《数据库原理》实验报告

实验名称　　数据库的事务创建与运行实验

班　　级　　2014211304

学　　号　　2014211218

姓　　名　　史文翰

实验七　数据库的事务创建与运行实验

一、　实验目的

通过实验，了解 kingbase 数据库系统中各类数据库事务的定义机制和基于锁的并发控制机制，掌握 kingbase 数据库系统的事务控制机制。

二、　实验环境

MySQL 5.7 on win10 x64

三、　实验内容

1、定义三种模式的数据库事务
2、查看事务的隔离级别

四、　实验步骤及结果分析

1、MySQL 中三种（两大类）数据库事务模式

- **显示事务**

使用 set autocommit = 0 的命令来使 MySQL 工作在显示事务模式下。在这种模式下，关键字 begin 表示一个事务的开始，commit 表示提交这个事务，rollback 表示删除这个事务。

```
mysql> set autocommit = 0;
Query OK, 0 rows affected (0.00 sec)
```

我们假设 course 表的初始状态如下所示：

```
mysql> select * from course;
+-----------+----------------------+-------+--------+----------+
| course_no | course_name          | hours | credit | semester |
+-----------+----------------------+-------+--------+----------+
| C01       | 编译原理             |    51 |      3 | 秋       |
| C02       | 数据库原理           |    51 |      3 | 春       |
| C03       | 操作系统             |    51 |      2 | 秋       |
| C04       | JAVA 程序设计        |    40 |      2 | 秋       |
| C05       | 计算机组成原理       |    30 |      2 | 春       |
+-----------+----------------------+-------+--------+----------+
5 rows in set (0.00 sec)
```

我们定义如下事务：

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into course values ('C77','',0,0,'');
Query OK, 1 row affected (0.00 sec)

mysql> savepoint s1;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into course values ('C88','',0,0,'');
Query OK, 1 row affected (0.00 sec)

mysql> savepoint s2;
Query OK, 0 rows affected (0.00 sec)
```

可以在存储点 s1，course 表只增加了 C77 一行，而 s2 已经再增加了 C88 一行。此时该事务还没有被提交，我们利用 select 来查看当前的状态：

```
mysql> select * from course;
+-----------+----------------------+-------+--------+----------+
| course_no | course_name          | hours | credit | semester |
+-----------+----------------------+-------+--------+----------+
| C01       | 编译原理             |    51 |      3 | 秋       |
| C02       | 数据库原理           |    51 |      3 | 春       |
| C03       | 操作系统             |    51 |      2 | 秋       |
| C04       | JAVA 程序设计        |    40 |      2 | 秋       |
| C05       | 计算机组成原理       |    30 |      2 | 春       |
| C77       |                      |     0 |      0 |          |
| C88       |                      |     0 |      0 |          |
+-----------+----------------------+-------+--------+----------+
7 rows in set (0.00 sec)
```

可知两行均“已”插入，我们将事务回滚到 s1：

```
mysql> rollback to s1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from course;
+-----------+----------------------+-------+--------+----------+
| course_no | course_name          | hours | credit | semester |
+-----------+----------------------+-------+--------+----------+
| C01       | 编译原理             |    51 |      3 | 秋       |
| C02       | 数据库原理           |    51 |      3 | 春       |
| C03       | 操作系统             |    51 |      2 | 秋       |
| C04       | JAVA 程序设计        |    40 |      2 | 秋       |
| C05       | 计算机组成原理       |    30 |      2 | 春       |
| C77       |                      |     0 |      0 |          |
+-----------+----------------------+-------+--------+----------+
6 rows in set (0.00 sec)
```

可见确实 course 表回到了只插入 C77 的状态。如果我们 rollback，相当于整个事务没有被执行。

```
mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from course;
+-----------+----------------------+-------+--------+----------+
| course_no | course_name          | hours | credit | semester |
+-----------+----------------------+-------+--------+----------+
| C01       | 编译原理             |    51 |      3 | 秋       |
| C02       | 数据库原理           |    51 |      3 | 春       |
| C03       | 操作系统             |    51 |      2 | 秋       |
| C04       | JAVA 程序设计        |    40 |      2 | 秋       |
| C05       | 计算机组成原理       |    30 |      2 | 春       |
+-----------+----------------------+-------+--------+----------+
5 rows in set (0.00 sec)
```

如果想真实地插入一个数据，我们可以如下定义事务并最终提交这个事务：

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into course values ('C77','',0,0,'');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from course;
+-----------+----------------------+-------+--------+----------+
| course_no | course_name          | hours | credit | semester |
+-----------+----------------------+-------+--------+----------+
| C01       | 编译原理             |    51 |      3 | 秋       |
| C02       | 数据库原理           |    51 |      3 | 春       |
| C03       | 操作系统             |    51 |      2 | 秋       |
| C04       | JAVA 程序设计        |    40 |      2 | 秋       |
| C05       | 计算机组成原理       |    30 |      2 | 春       |
| C77       |                      |     0 |      0 |          |
+-----------+----------------------+-------+--------+----------+
6 rows in set (0.00 sec)
```

- **隐式事务（自动提交事务）**

当我们按照默认的情况即 autocommit=1 时，每当我们执行一条 SQL 语句，这条语句就被当做一个完整的事务并提交，因此在之前实验的操作均属于被自动提交的事务，这种模式没有回滚等机制的支持。

然而即使使autocommit=0,开启显示事务,也有一部分SQL语句被视为隐式事务。例如 drop table：

```
mysql> drop table course;
Query OK, 0 rows affected (0.11 sec)

mysql> show tables;
+---------------+
| Tables_in_stu |
+---------------+
| info_sel      |
| sel           |
| student       |
| tmp           |
+---------------+
4 rows in set (0.00 sec)
```

我们并没有在这里提交 drop 事务，但是由于 MySQL 的隐式事务机制，该事务已经被自动提交。我们用 rollback 来验证这一结果：

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+---------------+
| Tables_in_stu |
+---------------+
| info_sel      |
| sel           |
| student       |
| tmp           |
+---------------+
4 rows in set (0.00 sec)
```

可见 course 表并没有成功回滚，这个事务已经被提交。

2、查看事务隔离级别

MySQL 提供了如下四种隔离级别：

（1）SERIALIZABLE(序列化)

//以序列的形式处理事务，只有事务提交后，用户才能看到，但是该级别的孤立会影响 mysql 的性能，因为需要占用大量的资源，以保证使大量事务在任意时间不被用户看到。

（2）REPEATABLE　READ(可重读)

// 相比序列化该级别在应用的安全性上做了部分妥协。

（3）READ COMMITTED(提交后读)

// 提交后读的安全性比可重读还要低。在这一级的事务，用户可以看到其他事务添加的新记录，在事务处理时，如果存在其他用户同时对事务的相应表进行修改，那么同一事务在不同时间使用 select 查询得到的结果集可能不同。

（4）READ　UNCOMMITTED(未提交读)

//安全性相比提交后读就更低，同时该孤立及也是事务之间最小的间隔（孤立程度），该孤立级容易产生虚幻读操作。其他用户可以在该孤立级上看到未提交的事务。

查看事务的隔离级别如下：

```
mysql> select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)
```

五、 实验小结

本实验主要帮助初学者加深对数据库事务和隔离级别的理解。何为“事务”？我理解为事务是一种被设计出来的机制，该机制对数据库的一系列操作提出了一些约束，比如原子性约束（语句组的不可分割性），一致性（修改的一致性，为了 rollback 提供前提），隔离性（每个事务都有自己的空间，不同的事务之间不发生影响）和持久性（事务一旦被提交，对数据的修改就是永久的）。这种机制为数据库的可靠性提供了保障，比如提出了回滚机制，并很好地控制和处理并发请求的问题。

而这其中有一对矛盾：事务越独立，并发的副作用就越小，但付出的代价就更大，这是因为隔离的本质是串行，这与并发是冲突的。为了平衡独立性和并发性，数据库用隔离级别来平衡这两个指标以适应不同情境下的应用。