Индексы в MySQL

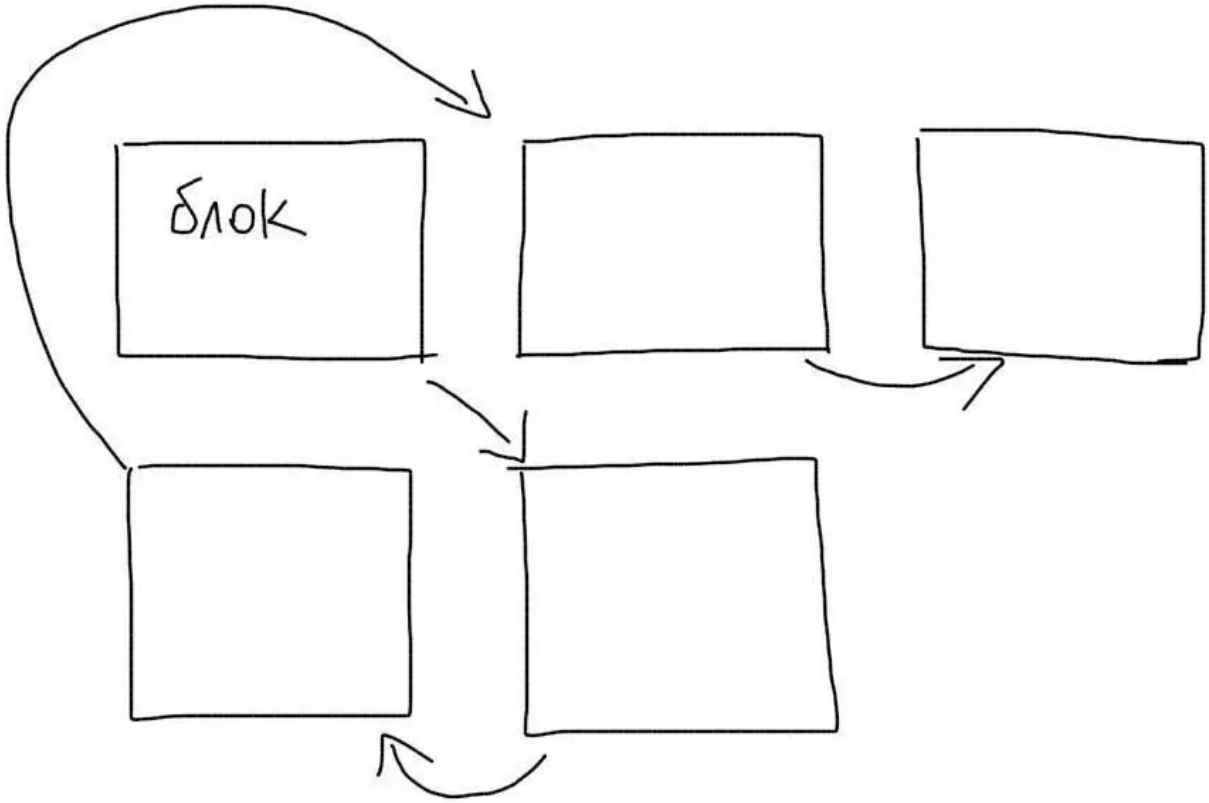
highload.today/indeksy-v-mysql

admin 8 листопада 2019 р.

Индексы в MySQL (Mysql indexes) — отличный инструмент для оптимизации SQL запросов. Чтобы понять, как они работают, посмотрим на работу с данными без них.

1. Чтение данных с диска

На жестком диске нет такого понятия, как файл. Есть понятие блок. Один файл обычно занимает несколько блоков. Каждый блок знает, какой блок идет после него. Файл делится на куски и каждый кусок сохраняется в пустой блок.



При чтении файла, мы по очереди проходимся по всем блокам и собираем файл из кусков. Блоки одного файла могут быть раскиданы по диску (фрагментация). Тогда чтение файла замедлится, т.к. понадобится прыгать разным участкам диска.

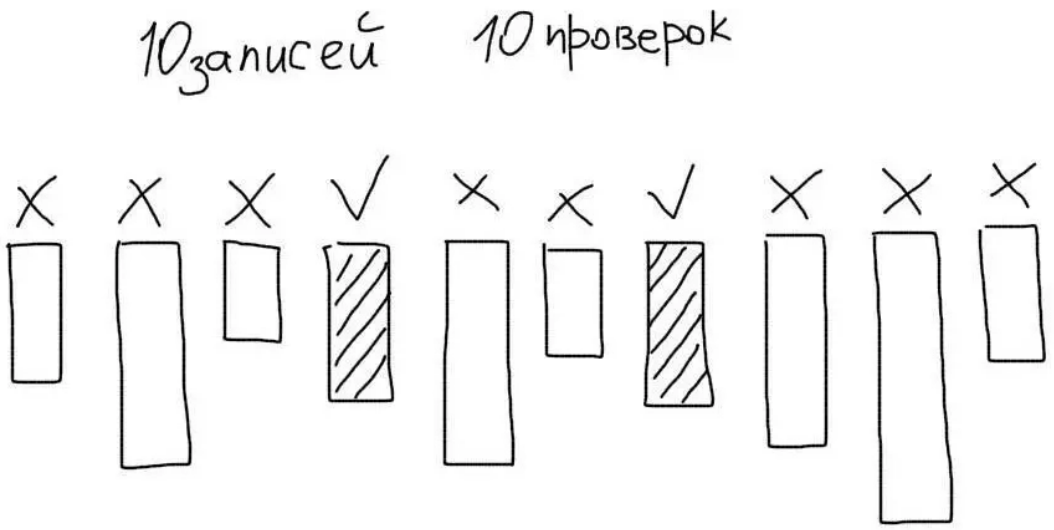
Когда мы ищем что-то внутри файла, нам понадобится пройтись по всем блокам, в которых он сохранен. Если файл очень большой, то и количество блоков будет значительным. Необходимость перепрыгивать с блока на блок, которые могут находиться в разных местах, сильно замедлит поиск данных.

2. Поиск данных в MySQL

Таблицы MySQL – это обычные файлы. Выполним запрос такого вида:

SELECT * FROM users WHERE age = 29
MySQL при этом открывает файл, где хранятся данные из таблицы **users**. А дальше — начинает перебирать весь файл, чтобы найти нужные записи.

Кроме этого, MySQL будет сравнивать данные в **каждой строке таблицы** со значением в запросе. Допустим работа ведется с таблицей, в которой есть 10 записей. Тогда MySQL прочитает все 10 записей, сравнит колонку age каждой из них со значением 29 и отберет только подходящие данные:

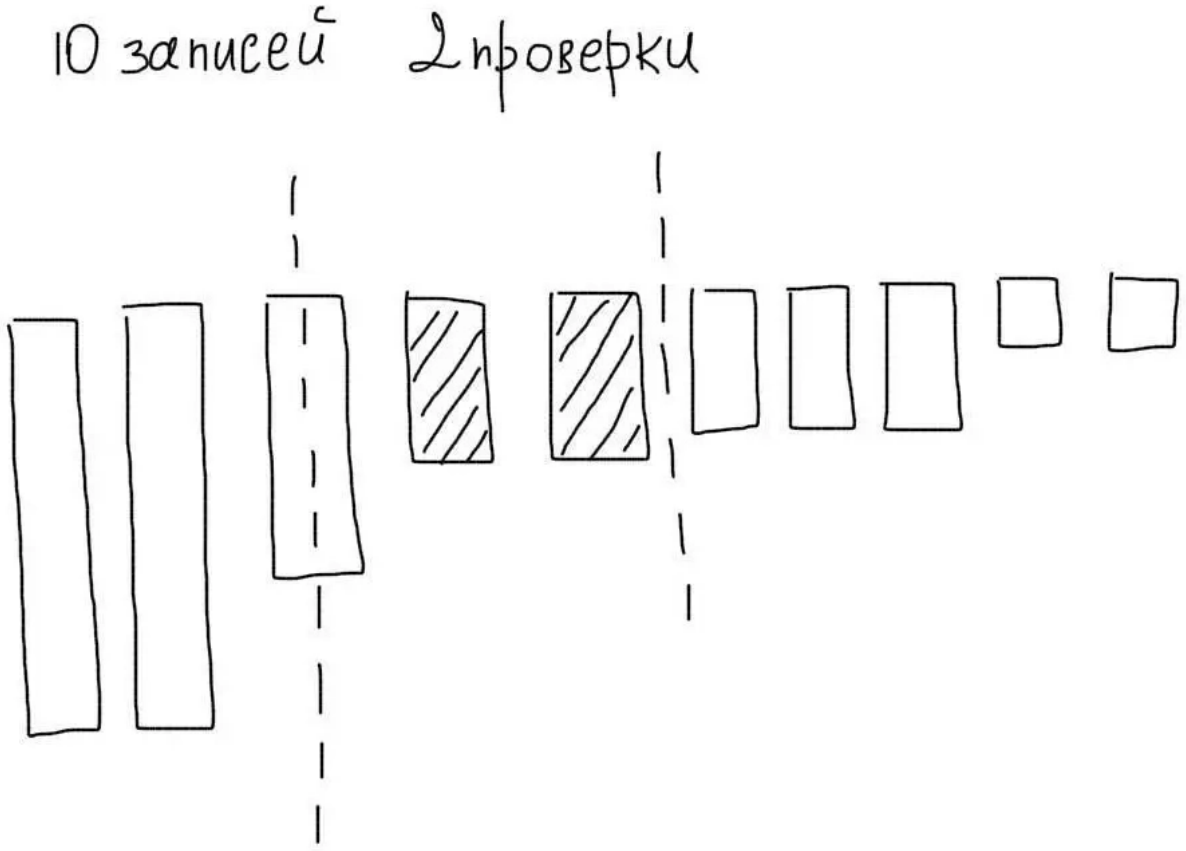


Итак, есть две проблемы при чтении данных:

- Низкая скорость чтения файлов из-за расположения блоков в разных частях диска (фрагментация).
- Большое количество операций сравнения для поиска нужных данных.

3. Сортировка данных

Представим, что мы отсортировали наши 10 записей по убыванию. Тогда используя алгоритм бинарного поиска, мы могли бы максимум за 4 операции отобрать нужные нам значения:

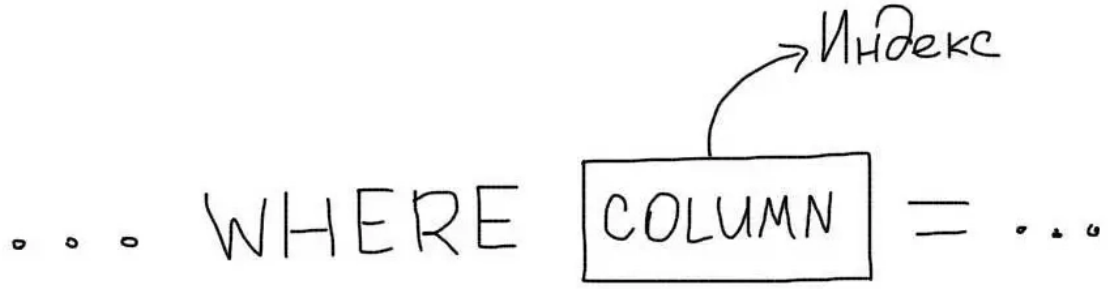


Кроме меньшего количества операций сравнения, мы сэкономили бы на чтении ненужных записей.

Индекс – это и есть отсортированный набор значений. В **MySQL индексы** всегда строятся для какой-то конкретной колонки. Например, мы могли бы построить индекс для колонки age из примера.

4. Выбор индексов в MySQL

В самом простом случае, индекс необходимо создавать для тех колонок, которые присутствуют в условии WHERE.



Рассмотрим запрос из примера:

SELECT * FROM users WHERE age = 29

Нам необходимо создать индекс на колонку age:

CREATE INDEX age ON users(age);
После этой операции MySQL начнет использовать индекс age для выполнения подобных запросов. Индекс будет использоваться и для выборок по диапазонам значений этой колонки:

SELECT * FROM users WHERE age < 29

Сортировка

Для запросов такого вида:

SELECT * FROM users ORDER BY register_date
действует такое же правило – создаем индекс на колонку, по которой происходит сортировка:

CREATE INDEX register_date ON users(register_date);

Внутренности хранения индексов

Представим, что наша таблица выглядит так:

```
id | name   | age
1  | Den    | 29
2  | Alyona | 15
3  | Putin  | 89
4  | Petro  | 12
```

После создания индекса на колонку age, MySQL сохранит все ее значения в отсортированном виде:

```
age index
12
15
29
89
```

Кроме этого, будет сохранена связь между значением в индексе и записью, которой соответствует это значение. Обычно для этого используется первичный ключ:

```
age index и связь с записями
12: 4
15: 2
29: 1
89: 3
```

Уникальные индексы

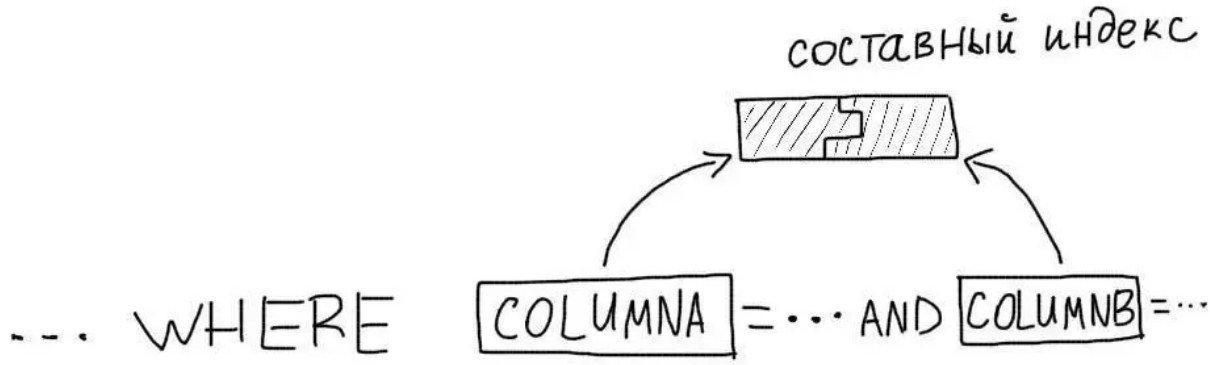
MySQL поддерживает уникальные индексы. Это удобно для колонок, значения в которых должны быть уникальными по всей таблице. Такие индексы улучшают эффективность выборки для уникальных значений. Например:

SELECT * FROM users WHERE email = 'golotyuk@gmail.com';
На колонку email необходимо создать уникальный индекс:

CREATE UNIQUE INDEX email ON users(email)
Тогда при поиске данных, MySQL остановится после обнаружения первого соответствия. В случае обычного индекса будет обязательно проведена еще одна проверка (следующего значения в индексе).

5. Составные индексы

MySQL может использовать только **один индекс для запроса** (кроме случаев, когда MySQL способен объединить результаты выборок по нескольким индексам). Поэтому, для запросов, в которых используется несколько колонок, необходимо использовать **составные индексы**.



Рассмотрим такой запрос:

SELECT * FROM users WHERE age = 29 AND gender = 'male'
Нам следует создать составной индекс на обе колонки:

CREATE INDEX age_gender ON users(age, gender);

Устройство составного индекса

Чтобы правильно использовать составные индексы, необходимо понять структуру их хранения. Все работает точно так же, как и для обычного индекса. Но для значений используются значений всех входящих колонок сразу. Для таблицы с такими данными:

```
id | name   | age | gender
1  | Den    | 29 | male
2  | Alyona | 15 | female
3  | Putin  | 89 | tsar
4  | Petro  | 12 | male
```

значения составного индекса будут такими:

```
age_gender
12male
15female
29male
89tsar
```

Это означает, что очередность колонок в индексе будет играть большую роль. Обычно колонки, которые используются в условиях WHERE, следует ставить в начало индекса. Колонки из ORDER BY — в конец.

Поиск по диапазону

Представим, что наш запрос будет использовать не сравнение, а поиск по диапазону:

SELECT * FROM users WHERE age <= 29 AND gender = 'male'
Тогда MySQL не сможет использовать полный индекс, т.к. значения gender будут отличаться для разных значений колонки age. В этом случае база данных попытается использовать часть индекса (только age), чтобы выполнить этот запрос:

```
age_gender
12male
15female
29male
89tsar
```

Сначала будут отфильтрованы все данные, которые подходят под условие *age <= 29*. Затем, поиск по значению “male” будет произведен без использования индекса.

Сортировка

Составные индексы также можно использовать, если выполняется сортировка:

SELECT * FROM users WHERE gender = 'male' ORDER BY age
В этом случае нам нужно будет создать индекс в другом порядке, т.к. сортировка (ORDER) происходит после фильтрации (WHERE):

CREATE INDEX gender_age ON users(gender, age);
Такой порядок колонок в индексе позволит выполнить фильтрацию по первой части индекса, а затем отсортировать результат по второй.

Колонок в индексе может быть больше, если требуется:

SELECT * FROM users WHERE gender = 'male' AND country = 'UA' ORDER BY age, register_time
В этом случае следует создать такой индекс:

CREATE INDEX gender_country_age_register ON users(gender, country, age, register_time);

6. Использование EXPLAIN для анализа индексов

Инструкция EXPLAIN покажет данные об использовании индексов для конкретного запроса. Например:

mysql> EXPLAIN SELECT * FROM users WHERE email = 'golotyuk@gmail.com';

```
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | users | ALL  | NULL          | NULL | NULL    | NULL |  336 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
```

Колонка **key** показывает используемый индекс. Колонка **possible_keys** показывает все индексы, которые могут быть использованы для этого запроса. Колонка **rows** показывает число записей, которые пришлось прочитать базе данных для выполнения этого запроса (в таблице всего 336 записей).

Как видим, в примере не используется ни один индекс. После создания индекса:

mysql> EXPLAIN SELECT * FROM users WHERE email = 'golotyuk@gmail.com';

```
+----+-------------+-------+-------+---------------+-------+---------+-------+------+-------+
| id | select_type | table | type  | possible_keys | key   | key_len | ref   | rows | Extra |
+----+-------------+-------+-------+---------------+-------+---------+-------+------+-------+
|  1 | SIMPLE      | users | const | email         | email | 386     | const |    1 |       |
+----+-------------+-------+-------+---------------+-------+---------+-------+------+-------+
```

Прочитана всего одна запись, так как был использован индекс.

Проверка длинны составных индексов

Explain также поможет определить правильность использования составного индекса. Проверим запрос из примера (с индексом на колонки age и gender):

mysql> EXPLAIN SELECT * FROM users WHERE age = 29 AND gender = 'male';

```
+----+-------------+--------+------+---------------+------------+---------+-------------+------+-------------+
| id | select_type | table  | type | possible_keys | key        | key_len | ref         | rows | Extra       |
+----+-------------+--------+------+---------------+------------+---------+-------------+------+-------------+
|  1 | SIMPLE      | users  | ref  | age_gender    | age_gender | 24      | const,const |    1 | Using where |
+----+-------------+--------+------+---------------+------------+---------+-------------+------+-------------+
```

Значение **key_len** показывает используемую длину индекса. В нашем случае 24 байта – длинна всего индекса (5 байт age + 19 байт gender).

Если мы выполним изменим точное сравнение на поиск по диапазону, увидим что MySQL использует только часть индекса:

mysql> EXPLAIN SELECT * FROM users WHERE age <= 29 AND gender = 'male';

```
+----+-------------+--------+------+---------------+------------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key        | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------------+---------+------+------+-------------+
|  1 | SIMPLE      | users  | ref  | age_gender    | age_gender | 5       |      |   82 | Using where |
+----+-------------+--------+------+---------------+------------+---------+------+------+-------------+
```

Это сигнал о том, что созданный индекс не подходит для этого запроса. Если же мы создадим правильный индекс:

mysql> Create index gender_age on users(gender, age); mysql> EXPLAIN SELECT * FROM users WHERE age < 29 and gender = 'male';

```
+----+-------------+--------+-------+----------------------+------------+---------+------+------+-------------+
| id | select_type | table  | type  | possible_keys        | key        | key_len | ref  | rows | Extra       |
+----+-------------+--------+-------+----------------------+------------+---------+------+------+-------------+
|  1 | SIMPLE      | users  | range | age_gender,gender_age | gender_age | 24      | NULL |   47 | Using where |
+----+-------------+--------+-------+----------------------+------------+---------+------+------+-------------+
```

В этом случае MySQL использует весь индекс gender_age, т.к. порядок колонок в нем позволяет сделать эту выборку.

7. Селективность индексов

Вернемся к запросу:

SELECT * FROM users WHERE age = 29 AND gender = 'male'
Для такого запроса необходимо создать составной индекс. Но как правильно выбрать последовательность колонок в индексе? Варианта два:

- age, gender
- gender, age

Подойдут оба. Но работать они будут с разной эффективностью.

Чтобы понять это, рассмотрим уникальность значений каждой колонки и количество соответствующих записей в таблице:

mysql> select age, count(*) from users group by age;

```
+------+----------+
| age  | count(*) |
+------+----------+
|   15 |      160 |
|   16 |      250 |
|        ...      |
|   76 |      210 |
|   85 |      230 |
+------+----------+
```

68 rows in set (0.00 sec)

mysql> select gender, count(*) from users group by gender;

```
+--------+----------+
| gender | count(*) |
+--------+----------+
| female |     8740 |
| male   |     4500 |
+--------+----------+
2 rows in set (0.00 sec)
```

Эта информация говорит нам вот о чем:

1. Любое значение колонки age обычно содержит около 200 записей.

2. Любое значение колонки gender – около 6000 записей.

Если колонка age будет идти первой в индексе, тогда MySQL после первой части индекса сократит количество записей до 200. Останется сделать выборку по ним. Если же колонка gender будет идти первой, то количество записей будет сокращено до 6000 после первой части индекса. Т.е. на порядок больше, чем в случае age.

Это значит, что индекс age_gender будет работать лучше, чем gender_age.

Селективность колонки определяется количеством записей в таблице с одинаковыми значениями. Когда записей с одинаковым значением мало – селективность высокая. Такие колонки необходимо использовать первыми в составных индексах.

8. Первичные ключи

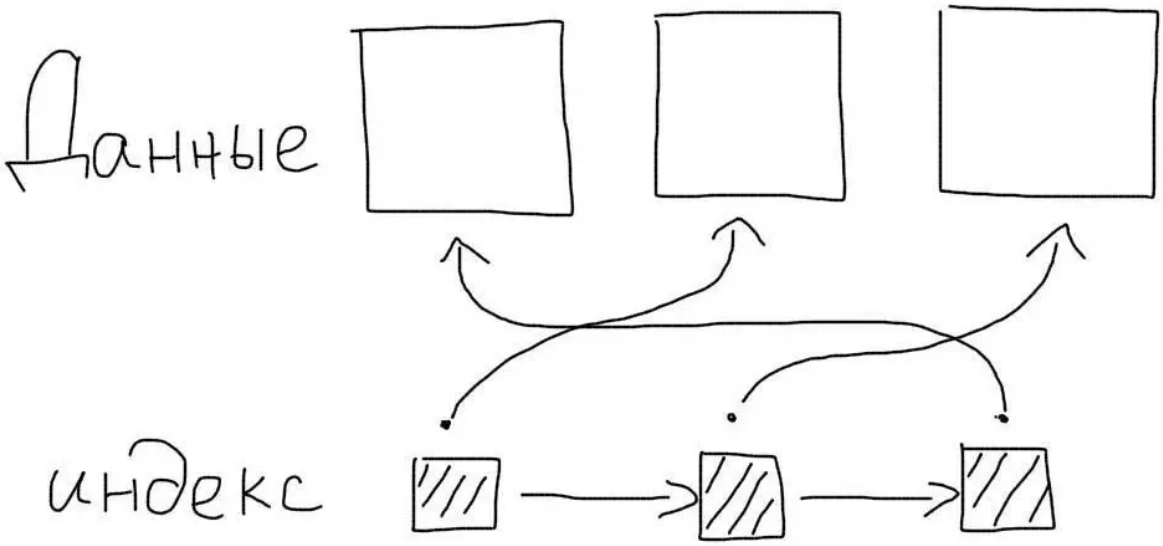
Первичный ключ (Primary Key) — это особый тип индекса, который является идентификатором записей в таблице. Он обязательно уникальный и указывается при создании таблиц:

```
CREATE TABLE `users` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`email` varchar(128) NOT NULL,
`name` varchar(128) NOT NULL,
PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

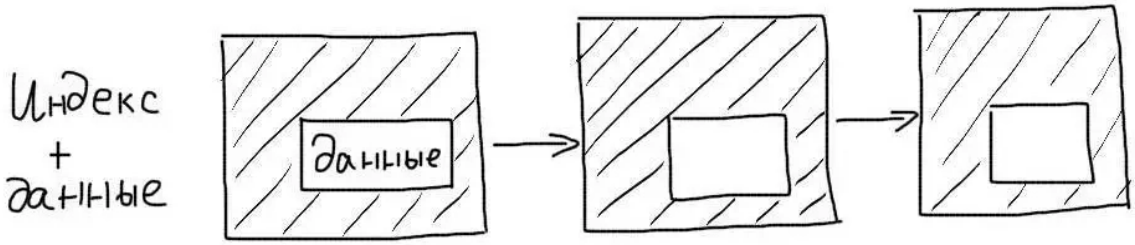
При использовании таблиц InnoDB **всегда определяйте первичные ключи**. Если первичного ключа нет, MySQL все равно создаст виртуальный скрытый ключ.

Кластерные индексы

Обычные индексы являются некластерными. Это означает, что сам индекс хранит только ссылки на записи таблицы. Когда происходит работа с индексом, определяется только список записей (точнее список их первичных ключей), подходящих под запрос. После этого происходит еще один запрос — для получения данных каждой записи из этого списка.



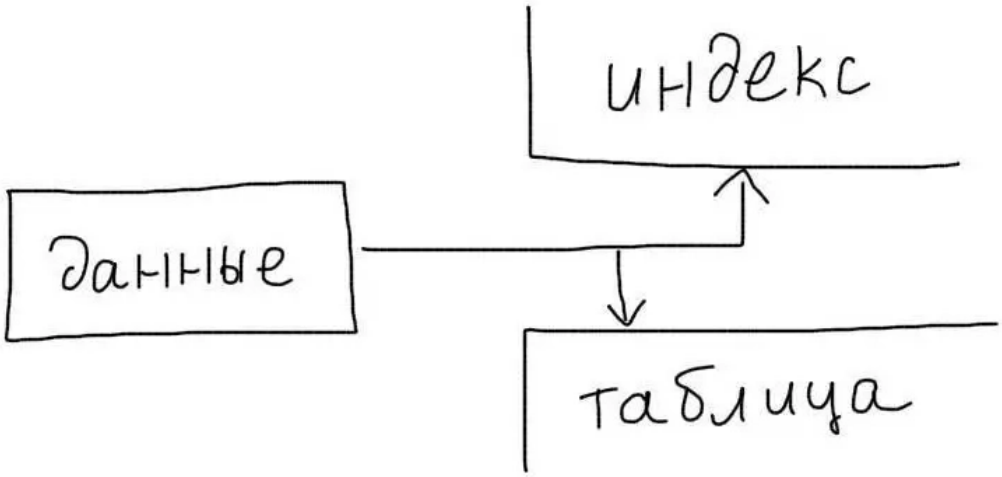
Кластерные индексы сохраняют данные записей целиком, а не ссылки на них. При работе с таким индексом не требуется дополнительной операции чтения данных.



Первичные ключи таблиц InnoDB являются кластерными. Поэтому выборки по ним происходят очень эффективно.

Overhead

Важно помнить, что индексы предполагают дополнительные операции записи на диск. При каждом обновлении или добавлении данных в таблицу, происходит также запись и обновление данных в индексе.



Создавайте только необходимые индексы, чтобы не расходовать зря ресурсы сервера. Контролируйте размеры индексов для Ваших таблиц:

mysql> show table status;

```
+------------------+--------+---------+------------+--------+----------------+---
----------+-----------------+--------------+-----------+----------------+---------
------------+-------------+------------+-----------------+----------+-------------
---+---------+
| Name             | Engine | Version | Row_format | Rows   | Avg_row_length |
Data_length | Max_data_length | Index_length | Data_free | Auto_increment |
Create_time         | Update_time | Check_time | Collation       | Checksum |
Create_options | Comment |
+------------------+--------+---------+------------+--------+----------------+---
----------+-----------------+--------------+-----------+----------------+---------
------------+-------------+------------+-----------------+----------+-------------
---+---------+
...
| users            | InnoDB |      10 | Compact    |    314 |            208 |
65536 |              0 |        16384 |         0 |           355 | 2014-07-11
01:12:17 | NULL        | NULL       | utf8_general_ci |     NULL |
|         |
+------------------+--------+---------+------------+--------+----------------+---
----------+-----------------+--------------+-----------+----------------+---------
------------+-------------+------------+-----------------+----------+-------------
---+---------+
18 rows in set (0.06 sec)
```

Когда создавать индексы?

- Индексы следует создавать по мере обнаружения медленных запросов. В этом поможет slow log в MySQL. Запросы, которые выполняются более 1 секунды, являются первыми кандидатами на оптимизацию.

- Начинайте создание индексов с самых частых запросов. Запрос, выполняющийся секунду, но 1000 раз в день наносит больше ущерба, чем 10-секундный запрос, который выполняется несколько раз в день.
- Не создавайте индексы на таблицах, число записей в которых меньше нескольких тысяч. Для таких размеров выигрыш от использования индекса будет почти незаметен.
- Не создавайте индексы заранее, например, в среде разработки. Индексы должны устанавливаться исключительно под форму и тип нагрузки работающей системы.
- Удаляйте неиспользуемые индексы.

Самое важное

Выделяйте достаточно времени на анализ и организацию индексов в MySQL (и других базах данных). На это может уйти намного больше времени, чем на проектирование структуры базы данных. Удобно будет организовать тестовую среду с копией реальных данных и проверять там разные структуры индексов.

Не создавайте индексы на каждую колонку, которая есть в запросе, MySQL так не работает. Используйте уникальные индексы, где необходимо. Всегда устанавливайте первичные ключи.

Не ссыте.