数据库系统原理作业#6

**一、存储过程和触发器小实验**

1、（1）实现单表查询的存储过程如下：

|  |
| --- |
| #创建单表查询的存储过程singleselect  delimiter //  create procedure singleselect()  begin  select \* from train;  end; //  #调用存储过程singleselect  call singleselect; |

调用该存储过程，得到如下结果：



实现多表查询的存储过程如下：

|  |
| --- |
| #创建多表查询的存储过程 multiselect  delimiter //  create procedure multiselect()  begin  SELECT \*  FROM (SELECT u.id,i.originating\_station AS originating\_station,i.arrival\_station AS arrival\_station,t.route  FROM user u,booking b,information i,train t  WHERE u.id=b.id AND b.no=i.no AND i.trips=t.trips) t  WHERE t.id=10000;  end; //  #调用存储过程multiselect  call multiselect; |

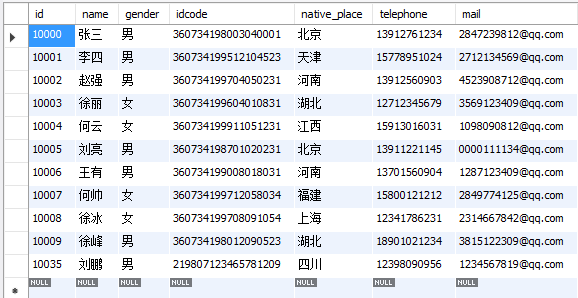
调用该存储过程，得到如下结果：



（2）实现数据插入的存储过程如下：

|  |
| --- |
| #创建数据插入的存储过程insert\_user  delimiter //  create procedure insert\_user()  begin  insert into user values(10009,'徐峰','男','360734198012090523','湖北',18901021234,'3815122309@qq.com');  end; // |

调用该存储过程，得到如下结果：

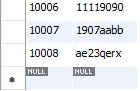


通过查询user表发现id=10009的元组已经成功插入。

（3）实现数据删除的存储过程如下：

|  |
| --- |
| #创建数据删除的存储过程delete\_account  delimiter //  create procedure delete\_account()  begin  delete from account where id=10035;  end; //  #调用存储过程  call delete\_account; |

调用该存储过程，可得如下结果：



发现account表中id=10035的元组已被删除。

（4）创建数据修改的存储过程如下：

|  |
| --- |
| #创建数据修改的存储过程 update\_user  delimiter //  create procedure update\_user()  begin  update user  set gender='男'  where id=10035;  end; //  #调用存储过程  call update\_user; |

调用该存储过程，可得：

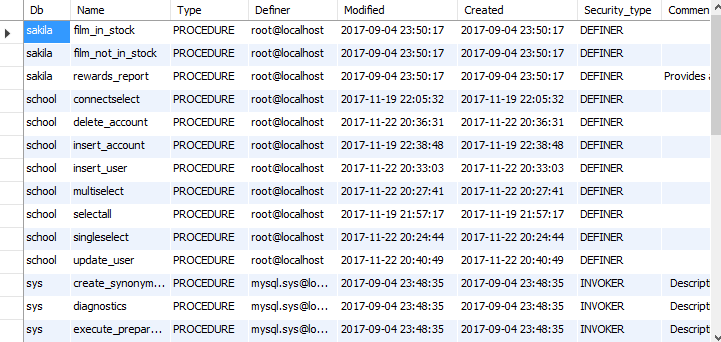


发现user表中id=10035的元组的gender属性值已被修改。

（5）其实，通过查询数据字典，也可以检验存储过程是否已被建立：

|  |
| --- |
| #查询存储过程  show procedure status; |

结果为：

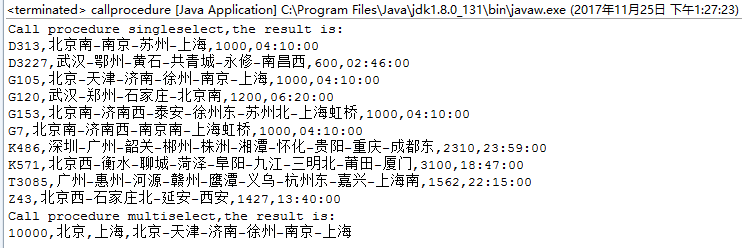


发现，名为singleselect、multiselect、insert\_user、delete\_account、update\_user的存储过程已经存在于数据字典中。

2、为测试方便，本处通过C/S架构调用后台的singleselect、multiselect两个存储过程，java代码如下：**（详见callprocedure.java文件）**

|  |
| --- |
| package procedure;  import java.sql.CallableStatement;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  public class callprocedure {  public static final String DRIVER\_CLASS ="com.mysql.jdbc.Driver";  public static final String URL="jdbc:mysql://localhost:3306/school";  public static final String USERNAME="root";  public static final String PASSWORD="15281253";    public static void main(String[]args)throws Exception{  Class.forName(DRIVER\_CLASS);  Connection connection=DriverManager.getConnection(URL,USERNAME,PASSWORD);  String sql1="{call singleselect}"; //调用存储过程singleselect  String sql2="{call multiselect}"; //调用存储过程multiselect  CallableStatement cstm= connection.prepareCall(sql1); //实例化对象cstm  ResultSet rs=cstm.executeQuery();  System.out.println("Call procedure singleselect,the result is: ");  while(rs.next()){  String trips=rs.getString(1);  String route=rs.getString(2);  String driven\_distance=rs.getString(3);  String driven\_time=rs.getString(4);  System.out.println(trips+","+route+","+driven\_distance+","+driven\_time);  }  CallableStatement cstm2=connection.prepareCall(sql2);//实例化对象cstm2  ResultSet rs2=cstm2.executeQuery();  System.out.println("Call procedure multiselect,the result is: ");  while(rs2.next()){  int id=rs2.getInt(1);  String originating\_station=rs2.getString(2);  String arrival\_station=rs2.getString(3);  String route2=rs2.getString(4);  System.out.println(id+","+originating\_station+","+arrival\_station+","+route2);  }  cstm2.close();  cstm.close();  connection.close(); //关闭数据库连接  }  } |

该代码的运行结果如下图所示：



3、（1）由数据插入引起的触发：

|  |
| --- |
| #创建一个由数据插入所引起的触发器:在对account表执行插入操作后information表中G105车次的price属性自增50  delimiter //  create trigger InsertInformation  after insert on account  for each row  begin  update information  set price=price+50  where trips='G105';  end; //  #测试触发器  insert into account(id,password) values(10009,'19183901'); |

创建该触发器后，对account表执行插入操作可得：



Price的值由650变为700.

（2）由数据更新引起的触发：

|  |
| --- |
| #创建一个由数据更新所引起的触发器  delimiter //  create trigger UpdateInformation  before update on account  for each row  begin  update information  set price=price+50  where trips='G105';  end;//  #测试触发器  Update account  set password='2222222'  where id=10009; |

执行更新语句，再查询information表，可得：



Price的值由700变为750.

（3）由数据删除引起的触发：

|  |
| --- |
| #1.8 创建一个由数据删除所引起的触发器  delimiter //  create trigger DeleteInformation  after delete on account  for each row  begin  update information  set price=price+50  where trips='G105';  end; //  #测试触发器  delete from account where id=10009; |

执行删除语句，再查询information表，可得：

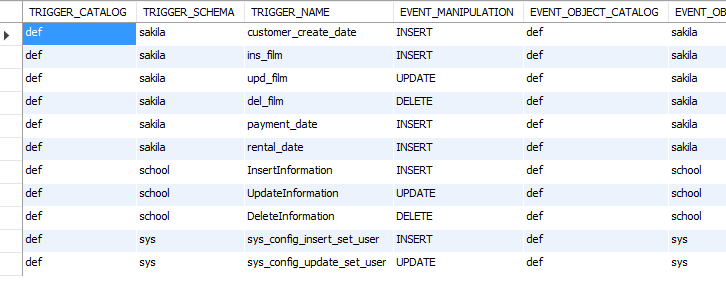


Price的值由750变为800.

查询数据字典，看有哪些触发器被建立：

|  |
| --- |
| #查询触发器  SELECT \* FROM information\_schema.`TRIGGERS`; |

结果为：



可见，名为InsertInformation、UpdateInformation、DeleteInformation的触发器已存在于数据字典中。

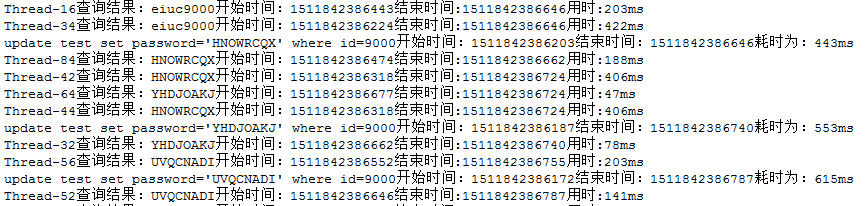
**二、并发模拟实验**

1、该题采用多线程同步对数据库进行插入和查询操作，模拟并发操作，体验并发操作带来的问题。

**具体代码：请参照ThreadToMysql工程**

2、当执行代码时，发现在数据修改完之后，线程查询出的结果依然是修改前的。

运行截图如下：



如第6行、第10行都出现了这个问题。

分析：由于没有进行并发控制，不同线程同时对数据库进行访问操作，导致数据库的读写出现问题，这是应该避免的，我们可以通过加入锁的机制来避免此类问题，这也是对数据库安全性的一种维护。

**三、压力测试与索引实验**

1、调用前端程序向数据库插入数据，**具体参照：insert工程**

2、无索引压力测试实验

查询test表和向test表插入数据耗时如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 插入数据量级 | 100 | 1000 | 5000 | 10000 | 50000 | 100000 | 1000000 |
| 插入耗时 | 154ms | 459ms | 955ms | 2224ms | 3984ms | 6987ms | 60552ms |
| 查询耗时 | 0.000s | 0.000s | 0.047s | 0.000s | 0.000s | 0.000s | 0.046s |

3、有索引的压力测试

清空test表，重新插入数据。对password属性建立唯一索引，重新进行查询和插入测试，结果如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 插入数据量级 | 100 | 1000 | 5000 | 10000 | 50000 | 100000 | 1000000 |
| 插入耗时 | 160ms | 472ms | 1023ms | 2356ms | 4100ms | 7100ms | 61240ms |
| 查询耗时 | 0.000s | 0.000s | 0.000s | 0.000ms | 0.000s | 0.012s | 0.022s |

4、无索引压力测试的图表如下：

分析：由图表可知数据插入操作随数据规模的增大线性增加；而查询操作耗时较少。

有索引的压力测试图表如下：

分析：该表与上表类似，数据插入操作的耗时随着数据规模的增大线性递增，相比无索引时插入时间稍微变长一些；数据查询耗时相比无索引时时间稍微减少，原因是对password属性建立唯一索引需要排序，所以需要一定耗时，但由于索引的建立，所以查询速率有上升。

**四、备份和日志初步实验**

1、数据库备份和还原

数据备份的方法：

①使用mysqldump命令备份

Mysqldump命令将数据库的数据备份成一个文本文件，表的结构和表中的数据将存储在生成的文本文件中。

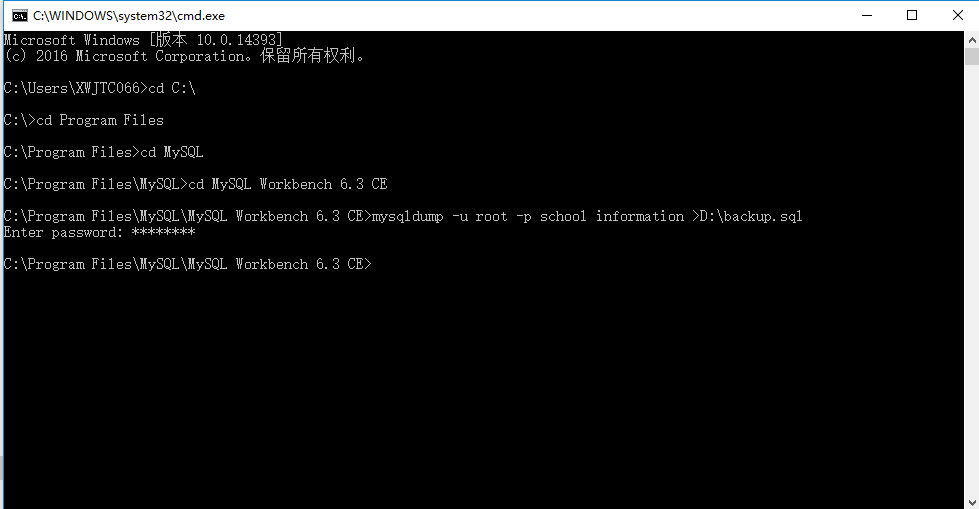
原理：它先查出需要备份的表的结构，再在文本文件中生成一个Create语句，然后将表中的记录转换成一条INSERT语句。然后通过这些语句，就能够创建表并插入数据。

基本语法:

Mysqldump –u username –p dbname table1 table2…（不带表名则表示对整个数据库进行备份）> BackUpName.sql

如：使用root用户备份school数据库下的information表

|  |
| --- |
| mysqldump –u root –p school information >D:\backup.sql |



**Information的备份内容详见：backup.sql**

如：使用root用户备份school数据库

|  |
| --- |
| mysqldump –u root –p school >D:\backup1.sql |

**数据库school的备份内容详见：backup1.sql**

如：使用root用户备份多个数据库

|  |
| --- |
| mysqldump –u root –p --databases school sakila >D:\backup2.sql |

**school、sakila数据库的备份内容见：back2.sql**

如：使用root用户备份所有数据库

|  |
| --- |
| mysqldump –u root –p –all-databases >D:\all.sql |

②直接复制整个数据库目录

MySQL有一种非常简单的备份方法，就是将MySQL中的数据库文件直接复制出来。这是最简单，速度最快的方法。

不过在此之前，要先将服务器停止，这样才可以保证在复制期间数据库的数据不会发生变化。如果在复制数据库的过程中还有数据写入，就会造成数据不一致。这种情况在开发环境可以，但是在生产环境中很难允许备份服务器。

　　注意：这种方法不适用于InnoDB存储引擎的表，而对于MyISAM存储引擎的表很方便。同时，还原时MySQL的版本最好相同。

数据还原的方法：

①还原使用mysqldump命令备份的数据库的语法如下：

*mysql -u root -p [dbname] < backup.sql*

如：还原school数据库

|  |
| --- |
| mysql –u root –p < D:\backup1.sql |

②还原直接复制目录的备份

通过这种方式还原时，必须保证两个MySQL数据库的版本号是相同的。MyISAM类型的表有效，对于InnoDB类型的表不可用，InnoDB表的表空间不能直接复制。

2、数据库日志

日志文件是用来记录事务对数据库的更新操作的文件。

日志文件在数据库恢复中起重要作用，可以用来进行事务故障恢复和系统故障恢复，并协助后备副本进行介质故障恢复。

具体如下：

（1）事务故障恢复和系统故障恢复必须用日志文件；

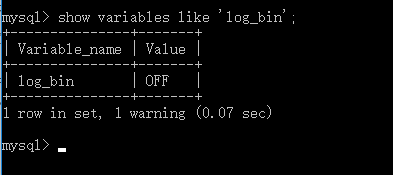
（2）在动态转储方式中必须建立日志文件，后备副本（备份）和日志文件结合起来才能有效地恢复数据库；

（3）在静态转储方式中也可以建立日志，当数据库破坏后可重新装入后援副本把数据库恢复到转储结束时刻的正确状态，然后利用日志文件把已完成的事务进行重做处理，对故障发生时尚未完成的事务进行撤销处理。这样不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态。

3、MySQL数据库平台的日志

查看数据库日志：

①确定日志是否已经启用：



②发现日志已经关闭，所以先启用日志：在my.ini里进行配置

|  |
| --- |
| # 在[mysqld] 中輸入  #log  log-error="E:/PROGRA~1/EASYPH~1.0B1/mysql/logs/error.log"  log="E:/PROGRA~1/EASYPH~1.0B1/mysql/logs/mysql.log"  long\_query\_time=2  log-slow-queries= "E:/PROGRA~1/EASYPH~1.0B1/mysql/logs/slowquery.log" |

对于没有my.ini文件的，可以进行如下操作：

|  |
| --- |
| /\* 首先通过命令行登录mysql,命令为:\*/  mysql -h localhost -u root -p  //接下来输入对应的passwprd,进入mysql操作界面  mysql>set global log\_output=file;//输出到文件  mysql>set global general\_log\_file='D:/file-path.log';//输出文件地址  mysql>set global general\_log=on;//开启通用日志 |

这样，日志就会输出到file\_path.log文件中。

当对information表作修改操作时：

|  |
| --- |
| Update information  set price=209  where trips='G105'; |

此时日志文件中的日志记录为：

|  |
| --- |
| 2017-11-25T11:41:06.619167Z 10 Query Update information  set price=209  where trips='G105'  2017-11-25T11:41:06.727456Z 9 Query SELECT st.\* FROM performance\_schema.events\_statements\_current st JOIN performance\_schema.threads thr ON thr.thread\_id = st.thread\_id WHERE thr.processlist\_id = 10  2017-11-25T11:41:06.728959Z 9 Query SELECT st.\* FROM performance\_schema.events\_stages\_history\_long st WHERE st.nesting\_event\_id = 30  2017-11-25T11:41:06.728959Z 9 Query SELECT st.\* FROM performance\_schema.events\_waits\_history\_long st WHERE st.nesting\_event\_id = 30 |

当对account作插入操作时：

|  |
| --- |
| Insert into account values(10009,’ajshqskkw’); |

此时日志文件中的日志记录为：

|  |
| --- |
| 2017-11-25T11:47:04.245202Z 10 Query Insert into account values(10009,'ajshqskkw') |

解释该记录如下：

此日志为通用日志，其中，2017-11-25T11:47:04.245202Z代表该操作执行的时间；10代表该操作的id；Query指的是command；后面的语句代表Argument。

**具体请参照日志文件：file\_path.log**