第4章 查 询 优 化

关系数据库的世界是一个表与集合、表与集合上的运算占统治地位的世界。数据库是一个表的集合，而表又是行和列的集合。在发布一条 SELECT 查询从表中进行检索行时，得到另一个行和列的集合。这些都是一些抽象的概念，对于数据库系统用来操纵表中数据的基本表示没有多少参考价值。另一个抽象概念是，表上的运算都同时进行；查询是一种概念性的集合运算，并且集合论中没有时间概念。

当然，现实世界是相当不同的。数据库管理系统实现了抽象的概念，但是在实际的硬件范围内要受到实际的物理约束。结果是，查询要花时间，有时要花很长的时间。而人类很容易不耐烦，不喜欢等待，因此我们丢下了集合上的那些瞬间的数学运算的抽象世界去寻求加速查询的方法。幸运的是，有几种加速运算的技术，可对表进行索引使数据库服务器查找行更快。可考虑怎样充分利用这些索引来编写查询。可编写影响服务器调度机制的查询，使来自多个客户机的查询协作得更好。我们思考基本硬件怎样运行，以便想出怎样克服其物理约束对性能进行改善的方法。

这些正是本章所要讨论的问题，其目标是优化数据库系统的性能，使其尽可能快地处理各种查询。MySQL 已经相当快了，但即使是最快的数据库，在人的设计下还能运行得更快。

4.1 使用索引

我们首先讨论索引，因为它是加快查询的最重要的工具。还有其他加快查询的技术，但是最有效的莫过于恰当地使用索引了。在 MySQL 的邮件清单上，人们通常询问关于使查询更快的问题。在大量的案例中，都是因为表上没有索引，一般只要加上索引就可以立即解决问题。但这样也并非总是有效，因为优化并非总是那样简单。然而，如果不使用索引，在许多情形下，用其他手段改善性能只会是浪费时间。应该首先考虑使用索引取得最大的性能改善，然后再寻求其他可能有帮助的技术。

本节介绍索引是什么、它怎样改善查询性能、索引在什么情况下可能会降低性能，以及怎样为表选择索引。下一节，我们将讨论 MySQL 的查询优化程序。除了知道怎样创建索引外，了解一些优化程序的知识也是有好处的，因为这样可以更好地利用所创建的索引。某些编写查询的方法实际上会妨碍索引的效果，应该避免这种情况出现。（虽然并非总会这样。有时也会希望忽略优化程序的作用。我们也将介绍这些情况。）

4.1.1 索引的益处

让我们从一个无索引的表着手来考察索引是怎样起作用的。无索引的表就是一个无序的行集。例如，图4-1给出了我们在第1章“ MySQL 与 SQL 介绍 ”中首先看到的 ad 表。这个表上没有索引，因此如果我们查找某个特定公司的行时，必须查看表中的每一行，看它是否与所需的值匹配。这是一个全表扫描，很慢，如果表中只有少数几个记录与搜索条件相匹配，则其效率是相当低的。

图4-2给出了相同的表，但在表的 company_num 列上增加了一个索引。此索引包含表中每行的一项，但此索引是在 company_num 上排序的。现在，不需要逐行搜索全表查找匹配的条款，而是可以利用索引进行查找。假如我们要查找公司 13的所有行，那么可以扫描索引，结果得出3行。然后到达公司14的行，这是一个比我们正在查找的要大的号码。索引值是排序的，因此在读到包含 14的记录时，我们知道不会再有匹配的记录，可以退出了。如果查找一个值，它在索引表中某个中间点以前不会出现，那么也有找到其第一个匹配索引项的定位算法，而不用进行表的顺序扫描（如二分查找法）。这样，可以快速定位到第一个匹配的值，以节省大量搜索时间。数据库利用了各种各样的快速定位索引值的技术，这些技术是什么并不重要，重要的是它们工作正常，索引技术是个好东西。

有人会问，为什么不只对数据文件进行排序，省掉索引文件？这样不也在搜索时产生相同的效果吗？问得好，如果只有单个索引时，是这样的。不过有可能会用到第二个索引，但同时以两种不同的方法对同一个数据文件进行排序是不可能的。（如，想要一个顾客名的索引，同时又要一个顾客 ID 号或电话号码的索引。）将索引文件作为一个与数据文件独立的实体就解决了这个问题，而且允许创建多个索引。此外，索引中的行一般要比数据文件中的行短。在插入或删除值时，为保持排序顺序而移动较短的索引值与移动较长的数据行相比更为容易。

ad 表

company_num	ad_num	hit_fee
14	48	0.01
23	49	0.02
17	52	0.01
13	55	0.03
23	62	0.02
23	63	0.01
23	64	0.02
13	77	0.03
23	99	0.03
14	101	0.01
13	102	0.01
17	119	0.02

图4-1 无索引的 ad 表

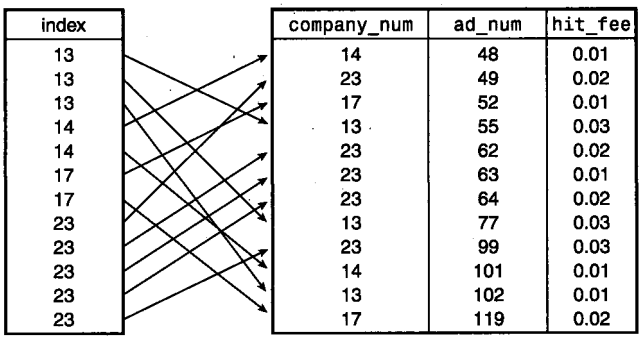

图4-2 有索引的 ad 表

这个例子与 MySQL 索引表的方法相符。表的数据行保存在数据文件中，而索引值保存在索引文件中。一个表上可有不止一个索引；如果确实有不止一个索引，它们都保存在同一个索引文件中。索引文件中的每个索引由排过序的用来快速访问数据文件的键记录数组构成。

前面的讨论描述了单表查询中索引的好处，其中使用索引消除了全表扫描，极大地加快了搜索的速度。在执行涉及多个表的连接查询时，索引甚至会更有价值。在单个表的查询中，每列需要查看的值的数目就是表中行的数目。而在多个表的查询中，可能的组合数目极大，因为这个数目为各表中行数之积。

假如有三个未索引的表 t1、t2、t3，分别只包含列 c1、c2、c3，每个表分别由含有数值 1

到 1000 的 1000 行组成。查找对应值相等的表行组合的查询如下所示：

```
SELECT c1, c2, c3
FROM t1, t2, t3
WHERE c1 = c2 AND c1 = c3
```

此查询的结果应该为 1000 行，每个组合包含 3 个相等的值。如果我们在无索引的情况下处理此查询，则不可能知道哪些行包含那些值。因此，必须寻找出所有组合以便得出与 WHERE 子句相配的那些组合。可能的组合数目为 1000×1000×1000（十亿），比匹配数目多一百万倍。很多工作都浪费了，并且这个查询将会非常慢，即使在如像 MySQL 这样快的数据库中执行也会很慢。而这还是每个表中只有 1000 行的情形。如果每个表中有一百万行时，将会怎样？很显然，这样将会产生性能极为低下的结果。如果对每个表进行索引，就能极大地加速查询进程，因为利用索引的查询处理如下：

1) 如下从表 t1 中选择第一行，查看此行所包含的值。

2) 使用表 t2 上的索引，直接跳到 t2 中与来自 t1 的值匹配的行。类似，利用表 t3 上的索引，直接跳到 t3 中与来自 t1 的值匹配的行。

3) 进到表 t1 的下一行并重复前面的过程直到 t1 中所有的行已经查过。

在此情形下，我们仍然对表 t1 执行了一个完全扫描，但能够在表 t2 和 t3 上进行索引查找直接取出这些表中的行。从道理上说，这时的查询比未用索引时要快一百万倍。

如上所述，MySQL 利用索引加速了 WHERE 子句中与条件相配的行的搜索，或者说在执行连接时加快了与其他表中的行匹配的行的搜索。它也利用索引来改进其他操作的性能：

- 在使用 MIN( ) 和 MAX( ) 函数时，能够快速找到索引列的最小或最大值。
- MySQL 常常能够利用索引来完成 ORDER BY 子句的排序操作。
- 有时，MySQL 可避免对整个数据文件的读取。假如从一个索引数值列中选择值，而且不选择表中其他列。这时，通过对索引值的读取，就已经得到了读取数据文件所要得到的值。没有对相同的值进行两次读取的必要，因此，甚至无需涉及数据文件。

4.1.2 索引的弊端

一般情况下，如果 MySQL 能够知道怎样用索引来更快地处理查询，它就会这样做。这表示，在大多数情况下，如果您不对表进行索引，则损害的是您自己的利益。可以看出，作者描绘了索引的诸多好处。但有不利之处吗？是的，有。实际上，这些缺点被优点所掩盖了，但应该对它们有所了解。

首先，索引文件要占磁盘空间。如果有大量的索引，索引文件可能会比数据文件更快地达到最大的文件尺寸。其次，索引文件加快了检索，但增加了插入和删除，以及更新索引列中的值的时间（即，降低了大多数涉及写入的操作的时间），因为写操作不仅涉及数据行，而且还常常涉及索引。一个表拥有的索引越多，则写操作的平均性能下降就越大。在 4.4节“有效地装载数据”中，我们将更为详细地介绍这些性能问题，并讨论怎样解决。

4.1.3 选择索引

创建索引的语法已经在3.4.3节“创建和删除索引”中进行了介绍。这里，我们假定您已经阅读过该节。但是知道语法并不能帮助确定表怎样进行索引。要决定表怎样进行索引需要考虑表的使用方式。本节介绍一些关于怎样确定和挑选索引列的准则：

■ **搜索的索引列，不一定是所要选择的列**。换句话说，最适合索引的列是出现在 WHERE 子句中的列，或连接子句中指定的列，而不是出现在 SELECT 关键字后的选择列表中的列：

```
SELECT
      col_a                ←不适合作索引列
FROM
 Tbl1 LEFT JOIN tbl2
      ON tbl1.col_b = tbl2.col_c     ←适合作索引列
WHERE
      col_d = expr              ←适合作索引列
```

当然，所选择的列和用于 WHERE 子句的列也可能是相同的。关键是，列出现在选择列表中不是该列应该索引的标志。

出现在连接子句中的列或出现在形如 col1 = col2 的表达式中的列是很适合索引的列。查询中的 col_b 和 col_c 就是这样的例子。如果 MySQL 能利用连接列来优化一个查询，表示它通过消除全表扫描相当可观地减少了表行的组合。

■ **使用惟一索引**。考虑某列中值的分布。对于惟一值的列，索引的效果最好，而具有多个重复值的列，其索引效果最差。例如，存放年龄的列具有不同值，很容易区分各行。而用来记录性别的列，只含有“ M ”和“ F ”，则对此列进行索引没有多大用处（不管搜索哪个值，都会得出大约一半的行）。

■ **使用短索引**。如果对串列进行索引，应该指定一个前缀长度，只要有可能就应该这样做。例如，如果有一个 CHAR(200) 列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。对前 10 个或 20 个字符进行索引能够节省大量索引空间，也可能会使查询更快。较小的索引涉及的磁盘 I/O 较少，较短的值比较起来更快。更为重要的是，对于较短的键值，索引高速缓存中的块能容纳更多的键值，因此，MySQL 也可以在内存中容纳更多的值。这增加了找到行而不用读取索引中较多块的可能性。（当然，应该利用一些常识。如仅用列值的第一个字符进行索引是不可能有多大好处的，因为这个索引中不会有许多不同的值。）

■ **利用最左前缀。**在创建一个 n 列的索引时，实际是创建了 MySQL 可利用的 n 个索引。多列索引可起几个索引的作用，因为可利用索引中最左边的列集来匹配行。这样的列集称为最左前缀。（这与索引一个列的前缀不同，索引一个列的前缀是利用该的前 n 个字符作为索引值。）

假如一个表在分别名为 state、city 和 zip 的三个列上有一个索引。索引中的行是按 state/city/zip 的次序存放的，因此，索引中的行也会自动按 state/city 的顺序和 state 的顺序存放。这表示，即使在查询中只指定 state 值或只指定 state 和 city 的值，MySQL 也可以利用索引。因此，此索引可用来搜索下列的列组合：

```
state, city, zip
state, city
state
```

MySQL 不能使用不涉及左前缀的搜索。例如，如果按 city 或 zip 进行搜索，则不能使用该索引。如果要搜索某个州以及某个 zip 代码（索引中的列1和列3），则此索引不能用于相应值的组合。但是，可利用索引来寻找与该州相符的行，以减少搜索范围。

■ **不要过度索引**。不要以为索引“越多越好”，什么东西都用索引是错的。每个额外的索引都要占用额外的磁盘空间，并降低写操作的性能，这一点我们前面已经介绍过。在修改表的内容时，索引必须进行更新，有时可能需要重构，因此，索引越多，所花的时间越长。如果有一个索引很少利用或从不使用，那么会不必要地减缓表的修改速度。此外，MySQL 在生成一个执行计划时，要考虑各个索引，这也要费时间。创建多余的索引给查询优化带来了更多的工作。索引太多，也可能会使 MySQL 选择不到所要使用的最好索引。只保持所需的索引有利于查询优化。

如果想给已索引的表增加索引，应该考虑所要增加的索引是否是现有多列索引的最左索引。如果是，则就不要费力去增加这个索引了，因为已经有了。

■ **考虑在列上进行的比较类型**。索引可用于“ < ”、“ <= ”、“ = ”、“ >= ”、“ > ”和 BETWEEN 运算。在模式具有一个直接量前缀时，索引也用于 LIKE 运算。如果只将某个列用于其他类型的运算时（如 STRCMP( )），对其进行索引没有价值。

4.2 MySQL 查询优化程序

在发布一个选择行的查询时，MySQL 进行分析，看是否能够对它进行优化，使它执行更快。本节中，我们将研究查询优化程序怎样工作。更详细的信息，可参阅 MySQL 参考指南中的“ Getting Maximum Performance from MySQL ”，该章描述了 MySQL 采用的各种优化措施。该章中的信息会不断变化，因为 MySQL 的开发者不断对优化程序进行改进，因此，有必要经常拜访一下该章，看看是否有可供利用的新技巧。（http://www.mysql.com/ 处的 MySQL 联机参考指南在不断地更新。）

MySQL 查询优化程序利用了索引。当然，它也利用了其他信息。例如，如果发布下列查询，MySQL 将非常快地执行它，不管相应的表有多大：

```
SELECT * FROM tbl_name WHERE 1 = 0
```

在此情形中，MySQL 考察 WHERE 子句，如果认识到不可能有满足该查询的行，就不会对该表进行搜索。可利用 EXPLAIN 语句知道这一点，EXPLAIN 语句要求 MySQL 显示某些有关它应该执行一条 SELECT 查询，而实际没有执行的信息。为了使用 EXPLAIN，只需要 SELECT 语句前放置 EXPLAIN 即可，如下所示：

```
EXPLAIN SELECT * FROM tbl_name WHERE 1 = 0
```

EXPLAIN 的输出结果为：

```
+-----------------+
¦ Comment         ¦
+-----------------+
¦ Impossible WHERE ¦
+-----------------+
```

通常，EXPLAIN 返回的信息比这个多，包括将用来扫描表的索引、将要使用的连接类型以及需要在每个表中扫描的行数估计等等。

4.2.1 优化程序怎样工作

MySQL 查询优化程序有几个目标，但其主要目标是尽量利用索引，而且尽量使用最具有限制性的索引以排除尽可能多的行。这样做可能会适得其反，因为发布一条 SELECT 语句的目的是寻找行，而不是拒绝它们。优化程序这样工作的原因是从要考虑的行中排除行越快，

那么找到确实符合给出标准的行就越快。如果能够首先进行最具限制性的测试，则查询可以进行得更快。假如有一个测试两列的查询，每列上都有一个索引：

```
WHERE col1 = "some value" AND col2 = "some other value"
```

还假定，与 col1 上的测试相符的有 900 行，与 col2 上的测试相符的有 300 行，而两个测试都通过的有 30 行。如果首先测试 col1，必须检查 900 行以找到也与 col2 值相符的 30 行。那么测试中有 870 将失败。如果首先测试 col2，要找到也与 col1 值相符的30 行，只需检查 300 行。测试中有失败 270 次，这样所涉及的计算较少，磁盘 I/O 也较少。

遵循下列准则，有助于优化程序利用索引：

- 比较具有相同类型的列。在比较中利用索引列时，应该使用那些类型相同的列。例如，CHAR(10) 被视为与 CHAR(10) 或 VARCHAR(10) 相同，但不同于 CHAR(12) 和 VARCHAR(12)。INT 与 BIGINT 不同。在 MySQL 3.23 版以前，要求使用相同类型的列，否则列上的索引将不起作用。自 3.23 版后，不严格要求这样做，但相同的列类型比不同类型提供更好的性能。如果所比较的两列类型不同，可使用 ALTER TABLE语句修改其中之一使它们的类型相配。
- 比较中应尽量使索引列独立。如果在函数调用或算术表达式中使用一个列，则 MySQL 不能使用这样的索引，因为它必须对每行计算表达式的值。有时，这是不可避免的，但很多时候，可以重新编写只取索引列本身的查询。

  下面的 WHERE 子句说明了怎样进行这项工作。第一行中，优化程序将简化表达式 4/2 为值 2，然后使用 my_col 上的索引快速地找到小于 2 的值。而在第二个表达式中，MySQL 必须检索出每行的 my_col 值，乘以 2，然后将结果与 4 比较。没索引可用，因为列中的每个值都要检索，以便能对左边的表达式求值：

```
WHERE my_col < 4 / 2
WHERE my_col * 2 < 4
```

  让我们考虑另一个例子。假如有一个索引列 date_col。如果发布如下的查询，相应的索引未被使用：

```
SELECT * FROM my_tbl WHERE YEAR(date_col) < 1990
```

  其中表达式并不将索引列与 1990 比较，而是将从列值计算出的值用于比较，而且必须计算每行的这个值。结果是，date_col 上的索引不可能得到使用。怎样解决？使用一个文字日期即可，这时将会使用 date_col 上的索引：

```
WHERE date_col < "1990-01-01"
```

  但是假如没有特定的日期值，那么可能会对找到具有出现在距今一定天数内的日期的记录感兴趣。有几种方法来编写这样的查询，但并非所有方法都很好。三种可能的方法如下：

```
WHERE TO_DAYS(date_col) - TO_DAYS(CURRENT_DATE) < cutoff
WHERE TO_DAYS(date_col) < cutoff + TO_DAYS(CURRENT_DATE)
WHERE date_col < DATE_ADD(CURRENT_DATE, INTERVAL cutoff DAY)
```

  其中第一行不能利用索引，因为必须为每行检索列，以便能够计算 TO_DAYS(date_col) 的值。第二行要好一些。cutoff 和 TO_DAYS(CURRENT_DATE) 两者都是常量，因此比较表达式的右边可在查询处理前由优化程序一次计算出来，而不是每行计算一次。但 date_col 列仍然出现在一个函数调用中，因此，没有使用索引。第三行是最好的方法。比较表达式的右边可在执行查询前作为常量一次计算出来，但现在其值是一个

日期。这个值可直接与 date_col 的值进行比较，不再需要转换为天数，可以利用索引。

- **在 LIKE 模式的起始处不要使用通配符**。有时，有的人会用下列形式的 WHERE 子句来搜索串：

```
WHERE col_name LIKE "%string%"
```

如果希望找到 string，不管它出现在列中任何位置，那么这样做是对的。但不要出于习惯在串的两边加“ % ”。如果实际要查找的只是出现在列的开始处的串，则不应该要第一个“ % ”号。例如，如果在一个包含姓的列中查找“ Mac ”起始的姓，应该编写如下的 WHERE 子句：

优化程序考虑模式中的开始的文字部分，然后利用索引找到相符合的行。不过宁

```
WHERE last_name LIKE "Mac%"
```

可写成如下的表达式，它允许使用 last_name 上的索引：

```
WHERE last_name >= "Mac" AND last_name < "Mad"
```

这种优化对使用 REGEXP 操作符的模式匹配不起作用。

- **帮助优化程序更好地评估索引的有效性。**缺省时，如果将索引列中的值与常量进行比较，优化程序将假定键字是均匀地分布在索引中的。优化程序还将对索引进行一个快速的检查，以估计在确定相应的索引是否应该用于常量的比较时要使用多少条目。可利用 myisamchk 或 isamchk 的 --analyze 选项给优化程序提供更好的信息，以便分析键值的分布。myisamchk 用于 MyISAM 表，isamchk 用于 ISAM 表。为了完成键值分析，必须能够登录到 MySQL 服务器主机中，而且必须对表文件具有写访问权限。
- **利用 EXPLAIN 检验优化程序操作**。检查用于查询中的索引是否能很快地排除行。如果不能，那么应该试一下利用 STRAIGHT_JOIN 强制按特定次序使用表来完成一个连接。查询的执行方式不那么显然；MySQL 可能会有很多理由不以您认为最好的次序使用索引。
- **测试查询的其他形式，而且不止一次地运行它们**。在测试一个查询的其他形式时，应该每种方法运行几次。如果对两个不同方法中的每种只运行查询一次，通常会发现第二个查询更快，因为来自第一个查询的信息在磁盘高速缓存中，不需要实际从磁盘上读出。还应该尽量在系统负载相对平稳的时候运行查询，以避免受系统中其他活动的影响。

4.2.2 忽略优化

这可能听起来有点奇怪，但在以下情况中，要废除 MySQL 的优化功能：

- **强迫 MySQL 慢慢地删除表的内容**。在需要完全删空一个表时，利用无 WHERE 子句的 DELETE 语句删除整个表的内容是最快的，如下所示：

```
DELETE FROM tbl_name
```

MySQL 对这种特殊情况的 DELETE 进行优化；它利用表信息文件中的表说明从头开始创建空数据文件和索引文件。这种优化使 DELETE 操作极快，因为 MySQL 无需单独地删除每一行。但在某些情况下，这样做会产生一些不必要的负作用：

  - MySQL 报告所涉及的行数为零，即使表不为空也是如此。很多时候这没有关系（虽然，如果事先没有思想准备，会感到困惑不解），但对于那些确实需要知道真实行数的应用程序来说，这是不恰当的。

■ 如果表含有一个 AUTO_INCREMENT 列，则该列的顺序编号会以 1 从头开始。这是真实的事情，即使在 MySQL 3.23 中对 AUTO_INCREMENT 的处理进行了改进后也是这样。关于这个改进的介绍请参阅第2章中的“使用序列”小节。可增加 WHERE 1 > 0 子句对 DELETE 语句“不优化”。

```
DELETE FROM tbl_name WHERE 1 > 0
```

这迫使 MySQL 进行逐行的删除。相应的查询执行要慢得多，但将返回真正删除的行数。它还将保持当前的 AUTO_INCREMENT 序列的编号，不过只对 MyISAM 表（MySQL 3.23 以上的版本可用）有效。而对于 ISAM 表，序列仍将重置。

■ **避免更新循环不终止**。如果更新一个索引列，如果该列用于 WHERE 子句且更新将索引值移入至今尚未出超的取值范围内时，有可能对所更新的行进行不终止的更新。假如表 my_tbl 有一个索引了的整数列 key_col。下列的查询会产生问题：

```
UPDATE my_tbl SET key_col = key_col+1 WHERE key_col > 0;
```

这个问题的解决方法是在 WHERE 子句中将 key_col 用于一个表达式，使 MySQL 不能使用索引：

```
UPDATE my_tbl SET key_col = key_col+1 WHERE key_col+0 > 0;
```

实际上，还有另外的方法，即升级到 MySQL 3.23.2 或更高的版本，它们已经解决了这样的问题。

■ **以随机次序检索结果**。自 MySQL 3.23.3 以来，可使用 ORDER BY RAND( ) 随机地对结果进行排序。另一技术对 MySQL 更旧的版本很有用处，那就是选择一个随机数列，然后在该列上进行排序。但是，如果按如下编写查询，优化程序将会让您的愿望落空：

```
SELECT ..., RAND() as rand_col FROM ... ORDER BY rand_col
```

这里的问题是 MySQL 认为该列是一个函数调用，将认为相应的列值是一个常数，而对 ORDER BY 子句进行优化，使此查询失效。可在表达式中引用某个表列来蒙骗优化程序。例如，如果表中有一个名为 age 的列，可编写如下查询：

```
SELECT ..., age*0+RAND() as rand_col FROM ... ORDER BY rand_col
```

■ **忽略优化程序的表连接次序**。可利用 STRIGHT_JOIN 强迫优化程序以特定的次序使用表。如果这样做，应该规定表的次序，使第一个表为从中选择的行数最少的表。（如果不能肯定哪个表满足这个要求，可将行数最多的表作为第一个表。）换句话说，应尽量规定表的次序，使最有限制性的选择先出现。排除可能的候选行越早，查询执行得就越快。要保证测试相应的查询两次；可能会有某些原因使优化程序不以您所想像的方式对表进行连接，并且 STRAIGHT_JOIN 也可能实际上不起作用。

4.3 列类型选择与查询效率

要选择有助于使查询执行更快的列，应遵循如下规则（这里，“BLOB 类型”应该理解为即包含 BLOB也包含TEXT 类型）：

■ **使用定长列，不使用可变长列。**这条准则对被经常修改，从而容易产生碎片的表来说特别重要。例如，应该选择 CHAR 列而不选择 VARCHAR 列。所要权衡的是使用定长列时，表所占用的空间更多，但如果能够承担这种空间的耗费，使用定长行将比使用

可变长的行处理快得多。

- **在较短的列能够满足要求时不要使用较长的列。**如果正使用的是定长的 CHAR 列，应该使它们尽量短。如果列中所存储的最长值为 40 个字符，那么就不要将其定义为 CHAR(255)；只要定义为 CHAR(40) 即可。如果能够使用 MEDIUMINT 而不是 BIGINT，表将会更小（磁盘 I/O 也较少），其值在计算中也可以处理得更快。
- **将列定义为** NOT NULL。这样处理更快，所需空间更少。而且有时还能简化查询，因为不需要检查是否存在特例 NULL。
- **考虑使用** ENUM **列**。如果有一个只含有限数目的特定值的列，那么应该考虑将其转换为 ENUM 列。ENUM 列的值可以更快地处理，因为它们在内部是以数值表示的。
- **使用** PROCEDURE ANALYSE( )。如果使用的是 MySQL 3.23 或更新的版本，应该执行 PROCEDURE ANALYSE( )，查看它所提供的关于表中列的信息：

```
SELECT * FROM tbl_name PROCEDURE ANALYSE()
SELECT * FROM tbl_name PROCEDURE ANALYSE(16,256)
```

相应输出中有一列是关于表中每列的最佳列类型的建议。第二个例子要求PROCEDURE ANALYSE( ) 不要建议含有多于 16 个值或取多于 256 字节的 ENUM 类型（可根据需要更改这些值）。如果没有这样的限制，输出可能会很长 ;ENUM 的定义也会很难阅读。

根据 PROCEDURE ANALYSE( ) 的输出，会发现可以对表进行更改以利用更有效的类型。如果希望更改值类型，使用 ALTER TABLE 语句即可。

- **将数据装入** BLOB。用 BLOB 存储应用程序中包装或未包装的数据，有可能使原来需要几个检索操作才能完成的数据检索得以在单个检索操作中完成。而且还对存储标准表结构不易表示的数据或随时间变化的数据有帮助。在第 3 章 ALTER TABLE 语句的介绍中，有一个例子处理存储来自 Web 问卷的结果的表。该例子中讨论了在问卷中增加问题时，怎样利用 ALTER TABLE 向该表追加列。

  解决该问题的另一个方法是让处理 Web 的应用程序将数据包装成某种数据结构，然后将其插入单个 BLOB 列。这样会增加应用程序对数据进行解码的开销（而且从表中检索出记录后要对其进行编码），但是简化了表的结构，并且不用在更改问卷时对表进行更改。

  另一方面，BLOB 值也有自己的固有问题，特别是在进行大量的 DELETE 或 UPDATE 操作时更是如此。删除 BLOB 会在表中留下一个大空白，在以后将需用一个记录或可能是不同大小的多个记录来填充。
- **对容易产生碎片的表使用** OPTIMIZE TABLE。大量进行修改的表，特别是那些含有可变长列的表，容易产生碎片。碎片不好，因为它在存储表的磁盘块中产生不使用的空间。随着时间的增长，必须读取更多的块才能取到有效的行，从而降低了性能。任意具有可变长行的表都存在这个问题，但这个问题对 BLOB 列更为突出，因为它们尺寸的变化非常大。经常使用 OPTIMIZE TABLE 有助于保持性能不下降。
- **使用合成索引**。合成索引列有时很有用。一种技术是根据其他列建立一个散列值，并将其存储在一个独立的列中，然后可通过搜索散列值找到行。这只对精确匹配的查询有效。（散列值对具有诸如“ < ”或“ >= ”这样的操作符的范围搜索没有用处）。在 MySQL 3.23版及以上版本中，散列值可利用 MD5( ) 函数产生。

  散列索引对 BLOB 列特别有用。有一事要注意，在 MySQL 3.23.2 以前的版本中，

不能索引 BLOB 类型。甚至是在 3.23.2 或更新的版本中，利用散列值作为标识值来查找 BLOB 值也比搜索 BLOB 列本身更快。

- **除非有必要，否则应避免检索较大的 BLOB 或 TEXT 值。**例如，除非肯定 WHERE 子句能够将结果恰好限制在所想要的行上，否则 SELECT * 查询不是一个好办法。这样做可能会将非常大的 BLOB 值无目的地从网络上拖过来。这是存储在另一列中的 BLOB 标识信息很有用的另一种情形。可以搜索该列以确定想要的行，然后从限定的行中检索 BLOB 值。
- **将 BLOB 值隔离在一个独立的表中。**在某些情况下，将 BLOB 列从表中移出放入另一个副表可能具有一定的意义，条件是移出 BLOB 列后可将表转换为定长行格式。这样会减少主表中的碎片，而且能利用定长行的性能优势。

4.4 有效地装载数据

很多时候关心的是优化 SELECT 查询，因为它们是最常用的查询，而且确定怎样优化它们并不总是直截了当。相对来说，将数据装入数据库是直截了当的。然而，也存在可用来改善数据装载操作效率的策略，其基本原理如下：

- 成批装载较单行装载更快，因为在装载每个记录后，不需要刷新索引高速缓存；可在成批记录装入后才刷新。
- 在表无索引时装载比索引后装载更快。如果有索引，不仅必须增加记录到数据文件，而且还要修改每个索引以反映增加了的新记录。
- 较短的 SQL 语句比较长的 SQL 语句要快，因为它们涉及服务器方的分析较少，而且还因为将它们通过网络从客户机发送到服务器更快。

这些因素中有一些似乎微不足道（特别是最后一个因素），但如果要装载大量的数据，即使是很小的因素也会产生很大的不同结果。我们可以利用上述的一般原理推导出几个关于如何最快地装载数据的实际结论：

- LOAD DATA（包括其所有形式）比 INSERT 效率高，因为其成批装载行。索引刷新较少，并且服务器只需分析和解释一条语句而不是几条语句。
- LOAD DATA 比 LOAD DATA LOCAL 效率更高。利用 LOAD DATA，文件必须定位在服务器上，而且必须具有 FILE 权限，但服务器可从磁盘直接读取文件。利用 LOAD DATA LOCAL，客户机读取文件并将其通过网络发送给服务器，这样做很慢。
- 如果必须使用 INSERT，应该利用允许在单个语句中指定多行的形式，例如：

```
INSERT INTO tbl_name VALUES(...),(...),...
```

可在语句中指定的行越多越好。这样会减少所需的语句数目，降低索引刷新量。

如果使用 mysqldump 生成数据库备份文件，应该使用--extended-insert 选项，使转储文件包含多行 INSERT 语句。还可以使用 --opt（优化），它启用 --extended-insert 选项。反之，应该避免使用 mysqldump 的 --complete-insert 选项；此选项会导致 INSERT 语句为单行，执行时间更长，比不用 --complete-insert 选项生成的语句需要更多的分析。

- 使用压缩了的客户机 /服务器协议以减少网络数据流量。对于大多数 MySQL 客户机，可以用 --compress 命令行选项来指定。它一般只用于较慢的网络，因为压缩需要占用大量的处理器时间。

- 让 MySQL 插入缺省值；不要在 INSERT 语句中指定将以任意方式赋予缺省值的列。平均来说，这样做语句会更短，能减少通过网络传送给服务器的字符数。此外，语句包含的值较少，服务器所进行的分析和转换就会较少。
- 如果表是索引的，则可利用批量插入（LOAD DATA 或多行的 INSERT 语句）来减少索引的开销。这样会最小化索引更新的影响，因为索引只需要在所有行处理过时才进行刷新，而不是在每行处理后就刷新。
- 如果需要将大量数据装入一个新表，应该创建该表且在未索引时装载，装载数据后才创建索引，这样做较快。一次创建索引（而不是每行修改一次索引）较快。
- 如果在装载之前删除或禁用索引，装入数据后再重新创建或启用索引可能使装载更快。

如果想对数据装载使用删除或禁用策略，一定要做一些实验，看这样做是否值得（如果将少量数据装入一个大表中，重建和索引所花费的时间可能比装载数据的时间还要长）。

可用 DROP INDEX 和 CREATE INDEX 来删除和重建索引。另一种可供选择的方法是利用 myisamchk 或 isamchk 禁用和启用索引。这需要在 MySQL 服务器主机上有一个帐户，并对表文件有写入权。为了禁用表索引，可进入相应的数据库目录，执行下列命令之一：

```
% myisamchk --keys-used=0 tbl_name
% isamchk --keys-used=0 tbl_name
```

对具有 .MYI 扩展名的索引文件的 MyISAM 表使用 myisamchk，对具有 .ISM 扩展名的索引文件的 ISAM 表使用 isamchk。在向表中装入数据后，按如下激活索引：

```
% myisamchk --recover --quick --keys-used=n tbl_name
% isamchk --recover --quick --keys-used=n tbl_name
```

n 为表具有的索引数目。可用 --description 选项调用相应的实用程序得出这个值：

```
% myisamchk --description tbl_name
% isamchk --description tbl_name
```

如果决定使用索引禁用和激活，应该使用第 13章中介绍的表修复锁定协议以阻止服务器同时更改锁（虽然此时不对表进行修复，但要对它像表修复过程一样进行修改，因此需要使用相同的锁定协议）。

上述数据装载原理也适用于与需要执行不同操作的客户机有关的固定查询。例如，一般希望避免在频繁更新的表上长时间运行 SELECT 查询。长时间运行 SELECT 查询会产生大量争用，并降低写入程序的性能。一种可能的解决方法为，如果执行写入的主要是 INSERT 操作，那么先将记录存入一个临时表，然后定期地将这些记录加入主表中。如果需要立即访问新记录，这不是一个可行的方法。但只要能在一个较短的时间内不访问它们，就可以使用这个方法。使用临时表有两个方面的好处。首先，它减少了与主表上 SELECT 查询语句的争用，因此，执行更快。其次，从临时表将记录装入主表的总时间较分别装载记录的总时间少；相应的索引高速缓存只需在每个批量装载结束时进行刷新，而不是在每行装载后刷新。

这个策略的一个应用是进入 Web 服务器的Web 页访问 MySQL 数据库。在此情形下，可能没有保证记录立即进入主表的较高权限。

如果数据并不完全是那种在系统非正常关闭事件中插入的单个记录，那么减少索引刷新的另一策略是使用 MyISAM 表的 DELAYED_KEY_WRITE 表创建选项（如果将 MySQL 用于某些数据录入工作时可能会出现这种情况）。此选项使索引高速缓存只偶尔刷新，而不是在每次插入后都要刷新。

如果希望在服务器范围内利用延迟索引刷新，只要利用 --delayed-key-write 选项启动 mysqld 即可。在此情形下，索引块写操作延迟到必须刷新块以便为其他索引值腾出空间为止，或延迟到执行了一个 flush-tables 命令后，或延迟到该索引表关闭。

4.5 调度与锁定问题

前面各段主要将精力集中在使个别的查询更快上。MySQL 还允许影响语句的调度特性，这样会使来自几个客户机的查询更好地协作，从而单个客户机不会被锁定太长的时间。更改调度特性还能保证特定的查询处理得更快。我们先来看一下 MySQL 的缺省调度策略，然后来看看为改变这个策略可使用什么样的选项。出于讨论的目的，假设执行检索（SELECT）的客户机程序为读取程序。执行修改表操作（DELETE，INSERT，REPLACE 或 UPDATE）的另一个客户机程序为写入程序。

MySQL 的基本调度策略可总结如下：

- 写入请求应按其到达的次序进行处理。
- 写入具有比读取更高的优先权。

在表锁的帮助下实现调度策略。客户机程序无论何时要访问表，都必须首先获得该表的锁。可以直接用 LOCK TABLES 来完成这项工作，但一般服务器的锁管理器会在需要时自动获得锁。在客户机结束对表的处理时，可释放表上的锁。直接获得的锁可用 UNLOCK TABLES 释放，但服务器也会自动释放它所获得的锁。

执行写操作的客户机必须对表具有独占访问的锁。在写操作进行中，由于正在对表进行数据记录的删除、增加或更改，所以该表处于不一致状态，而且该表上的索引也可能需要作相应的更新。如果表处于不断变化中，此时允许其他客户机访问该表会出问题。让两个客户机同时写同一个表显然不好，因为这样会很快使该表不可用。允许客户机读不断变化的表也不是件好事，因为可能在读该表的那一刻正好正在对它进行更改，其结果是不正确的。

执行读取操作的客户机必须有一把防止其他客户机写该表的锁，以保证读表的过程中表不出现变化。不过，该锁无需对读取操作提供独占访问。此锁还允许其他客户机同时对表进行读取。读取不会更改表，所有没必要阻止其它客户机对该表进行读取。

MySQL 允许借助几个查询限修饰符对其调度策略施加影响。其中之一是 DELETE、INSERT、LOAD DATA、REPLACE 和 UPDATE 语句的 LOW_PRIORITY 关键字。另一个是 SELECT 语句的 HIGH_PRIORITY 关键字。第三个是 INSERT 和 REPLACE 语句的 DELAYED 关键字。

LOW_PRIORITY 关键字按如下影响调度。一般情况下，如果某个表的写入操作在表正被读取时到达，写入程序被阻塞，直到读取程序完成，因为一旦某个查询开始，就不能中断。如果另一读取请求在写入程序等待时到达，此读取程序也被阻塞，因为缺省的调度策略为写入程序具有比读取程序高的优先级。在第一个读取程序结束时，写入程序继续，在此写入程序结束时，第二个读取程序开始。

如果写入请求为 LOW_PRIORITY 的请求，则不将该写入操作视为具有比读取操作优先级高的操作。在此情形下，如果第二个读取请求在写入程序等待时到达，则让第二个读取操作排在等待的写入操作之前。仅当没有其他读取请求时，才允许写入程序执行。这种调度的更改从理论上说，其含义为 LOW_PRIORITY 写入可能会永远被阻塞。当正在处理前面的读

取请求时，只要另一个读取请求到达，这个新的请求允许排在 LOW_PRIORITY 写入之前。

SELECT 查询的 HIGH_PRIORITY 关键字作用类似。它使 SELECT 插在正在等待的写入操作之前，即使该写入操作具有正常的优先级。

INSERT 的 DELAYED 修饰符作用如下，在表的一个 INSERT DELAYED 请求到达时，服务器将相应的行放入一个队列，并立即返回一个状态到客户机程序，以便该客户机程序可以继续执行，即使这些行尚未插入表中。如果读取程序正在对表进行读取，那么队列中的行挂起。在没有读取时，服务器开始开始插入延迟行队列中的行。服务器不时地停下来看看是否有新的读取请求到达，并进行等待。如果是这样，延迟行队列将挂起，并允许读取程序继续。在没有其他的读取操作时，服务器再次开始插入延迟行。这个过程一直进行到延迟行队列空为止。

此调度修饰符并非出现在所有 MySQL 版本中。下面的表列出了这些修饰符和支持这些修饰符的 MySQL 版本。可利用此表来判断所使用的 MySQL 版本具有什么样的功能：

语句类型	开始出现的版本
DELETE LOW_PRIORITY	3.22.5
INSERT LOW+PRIORITY	3.22.5
INSERT DELAYED	3.22.15
LOAD DATA LOW_PRIORITY	3.23.0
LOCK TABLES ... LOW_PRIORITY	3.22.8
REPLACE LOW_PRIORITY	3.22.5
REPLACE DELAYED	3.22.15
SELECT ... HIGH_PRIORITY	3.22.9
UPDATE LOW_PRIORITY	3.22.5
SET SQL_LOW_PRIORITY_UPDATES	3.22.5

INSERT DELAYED 在客户机方的作用

如果其他客户机可能执行冗长的 SELECT 语句，而且您不希望等待插入完成，此时 INSERT DELAYED 很有用。发布 INSERT DELAYED 的客户机可以更快地继续执行，因为服务器只是简单地将要插入的行插入。

不过应该对正常的 INSERT 和 INSERT DELAYED 性能之间的差异有所认识。如果 INSERT DELAYED 存在语法错误，则向客户机发出一个错误，如果正常，便不发出信息。例如，在此语句返回时，不能相信所取得的 AUTO_INCREMENT 值。也得不到惟一索引上的重复数目的计数。之所以这样是因为此插入操作在实际的插入完成前返回了一个状态。其他还表示，如果 INSERT DELAYED 语句的行在等待插入中被排队，并且服务器崩溃或被终止（用 kill -9），那么这些行将丢失。正常的 TERM 终止不会这样，服务器会在退出前将这些行插入。

4.6 管理员的优化

前面各段介绍了普通的 MySQL 用户利用表创建和索引操作，以及利用查询的编写能够进行的优化。不过，还有一些只能由 MySQL 管理员和系统管理员来完成的优化，这些管理员在 MySQL 服务器或运行 MySQL 的机器上具有控制权。有的服务器参数直接适用于查询处理，可将它们打开。而有的硬件配置问题直接影响查询处理速度，应该对它们进行调整。

4.6.1 服务器参数

服务器有几个能够改变从而影响其操作的参数（或称变量）。有关服务器参数优化的综合介绍请参见第11章，但其中几个参数主要与查询有关，有必要在此提一下：

- delayed_queue_size

  此参数在执行其他 INSERT DELAYED 语句的客户机阻塞以前，确定来自 INSERT DELAYED 语句的放入队列的行的数目。增加这个参数的值使服务器能从这种请求中接收更多的行，因而客户机可以继续执行而不阻塞。

- key_buffer_size

  此参数为用来存放索引块的缓冲区尺寸。如果内存多，增加这个值能节省索引创建和修改的时间。较大的值使 MySQL 能在内存中存储更多的索引块，这样增加了在内存中找到键值而不用读磁盘块的可能性。

在 MySQL 3.23 版及以后的版本中，如果增加了键缓冲区的尺寸，可能还希望用 --init-file 选项启动服务器。这样能够指定一个服务器启动时执行的 SQL 语句文件。如果有想要存放在内存中的只读表，可将它们拷贝到索引查找非常快的 HEAP 表。

4.6.2 硬件问题

可利用硬件更有效地改善服务器的性能：

- **在机器中安装更多的内存。**这样能够增加服务器的高速缓存和缓冲区的尺寸，使服务器更经常地使用存放在内存中的信息，降低从磁盘取信息的要求。
- **如果有足够的 RAM 使所有交换在内存文件系统中完成，那么应该重新配置系统，去掉所有磁盘交换设置。**否则，即使有足以满足交换的 RAM，某些系统仍然要与磁盘进行交换。
- **增加更快的磁盘以减少 I/O 等待时间。**寻道时间是这里决定性能的主要因素。逐字地移动磁头是很慢的，一旦磁头定位，从磁道读块则较快。
- **在不同的物理设备上设法重新分配磁盘活动。**如果可能，应将您的两个最繁忙的数据库存放在不同的物理设备上。请注意，使用同一物理设备上的不同分区是不够的。这样没有帮助，因为它们仍将争用相同的物理资源（磁盘头）。移动数据库的过程在第 10 章中介绍。

  在将数据重新放到不同设备之前，应该保证了解该系统的装载特性。如果在特定的物理设备上已经有了某些特定的主要活动，将数据库放到该处实际上可能会使性能更坏。例如，不要把数据库移到处理大量 Web 通信的Web 服务器设备上。
- **在设置 MySQL 时，应该配置其使用静态库而不是共享库。**使用共享库的动态二进制系统可节省磁盘空间，但静态二进制系统更快（然而，如果希望装入用户自定义的函数，则不能使用静态二进制系统，因为 UDF 机制依赖于动态连接）。