ТЕХНОТРЕК

Занятие №7

Оптимизация

Дина Сафина

Оптимизация

1. Индексирование
2. Хранение индексов
3. Профилирование
4. Выполнение запросов в MySQL
5. EXPLAIN
6. Рецепты оптимизации

Всё как-то медленно

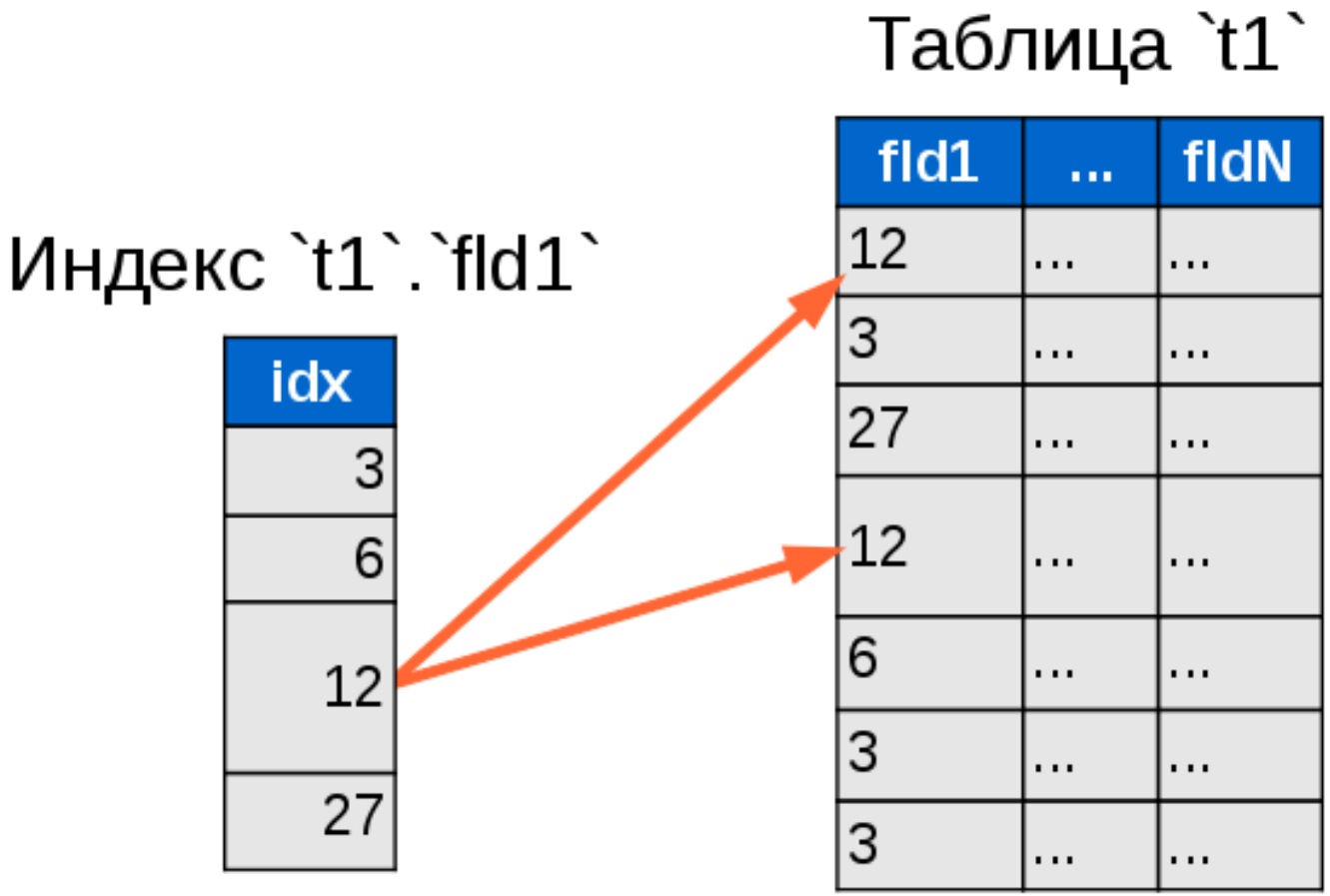
Что делать?

- Перепроектируй
- Проиндексируй
- Перепиши запрос

ТЕХНОТРЕК

Индексирование

Что такое индекс?



Типы индексов

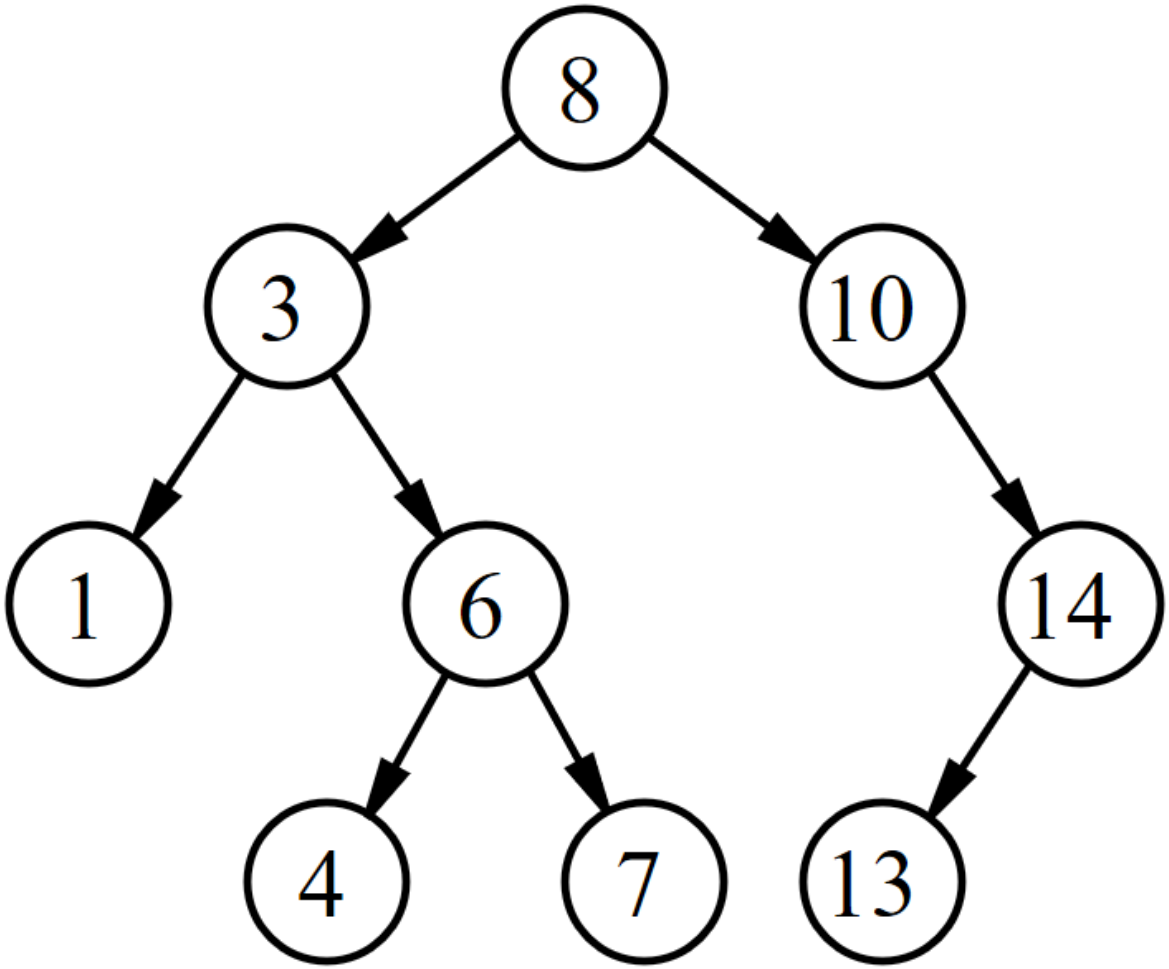
- B-дерево
- Hash-индексы
- Пространственные индексы
- Полнотекстовые индексы
- Кластерные индексы

Использование индексов

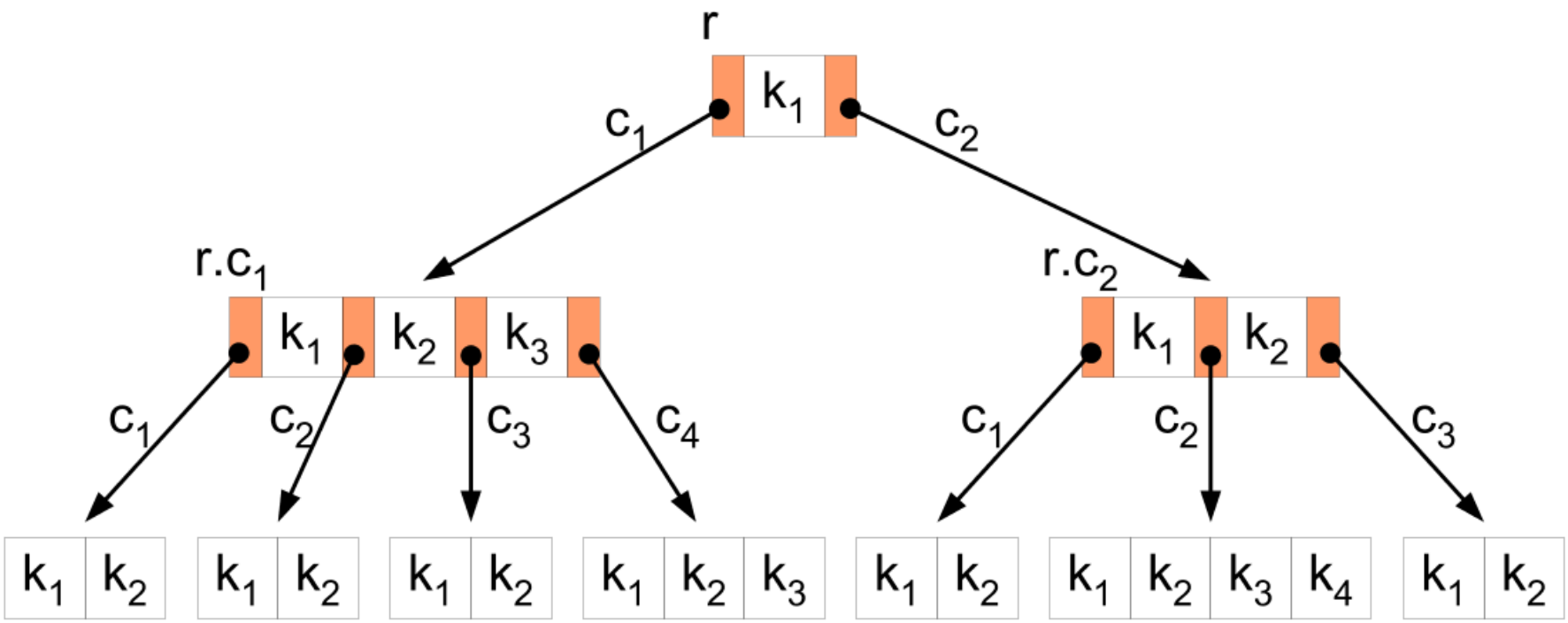
+ Быстрая фильтрация
+ Быстрый JOIN
+ Уменьшение количества просматриваемых записей
+ Быстрый поиск MAX и MIN
+ Быстрая сортировка и группировка
+ Извлечения данных прямиком из индексного файла

- Замедление операция INSERT, UPDATE, DELETE
- Занимают дополнительный объём памяти

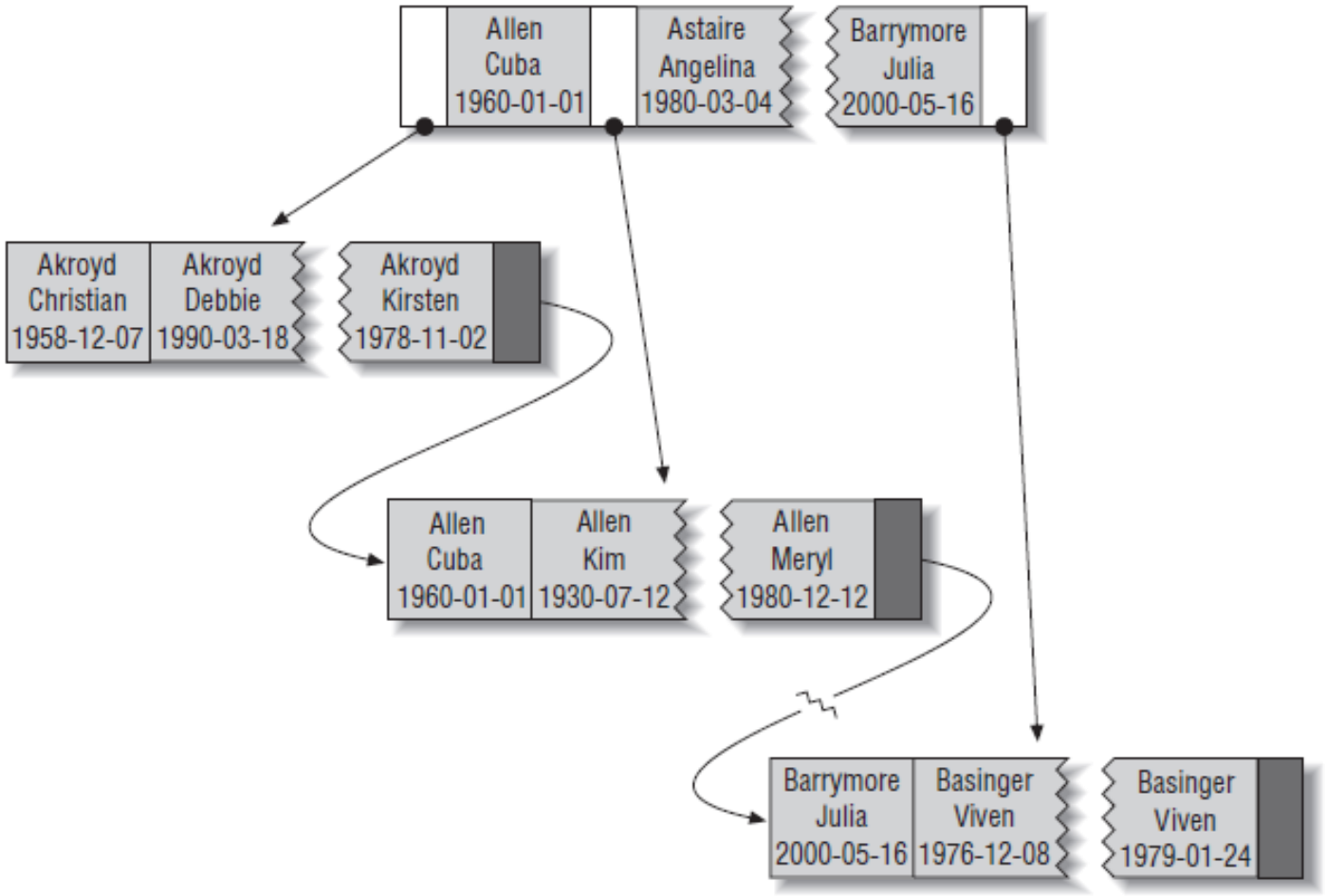
Бинарное дерево



B-дерево


r
k1
c1
c2
r.c1
k1
k2
k3
r.c2
k1
k2
c1
c2
c3
c4
c1
c2
c3
k1 k2
k1 k2
k1 k2
k1 k2 k3
k1 k2
k1 k2 k3 k4
k1 k2

B-дерево. Многостолбцовый индекс



B-дерево. Многостолбцовый индекс

- Порядок полей имеет значение
- Первое поле – самое используемое в фильтрации/сортировке/агрегации
- Или с наибольшей селективностью
- Иногда наименее селективное поле идёт первым
- Тогда при фильтрации добавляется в IN (…)

- Несколько одиночных индексов – это плохо
- Хотя MySQL может использовать **слияние индексов**
- Это значит индексация хромает

B-дерево. Особенности

Можно:

- Поиск по полному значению
- Поиск по самому левому префиксу
- Поиск по префиксу столбца
- Поиск по диапазону значений
- Поиск по полному совпадению одной части и диапазону в другой части
- Запросы только по индексу

Нельзя:

- Поиск без использования левой части ключа
- Нельзя пропускать столбцы
- Оптимизация после поиска в диапазоне

Hash-индекс

fname	lname
Arjen	Lentz
Baron	Schwartz
Peter	Zaitsev
Vadim	Tkachenko

f(‘Arjen’) = 2323
f(‘Baron’) = 7437
f(‘Peter’) = 8784
f(‘Vadim’) = 2458

Ячейка	Значение
2323	Указатель на строку 1
2458	Указатель на строку 4
7437	Указатель на строку 2
8784	Указатель на строку 3

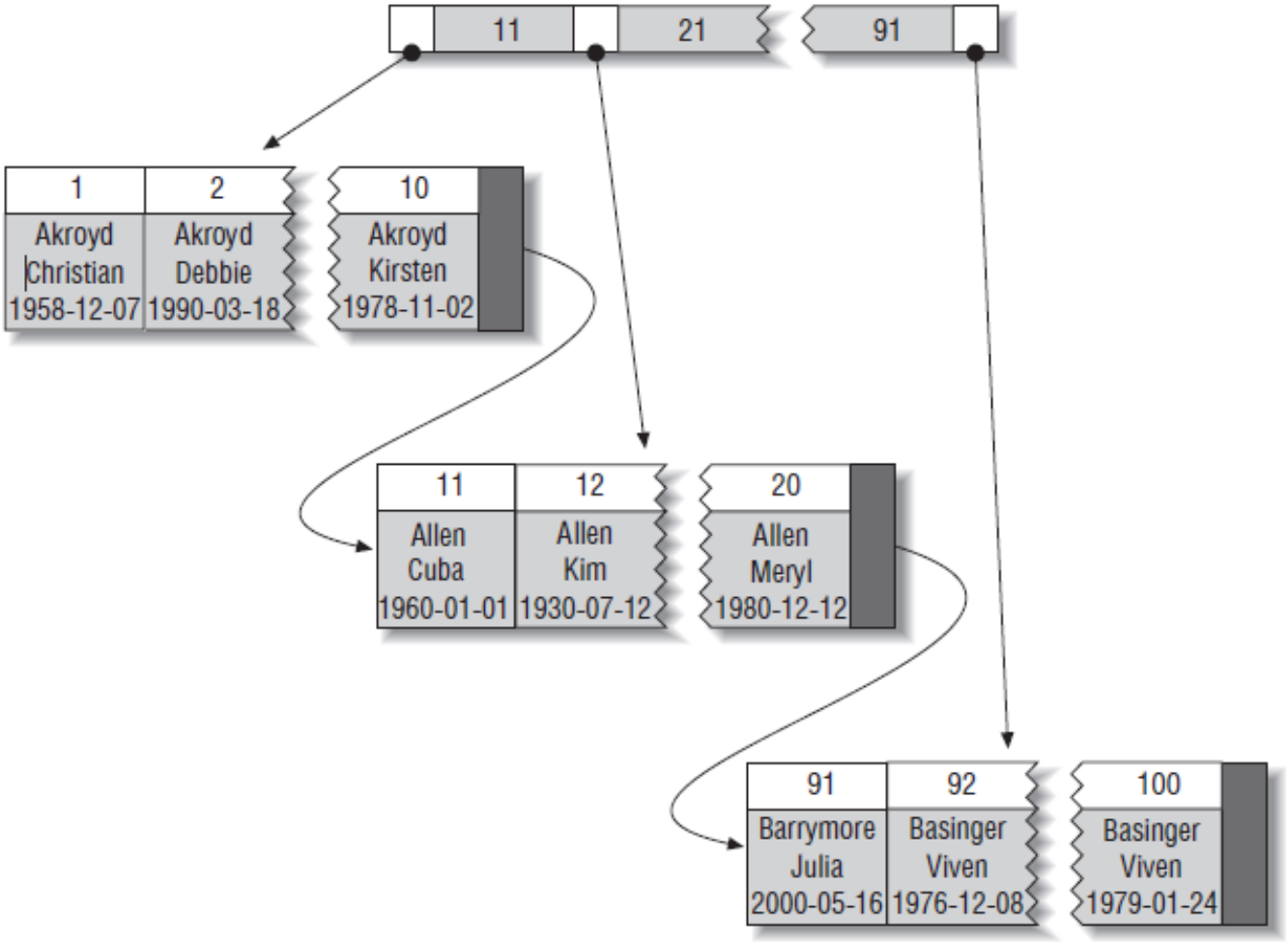
Hash-индекс. Особенности

- Строки читать придётся
- Не помогает в сортировке
- Поиск по частичному ключу не работает
- Сравнение только операторами =, IN() и <=>
- Быстро, если мало коллизий
- Медленно, если коллизий много

Hash-индекс. MySQL

- Поддерживается только движком **Memory**
- В InnoDB есть **адаптивные хэш-индексы**
- Можно построить собственные хэш-индексы

Кластерный индекс

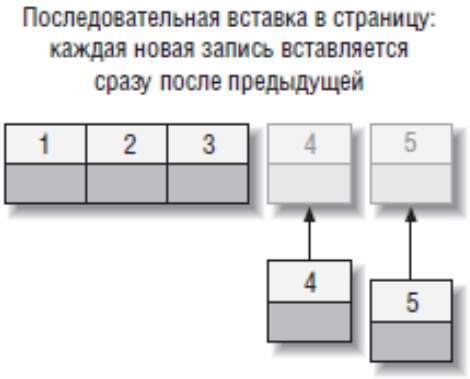


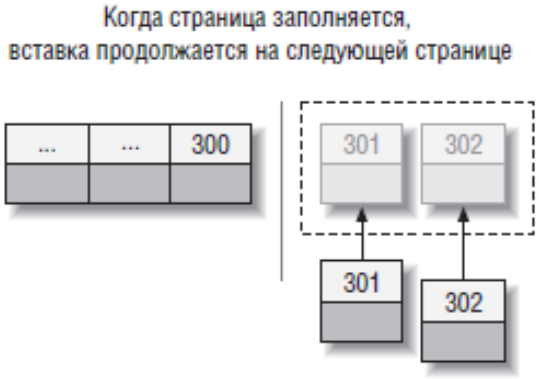
Кластерный индекс

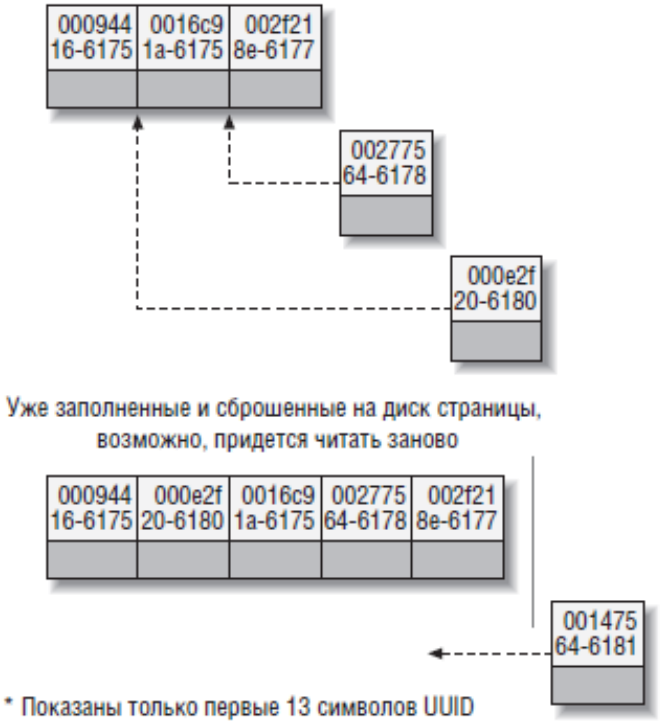
+ Связанные данные хранятся рядом
+ Быстрый доступ к данным
+ Вторичные ключи с указателями на первичные ключи могут не ходить в кластерный индекс

- Нет смысла для таблиц, хранимых в памяти
- При беспорядочных INSERT’ах используйте OPTIMIZE TABLE
- Дорогое обновление столбцов кластерного индекса
- Дорогое полное сканирование кластерных таблиц
- Вторичные индексы больше из-за того, что содержат первичный ключ, а не указатель
- Для доступа по вторичному индексу читаются два индекса

Кластерный индекс. AUTO_INCREMENT vs UUID







Кастомные индексы

- Псевдо-хэш-индексы
- Префиксные индексы
- Суффиксные индексы
- Функциональные индексы

Покрывающие индексы

Покрывающий индекс — индекс, который содержит все данные, необходимые для формирования запроса

Вторичные индексы в InnoDB могут быть покрывающими

Хэш-индексы, пространственные и полнотекстовые индексы не могут быть покрывающими

EXPLAIN: Using index в поле EXTRA

Плохие индексы

Дублирующиеся индексы — индексы одного типа, созданные на основе одного и того же набора столбцов в одинаковом порядке.

Избыточный индекс — индекс, чья функциональность входит в другой индекс.

Индекс (A) будет избыточным при наличии индекса (A, B).

Неиспользуемые индексы — индексы, которые сервер не задействует

CREATE INDEX Syntax

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

CREATE INDEX Example

```sql
CREATE INDEX part_of_name ON customer (name(10));
```

```sql
CREATE TABLE t1 (col1 INT, col2 INT, INDEX func_index ((ABS(col1))));
CREATE INDEX idx1 ON t1 ((col1 + col2));
CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);
ALTER TABLE t1 ADD INDEX ((col1 * 40) DESC);
```

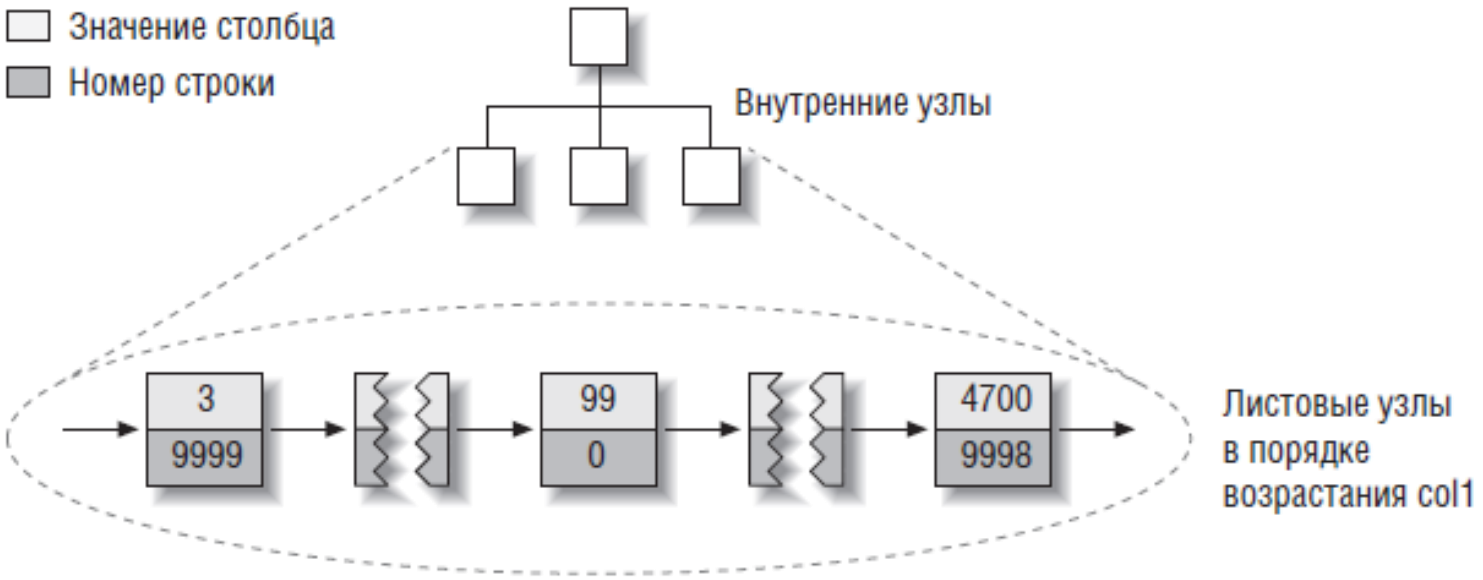
ТЕХНОТРЕК

Хранение индексов

MyISAM. Первичный индекс

```
CREATE TABLE
layout_test (
    col1 int NOT NULL,
    col2 int NOT NULL,
    PRIMARY KEY(col1),
    KEY(col2)
);
```

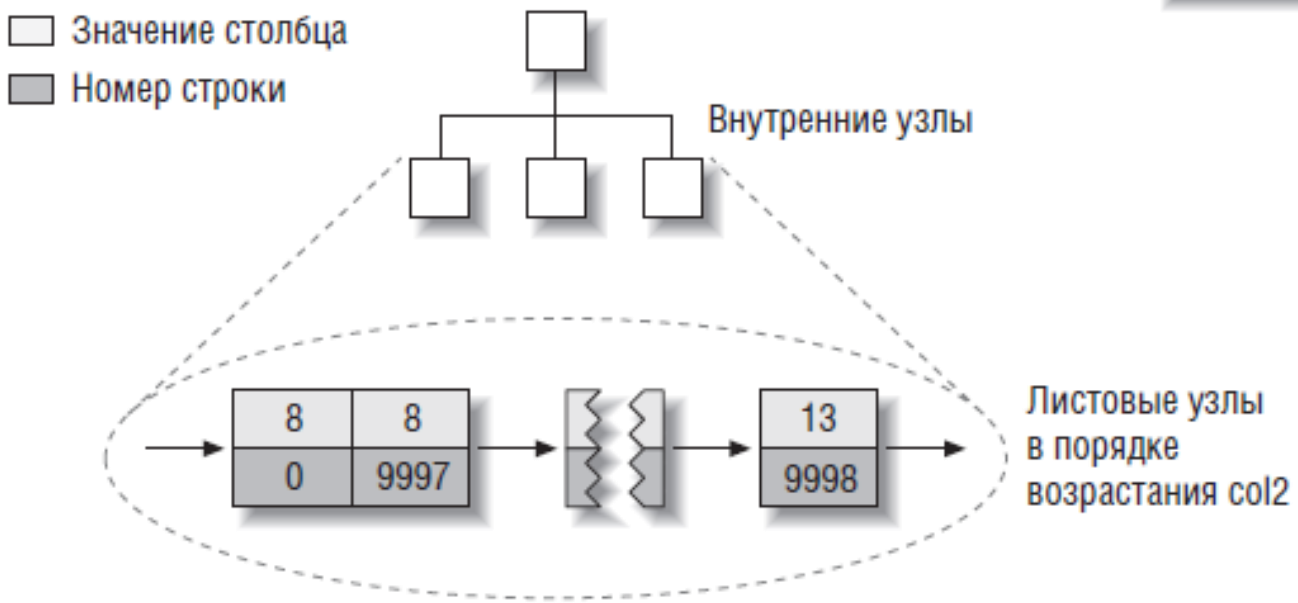
Номер строки	col1	col2
0	99	8
1	12	56
2	3000	62
9997	18	8
9998	4700	13
9999	3	93



MyISAM. Вторичный индекс

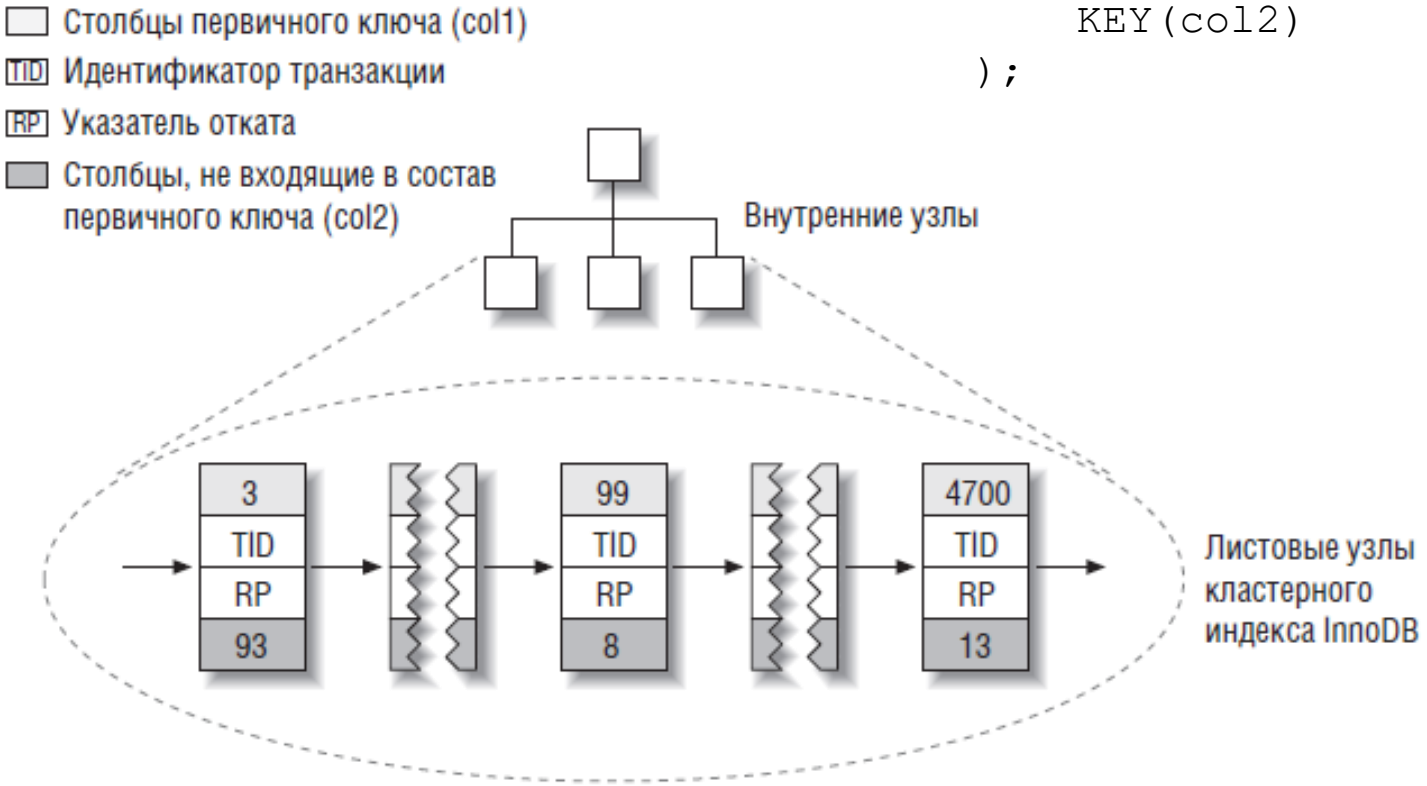
```
CREATE TABLE
layout_test (
    col1 int NOT NULL,
    col2 int NOT NULL,
    PRIMARY KEY(col1),
    KEY(col2)
);
```

Номер строки	col1	col2
0	99	8
1	12	56
2	3000	62
9997	18	8
9998	4700	13
9999	3	93



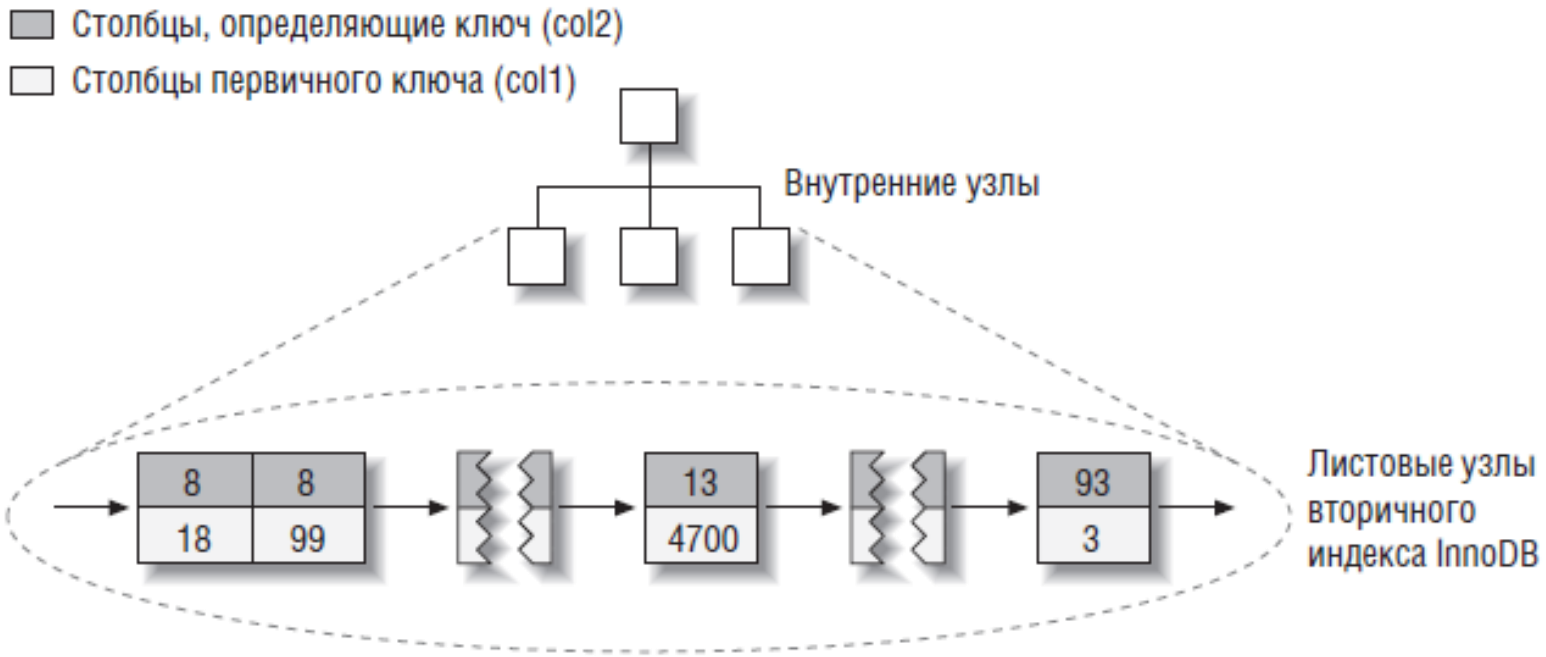
InnoDB. Первичный ключ

```
CREATE TABLE layout_test (
    col1 int NOT NULL,
    col2 int NOT NULL,
    PRIMARY KEY(col1),
    KEY(col2)
);
```



InnoDB. Вторичный ключ

```
CREATE TABLE layout_test (
    col1 int NOT NULL,
    col2 int NOT NULL,
    PRIMARY KEY(col1),
    KEY(col2)
);
```



ТЕХНОТРЕК

Профилирование

Профилирование

- SHOW PROFILE;
- SHOW PROFILES;

- Slow Query Log
- mysqldumpslow

Профайлер MySQL

```
mysql> SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)

```

Профайлер MySQL

```
mysql> SHOW PROFILES;
+----------+----------+--------------------------+
| Query_ID | Duration | Query                    |
+----------+----------+--------------------------+
|        0 | 0.000088 | SET PROFILING = 1        |
|        1 | 0.000136 | DROP TABLE IF EXISTS t1  |
|        2 | 0.011947 | CREATE TABLE t1 (id INT) |
+----------+----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000040 |
| creating table       | 0.000056 |
| After create         | 0.011363 |
| query end            | 0.000375 |
| freeing items        | 0.000089 |
| logging slow query   | 0.000019 |
| cleaning up          | 0.000005 |
+----------------------+----------+
7 rows in set (0.00 sec)
```

Профайлер MySQL

```
mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
4 rows in set (0.00 sec)
```

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

Не запрашиваю ли я лишние данные?

- Выбор всех строк для чтения X строк:
  – используй **LIMIT X**

- **SELECT ***
  – тебе нужны все эти данные? Серьёзно?
  – покрывающие индексы не помогут

- Повторный выбор одних и тех же данных:
  – кэшируй

Не многовато ли данных анализирует MySQL?

- Время отклика = Время работы + Время ожидания
  - высокая конкурентность?
  - блокировки?

- Проанализированные строки : Возвращённые строки
  - в идеале 1:1
  - агрегаты? Может сводные таблицы?

- Типы доступа от худшего к лучшему:
  - по индексу
  - по покрывающему индексу
  - Full Scan

Неинтуитивная декомпозиция

```
DELETE FROM messages
 WHERE created < DATE_SUB(NOW(), INTERVAL 3 MONTH);
```

```
row affected = 0
do {
    rows_affected = do_query("DELETE FROM messages \
                              WHERE created < DATE_SUB(NOW(), INTERVAL 3 MONTH) \
                              LIMIT 10000")
} while rows_affected > 0
```

Неинтуитивная декомпозиция

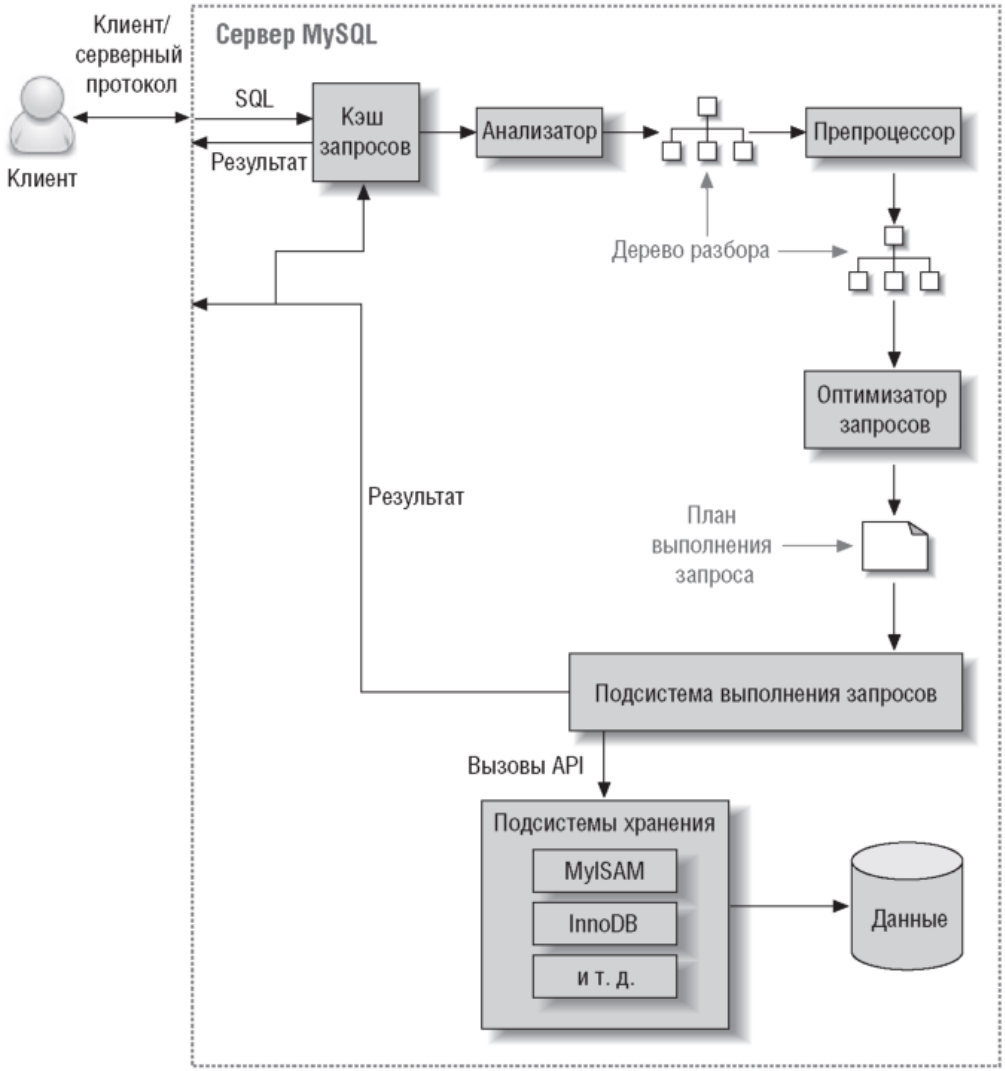
```
SELECT * FROM tag
  JOIN tag_post ON tag_post.tag_id = tag.id
  JOIN post ON tag_post.post_id = post.id
 WHERE tag.tag = 'mysql';
```

```
SELECT * FROM tag WHERE tag = 'mysql';
SELECT * FROM tag_post WHERE tag_id = 1234;
SELECT * FROM post WHERE post.id IN (123, 456, 9098);
```

ТЕХНОТРЕК

Выполнение запросов в MySQL

Выполнение запросов в MySQL



1. Отправка SQL-команды серверу

- Полудуплексный протокол
- Клиент обязан получить весь результирующий набор
- MySQL выталкивает строки, а не клиент вытягивает
- Клиент может весь запрос разместить в памяти
- Некоторые клиенты могут забирать по одной строке
- У каждого соединения есть состояние. Некоторые: sleep, query, locked, analyzing, statistics, copying to tmp table, sorting result, sending data

2. Кэш запросов

- Запрос должен быть идентичным включая даже регистр
- Проверяются привелеги
- Если всё хорошо, закэшированный результат сразу отправляется клиенту

3. Анализатор и препроцессор

- Анализатор разбивает запрос на лексемы и строит дерево разбора
- Анализатор проверяет синтаксис, порядок, парность кавычек и т.д.

- Препроцессор проверяет, что все таблицы и столбцы существуют, а ссылки на столбцы не допускают неоднозначного толкования
- Препроцессор проверяет привелегии

4. Оптимизатор

- Превращает дерево разбора в план выполнения
- План выполнения – дерево инструкций
- В MySQL затратный оптимизатор
- Иногда он ошибается
- Но реже, чем ты
- Статическая и динамическая оптимизации
- Про таблицы и индексы знают подсистемы хранения
- Важная часть оптимизатора – оптимизатор соединений
- Всё рассматривается как соединение таблиц
- Соединения методом NESTED LOOPS
- Если индекса для сортировки нет, включается файловая сортировка

4. Оптимизатор

Примеры оптимизации:

— изменение порядка соединений
— применение алгебраических правил
— выбор COUNT(), MIN(), MAX() из индексов
— вычисление константных выражений
— раннее завершение (например, LIMIT)

4. Подсистема выполнения запросов

- Следует инструкциям плана выполнения запросов

- Дёргает API, который поддерживает каждая система хранения

5. Возврат результатов клиенту

- На этом этапе план запроса и результат его выполнения могут помещаться в кэш

Подсказки оптимизатору

Подсказка	Описание
HIGH_PRIORITY, LOW_PRIORITY	Приоритет относительно других команд
DELAYED	Отложенная вставка
STRAIGHT_JOIN	Соединение таблиц в указанном порядке
SQL_SMALL_RESULT, SQL_BIG_RESULT	Сортировка для группировки (DISTINCT/GROUP BY) в памяти или на диске
SQL_BUFFER_RESULT	Результат поместить во временную таблицу, чтобы освободить блокировки
SQL_CACHE, SQL_NO_CACHE	Помещать запрос в кэш или нет
SQL_CALC_FOUND_ROWS	Считать весь набор несмотря на LIMIT
FOR UPDATE, LOCK IN SHARE MODE	Управление блокировками
USE INDEX, IGNORE INDEX, FORCE INDEX	Управление использованием индекса

ТЕХНОТРЕК

EXPLAIN

EXPLAIN

- Ничего не знает про хранимые подпрограммы
- Ничего не знает о динамической части оптимизации
- Часть информации – всего лишь оценка
- Не показывает всего
- Не делает различий между некоторыми операциями

EXPLAIN. Список полей

Поле	Назначение	Возможные значения
id	Номер обрабатываемого запроса	NULL, если UNION
select_type	Категория запроса	…
table	Название таблицы	<derivedN>, <unionX,Y>
type	Метод доступа	…
possible_keys	Возможные индексы	
key	Выбранный индекс	
key_len	Используется байтов индекса	
ref	Что используется в индексе	Поля из предыдущих таблиц
rows	Сколько строк придётся читать	Примерная оценка
filtered	Процент подходящих строк	Пессимистичная оценка
Extra	Дополнительная информация	…

EXPLAIN. select_type

- **SIMPLE** – простой запрос
- **SUBQUERY** – подзапрос в SELECT
- **DERIVED** – подзапрос во FROM
- **UNION** – второй и последующие таблицы в UNION
- **UNION RESULT** – временная таблица с UNION’ом
- **DEPENDENT** – зависимость от внешних запросов
- **UNCACHEABLE** – что-то мешает кэшированию

EXPLAIN. type

- **ALL** – Full Scan
- **index** – Full Scan в порядке, указанном индексом
- **range** – просмотр диапазона индекса
- **ref** – доступ по неуникальному индексу
- **eq_ref** – доступ по индексу с возвратом единственного значения
- **const**, **system** – замена выражения константой
- **NULL** – разрешение запроса на фазе оптимизации

EXPLAIN. Extra

- **Using index** – покрывающие индексы
- **Using where** – сервер фильтрует строки после фильтрации строк подсистемой хранения
- **Using temporary** – временная таблица для сортировки
- **Using filesort** – файловая сортировка
- **range checked for each record** – поиск индекса на лету

ТЕХНОТРЕК

Рецепты оптимизации

COUNT()

- COUNT(col) – считает NOT NULL значения
- COUNT(*) – считает количество строк
- В MyISAM COUNT(*) без WHERE мгновенный
- EXPLAIN оценит приблизительное количество строк

- Количество красных и синих штучек:

```
SELECT SUM(IF(color='blue', 1, 0)) AS blue,
       SUM(IF(color='red', 1, 0)) AS red
  FROM items;

SELECT COUNT(color='blue' OR NULL) AS blue,
       COUNT(color='red' OR NULL) AS red
  FROM items;
```

JOIN

- Индекс на соединительный столбец второй таблицы
- GROUP BY и ORDER BY по полям из одной таблицы
- Будьте осторожны при смене версии MySQL

JOIN’ы лучше подзапросов

GROUP BY WITH ROLLUP

Очень плохо оптимизируется

Иногда лучше дозапрос с UNION’ом

LIMIT и OFFSET

- Индекс на поле сортировки
- Синтетическое поле с номером строки (и индекс)
- Смещение с покрывающим индексом

```
SELECT film_id, description
  FROM sakila.film
 ORDER BY title LIMIT 50, 5;

SELECT film.film_id, film.description
  FROM sakila.film
 INNER JOIN (
       SELECT film_id FROM sakila.film
        ORDER BY title LIMIT 50, 5
       ) AS lim USING(film_id);
```

UNION

- UNION ALL всегда лучше
- Опускайте WHERE, LIMIT, ORDER внутрь запросов

SQL_CALC_FOUND_ROWS + LIMIT

- Считает всю выборку и отбрасывает ненужное
- Выбирайте на одну запись больше
- Выберите 1000 строк («более 1000 результатов»)
- Иногда достаточно COUNT(*)
- EXPLAIN выдаёт оценку количества строк