Вадим @WASD42
Backend-программист
86,2 карма 0,0 рейтинг

Профиль | 5 Публикации | 44 Комментарии | 30 Избранное | 6 Подписчики

25 сентября 2009 в 17:52

Разработка → Индексы в MySQL: многоколоночные индексы против комбинированных индексов перевод

MySQL*

Я часто вижу ошибки, связанные с созданием индексов в MySQL. Многие разработчики (и не только новички в MySQL) создают много индексов на тех колонках, которые будут использовать в выборках, и считают это оптимальной стратегией. Например, если мне нужно выполнить запрос типа **AGE=18 AND STATE='CA'**, то многие люди просто создадут 2 отдельных индекса на колонках AGE и STATE.

Намного лучшей (***здесь и далее прим. переводчика:*** *а обычно и единственной верной*) стратегией является создание комбинированного индекса вида (AGE,STATE). Давайте рассмотрим почему это так.

Обычно (но не всегда) индексы в MySQL являются BTREE-индексами — такой тип индекса способна быстро просматривать информацию, содержащуюся в своих префиксах, и перебирать диапазоны отсортированных значений. Например, когда Вы запрашиваете **AGE = 18** с BTREE-индексом по колонке AGE MySQL найдёт в таблице первую отвечающую запросу строку и продолжит поиск до тех пор, пока не найдёт первую неподходящую строку — тогда он останавливает поиск, т.к. считает, что дальше ничего подходящего не будет. Диапазоны, например запросы вида **BETWEEN 18 AND 20**, работают сходным образом — MySQL останавливается на других значениях.

Несколько сложнее ситуация с запросами типа **AGE IN (18,20,30)**, т.к. на самом деле MySQL приходится несколько раз проходить по индексу.

Итак, мы обсудили как MySQL ищет по индексу, но не определили что же он возвращает после поиска — обычно (если речь не идёт о покрывающих (covering) индексах) получает «указатель строки», который может быть значением первичного ключа (если используется движок InnoDB), физическое смещение в файле (для MyISAM) или что-нибудь в этом роде. Важно, что внутренний движок MySQL может по этому указателю найти полную строку со всеми необходимыми данными, отвечающими заданному значению индекса.

А какие есть варианты у MySQL, если Вы создали два отдельных индекса? Он может либо использовать только один из них, чтобы отобрать подходящие строки (а потом отфильтровать извлечённые данные, руководствуясь WHERE — *но уже без использования индексов*), либо может получить указатели на строки от всех подходящих индексов и вычислить их пересечение, а затем уже вернуть данные.

Какой из способов будет более подходящим зависит от избирательности и корреляции индексов. Если после отработки WHERE по первой колонке будет отобрано 5% строк, а применение далее WHERE по второй колонке отфильтровывает строки до 1% от общего количества, то применение пересечений, конечно, имеет смысл. Но если второй WHERE отфильтрует только до 4.5%, то обычно значительно выгоднее использовать только первый индекс и отфильтровать ненужные нас строки после извлечения данных.

Давайте рассмотрим несколько примеров:

```
CREATE TABLE `idxtest` (
`i1` int(10) UNSIGNED NOT NULL,
`i2` int(10) UNSIGNED NOT NULL,
`val` varchar(40) DEFAULT NULL,
KEY `i1` (`i1`),
KEY `i2` (`i2`),
KEY `combined` (`i1`,`i2`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Я создал колонки i1 и i2 независимыми друг от друга, причём каждая из них отбирает около 1% строк в таблице, которая содержит в общей сложности 10 млн. записей.

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest WHERE i1=50 AND i2=50;
```

```
+----+-------------+---------+------+---------------+----------+---------+-------------+------+-------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+----------+---------+-------------+------+-------+
| 1 | SIMPLE | idxtest | ref | i1,i2,combined | combined | 8 | const,const | 665 |
+----+-------------+---------+------+---------------+----------+---------+-------------+------+-------+
1 row IN SET (0.00 sec)
```

Как Вы можете видеть MySQL предпочёл использовать комбинированный индекс, и запрос выполнился меньше, чем за 10 мс!

А теперь предположим, что у нас есть индекс только по отдельным колонкам (сказать оптимизатору игнорировать комбинированный индекс):

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX (combined) WHERE i1=
AND i2=50;
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
---------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
---------+
| 1 | SIMPLE | idxtest | index_merge | i1,i2 | i1,i2 | 4,4 | NULL | 1032 | USING intersect(i1,i2); USING WHERE
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
---------+
1 row IN SET (0.00 sec)
```

Как Вы можете видеть в данном случае MySQL выполнил поиск пересечений индексов, а на выполнение запроса понадобилось 70 мс — **в 7 раз дольше!**

Теперь давайте посмотрим, что будет, если использовать только один индекс и отфильтровывать полученные данные:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX (combined,i2) WHERE
AND i2=50;
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| 1 | SIMPLE | idxtest | ref | i1 | i1 | 4 | const | 106222 | USING WHERE
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
1 row IN SET (0.00 sec)
```

На этот раз MySQL пришлось обойти значительно больше строк, а выполнение запроса заняло **290 мс**. Таким образом мы видим, что использование пересечения индексов намного лучше, чем использование одного индекса, но значительно лучше использовать комбинированные индексы.

Однако на этом проблемы с пересечениями индексов не заканчиваются. В настоящее время возможности использования этой процедуры в MySQL значительно ограничены, поэтому MySQL использует их далеко не всегда:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX (combined) WHERE i1=
AND i2 IN (49,50);
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| 1 | SIMPLE | idxtest | ref | i1,i2 | i1 | 4 | const | 106222 | USING WHERE
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
1 row IN SET (0.00 sec)
```

Как только запрос по одной из колонок становится не сравнением, а перечислением, MySQL больше не сможет использовать пересечение индексов, несмотря на то, что в данном случае при запросе **i2 IN (49,50)** это было бы более, чем разумно, т.к. запрос остаётся достаточно селективным.

Теперь давайте проведём ещё один тест. Я очистил таблицу и вновь наполнил её данными таким образом, чтобы значения в i1 и i2 сильно коррелировали. На самом деле они теперь вообще равны:

```
mysql [localhost] {msandbox} (test)> UPDATE idxtest SET i2=i1;
Query OK, 10900996 rows affected (6 min 47.87 sec)
Rows matched: 11010048 Changed: 10900996 Warnings: 0
```

Давайте посмотрим, что произойдёт в этом случае:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest WHERE i1=50 AND i2=50;
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
----------+
```

```
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
----------+
| 1 | SIMPLE | idxtest | index_merge | i1,i2,combined | i2,i1 | 4,4 | NULL | 959 | USING intersect(i2,i1); USING WI
+----+-------------+---------+-------------+---------------+-------+---------+------+------+----------------------
----------+
1 row IN SET (0.00 sec)
```

Оптимизатор решил использовать пересечение индексов, хотя это было едва ли не самым худшим решением! Выполнение запроса заняло **360 мс**. Также обратите внимания на большую погрешность в оценке примерного количества строк.

Это произошло из-за того, что MySQL считает значения в колонках i1 и i2 независимыми, и потому выбирает пересечение индексов. На самом деле он не может предположить другого, т.к. никакой статистики о корреляции значений в колонках у него нет.

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX(i2) WHERE i1=50 AND
i2=50;
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
| 1 | SIMPLE | idxtest | ref | i1,combined | i1 | 4 | const | 106222 | USING WHERE
+----+-------------+---------+------+---------------+------+---------+-------+--------+-------------+
1 row IN SET (0.00 sec)
```

А теперь, когда мы запретили MySQL использовать индекс по колонке i2 (а значит он не может и найти пересечение индексов), он использует индекс по одной колонке, а не комбинированный. Произошло так потому, что у MySQL есть статистика о примерном количестве подходящих строк, и так как оно равно для обоих индексов, то MySQL выбрал меньший по размеру. Выполнение запроса опять заняло **290 мс** — в точности столько же, сколько и в прошлый раз.

Заставим MySQL использовать только combined индекс:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX(i1,i2) WHERE i1=50 A
i2=50;
+----+-------------+---------+------+---------------+----------+---------+-------------+--------+-------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+----------+---------+-------------+--------+-------+
| 1 | SIMPLE | idxtest | ref | combined | combined | 8 | const,const | 121137 |
+----+-------------+---------+------+---------------+----------+---------+-------------+--------+-------+
1 row IN SET (0.00 sec)
```

Видно, что MySQL примерно на 20% ошибается в оценке количества перебираемых строк, что, конечно, неверно, т.к. используется тот же префикс, что и при использовании индекса только по колонке i1. MySQL не знает этого, т.к. просматривает статистику по отдельным индексам и не пытается согласовывать их.

Из-за того, что используемый комбинированный индекс больше, чем индекс по одной колонке, выполнение запроса заняло **300 мс**.

Таким образом мы видим, что MySQL может решить использовать пересечение индексов даже в том случае, если это худший вариант, хотя с технической точки зрения это, конечно, будет лучший план, учитывая, что другой статистики у него нет.

Есть простые способы заставить MySQL не использовать пересечение индексов, но, к сожалению, мне не известно как заставить его использовать пересечения, если он считает этот вариант неоптимальным. Надеюсь, что такая возможность в будущем будет добавлена.

Наконец, давайте рассмотрим ситуацию, когда процедура нахождения пересечения индексов работает значительно лучше, чем комбинированные индексы по нескольким колонкам. Речь идёт о случае, когда мы используете **OR** при выборке между колонками. В этом случае комбинированный индекс становится совершенно бесполезным, и у MySQL есть выбор между полным сканированием таблицы (FULL SCAN) и выполнением объединения (UNION) значений вместо поиска пересечения на данных, которые он получил из одной таблице.

Я вновь изменил взначения в столбцах i1 и i2 таким образом, чтобы в них содержались независимые данные (типичная ситуация для таблиц).

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest WHERE i1=50 OR i2=50;
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+----------------------
--------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
```

```
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+-----------------
--------+
| 1 | SIMPLE | idxtest | index_merge | i1,i2,combined | i1,i2 | 4,4 | NULL | 203803 | USING union(i1,i2); USING WHI
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+-----------------
--------+
1 row IN SET (0.00 sec)
```

Такой запросы выполняется **660 мс**. Отключив индекс по второй колонке мы получим FULL SCAN:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest IGNORE INDEX(i2) WHERE i1=50 OR :
+----+-------------+---------+------+---------------+------+---------+------+----------+-------------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+------+---------------+------+---------+------+----------+-------------+
| 1 | SIMPLE | idxtest | ALL | i1,combined | NULL | NULL | NULL | 11010048 | USING WHERE
+----+-------------+---------+------+---------------+------+---------+------+----------+-------------+
1 row IN SET (0.00 sec)
```

Обратите внимание, что MySQL указал ключи i1,combined как *возможные* к использованию, однако на самом деле такой возможности у него **нет**. Выполнение такого запросы занимает **3370 мс**!

Также обратите внимание на то, что выполнение запроса заняло в 5 раз больше времени несмотря на то, что FULL SCAN прошёл примерно в 50 раз больше строк. Это показывает очень большую разницу в производительности между полным проходом по таблице и доступе по ключу, который занимает в 10 раз больше времени (в смысле «стоимости» доступа на строку), несмотря на то, что выполняется в памяти.

В случае UNION оптизатор действует более продвинуто и вполне способен справится с диапазонами:

```
mysql [localhost] {msandbox} (test)> EXPLAIN SELECT avg(length(val)) FROM idxtest WHERE i1=50 OR i2 IN (49,50);
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+-----------------
-------------+
| id | select_type | TABLE | type | possible_keys | KEY | key_len | ref | rows | Extra
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+-----------------
-------------+
| 1 | SIMPLE | idxtest | index_merge | i1,i2,combined | i1,i2 | 4,4 | NULL | 299364 | USING sort_union(i1,i2); USII
WHERE
+----+-------------+---------+-------------+----------------+-------+---------+------+--------+-----------------
-------------+
1 row IN SET (0.00 sec)
```

Подводя итоги

В большинстве случаев использование комбинированных индексов по нескольким колонкам является лучшим решением, если вы используете AND между подобными колонками в WHERE. Использование пересечения индексов в принципе улучшает производительность, но она всё равно значительно хуже, чем при использовании комбинированных ключей. В случае, если Вы используете OR между колонками Вам потребуется иметь по индексу на каждой из колонок, чтобы MySQL смог найти их пересечения, а комбинированные индексы не могут использоваться в таких запросах.

mysql, index, индексы, индекс, multicolumn, многоколоночные индексы

Вадим @WASD42 карма 86,2 рейтинг 0,0
Backend-программист

Похожие публикации

+84 Кластерные и «обычные» индексы MySQL (InnoDB)
38,2k 486 33

+65 Как расширение индекса в InnoDB таблицах удивительным образом снижает производительность
2,2k 90 61