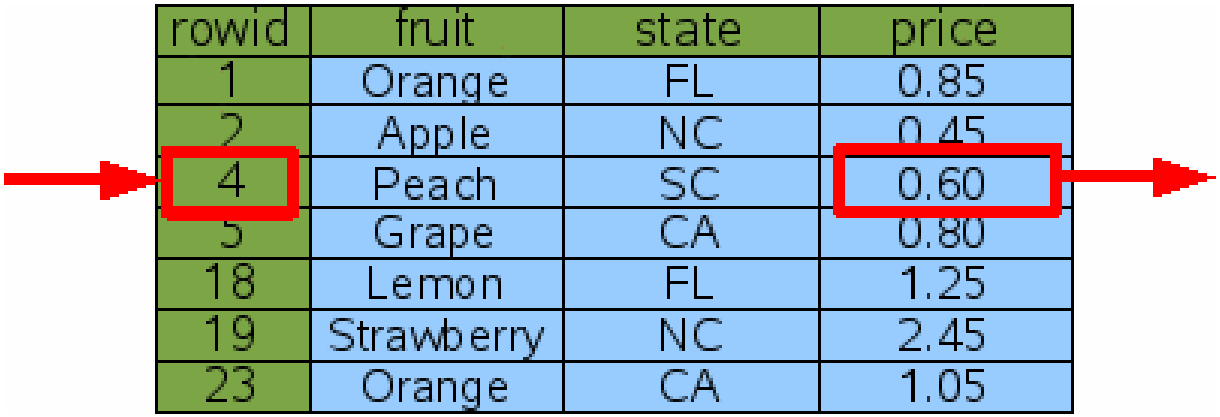
SELECT price FROM fruits WHERE fruit='Peach';



rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Сканирование по всей таблице => N операций

SELECT price FROM fruits WHERE rowid=4;



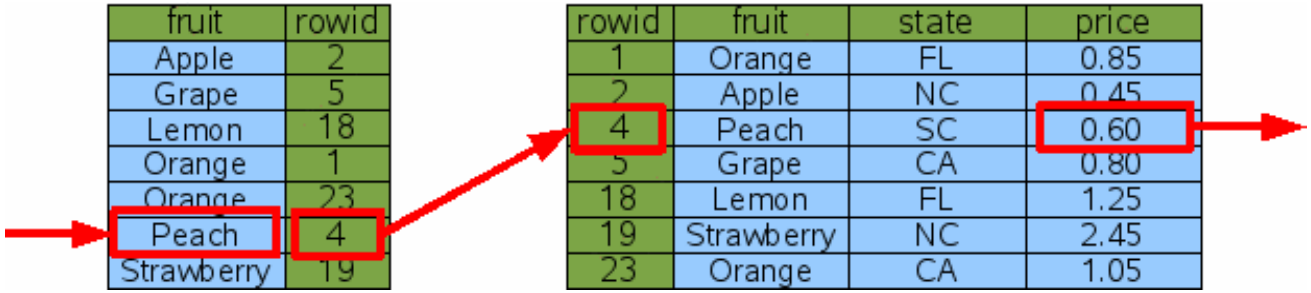
rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Поиск по rowid => $\log_2 N$ операций

CREATE INDEX Idx1 ON fruits(fruit);

Логическое представление индекса по столбцу fruit

SELECT price FROM fruits WHERE fruit='Peach';

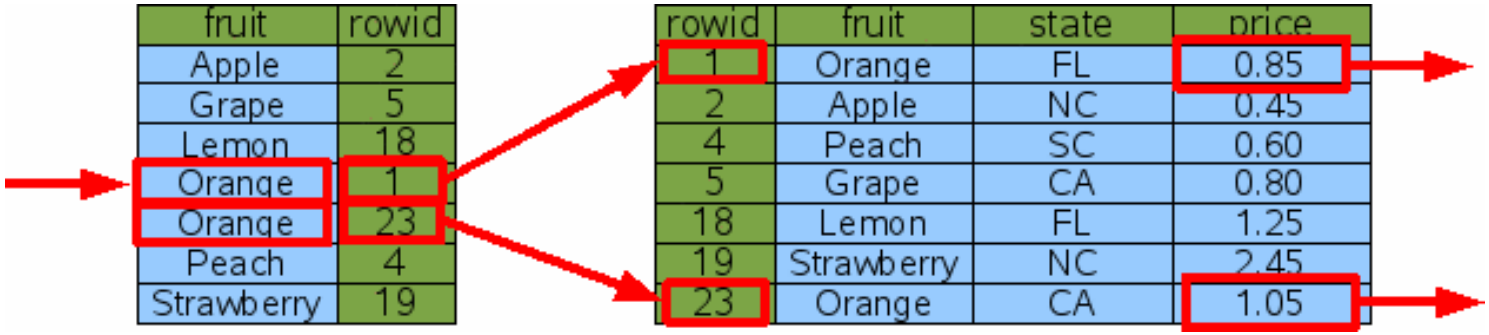


fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Поиск по индексу, один результат => $2*\log_2 N$ операций

SELECT price FROM fruits WHERE fruit='Orange';

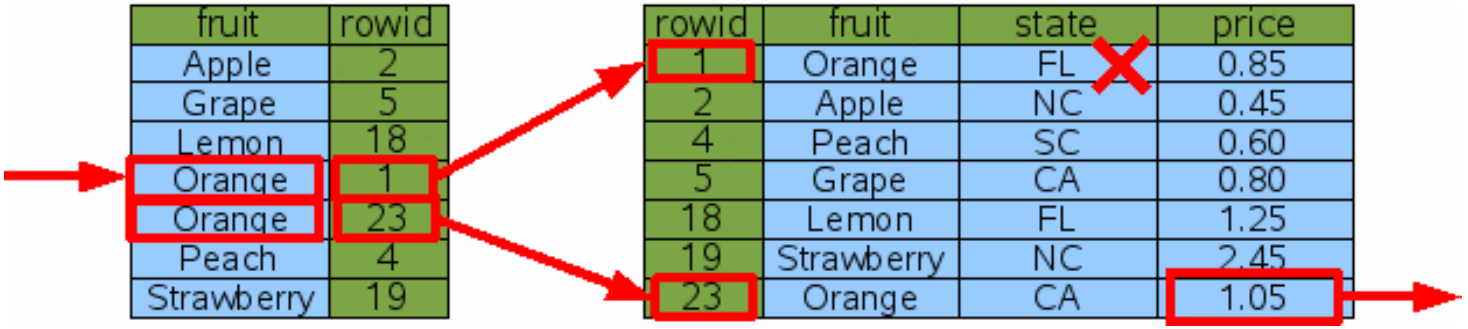


fruit	rowid
Apple	2
Grape	5
Lemon	18
Orange	1
Orange	23
Peach	4
Strawberry	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

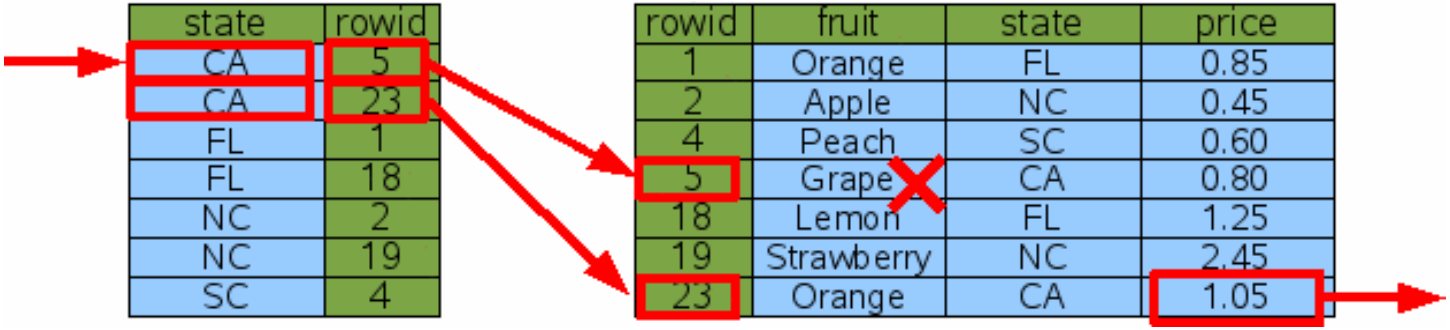
Несколько строк в результате => $(K+1)*\log_2 N$ операций

SELECT price FROM fruits WHERE fruit='Orange' AND state='CA';



Условие AND в WHERE

CREATE INDEX Idx2 ON fruits(state);

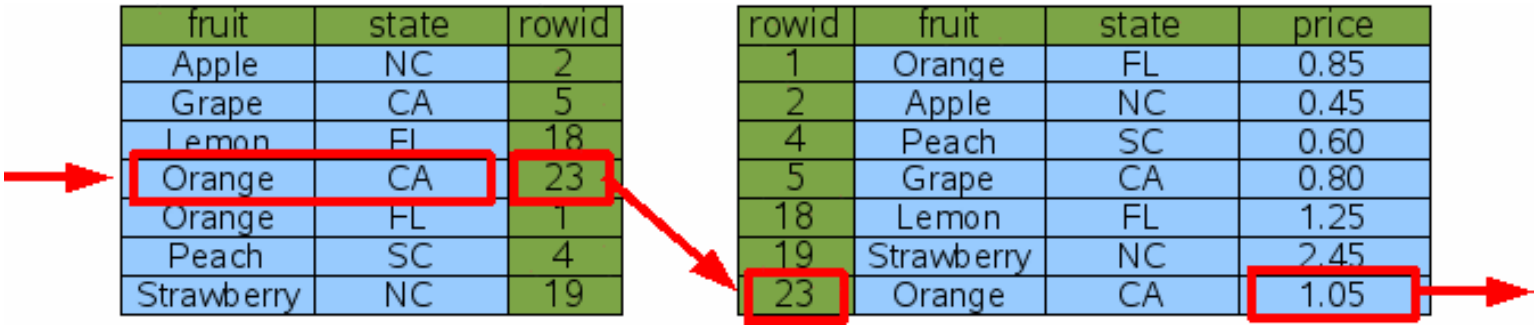


CREATE INDEX Idx3 ON Fruits(fruit, state);

fruit	state	rowid
Apple	NC	2
Grape	CA	5
Lemon	FL	18
Orange	CA	23
Orange	FL	1
Peach	SC	4
Strawberry	NC	19

Составной индекс

SELECT price FROM fruits WHERE fruit='Orange' AND state='CA';



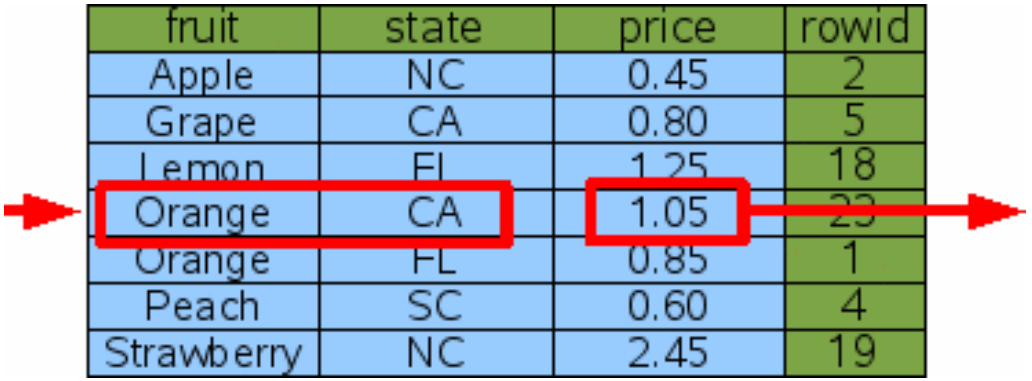
$2*\log_2 N$ операций

CREATE INDEX Idx4 ON Fruits(fruit, state, price);

fruit	state	price	rowid
Apple	NC	0.45	2
Grape	CA	0.80	5
Lemon	FL	1.25	18
Orange	CA	1.05	23
Orange	FL	0.85	1
Peach	SC	0.60	4
Strawberry	NC	2.45	19

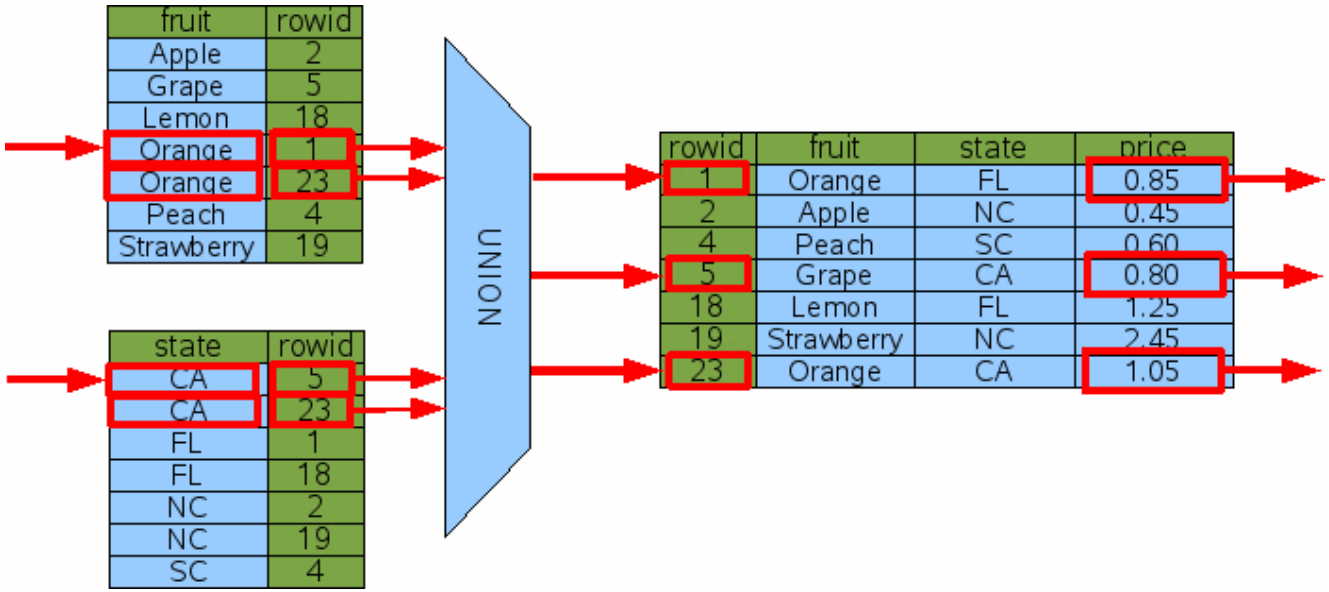
Покрывающий индекс

SELECT price FROM fruits WHERE fruit='Orange' AND state='CA';



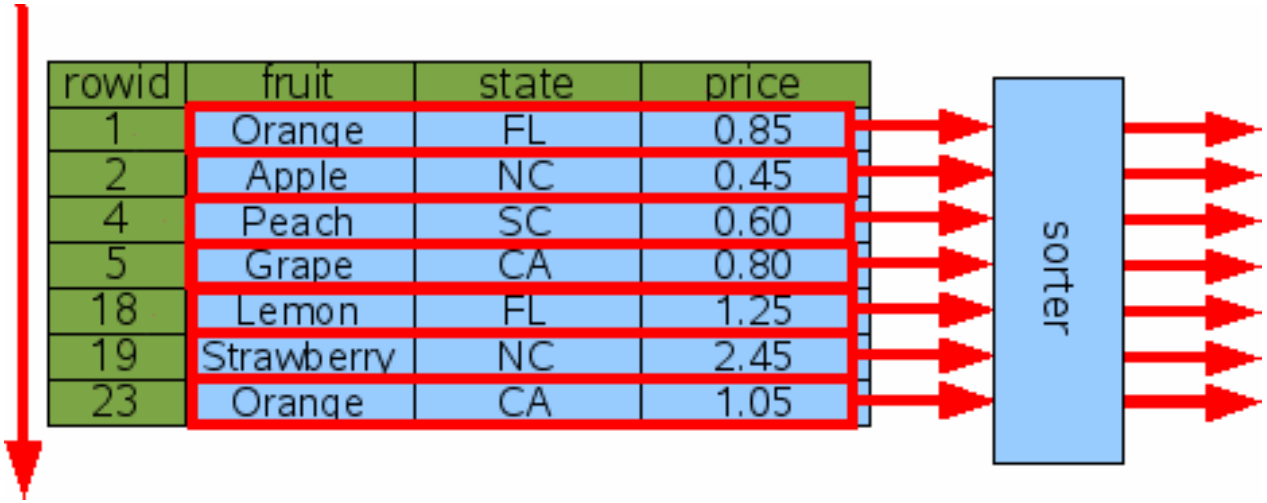
$\log_2 N$ операций

SELECT price FROM Fruits WHERE fruit='Orange' OR state='CA';



Условие OR в WHERE

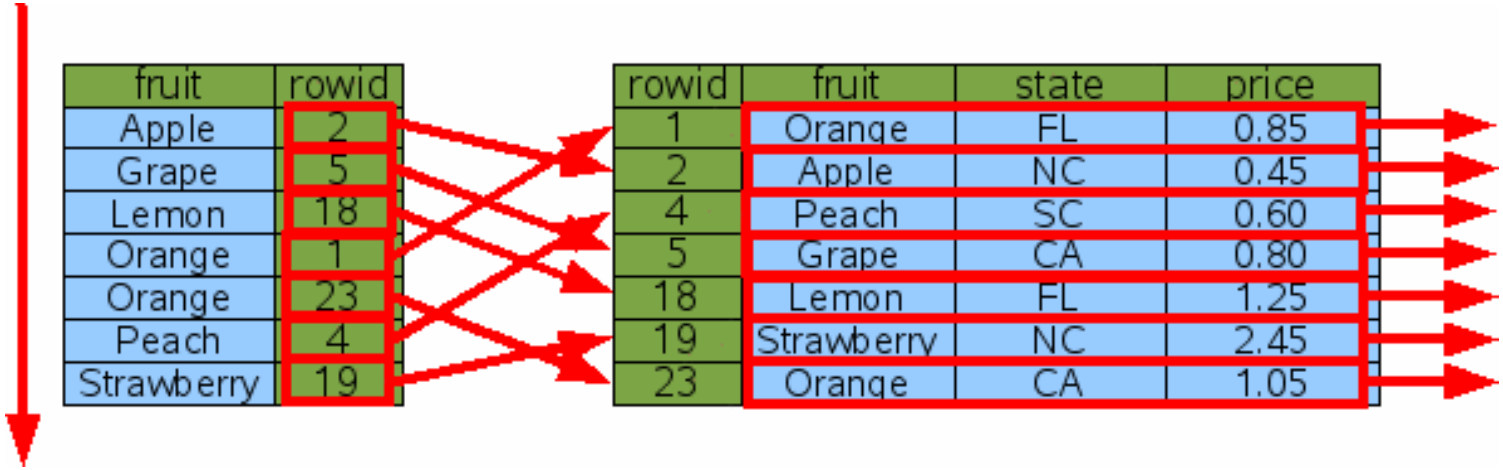
SELECT * FROM fruits ORDER BY fruit;



Сортировка без индекса => N*log2N операций

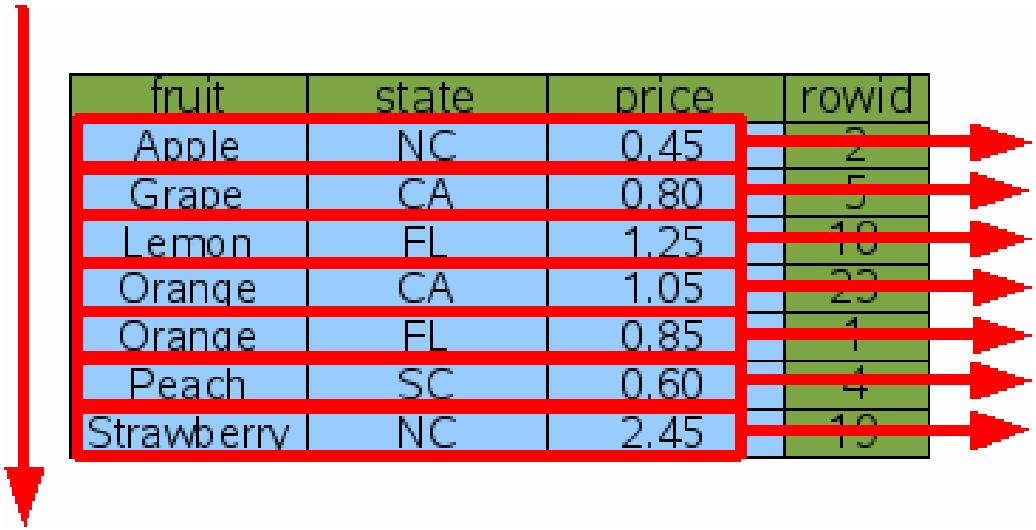
Также требуются временные хранилища для сортировки (b-tree)

SELECT * FROM fruits ORDER BY fruit;



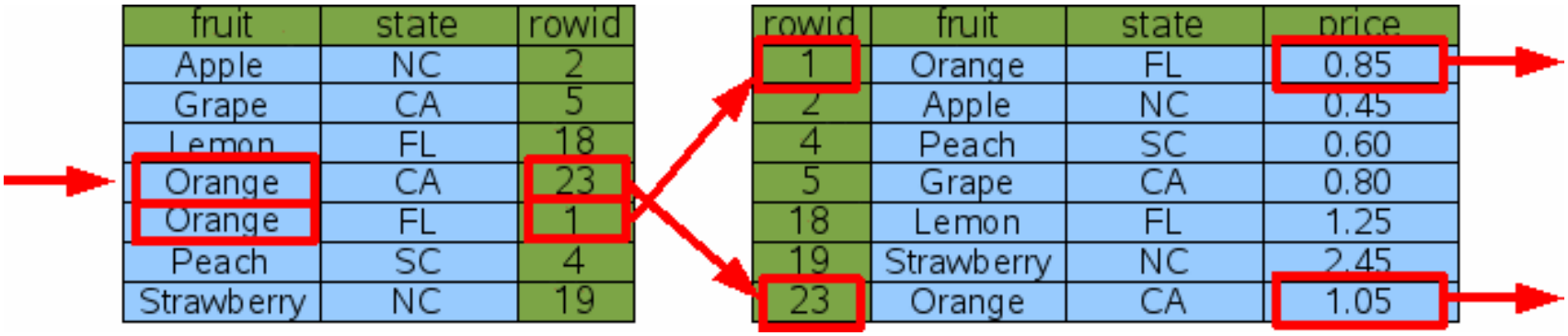
Сортировка с индексом => N*log2N операций

SELECT fruits, state, price FROM fruits ORDER BY fruit;



Сортировка с покрывающим индексом => N операций

SELECT price FROM fruits WHERE fruit='Orange' ORDER BY state

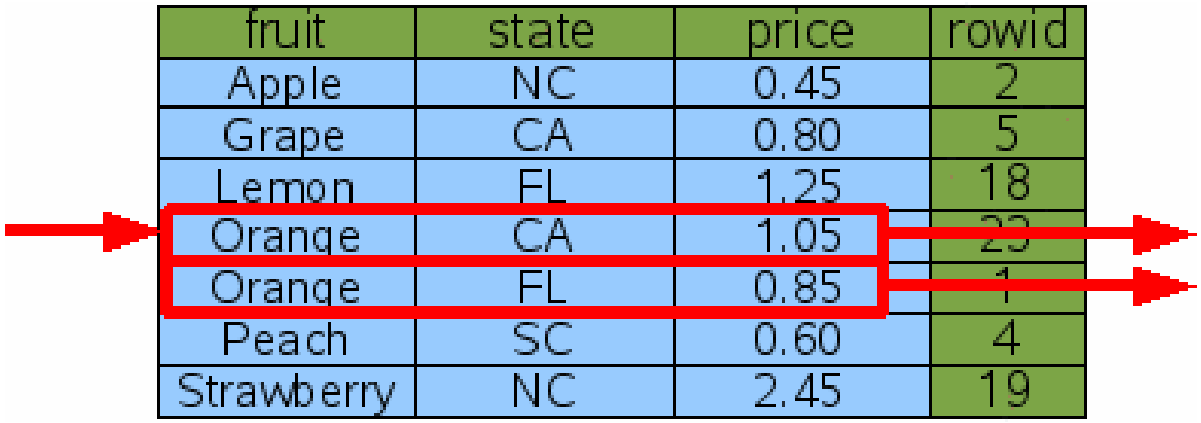


fruit	state	rowid
Apple	NC	2
Grape	CA	5
Lemon	FL	18
Orange	CA	23
Orange	FL	1
Peach	SC	4
Strawberry	NC	19

rowid	fruit	state	price
1	Orange	FL	0.85
2	Apple	NC	0.45
4	Peach	SC	0.60
5	Grape	CA	0.80
18	Lemon	FL	1.25
19	Strawberry	NC	2.45
23	Orange	CA	1.05

Поиск и сортировка с составным индексом => $O(\log_2 N)$ операций

SELECT price FROM fruits WHERE fruit='Orange' ORDER BY state



fruit	state	price	rowid
Apple	NC	0.45	2
Grape	CA	0.80	5
Lemon	FL	1.25	18
Orange	CA	1.05	23
Orange	FL	0.85	1
Peach	SC	0.60	4
Strawberry	NC	2.45	19

Поиск и сортировка с покрывающим индексом

EXPLAIN QUERY PLAN

- SCAN - перебор всех строк в таблице
- SEARCH - выбор некоторых строк
- USE TEMP B-TREE FOR xxx - временное хранилище для сортировки строк

Плата за индексы

- Место
- Перестройка индекса при обновлении данных