第 17 章

MySQL 的性能优化

本章内容提要

MySQL 性能优化是通过合理安排资源、调整系统参数等方法提高 MySQL 数据库的性能。性能优化的目的是为了使 MySQL 数据库运行速度更快、占用的磁盘空间更小。本章就来介绍 MySQL 的性能优化，主要内容包括查询速度的优化、数据库结构的优化、MySQL 服务器的优化等。

本章知识点

- 性能优化。
- 查询速度的优化。
- 数据库结构的优化。
- MySQL 服务器的优化。

17.1 认识 MySQL 性能优化

掌握优化 MySQL 数据库的方法是数据库管理员和数据库开发人员的必备技能。通过不同的优化方法以达到提高 MySQL 数据库性能的目的。MySQL 数据库优化是多方面的，原则是减少系统的瓶颈，减少资源的占用，增加系统的反应速度。

在 MySQL 中，可以使用 SHOW STATUS 语句查询 MySQL 数据库的性能参数，其语法如下：

```
SHOW STATUS LIKE 'value';
```

其中，value 是要查询的参数值，一些常用的性能参数如表 17-1 所示。

表 17-1 value 常用的参数值

参 数 名	功 能 简 介
Connections	连接 MySQL 服务器的次数
Uptime	MySQL 服务器的上线时间
Slow_queries	慢查询的次数
Com_select	查询操作的次数
Com_insert	插入操作的次数

续表

参 数 名	功 能 简 介
Com_update	更新操作的次数
Com_delete	删除操作的次数

【实例 1】查询 MySQL 服务器的连接次数，可以执行如下语句：

```
SHOW STATUS LIKE 'Connections';
```

按 Enter 键，即可返回查询结果，如图 17-1 所示。从结果可以得出当前 MySQL 服务器的连接次数为“8”。

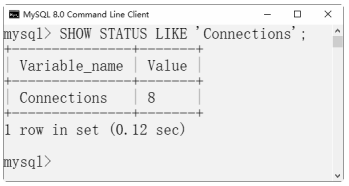

图 17-1 返回查询结果

【实例 2】查询 MySQL 服务器的慢查询次数，可以执行如下语句：

```
SHOW STATUS LIKE 'Slow_queries';
```

按 Enter 键，即可返回查询结果，如图 17-2 所示。从结果可以得出当前慢查询次数为“0”。

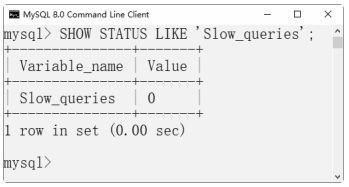

图 17-2 返回查询结果

提示：查询其他参数的方法和两个参数的查询方法相同。而且慢查询次数参数可以结合慢查询日志，找出慢查询语句，然后针对慢查询语句进行表结构优化或者查询语句优化。

17.2 查询速度的优化

在 MySQL 数据库中，对数据的查询操作是数据库中最频繁的操作，提高数据的查询速度可以有效地提高 MySQL 数据库的性能。

17.2.1 分析查询语句

通过分析查询语句，可以了解查询语句的执行情况，找出查询语句的不足之处，从而优化查询语句。在 MySQL 中，可以使用 EXPLAIN 和 DESCRIBE 语句来分析查询语句。

1. 使用 EXPLAIN 语句分析查询语句

EXPLAIN 语句的基本语法格式如下：

```
EXPLAIN [EXTENDED] SELECT select_options
```

主要参数介绍如下。

- EXTENDED 是关键字，EXPLAIN 语句将产生附加信息。

- select_options 参数是 SELECT 语句的查询选项，包括 FROM WHERE 子句等。

执行该语句，可以分析 EXPLAIN 后面的 SELECT 语句的执行情况，并且能够分析出所查询的表的一些特征。

【实例 3】使用 EXPLAIN 语句分析查询水果表 fruits 的语句，执行如下语句：

```
EXPLAIN SELECT * FROM fruits;
```

按 Enter 键，即可返回分析结果，如图 17-3 所示。

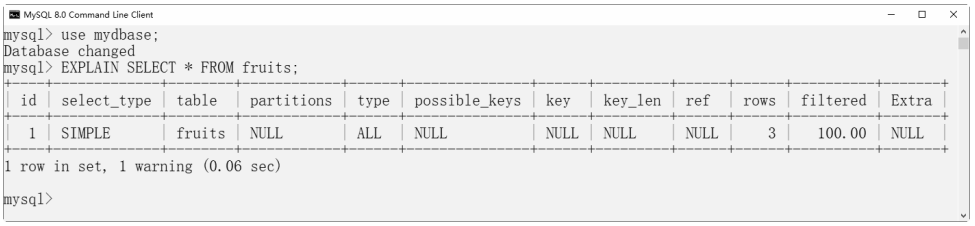

图 17-3　使用 EXPLAIN 语句分析

下面对查询结果中主要参数功能进行介绍。

（1）id：SELECT 识别符，这是 SELECT 的查询序列号。

（2）select_type：表示 SELECT 语句的类型。其主要取值如表 17-2 所示。

表 17-2　select_type 的取值

取　值	取 值 介 绍
SIMPLE	表示简单查询，其中不包括连接查询和子查询
PRIMARY	表示主查询，或者是最外层的查询语句
UNION	表示连接查询的第 2 个或后面的查询语句
DEPENDENT UNION	连接查询中的第 2 个或后面的 SELECT 语句，取决于外面的查询
UNION RESULT	连接查询的结果
SUBQUERY	子查询中的第 1 个 SELECT 语句
DEPENDENT SUBQUERY	子查询中的第 1 个 SELECT 语句，取决于外面的查询
DERIVED	导出表的 SELECT 语句（FROM 子句的子查询）

（3）table：表示查询的表。

（4）type：表示表的连接类型，下面按照从最佳类型到最差类型的顺序给出各种连接类型，如表 17-3 所示。

（5）possible_keys：指出 MySQL 能使用哪个索引在该表中找到行。如果该列是 NULL，则没有相关的索引。在这种情况下，可以通过检查 WHERE 子句看它是否引用某些列或适合索引的列来提高查询性能。如果是这样，可以创建适合的索引来提高查询的性能。

（6）key：表示查询实际使用到的索引，如果没有选择索引，该列的值是 NULL。要想强制 MySQL 使用或忽视 possible_keys 列中的索引，在查询中使用 FORCE INDEX、USE INDEX 或者 IGNORE INDEX。参见 SELECT 语法。

（7）key_len：表示 MySQL 选择的索引字段按字节计算的长度，如果键是 NULL，则长度为 NULL。注意通过 key_len 值可以确定 MySQL 将实际使用一个多列索引中的几个字段。

（8）ref：表示使用哪个列或常数与索引一起来查询记录。

（9）rows：显示 MySQL 在表中进行查询时必须检查的行数。

（10）Extra：表示 MySQL 在处理查询时的详细信息。

表 17-3 表的连接类型

连 接 类 型	功 能 介 绍
system	该表是仅有一行的系统表。这是 const 连接类型的一个特例
const	数据表最多只有一个匹配行，它将在查询开始时被读取，并在余下的查询优化中作为常量对待。const 表查询速度很快，因为它们只读取一次。const 用于使用常数值比较 PRIMARY KEY 或 UNIQUE 索引的所有部分的场合
eq_ref	对于每个来自前面的表的行组合，从该表中读取一行。当一个索引的所有部分都在查询中使用并且索引是 UNIQUE 或 PRIMARY KEY 时，即可使用这种类型。eq_ref 可以用于使用“=”操作符比较带索引的列。比较值可以为常量或一个在该表前面所读取的表的列的表达式
ref	对于来自前面的表的任意行组合，将从该表中读取所有匹配的行。这种类型用于索引既不是 UNIQUE 也不是 PRIMARY KEY 的情况，或者查询中使用了索引列的左子集，即索引中左边的部分列组合。ref 可以用于使用=或<=>操作符的带索引的列
ref_or_null	该连接类型如同 ref，但是添加了 MySQL 可以专门搜索包含 NULL 值的行。在解决子查询中经常使用该连接类型的优化
index_merge	该连接类型表示使用了索引合并优化方法。在这种情况下，key 列包含了使用的索引的清单，key_len 包含了使用的索引的最长的关键元素
unique_subquery	该类型替换了下面形式的 IN 子查询： value IN (SELECT primary_key FROM single_table WHERE some_expr) unique_subquery 是一个索引查找函数，可以完全替换子查询，效率更高
index_subquery	该连接类型类似于 unique_subquery，可以替换 IN 子查询，但只适合下列形式的子查询中的非唯一索引：value IN (SELECT key_column FROM single_table WHERE some_expr)
range	只检索给定范围的行，使用一个索引来选择行。当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符，用常量比较关键字列时，类型为 range
index	该连接类型与 ALL 基本相同，不过 index 连接只扫描索引树，这通常比 ALL 快一些，因为索引文件通常比数据文件小
ALL	对于前面的表的任意行组合，进行完整的表扫描

2. 使用 DESCRIBE 语句分析查询语句

DESCRIBE 语句的使用方法与 EXPLAIN 语句是一样的，其语法形式如下：

```
DESCRIBE SELECT select_options
```

其中，DESCRIBE 可以缩写成 DESC。

【实例 4】使用 DESCRIBE 语句分析查询水果表 fruits 的语句，执行如下语句：

```
DESCRIBE SELECT * FROM fruits;
```

按 Enter 键，即可返回分析结果，如图 17-4 所示。从结果可以得出这两种方法的分析结果是一样的。

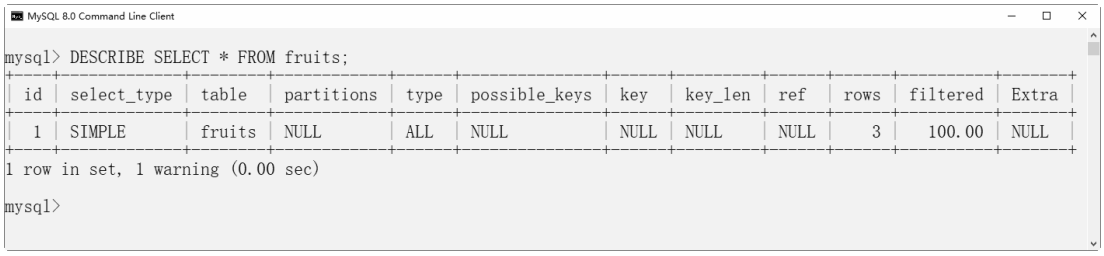

图 17-4 使用 DESCRIBE 语句分析

17.2.2　使用索引优化查询

索引可以快速地定位表中的某条记录，使用索引可以提高数据库的查询速度，从而提高数据库的性能。在数据量大的情况下，如果不使用索引，查询语句将扫描表中的所有记录，这样查询的速度会很慢。如果使用索引，查询语句可以根据索引快速定位到待查询记录，从而减少查询的记录数，达到提高查询速度的目的。

【实例 5】下面是查询语句中不使用索引和使用索引的对比。首先，分析未使用索引时的查询情况，EXPLAIN 语句执行如下：

```
EXPLAIN SELECT * FROM fruits WHERE fruits_name='苹果';
```

按 Enter 键，即可返回查询结果，如图 17-5 所示。可以看到，rows 列的值是 3，说明“SELECT * FROM fruits WHERE fruits_name='苹果';”这个查询语句扫描了表中的 3 条记录。

图 17-5　返回查询结果

下面在 fruits 表的 fruits_name 字段上加上索引，添加索引的语句如下：

```
CREATE INDEX index_name ON fruits(fruits_name);
```

按 Enter 键，即可完成添加索引的操作，如图 17-6 所示。

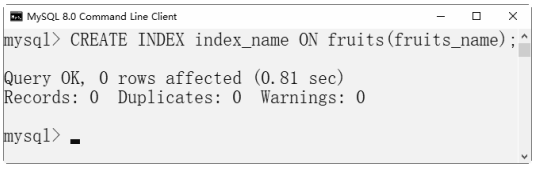

图 17-6　添加索引

现在，再分析上面的查询语句。执行的 EXPLAIN 语句如下：

```
EXPLAIN SELECT * FROM fruits WHERE fruits_name='苹果';
```

按 Enter 键，即可返回查询结果，如图 17-7 所示。从结果可以看出 rows 列的值为 1，这表示这个查询语句只扫描了表中的一条记录，其查询速度自然比扫描 3 条记录快。而且 possible_keys 和 key 的值都是 index_name，这说明查询时使用了 index_name 索引。

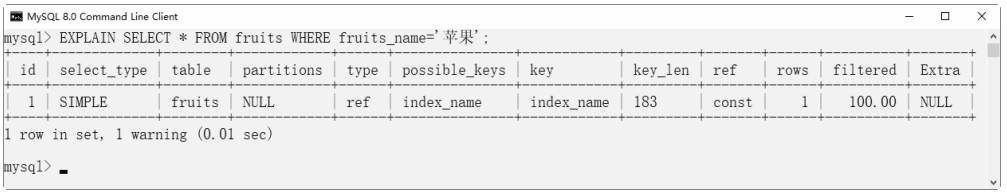

图 17-7　返回查询结果

17.2.3　使用索引查询的缺陷

索引可以提高查询的速度，但并不是使用带有索引的字段查询时，索引都会起作用。下面

介绍使用索引的几种特殊情况，在这些情况下，有可能使用带有索引的字段查询时，索引并没有起作用。

1. 使用 LIKE 关键字的查询语句

在使用 LIKE 关键字进行查询的查询语句中，如果匹配字符串的第一个字符为“%”，索引不会起作用。只有“%”不在第一个位置，索引才会起作用。下面将举例说明。

【实例 6】查询语句中使用 LIKE 关键字，并且匹配的字符串中含有“%”字符，EXPLAIN 语句执行如下：

```
mysql> EXPLAIN SELECT * FROM fruits WHERE f_name like '%x';
+----+-------------+---------+-------+-----------------+---------------+----------+--
-----+------+---------------+
| id | select_type | table   | type  | possible_keys | key      | key_len | ref   | rows |
Extra     |
+----+-------------+---------+-------+-----------------+---------------+----------+--
------+------+---------------+
|  1 | SIMPLE  | fruits  | ALL  | NULL      | NULL    | NULL  | NULL| 16  | Using where |
+----+-------------+---------+-------+-----------------+---------------+----------+--
------+------+---------------+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM fruits WHERE f_name like 'x%';
+----+-------------+---------+------+---------------+--------------+---------+-------
-+-------+-----------------+
| id | select_type | table   | type | possible_keys | key     | key_len | ref   | rows |
Extra      |
+----+-------------+---------+------+---------------+--------------+---------+-------
-+-------+-----------------+
|  1 | SIMPLE  | fruits  | range | index_name | index_name | 150  | NULL |  4  | Using where
|
+----+-------------+---------+------+---------------+--------------+---------+-------
-+-------+-----------------+
1 row in set (0.00 sec)
```

已知 f_name 字段上有索引 index_name。第 1 个查询语句执行后，rows 列的值为 16，表示这次查询过程中扫描了表中所有的 16 条记录；第 2 个查询语句执行后，rows 列的值为 4，表示这次查询过程扫描了 4 条记录。第 1 个查询语句索引没有起作用，因为第 1 个查询语句的 LIKE 关键字后的字符串以“%”开头，而第 2 个查询语句使用了索引 index_name。

2. 使用多列索引的查询语句

MySQL 可以为多个字段创建索引。一个索引可以包括 16 个字段。对于多列索引，只有查询条件中使用了这些字段中第 1 个字段时，索引才会被使用。

【实例 7】本例在表 fruits 中 f_id、f_price 字段创建多列索引，验证多列索引的使用情况。

```
mysql> CREATE INDEX index_id_price ON fruits(f_id, f_price);
Query OK, 0 rows affected (0.39 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> EXPLAIN SELECT * FROM fruits WHERE f_id='l2';
+----+-------------+--------+-------+-------------------------------+-------------+--
-------+-------+------+-------+
| id  | select_type | table  | type  | possible_keys        | key     | key_len | ref  | rows
| Extra |
+----+-------------+--------+-------+-------------------------------+-------------+-
--------+-------+------+-------+
|  1 | SIMPLE   | fruits | const | PRIMARY,index_id_price | PRIMARY | 20    | const |  1
|    |
+----+-------------+--------+-------+-------------------------------+-------------+-
```

```
--------+-------+------+--------+
    1 row in set (0.00 sec)

    mysql> EXPLAIN SELECT * FROM fruits WHERE f_price=5.2;
    +----+------------+--------+------+----------------+--------+---------+------+------+
---------------+
    | id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra
|
    +----+------------+--------+------+----------------+--------+---------+------+------+
---------------+
    |  1 | SIMPLE  | fruits | ALL  | NULL     | NULL | NULL | NULL |  16 | Using where|
    +----+-------------+--------+------+----------------+--------+---------+-------+------
+---------------+
    1 row in set (0.00 sec)
```

从第 1 条语句查询结果可以看出，“f_id= 'l2'”的记录有 1 条。第 1 条语句共扫描了 1 条记录，并且使用了索引 index_id_price。从第 2 条语句查询结果可以看出，rows 列的值是 16，说明查询语句共扫描了 16 条记录，并且 key 列值为 NULL，说明“SELECT * FROM fruits WHERE f_price=5.2;”语句并没有使用索引。因为 f_price 字段是多列索引的第 2 个字段，只有查询条件中使用了 f_id 字段才会使 index_id_price 索引起作用。

3. 使用 OR 关键字的查询语句

查询语句的查询条件中只有 OR 关键字，且 OR 前后的两个条件中的列都是索引时，查询中才使用索引。否则，查询将不使用索引。

【实例 8】查询语句使用 OR 关键字的情况。

```
    mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple' or s_id=101 \G
    *** 1. row ***
               id: 1
      select_type: SIMPLE
            table: fruits
             type: ALL
    possible_keys: index_name
              key: NULL
          key_len: NULL
              ref: NULL
             rows: 16
            Extra: Using where
    1 row in set (0.00 sec)

    mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple' or f_id='12' \G
    *** 1. row ***
               id: 1
      select_type: SIMPLE
            table: fruits
             type: index_merge
    possible_keys: PRIMARY,index_name,index_id_price
              key: index_name,PRIMARY
          key_len: 510,20
              ref: NULL
             rows: 2
            Extra: Using union(index_name,PRIMARY); Using where
    1 row in set (0.00 sec)
```

因为 s_id 字段上没有索引，第 1 条查询语句没有使用索引，总共查询了 16 条记录；第 2 条查询语句使用了 f_name 和 f_id 这两个索引，因为 id 字段和 name 字段上都有索引，查询的记录数为 2 条。

17.2.4 优化子查询

使用子查询可以进行 SELECT 语句的嵌套查询，即一个 SELECT 查询的结果作为另一个 SELECT 语句的条件。子查询可以一次性完成很多逻辑上需要多个步骤才能完成的 SQL 操作。子查询虽然可以使查询语句很灵活，但执行效率不高。执行子查询时，MySQL 需要为内层查询语句的查询结果建立一个临时表。然后外层查询语句从临时表中查询记录。查询完毕后，再撤销这些临时表。因此，子查询的速度会受到一定的影响。如果查询的数据量比较大，这种影响就会随之增大。

在 MySQL 中，可以使用连接（JOIN）查询来替代子查询。连接查询不需要建立临时表，其速度比子查询要快，如果查询中使用索引的话，性能会更好。连接查询之所以更有效率，是因为 MySQL 不需要在内存中创建临时表来完成查询工作。

17.3 数据库结构的优化

合理的数据库结构不仅可以使数据库占用更小的磁盘空间，而且能够使查询速度更快。数据库结构的设计，需要考虑数据冗余、查询和更新的速度、字段的数据类型是否合理等多方面的内容。

17.3.1 通过分解表来优化

对于字段较多的表，如果有些字段的使用频率很低，可以将这些字段分离出来形成新表。因为当一个表的数据量很大时，会由于使用频率低的字段的存在而变慢。

【实例 9】假设会员表存储会员登录认证信息，该表中有很多字段，如 id、姓名、密码、地址、电话、个人描述字段。其中地址、电话、个人描述等字段并不常用。可以将这些不常用字段分解出另外一个表。将这个表取名叫 members_detail。表中有 member_id、address、telephone、description 等字段。其中，member_id 是会员编号，address 字段存储地址信息，telephone 字段存储电话信息，description 字段存储会员个人描述信息。这样就把会员表分成两个表，分别为 members 表和 members_detail 表。

创建 members 表 SQL 语句如下：

```
CREATE TABLE members (
  Id int(11) NOT NULL AUTO_INCREMENT,
  username varchar(255) DEFAULT NULL ,
  password varchar(255) DEFAULT NULL ,
  last_login_time datetime DEFAULT NULL ,
  last_login_ip varchar(255) DEFAULT NULL ,
  PRIMARY KEY (Id)
) ;
```

创建 members_detail 表 SQL 语句如下：

```
CREATE TABLE members_detail (
  member_id int(11) NOT NULL DEFAULT 0,
  address varchar(255) DEFAULT NULL ,
  telephone varchar(16) DEFAULT NULL ,
  description text
);
```

下面查询 members 表结构，SQL 语句如下：

```
mysql>desc members;
```

按 Enter 键，即可返回 members 表的结构，如图 17-8 所示。

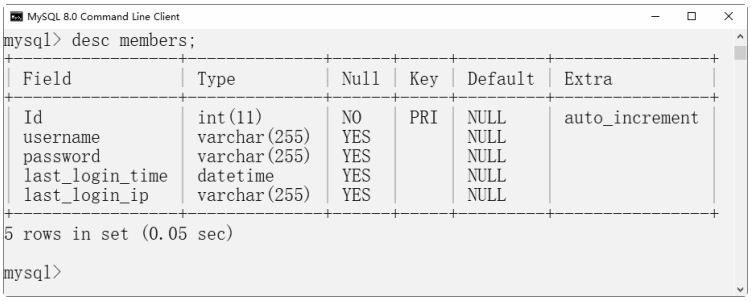

图 17-8　members 表结构

下面查询 members_detail 表结构，SQL 语句如下：

```
mysql> DESC members_detail;
```

按 Enter 键，即可返回 members_detail 表的结构，如图 17-9 所示。

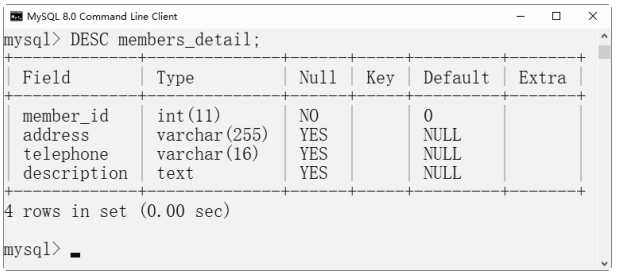

图 17-9　members_detail 表结构

如果需要查询会员的详细信息，可以用会员的 id 来查询。如果需要将会员的基本信息和详细信息同时显示，可以将 members 表和 members_detail 表进行联合查询，查询语句如下：

```
    SELECT * FROM members LEFT JOIN members_detail ON members.id=members_detail.
member_id;
```

通过这种分解，可以提高表的查询效率，对于字段很多且有些字段使用不频繁的表，可以通过这种分解的方式来优化数据库的性能。

17.3.2　通过中间表来优化

对于需要经常联合查询的表，可以建立中间表以提高查询效率。通过建立中间表，把需要经常联合查询的数据插入到中间表中，然后将原来的联合查询改为对中间表的查询，以此来提高查询效率。

【实例 10】会员信息表和会员组信息表的 SQL 语句如下：

```
    CREATE TABLE vip(
      Id int(11) NOT NULL AUTO_INCREMENT,
      username varchar(255) DEFAULT NULL,
      password varchar(255) DEFAULT NULL,
      groupId INT(11) DEFAULT 0,
      PRIMARY KEY (Id)
    ) ;
    CREATE TABLE vip_group (
      Id int(11) NOT NULL AUTO_INCREMENT,
      name varchar(255) DEFAULT NULL,
```

```
  remark varchar(255) DEFAULT NULL,
  PRIMARY KEY (Id)
) ;
```

查询会员信息表和会员组信息表。

```
mysql> DESC vip;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| Id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| username | varchar(255) | YES  |     | NULL    |                |
| password | varchar(255) | YES  |     | NULL    |                |
| groupId  | int(11)      | YES  |     | 0       |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> DESC vip_group;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| Id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255) | YES  |     | NULL    |                |
| remark | varchar(255) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

已知现在有一个模块需要经常查询带有会员组名称、会员组备注（remark）、会员用户名信息的会员信息。根据这种情况可以创建一个 temp_vip 表。temp_vip 表中存储用户名（user_name），会员组名称（group_name）和会员组备注（group_remark）信息。创建表的语句如下：

```
CREATE TABLE temp_vip (
  Id int(11) NOT NULL AUTO_INCREMENT,
  user_name varchar(255) DEFAULT NULL,
  group_name varchar(255) DEFAULT NULL,
  group_remark varchar(255) DEFAULT NULL,
  PRIMARY KEY (Id)
);
```

接下来，从会员信息表和会员组表中查询相关信息存储到临时表中：

```
mysql> INSERT INTO temp_vip(user_name, group_name, group_remark)
      SELECT v.username,g.name,g.remark
      FROM vip as v ,vip_group as g
      WHERE v.groupId =g.Id;
Query OK, 0 rows affected (0.95 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

以后，可以直接从 temp_vip 表中查询会员名、会员组名称和会员组备注，而不用每次都进行联合查询。这样可以提高数据库的查询速度。

17.3.3 通过冗余字段优化

设计数据库表时应尽量遵循范式理论的规约，尽可能减少冗余字段，让数据库设计看起来精致、优雅。但是，合理地加入冗余字段可以提高查询速度。

表的规范化程度越高，表与表之间的关系就越多，需要连接查询的情况也就越多。例如，员工的信息存储在 staff 表中，部门信息存储在 department 表中。通过 staff 表中的 department_id 字段与 department 表建立关联关系。如果要查询一个员工所在部门的名称，必须从 staff 表中查找员工所在部门的编号（department_id），然后根据这个编号去 department 表查找部门的名称。

如果经常需要进行这个操作，连接查询会浪费很多时间。可以在 staff 表中增加一个冗余字

段 department_name，该字段用来存储员工所在部门的名称，这样就不用每次都进行连接操作了。

不过，冗余字段会导致一些问题。比如，冗余字段的值在一个表中被修改了，就要想办法在其他表中更新该字段。否则就会使原本一致的数据变得不一致。

提示：分解表、中间表和增加冗余字段都浪费了一定的磁盘空间。从数据库性能来看，为了提高查询速度而增加少量的冗余大部分时候是可以接受的，而是否通过增加冗余来提高数据库性能，这要根据实际需求综合分析。

17.3.4　优化插入记录的速度

插入记录时，影响插入速度的主要是索引、唯一性校验、一次插入记录条数等，根据这些情况，可以分别进行优化。

1. 对于 MyISAM 引擎的表的优化

对于 MyISAM 引擎的表，常见的优化方法如下：

（1）禁用索引。

对于非空表，插入记录时，MySQL 会根据表的索引对插入的记录建立索引。如果插入大量数据，建立索引会降低插入记录的速度。为了解决这种情况，可以在插入记录之前禁用索引，数据插入完毕后再开启索引。禁用索引的语句如下：

```
ALTER TABLE table_name DISABLE KEYS;
```

其中 table_name 是禁用索引的表的表名。

重新开启索引的语句如下：

```
ALTER TABLE table_name ENABLE KEYS;
```

对于空表批量导入数据，则不需要进行此操作，因为 MyISAM 引擎的表是在导入数据之后才建立索引的。

（2）禁用唯一性检查。

插入数据时，MySQL 会对插入的记录进行唯一性校验。这种唯一性校验也会降低插入记录的速度。为了降低这种情况对查询速度的影响，可以在插入记录之前禁用唯一性检查，等到记录插入完毕后再开启。禁用唯一性检查的语句如下：

```
SET UNIQUE_CHECKS=0;
```

开启唯一性检查的语句如下：

```
SET UNIQUE_CHECKS=1;
```

（3）使用批量插入。

插入多条记录时，可以使用一条 INSERT 语句插入一条记录；也可以使用一条 INSERT 语句插入多条记录。插入一条记录的 INSERT 语句情形如下：

```
INSERT INTO score VALUES('A1','101','MySQL','89');
INSERT INTO score VALUES('A2','102','SQL Server','57')
INSERT INTO score VALUES('A3','103','Access','90')
```

使用一条 INSERT 语句插入多条记录的情形如下：

```
INSERT INTO score VALUES
('A1','101','MySQL','89'),
('A2','102','SQL Server','57'),
('A3','103','Access','90');
```

第 2 种情形的插入速度要比第 1 种情形快。

（4）使用 LOAD DATA INFILE 批量导入。

当需要批量导入数据时，如果能用 LOAD DATA INFILE 语句，就尽量使用。因为 LOAD DATA INFILE 语句导入数据的速度比 INSERT 语句快。

2. 对于 InnoDB 引擎的表的优化

对于 InnoDB 引擎的表，常见的优化方法如下：

（1）禁用唯一性检查。

插入数据之前执行 set unique_checks=0 来禁止对唯一索引的检查，数据导入完成之后再运行 set unique_checks=1。这个和 MyISAM 引擎的使用方法一样。

（2）禁用外键检查。

插入数据之前执行禁止对外键的检查，数据插入完成之后再恢复对外键的检查。禁用外键检查的语句如下：

```
SET foreign_key_checks=0;
```

恢复对外键的检查语句如下：

```
SET foreign_key_checks=1;
```

（3）禁止自动提交。

插入数据之前禁止事务的自动提交，数据导入完成之后，执行恢复自动提交操作。禁止自动提交的语句如下：

```
set autocommit=0;
```

恢复自动提交的语句如下：

```
set autocommit=1;
```

17.3.5 分析表、检查表和优化表

MySQL 提供了分析表、检查表和优化表的语句。分析表主要是分析关键字的分布；检查表主要是检查表是否存在错误；优化表主要是消除删除或者更新造成的空间浪费。

1. 分析表

MySQL 中提供了 ANALYZE TABLE 语句分析表，ANALYZE TABLE 语句的基本语法格式如下：

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name]…
```

主要参数介绍如下。

- LOCAL 关键字：是 NO_WRITE_TO_BINLOG 关键字的别名，二者都是执行过程不写入二进制日志。
- tbl_name：为分析的表的表名，可以有一个或多个。

使用 ANALYZE TABLE 分析表的过程中，数据库系统会自动对表加一个只读锁。在分析期间，只能读取表中的记录，不能更新和插入记录。ANALYZE TABLE 语句能够分析 InnoDB、BDB 和 MyISAM 类型的表。

【实例 11】使用 ANALYZE TABLE 来分析 fruits 表，SQL 语句如下：

```
ANALYZE TABLE fruits;
```

按 Enter 键，即可返回分析结果，如图 17-10 所示。

结果显示的信息说明如下。

（1）Table：表示分析的表的名称。

（2）Op：表示执行的操作。analyze 表示进行分析操作。

（3）Msg_type：表示信息类型，其值通常是状态（status）、信息（info）、注意（note）、

警告（warning）和错误（error）之一。

（4）Msg_text：显示信息。

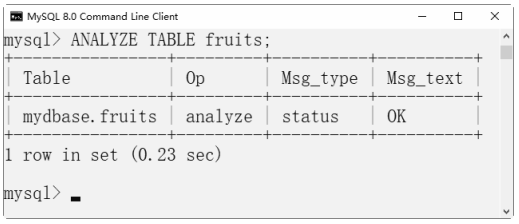

图 17-10　使用 ANALYZE TABLE 分析

2. 检查表

MySQL 中可以使用 CHECK TABLE 语句来检查表。CHECK TABLE 语句能够检查 InnoDB 和 MyISAM 类型的表是否存在错误。对于 MyISAM 类型的表，CHECK TABLE 语句还会更新关键字统计数据。而且，CHECK TABLE 也可以检查视图是否有错误，比如在视图定义中被引用的表已不存在。该语句的基本语法格式如下：

```
CHECK TABLE tbl_name [, tbl_name] … [option] …
option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

主要参数介绍如下。

- tbl_name：是表名。
- option：该参数有 5 个取值，分别是 QUICK、FAST、MEDIUM、 EXTENDED 和 CHANGED。各个选项的意义分别是：

（1）QUICK：不扫描行，不检查错误的连接。

（2）FAST：只检查没有被正确关闭的表。

（3）CHANGED：只检查上次检查后被更改的表和没有被正确关闭的表。

（4）MEDIUM：扫描行，以验证被删除的连接是有效的。也可以计算各行的关键字校验和，并使用计算出的校验和验证这一点。

（5）EXTENDED：对每行的所有关键字进行一个全面的关键字查找。这可以确保表是 100%一致的，但是花的时间较长。

注意：option 只对 MyISAM 类型的表有效，对 InnoDB 类型的表无效。CHECK TABLE 语句在执行过程中也会给表加上只读锁。

3. 优化表

MySQL 中使用 OPTIMIZE TABLE 语句来优化表。该语句对 InnoDB 和 MyISAM 类型的表都有效。但是，OPTIMIZE TABLE 语句只能优化表中的 VARCHAR、BLOB 或 TEXT 类型的字段。OPTIMIZE TABLE 语句的基本语法如下：

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] …
```

主要参数介绍如下。

- LOCAL | NO_WRITE_TO_BINLOG：该关键字的意义和分析表相同，都是指定不写入二进制日志。
- tbl_name：是表名。

通过 OPTIMIZE TABLE 语句可以消除删除和更新造成的文件碎片。OPTIMIZE TABLE 语句在执行过程中也会给表加上只读锁。

17.4 MySQL 服务器的优化

优化 MySQL 服务器主要从两个方面来优化，一方面是对硬件进行优化；另一方面是对 MySQL 服务的参数进行优化，对于可以定制参数的操作系统，也可以针对 MySQL 进行操作系统优化。

17.4.1 服务器硬件的优化

服务器的硬件性能直接决定着 MySQL 数据库的性能。硬件的性能瓶颈，直接决定 MySQL 数据库的运行速度和效率。对服务器硬件的优化，可以从以下几个方面进行。

（1）配置较大的内存。足够大的内存，是提高 MySQL 数据库性能的方法之一。内存的速度比磁盘 I/O 快得多，可以通过增加系统的缓冲区容量，使数据在内存停留的时间更长，以减少磁盘 I/O。

（2）配置高速磁盘系统，以减少读盘的等待时间，提高响应速度。

（3）合理分布磁盘 I/O，把磁盘 I/O 分散在多个设备上，以减少资源竞争，提高并行操作能力。

（4）配置多处理器，MySQL 是多线程的数据库，多处理器可同时执行多个线程。

17.4.2 MySQL 参数的优化

通过优化 MySQL 的参数可以提高资源利用率，从而达到提高 MySQL 服务器性能的目的。MySQL 服务的配置参数都在 my.cnf 或者 my.ini 文件的[MySQLd]组中。下面介绍几个对性能影响比较大的参数。

（1）key_buffer_size：表示索引缓冲区的大小。增加索引缓冲区可以得到更好处理的索引。当然，这个值也不是越大越好，它的大小取决于内存的大小。如果这个值太大，导致操作系统频繁换页，也会降低系统性能。

（2）table_cache：表示同时打开的表的个数。这个值越大，能够同时打开的表的个数越多。这个值不是越大越好，因为同时打开的表太多会影响操作系统的性能。

（3）query_cache_size：表示查询缓冲区的大小。该参数需要和 query_cache_type 配合使用。当 query_cache_type 值是 0 时，所有的查询都不使用查询缓冲区；当 query_cache_type=1 时，所有的查询都将使用查询缓冲区，除非在查询语句中指定 SQL_NO_CACHE，如 SELECT SQL_NO_CACHE * FROM tbl_name；当 query_cache_type=2 时，只有在查询语句中使用 SQL_CACHE 关键字，查询才会使用查询缓冲区。使用查询缓冲区可以提高查询的速度，这种方式只适用于修改操作少且经常执行相同的查询操作的情况。

（4）sort_buffer_size：表示排序缓存区的大小。这个值越大，进行排序的速度越快。

（5）read_buffer_size：表示每个线程连续扫描时为扫描的每个表分配的缓冲区的大小（字节）。当线程从表中连续读取记录时需要用到这个缓冲区。SET SESSION read_buffer_size=n 可以临时设置该参数的值。

（6）read_rnd_buffer_size：表示为每个线程保留的缓冲区的大小，与 read_buffer_size 相似，但主要用于存储按特定顺序读取出来的记录。也可以用 SET SESSION read_rnd_buffer_size=n 来临时设置该参数的值。如果频繁进行多次连续扫描，可以增加该值。

（7）innodb_buffer_pool_size：表示 InnoDB 类型的表和索引的最大缓存。这个值越大，查

询的速度就会越快。但是这个值太大会影响操作系统的性能。

（8）max_connections：表示数据库的最大连接数。这个连接数不是越大越好，因为这些连接会浪费内存的资源。过多的连接可能会导致 MySQL 服务器僵死。

（9）interactive_timeout：表示服务器在关闭连接前等待行动的秒数。

（10）thread_cache_size：表示可以复用的线程的数量。如果有很多新的线程，为了提高性能可以增大该参数的值。

（11）wait_timeout：表示服务器在关闭一个连接时等待行动的秒数。默认数值是 28 800。

总之，合理地配置这些参数可以提高 MySQL 服务器的性能。除上述参数以外，还有 innodb_log_ buffer_size、innodb_log_file_size 等参数。配置完参数以后，需要重新启动 MySQL 服务才会生效。

17.5　课后习题与练习

一、填充题

1. 在 MySQL 中，可以使用_____和_____关键字来分析查询语句。

答案：EXPLAIN，DESCRIBE

2. 使用 EXPLAIN 分析数据表时，输出的分析结果中_____字段的值表示输出行所引用的表。

答案：table

3. 优化数据库结果时，使用_____语句来分析数据表。

答案：ANALYZE TABLE

二、选择题

1. 下面关于索引优化的描述正确的是_____。

A. 使用索引可以提高数据库的查询速度，因此索引越多越好

B. LIKE 关键字配置的字符串不能以符号“%”开头，否则索引不起作用

C. 使用多列索引，查询条件必须要使用索引的第一个字符，索引才会被使用

D. 以上都不对

答案：B

2. 在 MySQL 中使用_____语句来优化表。

A. OPTIMIZE TABLE　　B. ANALYZE TABLE

C. EXPLAIN TABLE　　D. CHECK TABLE

答案：A

3. 优化 MySQL 服务器的配置参数时，_____参数表示查询缓存区的大小。

A. query_cache_size　B. key_buffer_size　C. sort_buffer_size　D. read_buffer_size

答案：A

三、简答题

1. 简述索引查询的缺陷。

2. 简述数据库结构优化的方法。

3. 简述 MySQL 服务器硬件和参数优化的方法。

17.6 新手疑难问题解答

疑问 1：在数据表中，建立的索引是不是越多越好？

解答：合理的索引可以提高查询的速度，但不是索引越多越好。在执行插入语句的时候，MySQL 要为新插入的记录建立索引，所以过多的索引会导致插入操作变慢。原则上是只有查询用的字段才建立索引。

疑问 2：在分析表时，为什么不能执行添加或删除数据记录？

解答：使用 ANALYZE TABLE 分析表的过程中，数据库系统会自动对表加一个只读锁。因此，在分析期间，只能读取表中的记录，不能执行添加、更新或删除数据记录。

17.7 实战训练

在 MySQL 中，针对数据库 Library 进行优化处理。

（1）查看 MySQL 服务器的连接数、查询次数和慢查询次数。

（2）分析数据库 Library 中读者信息表 Reader。

（3）检查数据库 Library 中读者信息表 Reader。

（4）优化数据库 Library 中读者信息表 Reader。