### 1、Why Index

从概念上讲，数据库是数据表的集合，数据表是数据行和数据列的集合。当你执行一个SELECT语句从数据表中查询部分数据行的时候，得到的就是另外一个数据表和数据行的集合。

当然，我们都希望获得这个新的集合的时间尽可能地短，效率尽可能地高，这就是优化查询。

提升查询速度的技术有很多，其中最重要的就是索引。当你发现自己的查询速度慢的时候，最快解决问题的方法就是使用索引。索引的使用是影响查询速度的重要因素。在使用索引之前其他的优化查询的动作纯粹是浪费时间，只有合理地使用索引之后，才有必要考虑其他优化方式。

### 2、索引是如何工作的

首先，在你的MySQL上创建t\_user\_action\_log表，方便下面进行演示。

通过查询分析器explain分析这条查询语句：

EXPLAIN SELECT id, name, ip\_address FROM t\_user\_action\_log

WHERE `action`=2;

分析结果如下：



其中type为ALL表示要进行全表扫描。这样效率无疑是极慢的。

下面为action列添加索引：

ALTER TABLE t\_user\_action\_log ADD INDEX (action);

然后再次执行查询分析，结果如下：



那么为什么索引会提高查询速度呢？原因是索引会根据索引值进行分类，这样就不用再进行全表扫描了。我们看到这次查询就使用索引了。加索引前Extra的值是Using Where，加索引后Extra的值为空。



比如上图，action值为2的索引值分类存储在了索引空间，可以快速地查询到索引值所对应的列。

### 3、创建索引

3种方式：

使用CREATE INDEX创建，语法如下：

CREATE INDEX indexName ON tableName (columnName(length));

例如我们对ip\_address这一列创建一个长度为16的索引：

CREATE INDEX index\_ip\_addr ON t\_user\_action\_log (ip\_address(16));

使用ALTER语句创建，语法如下：

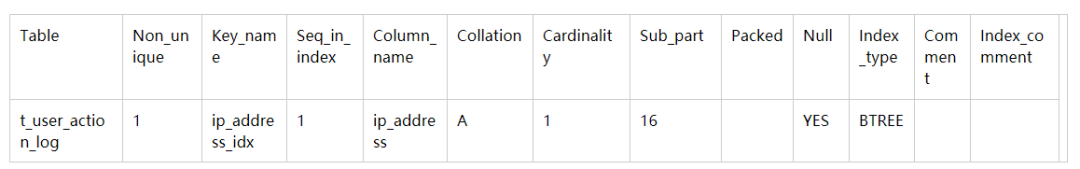
ALTER TABLE tableName ADD INDEX indexName(columnName);

ALTER语句创建索引前面已经有例子了。下面提供一个设置索引长度的例子：

ALTER TABLE t\_user\_action\_log

ADD INDEX ip\_address\_idx (ip\_address(16));

SHOW INDEX FROM t\_user\_action\_log;



CREATE TABLE tableName( 建表的时候创建索引：

id INT NOT NULL,

columnName columnType,

INDEX [indexName] (columnName(length))

);

### 4、查看索引

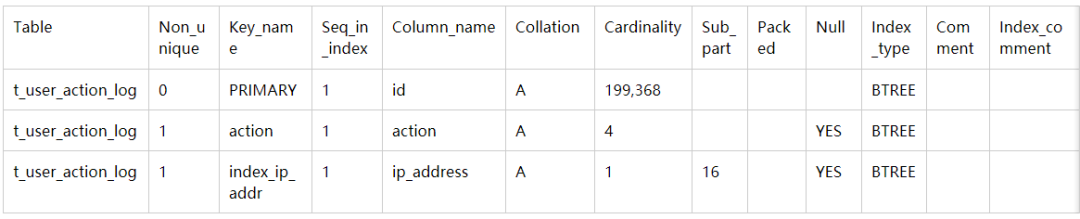
可以通过show语句查看索引：

SHOW INDEX FROM t\_user\_action\_log;

### 5、删除索引

使用ALTER命令可以删除索引，例如：

ALTER TABLE t\_user\_action\_log DROP INDEX index\_ip\_addr;



### 6、索引的使用原则

索引由于其提供的优越的查询性能，似乎不使用索引就是一个愚蠢的行为了。但是使用索引，是要付出时间和空间的代价的。因此，索引虽好不可贪多。

下面介绍几个索引的使用技巧和原则，在使用索引之前，你应该对它们有充分的认识。

写操作比较频繁的列慎重加索引

索引在提高查询速度的同时，也由于需要更新索引而带来了降低插入、删除和更新带索引列的速度的问题。一张数据表的索引越多，在写操作的时候性能下降的越厉害。

索引越多占用磁盘空间越大

与没有加索引比较，加索引会更快地使你的磁盘接近使用空间极限。

不要为输出列加索引

为查询条件、分组、连接条件的列加索引，而不是为查询输出结果的列加索引。

例如下面的查询语句：

select ip\_address from t\_user\_action\_log

where name='LiSi'

group by action

order by create\_time;

所以可以考虑增加在 name action create\_time 列上，而不是 ip\_address。

考虑维度优势

例如action列的值包含：1、2、3、4、5，那么该列的维度就是5。

维度越高（理论上维度的最大值就是数据行的总数），数据列包含的独一无二的值就越多，索引的使用效果越好。

对于维度很低的数据列，索引几乎不会起作用，因此没有必要加索引。

例如性别列的值只有男和女，每种查询结果占比大约50%。一般当查询优化处理器发现查询结果超过全表的30%的时候，就会跳过索引，直接进行全表扫描。

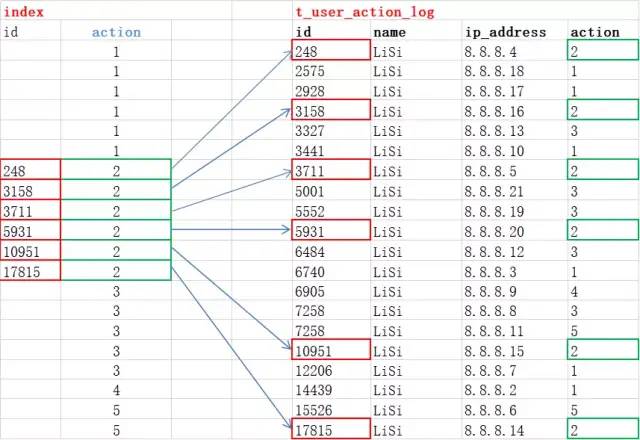
对短小的值加索引

对短小的值加索引，意味着索引所占的空间更小，可以减少I/O活动，同时比较索引的速度也更快。

尤其是主键，要尽可能短小。

另外，InnoDB使用的是聚集索引（clustered index），也就是把主键和数据行保存在一起。主键之外的其他索引都是二级索引，这些二级索引也保留着一份主键，这样在查询到索引以后，就可以根据主键找到对应的数据行。如果主键太长的话，会造成二级索引占用的空间变大。

比如下面的action索引保存了对应行的id。



为字符串前缀加索引

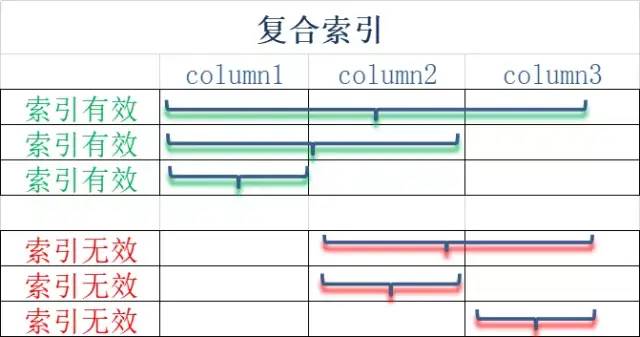
前边已经讲过短小索引的种种好处了，有时候一个字符串的前几个字符就能唯一标识这条记录，这个时候设置索引的长度就是非常划算的做法。

前面已经提供了设置索引length的例子，这里就不举例子了。

复合索引的左侧索引

创建复合索引的语法如下：

CREATE INDEX indexName ON tableName (column1 DESC, column2 DESC, column3 ASC);



我们可以看到，最左侧的column1索引总是有效的。

索引加锁

对于InnoDB来说，索引可以让查询锁住更少的行，从而可以在并发情况下拥有更佳表现。

下面演示一下查询锁与索引之间的关系。

前面使用的t\_user\_action\_log表目前有一个id为主键，还有一个二级索引action。

下面这条语句的修改范围是id值为1 2 3 4所在的行，查询锁会锁住id值为1 2 3 4 5所在的行。

update ijiangtao\_local\_db\_mysql.t\_user\_action\_log set name='c1' where id<5;

首先创建数据库连接1，开启事务，并执行update语句

set autocommit=0;

begin;

update ijiangtao\_local\_db\_mysql.t\_user\_action\_log set name='c1' where id<5;

然后开启另外一个连接2，分别执行下面几个update语句

-- 没有被锁

update ijiangtao\_local\_db\_mysql.t\_user\_action\_log set name='c2' where id=6;

-- 被锁

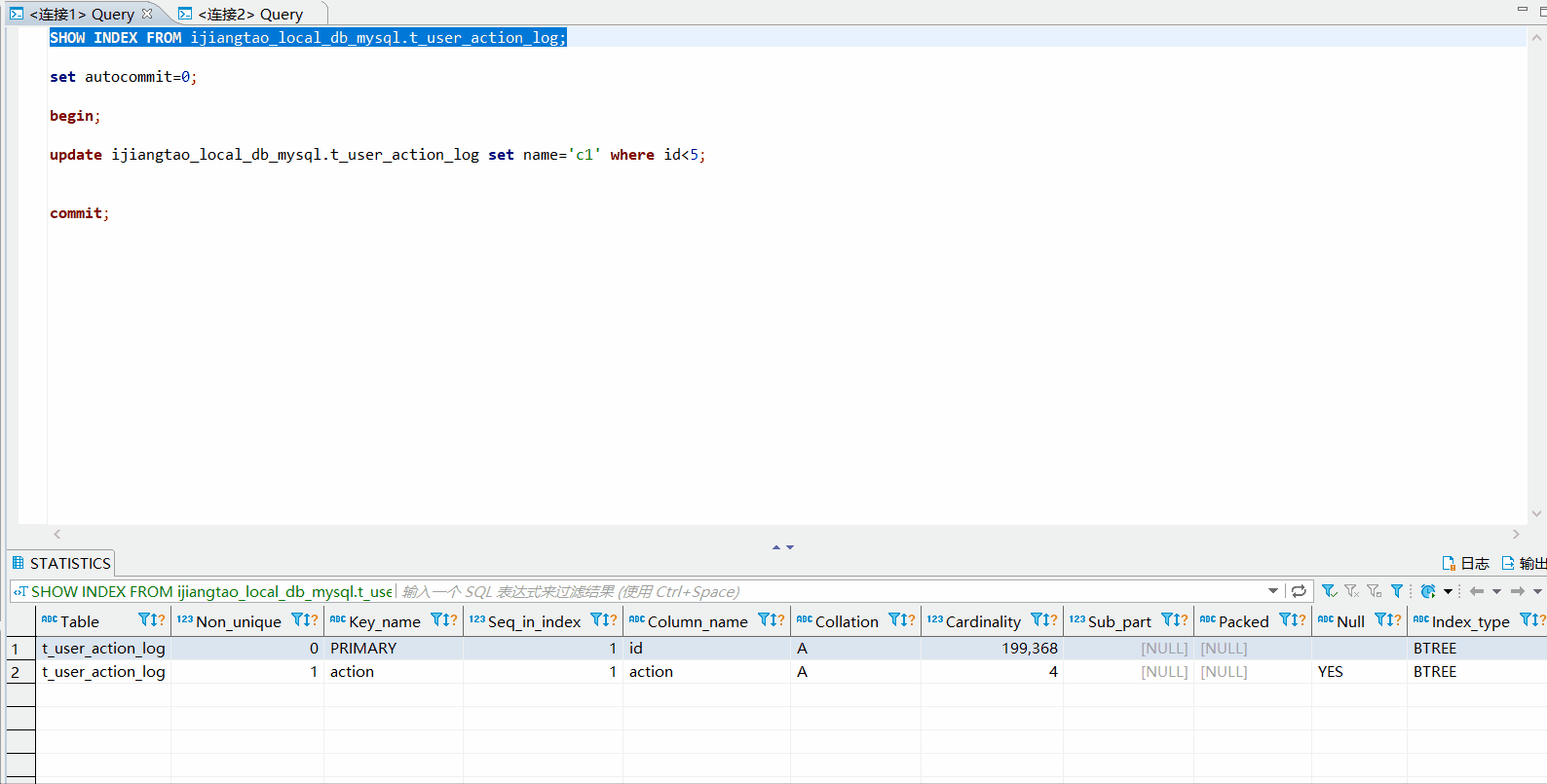
update ijiangtao\_local\_db\_mysql.t\_user\_action\_log set name='c2' where id=5;

你会发现id=5的数据行已经被锁定，id=6的数据行可以正常提交。

连接1提交事务，连接2的id=1和id=5的数据行可以update成功了。

-- 在连接1提交事务

commit;



如果不使用索引

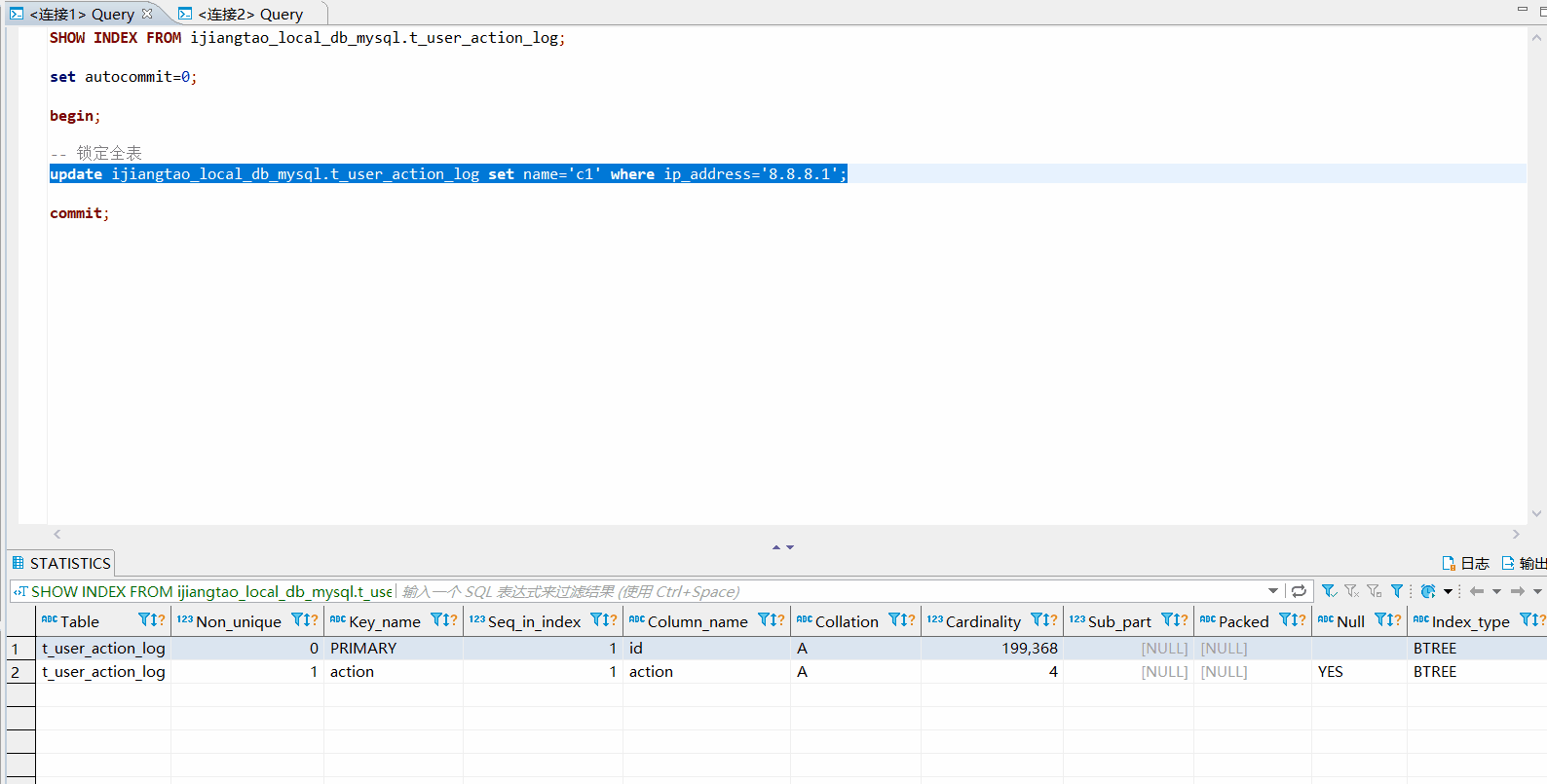
ip\_address没有索引的话，会锁定全表。

连接1开启事务以后commit;之前，连接2对该表的update全部需要等待连接1释放锁。

set autocommit=0;

begin;

update ijiangtao\_local\_db\_mysql.t\_user\_action\_log set name='c1' where ip\_address='8.8.8.1';



覆盖索引

如果索引包含满足查询的所有数据，就被称为覆盖索引(Covering Indexes)，覆盖索引非常强大，可以大大提高查询性能。

覆盖索引高性能的原因是：

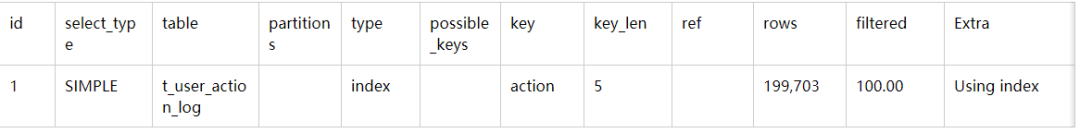
索引通常比记录要小，覆盖索引查询只需要读索引，而不需要读记录。

索引都按照值的大小进行顺序存储，相比与随机访问记录，需要更少的I/0。

大多数数据引擎能更好的缓存索引，例如MyISAM只缓存索引。

ijiangtao\_local\_db\_mysql表的action列包含索引。使用explain分析下面的查询语句，对于索引覆盖查询(index-covered query)，分析结果Extra的值是Using index，表示使用了覆盖索引 :

explain select `action` from ijiangtao\_local\_db\_mysql.t\_user\_action\_log;



聚簇索引

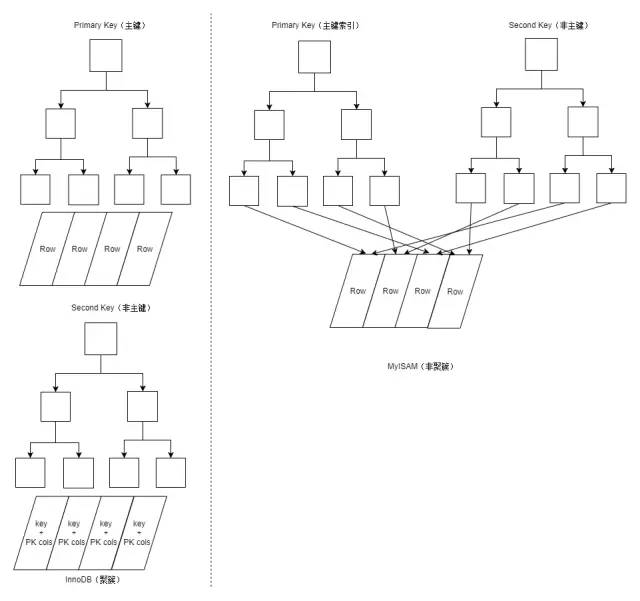
聚簇索引(Clustered Indexes)保证关键字的值相近的元组存储的物理位置也相同，且一个表只能有一个聚簇索引。

字符串类型不建议使用聚簇索引，特别是随机字符串，因为它们会使系统进行大量的移动操作。

并不是所有的存储引擎都支持聚簇索引，目前InnoDB支持。

如果使用聚簇索引，最好使用AUTO\_INCREMENT列作为主键，应该尽量避免使用随机的聚簇主键。

从物理位置上看，聚簇索引表比非聚簇的索引表，有更好的访问性能。



选择合适的索引类型

从数据结构角度来看，MySQL支持的索引类型有B树索引、Hash索引等。

B树索引

B树索引对于<、<=、 =、 >=、 >、 <>、!=、 between查询，进行精确比较操作和范围比较操作都有比较高的效率。

B树索引也是InnoDB存储引擎默认的索引结构。

Hash索引

Hash索引仅能满足=、<=>、in查询。

Hash索引检索效率非常高，索引的检索可以一次定位，不像B树索引需要从根节点到枝节点，最后才能访问到页节点这样多次的I/O访问，所以Hash索引的查询效率要远高于B树索引。但Hash索引不能使用范围查询。

查询优化建议

下面提供几个查询优化的建议。

使用explain分析查询语句

前面已经演示过如何使用explain命令分析查询语句了，这里再解释一下其中几个有参考价值的字段的含义：

select\_type

select\_type表示查询中每个select子句的类型，一般有下面几个值:

SIMPLE

简单SELECT,不使用UNION或子查询等。

PRIMARY

查询中若包含任何复杂的子部分,最外层的select被标记为PRIMARY。

UNION

UNION中的第二个或后面的SELECT语句。

DEPENDENT UNION

UNION中的第二个或后面的SELECT语句，取决于外面的查询。

UNION RESULT

UNION的结果。

SUBQUERY

子查询中的第一个SELECT。

DEPENDENT SUBQUERY

子查询中的第一个SELECT，取决于外面的查询。

DERIVED

派生表的SELECT, FROM子句的子查询。

UNCACHEABLE SUBQUERY

一个子查询的结果不能被缓存，必须重新评估外链接的第一行。

type

type表示MySQL在表中找到所需行的方式，又称“访问类型”，常用的类型有：

ALL, index, range, ref, eq\_ref, const, system, NULL。

从左到右，性能从差到好。

ALL：

Full Table Scan，MySQL将遍历全表以找到匹配的行。

index:

Full Index Scan，index与ALL区别为index类型只遍历索引树。

range:

只检索给定范围的行，使用一个索引来选择行。

ref:

表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值。

eq\_ref:

类似ref，区别就在使用的索引是唯一索引，对于每个索引键值，表中只有一条记录匹配，简单来说，就是多表连接中使用primary key或者 unique key作为关联条件。

const:

当MySQL对查询某部分进行优化，并转换为一个常量时，使用这些类型访问。

如将主键置于where列表中，MySQL就能将该查询转换为一个常量。

NULL:

MySQL在优化过程中分解语句，执行时甚至不用访问表或索引，例如从一个索引列里选取最小值可以通过单独索引查找完成。

Key

key列显示MySQL实际决定使用的键（索引），如果没有选择索引，键是NULL。

possible\_keys

possible\_keys指出MySQL能使用哪个索引在表中找到记录，查询涉及到的字段上如果存在索引则该索引将被列出，但不一定被查询使用。

ref

ref表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值。

rows

rows表示MySQL根据表统计信息，以及索引选用的情况，找到所需记录需要读取的行数。这个行数是估算的值，实际行数可能不同。

用好explain命令是查询优化的第一步 !

声明NOT NULL

当数据列被声明为NOT NULL以后，在查询的时候就不需要判断是否为NULL，由于减少了判断，可以降低复杂性，提高查询速度。

如果要表示数据列为空，可以使用0等代替。

考虑使用数值类型代替字符串

MySQL对数值类型的处理速度要远远快于字符串，而且数值类型往往更加节省空间。

例如对于“Male”和“Female”可以用“0”和“1”进行代替。

考虑使用ENUM类型

如果你的数据列的取值是确定有限的，可以使用ENUM类型代替字符串。因为MySQL会把这些值表示为一系列对应的数字，这样处理的速度会提高很多。

CREATE TABLE shirts (

name VARCHAR(40),

size ENUM('x-small', 'small', 'medium', 'large', 'x-large')

);

INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'),

('polo shirt','small');

SELECT name, size FROM shirts WHERE size = 'medium';

总结

索引是一个单独的，存储在磁盘上的数据结构，索引对数据表中一列或者多列值进行排序，索引包含着对数据表中所有数据的引用指针。

本教程从MySQL开始讲起，又介绍了MySQL中索引的使用，最后提供了使用索引的几条原则和优化查询的几个方法。

无论你是DBA还是软件开发，菜鸟程序员还是资深工程师，相信本节提到的关于索引的知识，对你都会有所帮助。