本节所讲内容：

- 存储过程
- 触发器
- 事物
- 找回密码

42.1 存 储 过 程

一、 存储过程

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重

要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

总结：存储语句就是一条或者多条 SQL 语句的集合。，提高执行效率及 SQL 代码封装功能。　在外部程序访问数据库中比如 php，当业务逻辑复杂的时候，一大堆的 SQL 和条件夹杂在 PHP 中是很混乱的，这个时候回用到 SQL 语句代码封装。

mysql 中的存储过程 封装 mysql 代码的。优点：

394

（1）存储过程相当于函数。方便你使用。

（2）一次编译（生成二进制文件），永久有效，提高执行效率。

为什么要使用存储过程

(1).存储过程增强了 SQL 语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

(2).存储过程允许标准组件是编程。存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的 SQL 语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。

(3).存储过程能实现较快的执行速度。如果某一操作包含大量的 Transaction-SQL 代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的 Transaction-SQL 语句在每次运行时都要进行编译和优化，速

度相对要慢一些。

(4).存储过程能过减少网络流量。针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的 Transaction-SQL 语句被组织程存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大增加了网络流量并降低了网络负载。

(5).存储过程可被作为一种安全机制来充分利用。系统管理员通过执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。

为什么不使用存储过程：

1）可移植性差

2）对于简单的 SQL 语句，存储过程没什么优势

3）如果存储过程中不一定会减少网络传输

4）如果只有一个用户使用数据库，那么存储过程对安全也没什么影响

5）团队开发时需要先统一标准，否则后期维护成本大

6）在大并发量访问的情况下，不宜写过多涉及运算的存储过程

7）业务逻辑复杂时，特别是涉及到对很大的表进行操作的时候，不如在前端先简化业务逻辑

42.1.1 定义存储过程

procedure [prəˈsi:dʒə(r)] 程序

```
create procedure 过程名（参数 1，参数 2....）
begin
  sql 语句
end
```

42.1.2 调用存储过程

```
call 过程名（参数 1，参数 2）;
```


默认 SQL 语句执行符号 ；

创建存储过程之前我们必须修改 mysql 语句默认结束符 ；

注释：

DELIMITER 是分割符的意思，因为 MySQL 默认以";"为分隔符，如果我们没有声明分割符，那么编译器会把存储过程当成 SQL 语句进行处理，则存储过程的编译过程会报错，所以要事先用 DELIMITER 关键字申明当前段分隔符，这样 MySQL 才会将";"当做存储过程中的代码，不会执行这些代码，用完了之后要把分隔符还原。

把 book 库导入进来，并且 USE book 库

修改执行符号

```
delimiter 新执行符号.
```

例：修改；为 //

```
delimiter //
```

注意：因为默认执行完 sql 语句遇到；后，就结束了。

【例 42.1】定一个存储过程，查看 category 表中所有数据

```
use book
mysql> delimiter //
mysql> create procedure selCg()
    -> begin
    -> select * from category;
    -> end   //
```

调用存储过程

```
mysql> call selCg()//
+---------+---------------+
| bTypeId | bTypeName       |
+---------+---------------+
|       1 | windows 应用 |
|       2 |  网站           |
|       3 | 3D 动画         |
|       4 | linux 学习     |
|       5 | Delphi 学习    |
|       6 |  黑客           |
|       7 |  网络技术    |
|       8 |  安全           |
|       9 |  平面           |
|      10 | AutoCAD 技术 |
```

总结：一次编译，永久执行。

42.1.3　存储过程参数传递

in 传入参数 把参数传递到过程内部。特点：读取外部变量值，且有效范围仅限存储过程内部

out 传出参数

into 赋值

【例 42.2】定义存储过程 getOneBook，当输入某书籍 id 后，可以调出对应书籍记录

语法：

```
create procedure 过程名（参数1，参数2....）
begin
  sql 语句：
end
//in---传入参数 ----------------------------------------
mysql> create procedure getOneBook(in b int)
    -> begin
    -> select * from books where bId=b;
    -> end //
Query OK, 0 rows affected (0.01 sec)

mysql> call getOneBook(3);//

+-----+------------------------------+---------+-------------
----------------+-------+------------+--------+-----------+
| bId | bName                          | bTypeId | publishing
| price | pubDate    | author | ISBN      |
+-----+------------------------------+---------+-------------
----------------+-------+------------+--------+-----------+
|   3 | 网络程序与设计一asp | 2       | 北方交通大学出版社 |    43 |
2005-02-01 | 王玥 | 75053815x |
+-----+------------------------------+---------+-------------
----------------+-------+------------+--------+-----------+
1 row in set (0.00 sec)
例：


//out --传出参数--------------------------------------
into 在 select 语句中赋值。
call demo(@a);  // @a mysql 中的变量
查看变量@a 中的值：
mysql> select @a
mysql> create procedure demo(out pa varchar(200))
begin
select bName into pa  from  books where bId=3;
end//调用，执行：
mysql> call demo(@a); //查看变量@a 中的值：
mysql> select @a;
+-----------------------------+
| @a                          |
+-----------------------------+
| 网络程序与设计一asp |
+-----------------------------+
1 row in set (0.00 sec)
```

42.1.4 过程内的变量的使用方法

申明：declare 变量名称 类型 ==》过程内的变量没有@

赋值：set 变量名 = (select 语句);

例：

```
mysql> create procedure demo1()
    -> begin
    -> declare str varchar(200);
    -> set str = (select bName from   books   where   bId=12);
    -> select str;
    -> end//
mysql> call demo1()//
+------------------------------+
| str                          |
+------------------------------+
| Fireworks 4 网页图形制作 |
+------------------------------+
1 row in set (0.06 sec)
```

查看存储过程由哪些
show procedure status \G

删除存储过程

语法：

方法一：DROP PROCEDURE 过程名

mysql> drop procedure p_inout;

方法二：DROP PROCEDURE IF EXISTS 存储过程名

这个语句被用来移除一个存储程序。不能在一个存储过程中删除另一个存储过程，只能调用另一个存储过程

42.2 触 发 器

与数据表有关，当表出现变化的时候（增、删、改），自动执行其他的特定的操作。

42.2.1 触发器的格式

语法：create　trigger 触发器名称　触发的时机　触发的动作

on 表名 for each row 触发器状态。

触发器名称：phptrigger 可以自己随意起

触发的时机： before /after　在执行动作之前还是之后

42.2.2　触发的动作

指的激发触发程序的语句类型

insert
update
delete

【例 42.3】当 category 表中，删除一个 bTypeid=3 的图书分类时，books 表中也要删除对应分类的图书信息。

```
mysql> create trigger delCategory after delete on category for each row
-> delete from books where bTypeid='3';
```

语法：create　trigger 触发器名称　触发的时机　触发的动作　on 表名 for each row

触发器状态。

在 category 执行删除前，查看 bTypeId=3 的图书分类：

```
mysql> select bName,bTypeId from books where bTypeId=3;
+-------------------------------------+---------+
| bName                                | bTypeId |
+-------------------------------------+---------+
| 3D MAX 3.0 创作效果百例              | 3       |
| 3DS MAX 4 横空出世                   | 3       |
| 3D MAX R3 动画制作与培训教程         | 3       |
| 3D Studio Max 3 综合使用             | 3       |
+-------------------------------------+---------+
```

删除 bTypeId=3 的记录

```
mysql> delete from category where bTypeId=3;
```

查看：是否还有 bTypeId=3 的图书记录。可以看出已经删除。

```
mysql> select bName,bTypeId from books where bTypeId=3;
Empty set (0.00 sec)
```

删除触发器

```
mysql> drop trigger delCategory;
```

42.3 事　　务

事务：（database transaction）:单个逻辑单元执行的一系列操作。

事务处理：可以确保非事务性单元的多个操作都能成功完成，否则不会更行数据资源。

优点：通过将一组操作组成一个，执行时，要么全部成功，要么全部失败的单元。

使程序更可靠，简化错误恢复。

语句：

START TRANSACTION　开启事务

COMMIT　提交当前事务

【例 42.4】创建一个事务，只有把 bId=1 和 bId=2 两条记录的 bName 都改成功后，才算修改成功

```
set autocommit=0;
mysql> start transaction;
    -> update books set   bName="cccccc" WHERE bId=1;
    -> update books set bName="dddddd" WHERE bId=2;
-> commit;//
```

测试，查看是否完成修改：

```
mysql> select bName from   books where bId=1 or bId=2;//
+--------+
| bName   |
+--------+
| cccccc |
```

| dddddd |

我们测试回滚操作，首先看我们的数据库存储引擎是否为 innodb
mysql> show create table books//\G

```
books | CREATE TABLE `books` (
  `bId` int(4) NOT NULL AUTO_INCREMENT,
  `bName` varchar(255) DEFAULT NULL,
  `bTypeId` enum('1','2','3','4','5','6','7','8','9','10') DEFAULT NULL,
  `publishing` varchar(255) DEFAULT NULL,
  `price` int(4) DEFAULT NULL,
  `pubDate` date DEFAULT NULL,
  `author` varchar(30) DEFAULT NULL,
  `ISBN` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`bId`)
) ENGINE=MyISAM AUTO_INCREMENT=45 DEFAULT CHARSET=utf8 |
```

因为 MyISAM 无法成功启动事务，虽然提交了，却无法回滚
修改数据库存储引擎为 innodb

402

delimiter ;
mysql> alter table books engine=innodb;
mysql> alter table category engine=innodb;

```
ENGINE=InnoDB AUTO_INCREMENT=45 DEFAULT CHARSET=utf8 |
```

```
mysql> select bName from books where bId=1 or bId=2;
+------------------------+
| bName                  |
+------------------------+
| 网站制作直通车         |
| 黑客与网络安全         |
+------------------------+
```

重新开启事务，并测试回滚

mysql> set autocommit=0;

mysql> delimiter //

mysql> start transaction;

-> update books set bName="HA" where bId=1;

-> update books set bName="LB" where bId=2;

-> commit;//

mysql> delimiter ;

```
mysql> select bName from books where bId=1 or bId=2;
+-------+
| bName |
+-------+
| HA    |
| LB    |
+-------+
```

无法回滚，因为我们 commit 已经提交了

mysql> delimiter //

mysql> start transaction; update books set bName="AH" where bId=1; update books set bName="BL" where bId=2;// 不提交

mysql> delimiter ;

```
mysql> select bName from books where bId=1 or bId=2;
+-------+
| bName |
+-------+
| AH    |
| BL    |
+-------+
```

回滚

mysql> rollback;

```
mysql> select bName from books where bId=1 or bId=2;
+-------+
| bName |
+-------+
| HA    |
| LB    |
+-------+
```

恢复了

举例 2：PHP 中使用事务实例(有 php 基础的可以拓展学习)

```
<?php
$handler=mysql_connect("localhost","root","password");
mysql_select_db("task");
mysql_query("SET AUTOCOMMIT=0");//设置为不自动提交 ,因为 MYSQL 默认立
```



```
即执行
mysql_query("BEGIN");//开始事务定义
if(!mysql_query("insert into trans (id) values('2')"))
{
mysql_query("ROLLBACK");//判断当执行失败时回滚
}
if(!mysql_query("insert into trans (id) values('4')"))
{
mysql_query("ROLLBACK");//判断执行失败回滚
}
mysql_query("COMMIT");//执行事务
mysql_close($handler);
```

?>

42.4 MySQL 架构

42.4.1 mysql 相关的配置文件

主配置文件： /etc/my.cnf

datadir=/var/lib/mysql

42.4.2 进程通讯 sock 文件

[root@xuegod63 ~]# ll /var/lib/mysql/mysql.sock
srwxrwxrwx 1 mysql mysql 0 Mar 27 17:48 /var/lib/mysql/mysql.sock
socket=/var/lib/mysql/mysql.sock

42.4.3 日志文件

错误日志文件

[mysqld_safe]
log-error=/var/log/mysqld.log

42.4.4 进程 ID 文件

pid-file=/var/run/mysqld/mysqld.pid

42.4.5 二进制日志文件 （二进制文件可以在读写分离时使用）

log-bin=mysql-bin.log

拓展：

mysqlbinlog 命令:解析 mysqlbinlog 日志的。

例如：mysql-bin.00001

bin-log 作用：用来记录 mysql 内部增删改查等对 mysql 数据库有更新的内容记录。

42.4.6 慢查询日志文件

在 mysql 配置文件 my.cnf 中增加

```
log-slow-queries=/var/lib/mysql/slowquery.log
#(指定日志文件存放位置 ,可以为空 ,系统会给一个缺省的文件 host_name-slow.log) ，
mysql 用户要对这个/var/lib/mysql/slowquery.log 文件有可写的权限
long_query_time=2       # (记录超过的时间，默认为 10s)
```


42.4.7 数据文件

```
[root@xuegod63 ~]# ls /var/lib/mysql/book/
books.frm   books.MYI       category.MYD   db.opt   t.MYD
books.MYD   category.frm    category.MYI   t.frm    t.MYI
```

- frm meta data 存储表定义
- MYD 存储数据文件
- MYI 存储索引文件

db.opt 作用 :MySQL 的每个数据库目录中有一个文件 db.opt,该文件主要 用来存储当前数据库的默认字符集和字符校验规则。

```
  eg.default-character-set=latin1
default-collation=latin1_swedish_ci
```

该文件中存储的是创建数据库时默认的字符集和字符集校验规则，则该数据库在以后创建表时如果没有指定字符集和校验规则，则该表的这两个属性将取自这两个表。

42.4.8 数据库系统架构

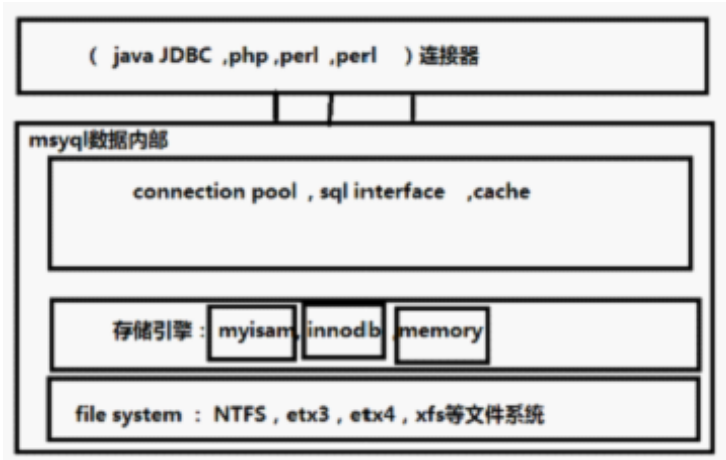


图 42.1　数据库系统架构

查看当前 myql 服务器支持的引擎：

mysql> show engines;

```
mysql> show engines;
+------------+---------+----------------------------------------------------------------+--------------+-----
| Engine     | Support | Comment                                                        | Transactions | XA
+------------+---------+----------------------------------------------------------------+--------------+-----
| MRG_MYISAM | YES     | Collection of identical MyISAM tables                          | NO           | NO
| CSV        | YES     | CSV storage engine                                             | NO           | NO
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         | NO           | NO
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign keys     | YES          | YES
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO
+------------+---------+----------------------------------------------------------------+--------------+-----
5 rows in set (0.00 sec)
```

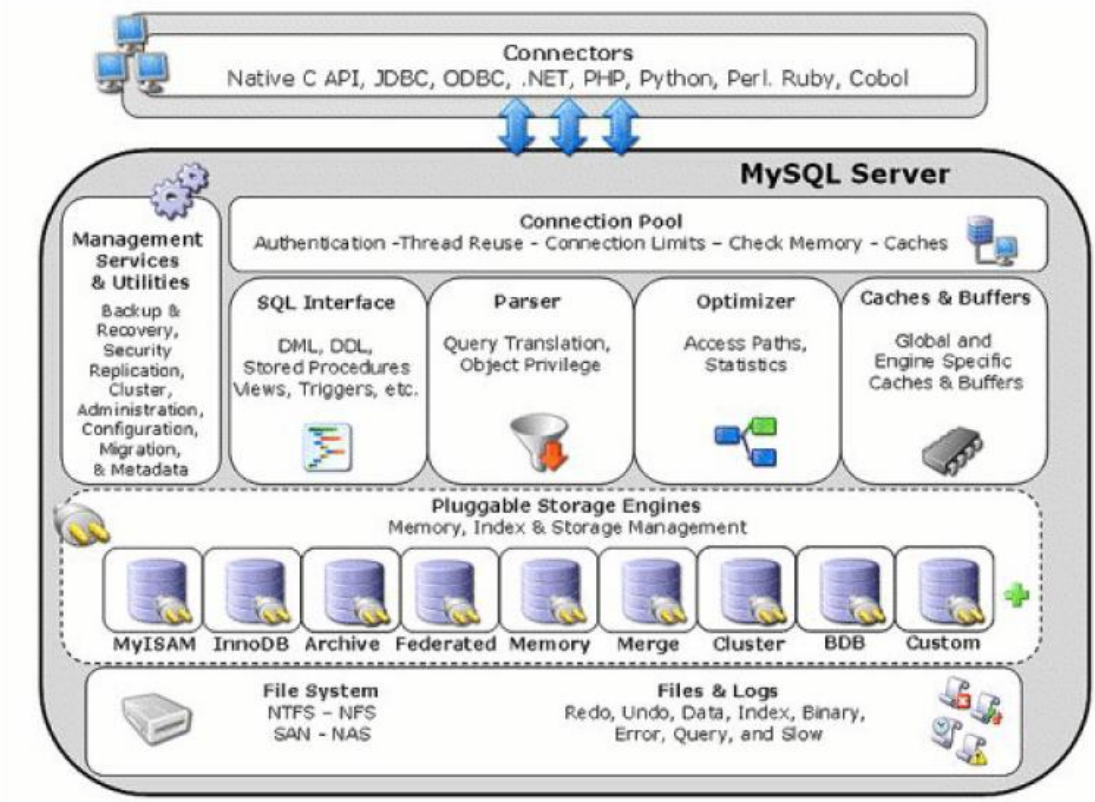
图 42.2　引擎

linux：文件系统 ext3　ext4　windows nfs　，存放视频都可以存放，但是占用空间大小和清晰度不一样，既然视频有这么多存储方式，而数据也有存储方式。这些存储方式我们叫他引擎　。

不同的引擎，占用空间大小，读取性能是不一样的。

存储引擎的系统结构图：

2.2 系统架构　　（sql 层 引擎层）



408

42.4.9　常见存储引擎介绍

myisam：　（mysql

特性：

- 不支持事务。宕机时会破坏表；
- 使用较小的内存和磁盘空间；
- 基于表的锁。表级锁；
- mysql 只缓存 index 索引， 数据由 OS 缓存。

典型应运：

（1）日志系统。

（2）大部分都是读是的操作。　门户网站，企业站点 www.xuegod.cn

（3）没有事务　，低并发。

➲ InnoDB 存储引擎：

MySQL 发展到今天，InnoDB 引擎已经作为绝对的主力，除了像大数据量分析等比较特殊领域需求外，它适用于众多场景

innoDB 是 MySQL 下使用最广泛的引擎，它是基于 MySQL 的高可扩展性和高性能存储引擎，从 5.5 版本开始，它已经成为了默认引擎。

InnODB 引擎支持众多特性：

a) 支持 ACID，简单地说就是支持事务完整性、一致性；

b) 支持行锁，以及类似 ORACLE 的一致性读，多用户并发；

c) 独有的聚集索引主键设计方式，可大幅提升并发读写性能；

d) 支持外键；

e) 支持崩溃数据自修复；

注意问题

a) 所有 InnoDB 数据表都创建一个和业务无关的自增数字型作为主键，对保证性能很有帮助；

b) 杜绝使用 text/blob，确实需要使用的，尽可能拆分出去成一个独立的表；

c) 时间建议使用 TIMESTAMP 类型存储；

d) IPV4 地址建议用 INT UNSIGNED 类型存储；

e) 性别等非是即非的逻辑，建议采用 TINYINT 存储，而不是 CHAR(1)；bool

f) 存储较长文本内容时，建议采用 JSON/BSON 格式存储；

InnoDB：

innodb 适用于：

InnoDB Good For:

1) 需要事务的应用。
2) 高并发的应用。
3) 自动恢复。
4) 较快速的基于主键的操作。

【例 42.5】修改默认引擎为 innodb

```
[root@xuegod63 mysql]# vim /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
```

```
default-storage-engine=innodb
[root@xuegod63 mysql]# service mysqld restart
```

创建表测试：

```
mysql>    create table ca (id int,name char(8));
Query OK, 0 rows affected (0.06 sec)
mysql> show create table ca \G
*************************** 1. row ***************************
           Table: ca
Create Table: CREATE TABLE `ca` (
   `id` int(11) DEFAULT NULL,
   `name` char(8) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

42.4.10　MyISAM 与 InnoDB 两者之间区别

（1）MyISAM 不支持事务，InnoDB 是事务类型的存储引擎当我们的表需要用到事务支持的时候，那肯定是不能选择 MyISAM 了。

（2）MyISAM 只支持表级锁， InnoDB 支持行级锁和表级锁默认为行级锁。

表级锁：直接锁定整张表，在锁定期间，其他进程无法对该表进行写操作，如果设置的是写锁，那么其他进程读也不允许。

（3）MyISAM 引擎不支持外键，InnoDB 支持外键。

（4） InnoDB 是为处理巨大数据量时的最大性能设计，它的 CPU 效率可能是任何其它基于磁盘的关系数据库引擎所不能匹敌的。

（5）MyISAM 支持全文索引（FULLTEXT），InnoDB 不支持。

（6）MyISAM 引擎的表的查询、更新、插入的效率要比 InnoDB 高。

42.5 锁

主要有：表/行级锁。

42.5.1 表级锁

对于 write，表锁定原理如下：

如果表上没有锁，在其上面放置一个写锁，否则，把锁定请求放在写锁队列中。

当一个锁定被释放时，表可被写锁定队列中的线程得到，然后才是读锁定队列中的线程。这意味着，如果你在一个表上有许多更新，那么你的 SELECT 语句将等到所有的写锁定线程执行完。

MyISAM 是表级锁定的存储引擎，它不会出现死锁问题

42.5.2 行级锁

只对指定的行进行锁定，其他进程还是可以对表中的其他行进行操作的。

行级锁是 Mysql 粒度最小的一种锁，它能大大的减少数据库操作的冲突，但是粒度越小实现成本也越大。例：公共厕所如图 42.5 所示。

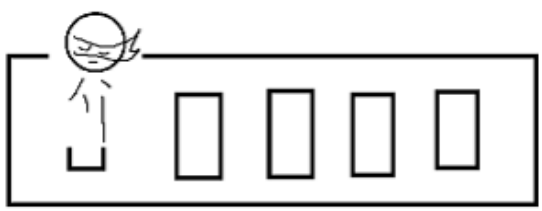


图 42.5 行级锁

行级锁可能会导致“死锁”，分析原因：Mysql 行级锁并不是直接锁记录，而是锁索引。索引分为主键索引和非主键索引两种，如果一条 sql 语句操作了主键索引，那么 Mysql 就会锁定这个主键索引，如果 sql 语句操作的是非主键索引，那么 Mysql 会先锁定这个非主键索引，再去锁定主键索引。

在 UPDATE 和 DELETE 操作时 Mysql 不仅会锁定所有 WHERE 条件扫描过得索引，还会锁定相邻的键值（被修改的字段）。

42.5.3 死锁

表 Test 字段结构：（ID,STATE,TIME） 主键索引：ID 非主键索引：STATE

当执行 SQL 语句 1：

"UPDATE STATE =1011 WHERE STATE=1000" 语句的时候会锁定 STATE 索引，由于 STATE 是非主键索引,所以 Mysql 还会去请求锁定 ID 索引

当另一个 SQL 语句 2 与语句 1 几乎同时执行时：

SQL 语句 2：“UPDATE STATE=1010 WHERE ID=1” 对于语句 2 Mysql 会先锁定 ID 索引，由于语句 2 操作了 STATE 字段，所以 Mysql 还会请求锁定 STATE 索引。这时。彼此锁定着对方需要的索引，又都在等待对方释放锁定。所以出现了"死锁"的情况。