GeekBrains

Базы данных. Интерактивный курс

Урок 9

Оптимизация запросов

Тип таблицы

СУБД MySQL поддерживает несколько видов таблиц, каждая из которых имеет свои возможности и ограничения. Задать тип таблицы можно при ее создании при помощи оператора **CREATE TABLE**:

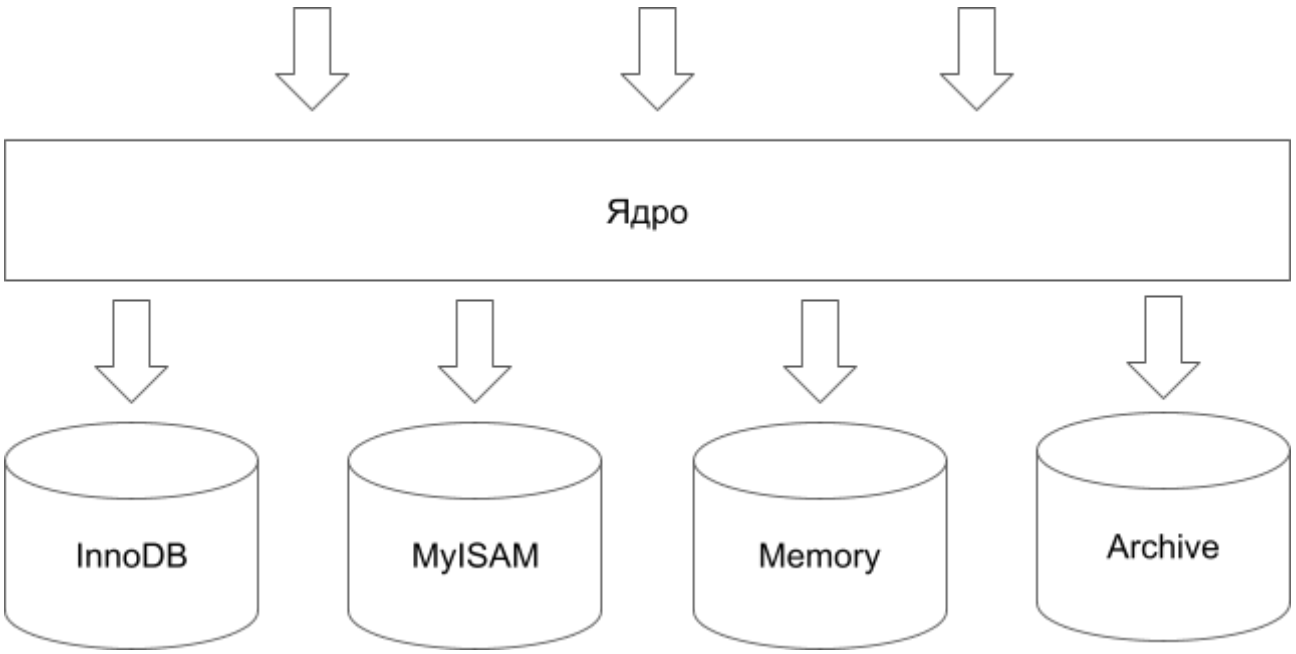
```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название раздела'
) COMMENT = 'Разделы интернет-магазина';
```

Для этого в конце в параметрах таблицы мы указываем ключевое слово **ENGINE**:

```
DROP TABLE IF EXISTS catalogs;
CREATE TABLE catalogs (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) COMMENT 'Название раздела'
) COMMENT = 'Разделы интернет-магазина' ENGINE=InnoDB;
```

Архитектура MySQL

В СУБД MySQL подсистемы хранения данных и ядро разделены.

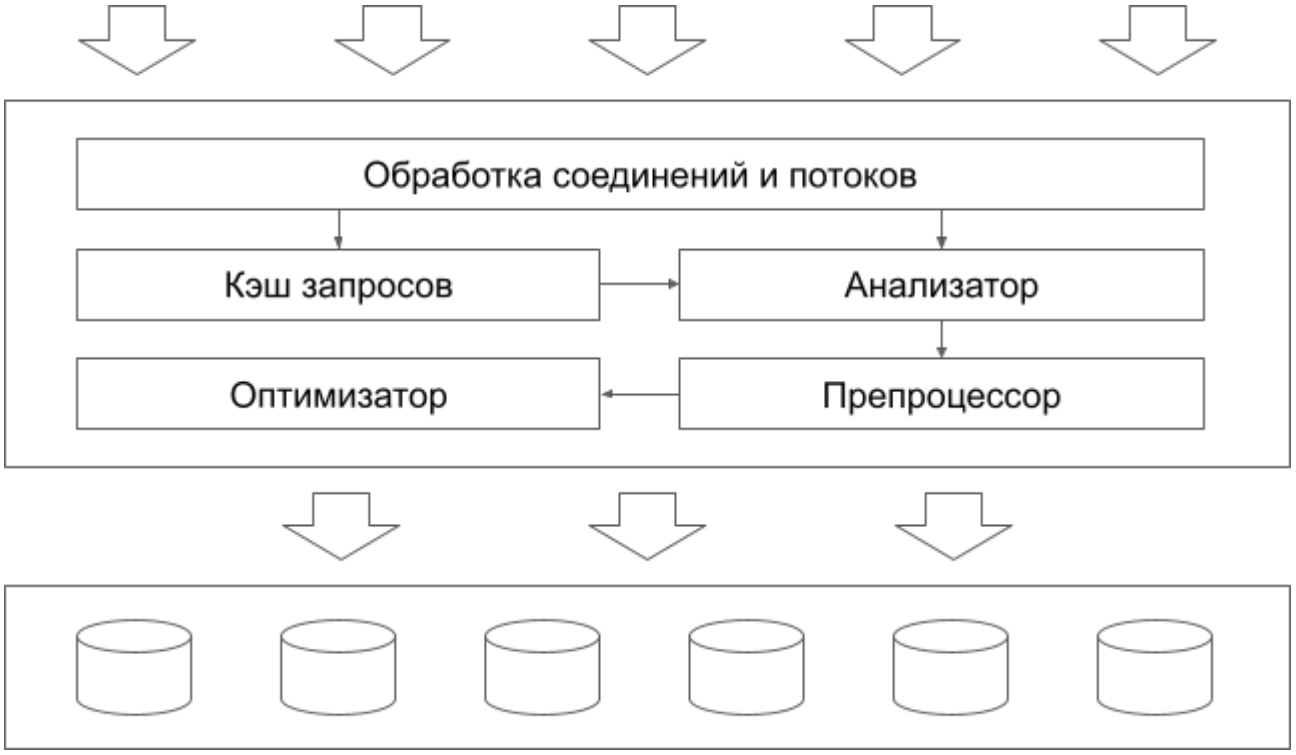


Поддержка соединения с клиентом, применение правил и логики языка SQL, кеширование запросов осуществляет ядро сервера MySQL. За хранение данных, индексирование, выполнение низкоуровневых операций с хранилищем отвечают подсистемы хранения, которые часто называют движками.

Их довольно много, на экране вы видите лишь часть. По умолчанию используется движок InnoDB. До текущего момента использовали его неявно. Так как он используется по умолчанию, нам не обязательно было его указывать при создании таблиц.

Однако вам придется явно указать тип движка, если вы захотите создать таблицу с подсистемой хранения, отличной от InnoDB. Например, можно задействовать таблицу типа **Memory**, полностью разместив данные в оперативной памяти, или использовать таблицу **Archive**, чтобы сжать данные. В этом случае вам придется явно указать тип движка при создании таблицы.

Прежде чем мы окунемся в особенности каждого движка, давайте кратко остановимся на ядре MySQL. Оно поддерживает пул соединений с клиентами, обеспечивая каждого из них сессией и обрабатывая его запросы. В случае, если на сервере включен кеш запросов, каждый SELECT-запрос проверяется сначала в кеше, если он присутствует там, ответ выдается из него.



Далее запрос идет в анализатор, который разбивает его на лексемы и строит дерево разбора. Для интерпретации и проверки запросов анализатор использует грамматику языка SQL. Проверяется, что все лексемы допустимы, следуют в нужном порядке и нет других ошибок, например непарных кавычек.

Затем получившееся дерево разбора передается препроцессору, который контролирует дополнительную семантику, не входящую в компетенцию анализатора. К примеру, проверяется, что указанные таблицы и столбцы существуют, а ссылки на столбцы не допускают неоднозначного толкования. Далее препроцессор проверяет привилегии.

Теперь, когда дерево разбора тщательно проверено, наступает очередь оптимизатора, который превращает его в план выполнения запроса. Часто существует множество способов выполнить запрос, и все они дают один и тот же результат. Задача оптимизатора — выбрать лучший из них.

Язык SQL — декларативный: мы не объясняем как выполнять запрос, мы запрашиваем то, что хотим получить. Поэтому анализатор имеет свободу действий по оптимизации запроса и выбора индексов.

Более того, он привлекает статистику сервера для выбора наиболее оптимального пути выполнения запроса.

После того, как план выполнения запроса выбран, происходит его выполнение с использованием API подсистемы хранения, т.е., ядро обращается к низкоуровневым вызовам движка.

Файлы таблицы

Мы с вами уже исследовали базы данных и выяснили, что это подкаталог в каталоге данных. Что же из себя представляют таблицы? В подкататалоге для каждой из таблиц создается файл с расширением **.frm** и именем, совпадающим с именем таблицы. В этом файле хранится определение таблицы, список столбцов, их типы и порядок следования. Этот файл одинаков для всех подсистем хранения, так как его обрабатывает ядро.

Данные и индексы для каждой системы хранения реализуются по-разному, за их работу отвечают движки таблиц. Например, для типа **MyISAM** данные сохраняются в одноименном файле с расширением **MYD**, а индексы — в файле **MYI**. В случае движка InnoDB данные и индексы всех таблиц и всех баз данных хранятся в едином табличном пространстве.

Впрочем, если в конфигурационном файле **my.cnf** включить директиву **innodb_file_per_table**, можно перевести MySQL в режим, когда для каждой таблицы будет организовываться свое собственное табличное пространство в файле с расширением **idb**.

Давайте перейдем в каталог данных и посмотрим его содержимое, путь к нему можно обнаружить среди системных переменных:

```
SHOW VARIABLES LIKE 'datadir';
```

Полученный путь используем в команде **cd**, чтобы перейти в каталог. Для просмотра содержимого каталога можно использовать команду **ls** в UNIX-подобной операционной системе и **dir** в Windows:

```
cd /usr/local/var/mysql
ls -la
```

Итак, мы можем видеть файл **ibdata1** с единым табличным пространством. Файл **ibtmp1** содержит временные таблицы, файлы **ib_logfile0** и i**b_logfile1** — журнал транзакций.

Давайте перейдем в каталог **shop**:

```
cd shop
```

Посмотрим его содержимое:

```
ls -la
```

Мы можем видеть здесь файлы со структурой **frm** и данными **ibd**, каждая пара соответствует таблице в базе данных **shop**. Кроме того, мы можем видеть несколько файлов с расширением **TRN**, которые предназначены для обеспечения работы триггеров.

Давайте переместимся в системную базу данных **mysql**:

```
cd ../mysql
```

Посмотрим ее содержимое:

```
ls -la
```

Как видим, тут есть таблицы **MyISAM**, например таблица **proc**, в которой содержатся хранимые процедуры и функции; для нее есть файл **frm** со структурой таблицы, MYD-файл с данными и **MYI** с индексами.

Чтобы определить, какая подсистема хранения используется для конкретной таблицы, не заглядывая в каталог данных, следует использовать команду **SHOW TABLE STATUS**.

```
SHOW TABLE STATUS LIKE 'catalogs'\G
```

Как видим, перед нами таблица типа InnoDB. Список доступных движков и их характеристики можно запросить при помощи запроса:

```
SHOW ENGINES\G
```

InnoDB была разработана для транзакционной обработки и автоматического восстановления после сбоя. Движок поддерживает внешние ключи и блокировку на уровне строк. Этот движок не кеширует информацию о количестве строк в таблице, поэтому в отчетах эта цифра часто приблизительная, а выполнение запросов с участием функции **COUNT(*)** — очень трудоемкое, так как каждый раз происходит полное сканирование таблицы.

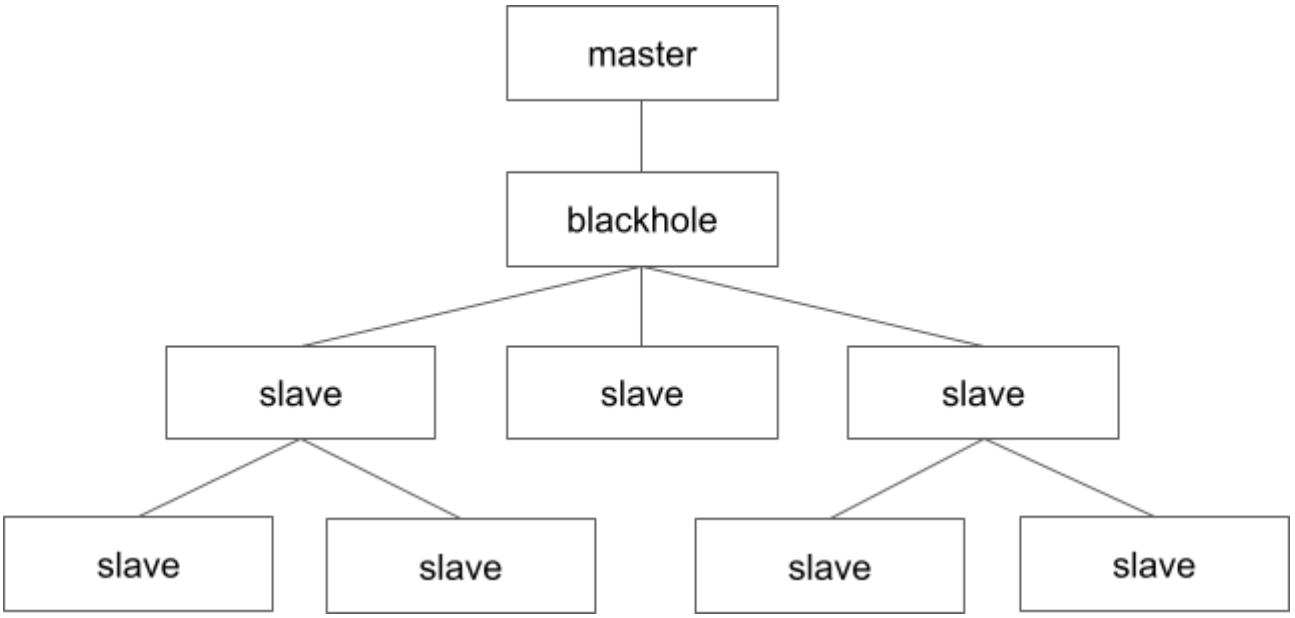
Memory-таблицы полностью расположены в оперативной памяти и не поддерживают транзакции. Поэтому при перезапуске содержимое таблиц обнуляется. За счет расположения в оперативной памяти операции с такой таблицей происходят исключительно быстро. При операциях вставки происходит табличная блокировка, т.е., нельзя одновременно вставлять две записи в таблицу, такие операции выстраиваются в очередь.

Для данного типа таблиц по умолчанию используются хэш-индексы. В настоящий момент эти таблицы редко используются, так как существуют более эффективные NoSQL-решения для таких задач, например, база данных Redis.

BLACKHOLE-таблицы не содержат данных, операции которые на них выполняются не фиксируются в базе данных. Однако фиксируются в бинарном журнале, поэтому BLACKHOLE-таблицы выступают в качестве ретранслятора бинарного журнала

Когда с этим не справляется реплика?

Например, в топологии «пирамида», когда необходимо к нагруженному мастер-серверу подключить множество подчиненных слейв-серверов. В этом случае сервер, который не выполняет запросов, справляется с передачей бинарного журнала гораздо лучше. На таком сервере таблицам переназначается тип **BLACKHOLE**.



MyISAM — файловый движок, не поддерживающий транзакции: при операциях вставки также происходит табличная блокировка. Можно задаться вопросом, зачем нужны движки без транзакций? Дело в том, что транзакции потребляют ресурсы и в тех случаях когда они не важны, а операции вставки не часты и применяются в отношении небольших таблиц, можно добиться значительного выигрыша в производительности.

MRG_MYISAM представляет собой объединение нескольких структурно одинаковых таблиц **MyISAM** в одну виртуальную; это до некоторой степени своеобразный материализованный UNION-запрос.

Система хранения **CSV** позволяет рассматривать текстовые CSV-файлы с разделителями-запятыми как таблицы. Такие файлы легко создаются вручную, любым скриптом или сохраняются из Excel-таблицы.

Archive позволяет выполнять только команды **INSERT** и **SELECT**. Таблицы этого типа потребляют исключительно мало оперативной памяти, снижают дисковый ввод-вывод, так как данные сжимаются библиотекой **zlib**. Этот тип чаще всего используется для протоколирования и сбора данных. Система хранения **Archive** поддерживает блокировку на уровне строк и специальный системный буфер для вставок с высокой степенью конкурентности. **Archive** не поддерживает транзакции. Она оптимизирована лишь для высокоскоростной вставки и хранения данных в сжатом виде.

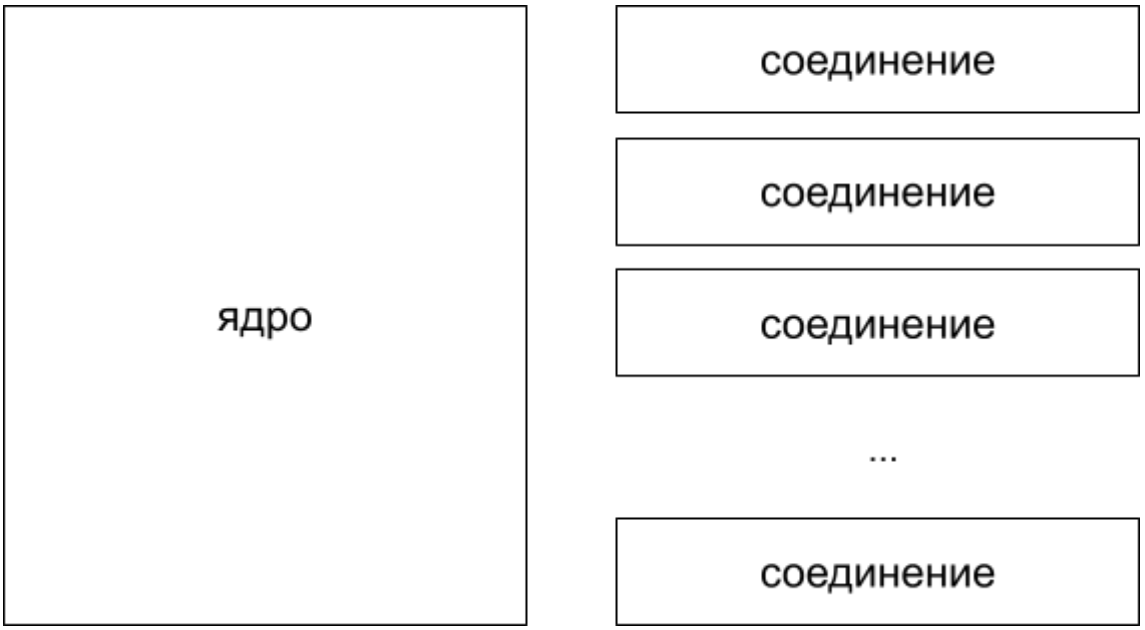
Индексы

Индексы представляют собой структуры, которые помогают MySQL эффективно извлекать данные. Они критичны для хорошей производительности. Важность индексов увеличивается по мере роста объема данных. Небольшие слабозагруженные базы данных зачастую могут удовлетворительно работать даже без правильно построенных индексов, но по мере роста объемов хранимой в базе данных информации производительность может упасть очень быстро.

Хорошая производительность достигается не только за счет того, что индексы хранят столбцы в заранее отсортированном виде, но и в том, что сами индексы стараются держать в быстрой оперативной памяти вместо хранения их на жестком диске.

Тем не менее, MySQL не может взять без спросу столько оперативной памяти, сколько имеется в системе — память потребуется выделять.

MySQL содержит большое количество самых разнообразных кешей. Основное их назначение — разместить в быстрой оперативной памяти информацию с более медленного диска. В результате данные моментально отдаются из оперативной памяти, вместо того, чтобы извлекаться с более медленного диска.



Одни кеши выделяются на ядро и являются общими для всех соединений, часть кешей выделяется на каждое соединение.

Например, кеши под индексы общие для всех клиентов, а кеши под сортировку данных — индивидуальные для каждого из соединений.

```
[mysqld]
...
query_cache_size = 0
key_buffer_size = 8M
innodb_buffer_pool_size = 1024M
innodb_additional_mem_pool_size = 8M
innodb_log_buffer_size = 8M
…
max_connections = 200
...
read_buffer_size = 1M
read_rnd_buffer_size = 1M
sort_buffer_size = 2M
thread_stack = 256K
join_buffer_size = 128K
```

```
Ядро:
0 +
8M +
1024M +
8M +
8M =
1056M

Соединение:
1M +
1M +
2M +
256K +
128K ~
3.5M
```

Почти всегда можно подсчитать максимальный размер, который займет MySQL кешами общего назначения и одним соединением. На рисунке можно видеть пример конфигурации MySQL-сервера, в котором ядро может занять чуть больше 1 Гб, а каждое из соединений — порядка 3.5 Мб.

Учитывая, что максимальное количество соединений — 200, при максимальной нагрузке на соединения уйдет около 700 Мб. В реальности, конечно, меньше, так как не все соединения будут использовать кеши. Кеши ядра тоже редко бывают заполнены под завязку.

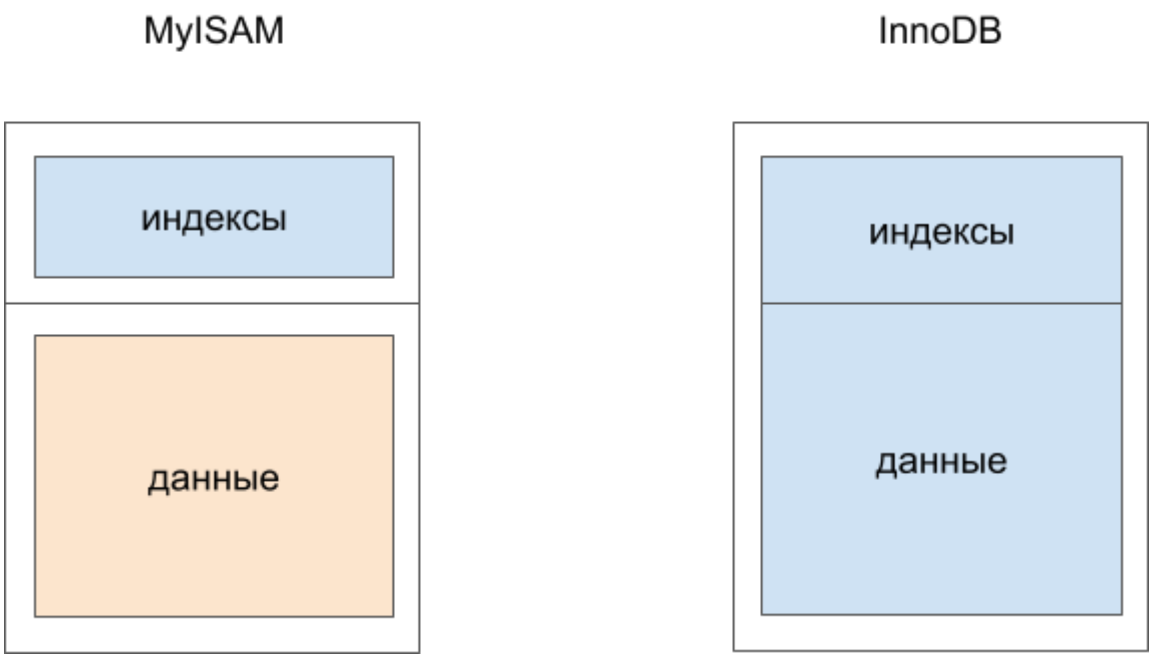
```
[mysqld]
...
query_cache_size = 0
key_buffer_size = 8M
innodb_buffer_pool_size = 8G
innodb_additional_mem_pool_size = 8M
innodb_log_buffer_size = 8M
…
max_connections = 500
...
read_buffer_size = 1M
read_rnd_buffer_size = 1M
sort_buffer_size = 2M
thread_stack = 256K
join_buffer_size = 128K
```

Ядро:
8G

Соединения:
500 * 3.5M = 1750M

Если на сервере есть свободная оперативная память, скажем, 16 Гб, можем смело выделить MySQL дополнительную память. Например, увеличить пул буферов InnoDB для кеширования индексов и данных до 8 Гб, а количество соединений — до 500. В результате при максимальной нагрузке MySQL-сервер будет потреблять чуть меньше 10 Гб.

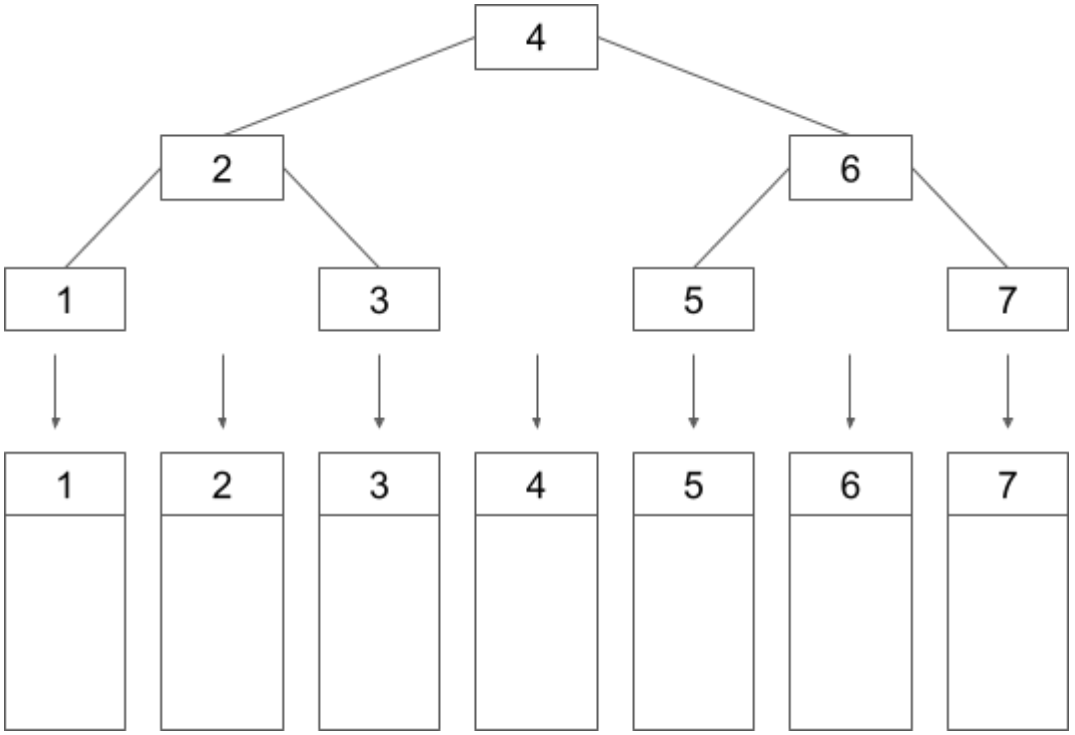
За реализацию индексов отвечают подсистемы хранения, поэтому в разных системах хранения кеширование индексов может осуществляться по-разному. Например, как мы видели в предыдущих роликах, в таблицах типа MyISAM индексы и данные разделены.



Данные всегда находятся на жестком диске, индексы по мере необходимости подтягиваются в оперативную память в кеш индексов.

Данные кешируются средствами операционной системы, поэтому, если MyISAM — основной тип используемых таблиц, важно оставлять на сервере какое-то количество свободной оперативной памяти.

В таблицах InnoDB индексы хранятся в едином табличном пространстве вместе с данными. Поэтому, когда мы говорим о кеше InnoDB, имеется в виду, что в оперативной памяти размещаются и индексы и данные.



Такое различие связано с тем, что для первичного ключа в InnoDB используется кластерный индекс — это фактически то же самое бинарное дерево, как и в случае BTREE-индексов, только в качестве листьев этого дерева выступают строки таблицы.

Таким образом, строки с близкими значениями первичного ключа хранятся по соседству, кроме того, над таблицей можно построить только один кластерный индекс, поскольку невозможно хранить один и тот же индекс в двух местах.

```
SHOW VARIABLES LIKE 'Key%';
```

Оценить объем оперативной памяти, выделенной под кеш MyISAM-индексов, можно при помощи команды. Значение **key_buffer_size** показывает объем в байтах выделенный под этот кеш: в данном случае это 8 Мб. Это значение можно увеличить через конфигурационный файл **my.cnf**, для этого используется одноименная директива **key_buffer_size**, максимально допустимый размер которой 4 Гб.

Мы преимущественно используем InnoDB, поэтому этот кеш не стоит увеличивать слишком сильно, тем не менее совершенно отключать этот кеш не стоит, так как таблицы типа MyISAM используются в системной базе данных MySQL:

```
SHOW STATUS LIKE 'Key%';
```

Оценить эффективность данного кеша можно при помощи команды **SHOW STATUS**, отфильтровав результаты по шаблону **Key**. **Key_blocks_used** сообщает количество занятых блоков в кеше, **Key_blocks_unused** показывает свободное количество блоков. Как видим, на этом сервере MySQL еще далеко до заполнения данного кеша и в его увеличении нет необходимости.

SHOW STATUS LIKE 'Key%';

Variable_name	Value
Key_blocks_not_flushed	0
Key_blocks_unused	0
Key_blocks_used	107171
Key_read_requests	18472428924
Key_reads	381826182
Key_write_requests	4483957
Key_writes	77799

Состояние **Key_read_requests** сообщает о количестве блоков, прочитанных из кеша. Состояние **Key_reads** сообщает о количестве блоков, прочитанных с жесткого диска в кеш.

Таким образом, в приведенном примере, операция обращения к жесткому диску осуществлялась на каждые 50 операций чтения из оперативной памяти.

Состояния **Key_write_requests** и **Key_writes** сообщают аналогичные сведения в отношении операций записи в кеш и на жесткий диск

```
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
```

Оценить объем пула буферов, выделенный под InnoDB, можно при помощи команды **SHOW VARIABLES**, отфильтровав результат по значению **innodb_buffer_pool_size**.

Как видно, переменная принимает значение 128 Мб. Это пул буферов в оперативной памяти и под индексы и под данные InnoDB. Значение **innodb_buffer_pool_size** можно регулировать через одноименную директиву конфигурационного файла **my.cnf**.

Если MySQL — это основное программное обеспечение на сервере, то под пул буферов рекомендуется выделять 50-80 %, т. е., 2, 4, 8, 16 Гб, если это позволяют ресурсы сервера. Так как кешируются не только индексы, но и данные, пул буферов должен быть объемным.

Если ситуация позволяет, желательно, чтобы база данных целиком проваливалась в пул буферов, а к жесткому диску MySQL обращалась бы только для записей транзакций.

Однако бесконечно увеличивать размер **innodb_buffer_pool_size** нельзя. Если оперативная память на сервере будет исчерпана, часть данных из оперативной памяти будет сбрасываться на жесткий диск в **swap**. При обращении к этим данным операционная система будет вынуждена выгрузить другие данные из оперативной памяти. В результате скорость работа системы MySQL-сервера, да и вообще операционной системы, может замедлиться критически. Все параметры MySQL-сервера лучше наращивать постепенно, тщательно отслеживая состояние операционной системы.

Оценить состояние эффективности кеша **InnoDB** можно при помощи команды **SHOW STATUS**:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_%';
```

Величина **Innodb_buffer_pool_pages_total** показывает общение количество блоков в кеше, значение **Innodb_buffer_pool_pages_free** сообщает о количестве свободных блоков. Если эта величина исчерпана, имеет смысл увеличить объем памяти под InnoDB.

Variable_name	Value
Innodb_buffer_pool_pages_data	61285
Innodb_buffer_pool_bytes_data	1004093440
Innodb_buffer_pool_pages_free	67315
Innodb_buffer_pool_pages_total	131071
Innodb_buffer_pool_read_requests	3028608196
Innodb_buffer_pool_reads	19805
Innodb_buffer_pool_write_requests	11944087

253

Приемы оптимизации

При выполнении любого запроса в консольном клиенте mysql всегда выводится время выполнения запроса:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_shop |
+----------------+
| hello      |
+----------------+
1 row in set (0,01 sec)
```

Довольно трудно делать какие-то выводы, так как время выполнения запросов на современных компьютерах зачастую составляет несколько миллисекунд. Тем не менее, когда сервер обрабатывает миллионы запросов, их общее время может выливаться в значительную нагрузку.

Оптимизатор старается переработать ваш запрос, переставить условные конструкции местами, чтобы они могли использовать индекс, если есть возможность вычислить значения заранее, вычислить результат по индексам, не используя данные таблицы, и т. д.

Тем не менее, ряд оптимизаций зависит от разработчика. Чем меньше данных возвращает запрос — тем лучше. MySQL затрачивает меньше усилий на их сбор, ответ быстрее пересылается по сети. Это касается и количества возвращаемых запросом строк, и данных в столбцах. Если из всей таблицы вам нужны лишь два-три столбца, не следует извлекать все данные при помощи * в SELECT-запросе. Лучше выбрать конкретные столбцы. Запрос на извлечения всех данных из таблицы пользователей менее эффективен, чем извлечение только части столбцов.

```
SELECT id, name FROM users;
```

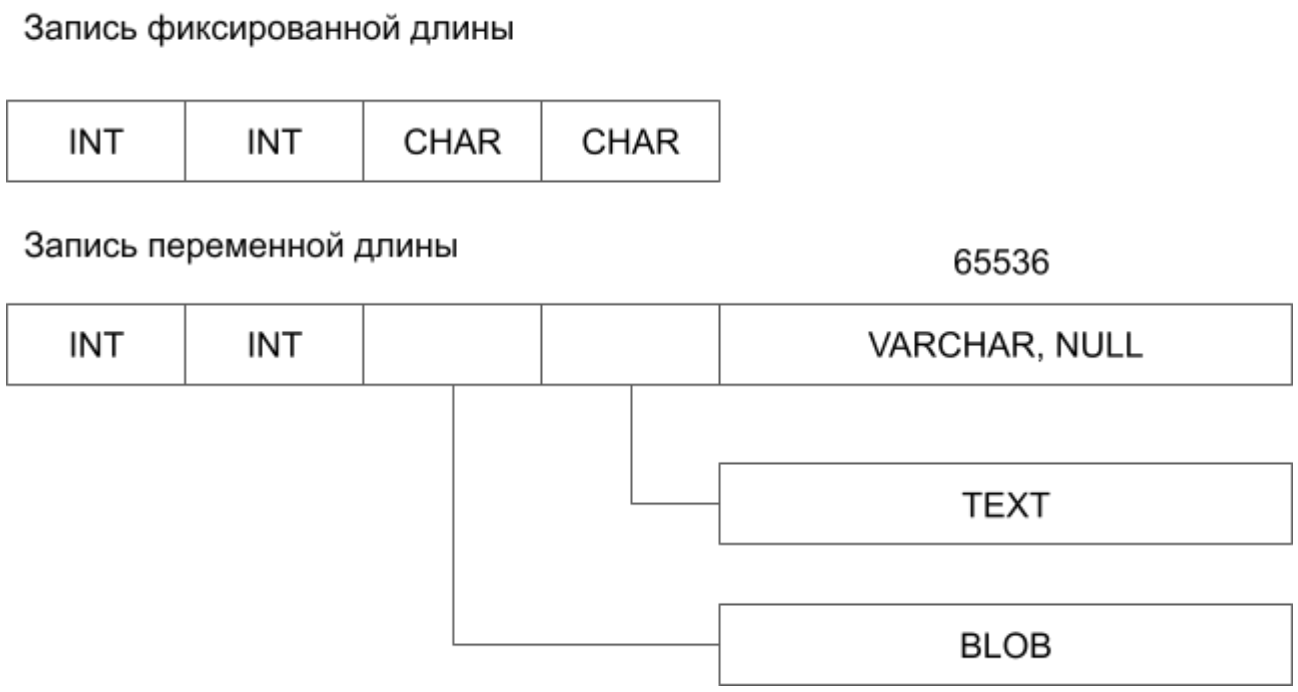
Запрос на извлечение всех данных таблицы менее эффективен, чем извлечение только ее части:

```
SELECT id, name FROM users LIMIT 2;
```

Например, только двух записей при помощи ключевого слова **LIMIT**.

Значения столбцов типа **VARCHAR** и **NULL** хранятся в специальной области для данных переменной длины. Эти данные обрабатываются менее эффективно, чем столбцы фиксированной длины, например целый тип. Поэтому, если вам не нужно NULL-значение столбца, лучше явно указывать атрибут **NOT NULL**.

Типы данных **BLOB** для хранения бинарных данных и **TEXT**, созданный на его основе, еще медленнее, чем остальные типы данных, так как хранятся отдельно от остальных столбцов таблицы. В силу такой особенности большинство операций с их участием часто протекают с использованием жесткого диска. Трудно построить базу данных совсем без этих типов, но если есть возможность, стоит исключить TEXT-столбцы из запроса.



Команда EXPLAIN

Основной способ узнать, какие решения принимает оптимизатор — воспользоваться командой EXPLAIN:

```
EXPLAIN SELECT id, name FROM catalogs ORDER BY id\G
```

Чтобы воспользоваться командой **EXPLAIN**, достаточно добавить слово **EXPLAIN** перед словом **SELECT** в запросе. MySQL пометит этот запрос специальным флагом. Во время его обработки этот флаг заставляет сервер сообщать информацию о каждом шаге плана выполнения, а не исполнять его.

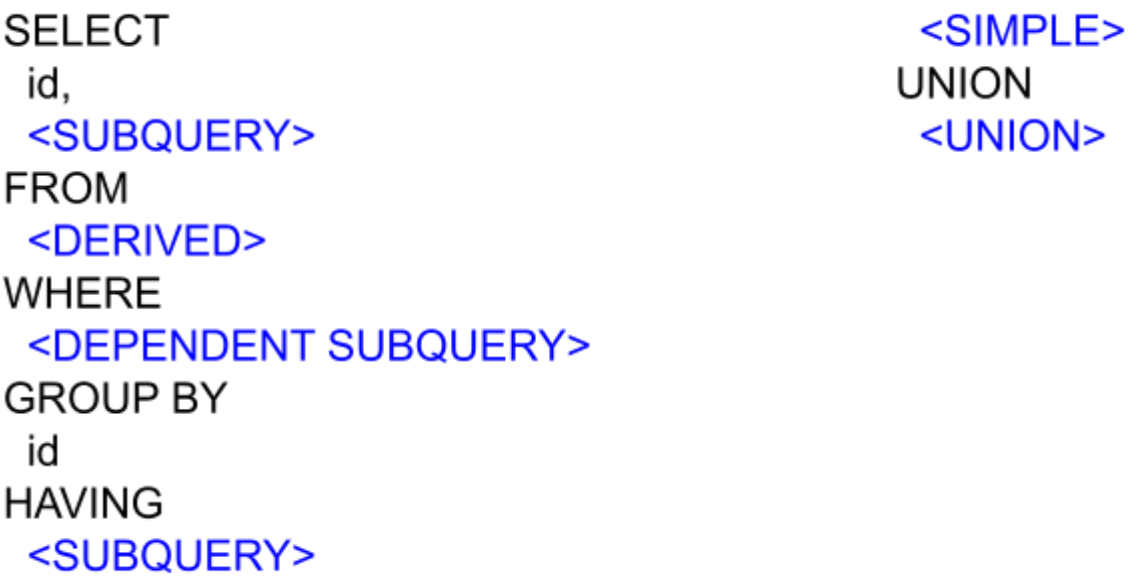
Для каждой встречающейся в запросе таблицы выводится одна строка. Если соединяются две таблицы, то будет выведено две строки.

```
EXPLAIN SELECT id, name FROM catalogs UNION ALL SELECT id, name FROM catalogs\G
```

Если одна и та же таблица попадается дважды, например при соединении таблицы с собой же, тоже будет выведено две строки. Команду **EXPLAIN** можно применить только для SELECT-запросов.

Для исследования других команд их предварительно следует перевести в SELECT-форму и применить **EXPLAIN**. Результаты, конечно, будут приблизительные, но это лучше, чем ничего.

Результат команды **EXPLAIN** всегда состоит из одних и тех же столбцов, изменяется лишь количество и содержимое строк. Столбец **select_type** показывает, соответствует ли строка простому или составному запросу **SELECT**.



На экране синим цветом показаны возможные значения поля **select_type** и где следует искать подзапросы, к которым относятся та или иная строка в отчете команды **EXPLAIN**.

Столбец **table** показывает, к какой таблице относится данная строка. Столбец **type** сообщает метод доступа к таблице: как MySQL будет искать строки в таблице. В данном случае используется метод

доступа **ALL**, т.е. будет произведено полное сканирование таблицы. Здесь это не страшно, так как мы и хотим извлечь все содержимое таблицы.

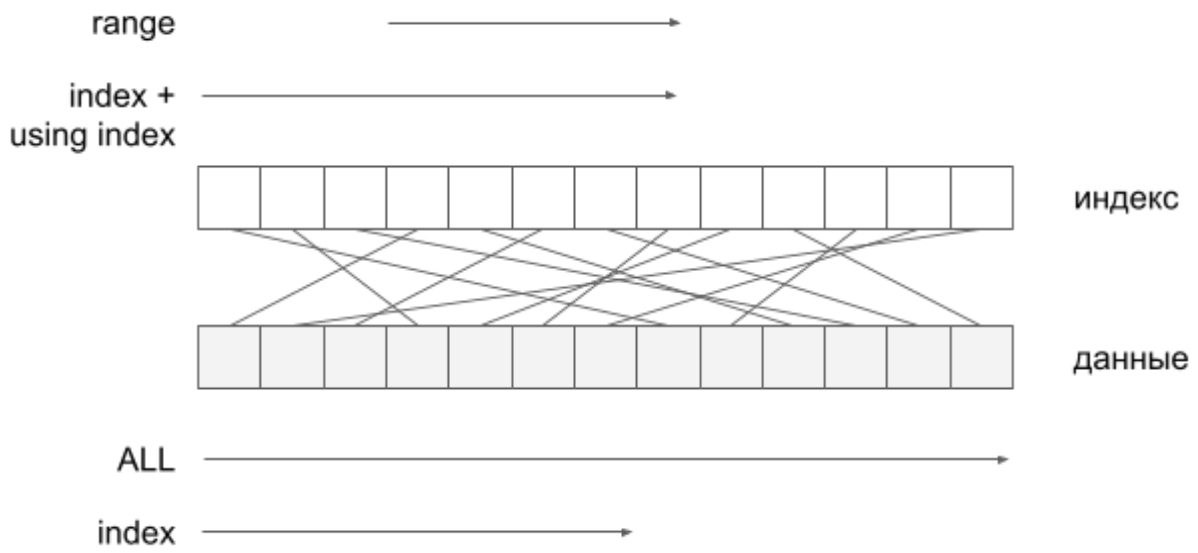
В поле **type** результирующей таблицы команды **EXPLAIN** могут встречаться следующие методы доступа:

- **ALL**,
- **index**,
- **range**,
- **ref**,
- **eq_ref**,
- **const**,
- **NULL**.

Методы доступа отсортированы от самых худших к наиболее предпочтительным.

ALL, как мы уже упоминали, означает полное сканирование таблицы. MySQL должна просмотреть таблицу от начала до конца, чтобы найти нужную строку. Существуют исключения, например запросы с фразой **LIMIT** — в этом случае MySQL прекратит перебор строк, как только будет набранно нужное количество.

index означает полное сканирование таблицы, только в порядке, задаваемом индексом. **range** означает просмотр диапазона индекса. Здесь тоже происходит сканирование, но уже индекса. Часто **range** можно увидеть в случае BETWEEN-условий или операций «меньше» или «больше».

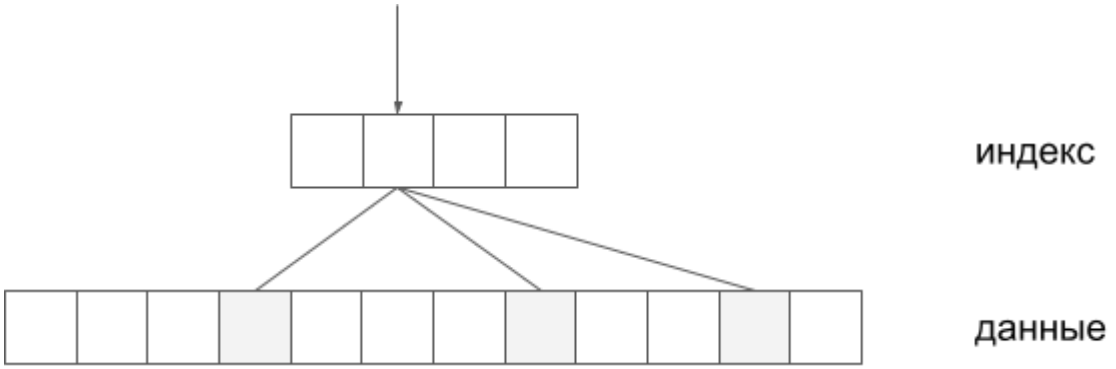


На экране представлены эти таблицы и индекс над ними. Индекс зачастую находится в оперативной памяти, работа с ним идет быстро за счет того, что его значения отсортированы. Данные чаще всего находятся на жестком диске.

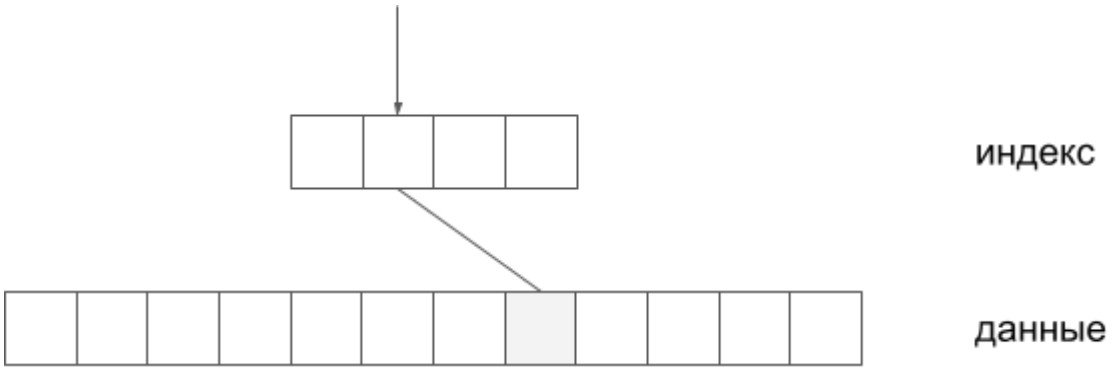
Методы **ALL** и **index** обращаются к данным таблицы. Метод **range** работает с индексом. Это означает, что данных, которые есть в индексе, достаточно для выполнения операции. Однако, потребуется перебор значений индекса в достаточно большом интервале.

Иногда совместно с типом **index** в дополнительном поле **Extra** можно увидеть сообщение **using index**. В этом случае речь идет о покрывающем индексе. Здесь тоже происходит сканирование индекса, но значений индекса достаточно, чтобы дать ответ, нет необходимости использовать данные таблицы с жесткого диска.

Методы доступа по **ref** и **eq_ref** — это доступ по индексу.



Метод доступа **ref** — это доступ по индексу, в результате которого возвращаются строки, соответствующие единственному заданному значению. Однако таких строк может быть несколько, поэтому поиск сочетается с просмотром. Данный тип доступа возможен лишь в случае неуникального индекса или неуникального префикса ключа в уникальном индексе.



eq_ref — поиск по индексу в случае, когда MySQL точно знает, что будет возвращено не более одного значения.

Если **EXPLAIN** возвращает **const**, это означает, что в процессе оптимизации какую-то часть запроса можно преобразовать в константу. **NULL** означает, что MySQL сумела разрешить запрос на фазе оптимизации, так что в ходе выполнения вообще не потребуется обращаться к таблице или индексу.

Столбец **possible_keys** показывает, какие индексы можно было бы задействовать для выполнения запроса. Сколько бы индексов ни было создано, в простом запросе может быть задействован только один. Выбранный оптимизатором индекс помещается в столбце **key**.

Столбец **key_len** показывает, сколько байт индекса использует MySQL. Если задействованы лишь некоторые индексированные столбцы составного индекса, то, зная это значение, можно определить, какие именно.

Столбец **ref** показывает, какие столбцы и константы из предыдущих таблиц используются для поиска в индексе. Столбец **rows** сообщает, сколько строк придется прочитать, чтобы найти запрошенные данные. Чем меньше это значение, тем быстрее будет выполняться запрос.

Столбец **filtered** показывает оценку доил-строк, которые удовлетворяют критерию выборки. Чем меньше этот процент, тем больше данных придется отбрасывать MySQL при сканировании таблицы.

Столбец **Extra** содержит дополнительную информацию, для которой не нашлось места в других столбцах:

- **Using index**.
- **Using where**.
- **Using temporary**.
- **Using filesort**.

Например **Using index** совместно с типом доступа **index** означает, что будет использоваться покрывающий индекс, т. е., данных индекса будет достаточно для выполнения запроса, обращаться к таблице не придется.

Using where означает, что сервер произведет дополнительную фильтрацию строк, отобранных подсистемой хранения. Чаще WHERE-условия обрабатываются системой хранения, например MyISAM или InnoDB, но иногда их возможностей не хватает и в дело вступает ядро.

Using temporary означает, что MySQL будет применять временную таблицу для сортировки результатов запроса. Такая таблица не обязательно размещается на жестком диске, если ее объем позволяет — она будет размещена в оперативной памяти. Максимально допустимый объем временной таблицы задается директивой **tmp_table_size**.

Using filesort означает, что MySQL прибегнет к обычной сортировке для упорядочения результатов, а не станет читать строки из таблицы в порядке, задаваемом индексом.

Используемые источники

1. https://dev.mysql.com/doc/refman/5.7/en/optimization.html
2. https://dev.mysql.com/doc/refman/5.7/en/execution-plan-information.html
3. Шварц Б., Зайцев П., Ткаченко В., Заводны Дж., Ленц А., Бэллинг Д. MySQL. Оптимизация производительности, 2-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 832 с.
4. Чарльз Белл, Мэтс Киндал и Ларс Талманн. Обеспечение высокой доступности систем на основе MySQL / Пер. с англ. — М. : Издательство "Русская редакция"; СПб. : БХВ-Петербург, 2012. — 624 с.
5. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
6. Гровф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.
7. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с.
8. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.
9. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.
10. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.
11. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид Групп, 2011. — 336 с.