



# 第三次上机 事务处理与并发控制

2019年12月17日

# 主要内容

---

1. MySQL事务编程的基本概念
2. 数据库事务编程（内容一、二、三）
3. 分布式数据库并发控制（内容四）

# MySQL事务的创建

1. autocommit=1（系统默认值），事务的开启与提交又分为两种状态：

①**手动开启手动提交**：当用户执行start transaction命令时（事务初始化），一个事务开启，当用户执行commit命令时当前事务提交。从用户执行start transaction命令到用户执行commit命令之间的一系列操作作为一个完整的事务周期。若不执行commit命令，系统则默认事务回滚。

②**自动开启自动提交**：如果用户未执行start transaction命令而对数据库进行了操作，系统则默认用户对数据库的**每一个操作**作为一个孤立的事务，也就是说用户每进行一次操作系都会即时提交或者即时回滚。这种情况下用户的每一个操作都是一个完整的事务周期。

2. autocommit=0，事务则在用户本次对数据进行操作时自动开启，在用户执行commit命令时提交，用户本次对数据库开始进行操作到用户执行commit命令之间的一系列操作作为一个完整的事务周期。若不执行commit命令，系统则默认事务回滚。当前情况下事务的状态是**自动开启手动提交**

有关的MySQL指令：

```
show variables like "autocommit";  
set autocommit = 0;
```



# 事务的四种隔离性

- ❑ **Serializable（串行化）**：一个事务在执行过程中完全看不到其他事务对数据库做的更新
- ❑ **Repeatable Read（可重复读）**：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，不能看到其他事务对已经提交的已有记录的更新
- ❑ **Read Committed（读已提交）**：一个事务在执行过程中可以看到其他事务已经提交的新插入的记录，而且能看到其他事务对已经提交的已有记录的更新
- ❑ **Read Uncommitted（读未提交）**：一个事务在执行过程中可以看到其他事务没有提交的新插入的记录，而且能看到其他事务没有提交的对已有记录的更新

# 使用Workbench模拟多用户访问

1. 打开MySQL Workbench, 新建connection连接至本地数据库服务器 (如dbcourse和dbcourse2)
2. 在不同的connection中执行SQL语句, 模拟多个用户对数据库的并发访问

## Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

[Browse Documentation >](#)

[Read the Blog >](#)

[Discuss on the](#)

### MySQL Connections ⊕ ⊖

Local instance MySQL Rou...

root  
localhost:3306

dbcourse

root  
127.0.0.1:3306

Local instance MySQL80

root  
localhost:3306

dbcourse2

root  
127.0.0.1:3306

# 使用JDBC模拟多用户并发访问

```
public class Main {
    public static void main(String[] args) throws ClassNotFoundException {
        try {
            Connection conn1 = DriverManager.getConnection("jdbc:mysql://localhost:3306/sys?useSSL=false", "admin", "admin");
            Connection conn2 = DriverManager.getConnection("jdbc:mysql://localhost:3306/sys?useSSL=false", "admin", "admin");
            conn1.setAutoCommit(false);
            conn2.setAutoCommit(false);
            Thread1 mTh1=new Thread1(conn1);
            Thread2 mTh2=new Thread2(conn2);
            mTh1.start();
            mTh2.start();
        }
        catch( Exception e )
        {
            e.printStackTrace();
        }
    }
}
```

- 每个线程模拟用户操作数据库的行为
- 通过JDBC合理的建立连接，对数据库进行操作
- 事务通过setAutoCommit (false) 及 conn.commit()实现
- 不推荐使用“root”访问数据库
- 在workbench中执行sql验证已经成功连接或操作数据库

```
class Thread1 extends Thread{
    private Connection conn;
    public Thread1(Connection conn) {
        this.conn=conn;
    }
    public void run() {
        try {
            System.out.println("Thread1 running");
            Statement stat = conn.createStatement();
            for(int i=1;i<=10;i++){
                String sql = "select * from test7 where test7_id = " + String.valueOf(i) + ";";
                ResultSet rs = stat.executeQuery(sql);
                while (rs.next()) {
                    System.out.println("val of id = " + String.valueOf(i)+" : " + rs.getString("val"));
                }
                rs.close();
            }
            conn.commit();
            System.out.println("Thread1 exits");
        }
        catch (SQLException ex) {
            ex.printStackTrace();
            try {
                //An error occurred so we rollback the changes.
                this.conn.rollback();
            } catch (SQLException ex1) {
                ex1.printStackTrace();
            }
        }
    }
}
```

# 封锁 (Locking)

封锁粒度与系统的并发度和并发控制的开销密切相关

- ❑ 封锁的粒度越大，数据库所能够封锁的数据单元就越少，并发度就越小，系统开销也越小；
- ❑ 封锁的粒度越小，并发度较高，但系统开销也就越大

封锁类型：

- ❑ 需要处理多个关系的大量元组的用户事务：以数据库为封锁单位（**页锁**）
- ❑ 需要处理大量元组的用户事务：以关系为封锁单位（**表锁**）
- ❑ 只处理少量元组的用户事务：以元组为封锁单位（**元组锁**）

# 表级锁 (table-level locking)

## □ 特点:

1. 开销小, 加锁快;
2. 不会出现死锁(因为MyISAM会一次性获得SQL所需的全部锁);
3. 锁定粒度大, 发生锁冲突的概率高, 并发度低
4. 使用表级锁的典型MySQL引擎: MyISAM

## □ 语法:

`LOCK TABLES tbl_name [AS alias] {READ [LOCAL] | WRITE}`

注: **LOCAL修饰符**: 允许在其他会话中, 对当前会话中获取READ锁的的表执行插入。对于InnoDB引擎, READ LOCAL与READ相同。



# 行级锁 (row-level locking)

## □ 特点:

1. 开销大, 加锁慢;
2. 会出现死锁;
3. 锁定粒度最小, 发生锁冲突的概率最低, 并发度高;
4. 使用表级锁的典型MySQL引擎: InnoDB

## □ 类型:

1. 共享锁: 共享锁又叫做读锁, 所有的事务只能对其进行读操作不能写操作, 加上共享锁后在事务结束之前其他事务只能再加共享锁, 不能增加其他任何类型的锁
2. 排它锁: 若某个事物对某一行加上了排他锁, 只能这个事务对其进行读写, 在此事务结束之前, 其他事务不能对其进行加任何锁, 其他进程可以读取, 但不能进行写操作

## □ 语法:

SELECT \* FROM table\_name WHERE .....LOCK IN SHARE MODE

SELECT \* FROM table\_name WHERE .....FOR UPDATE

# MYSQL分布式事务

1. 在数据库应用开发过程中，为了降低单点压力，通常会根据业务情况进行分表分库，将表分布在不同的库中（库可能分布在不同的机器上）。在这种场景下，事务的提交会变得相对复杂，因为多个节点（库）的存在，可能存在部分节点提交失败的情况，即事务的ACID特性需要在各个不同的数据库实例中保证
2. MySQL 从5.0.3开始支持XA分布式事务，且只有InnoDB存储引擎支持。MySQL Connector/J 从5.0.0版本之后开始直接提供对XA的支持。  
> show engines;

Engine	Support	Comment	Transactions	XA	Savepoints
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign key constraints	YES	YES	YES

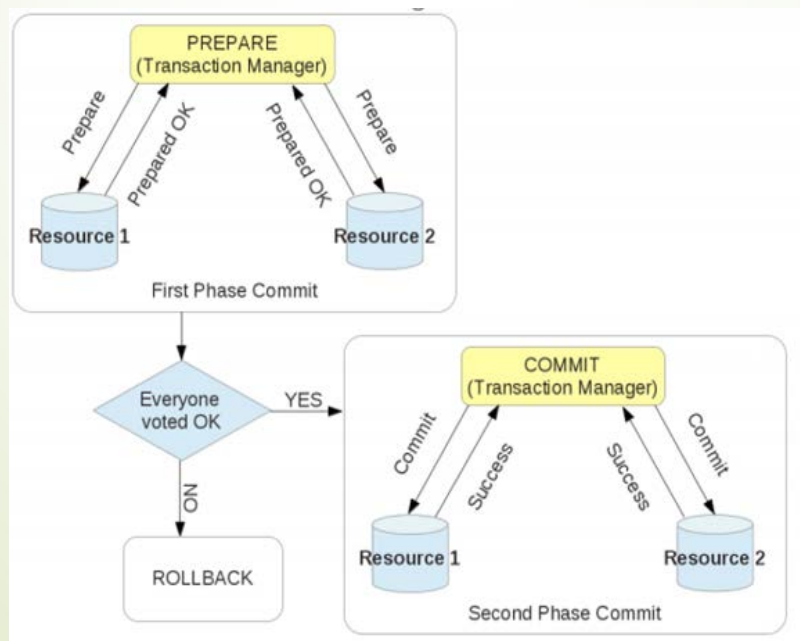
# MySQL支持的XA事务类型

1. 外部XA：用于跨多MySQL实例的分布式事务，需要应用层作为协调者，如通过JAVA调用JDBC实现事务管理，JAVA对应的逻辑就是协调者。应用层负责决定提交还是回滚，崩溃时的悬挂事务。
  - MySQL数据库外部XA可以用在分布式数据库代理层，实现对MySQL数据库的分布式事务支持，例如开源的代理工具：网易的DDB，淘宝的TDDL等等。
2. 内部XA：事务用于同一实例下跨多引擎事务，由Binlog作为协调者，如在一个存储引擎提交时，需要将提交信息写入二进制日志，这就是一个分布式内部XA事务，只不过二进制日志的参与者是MySQL本身。
  - Binlog作为内部XA的协调者，在binlog中出现的内部xid，在crash recover时，由binlog负责提交。

# 两阶段提交协议

分布式事务通常采用2PC协议（Two Phase Commitment Protocol）。该协议主要为了解决在分布式数据库场景下，所有节点间数据一致性的问题。分布式事务通过2PC协议将提交分成两个阶段：

- 准备（prepare）阶段：即所有的参与者准备执行事务并锁住需要的资源。参与者ready时，向transaction manager报告已准备就绪
- 提交阶段（commit）。当transaction manager确认所有参与者都ready后，向所有参与者发送commit命令

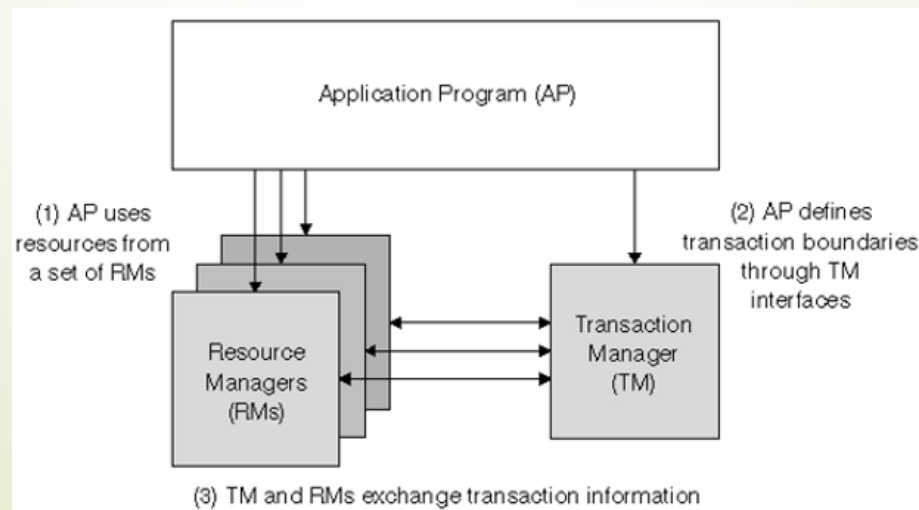


# XA事务处理架构

## 1. 包含两类主要角色：

- 资源管理器（Resource Manager, RM）：用来管理系统资源，数据库就是一种资源管理器。资源管理还具有管理事务提交或回滚的能力。
- 事务管理器（Transaction Manager, TM）：事务管理器是分布式事务的核心管理者。事务管理器与每个资源管理器（resource manager）进行通信，协调并完成事务的处理。事务的各个分支由唯一命名ID进行标识。

## 2. 在DTP（Distributed Transaction Processing）模型中，MySQL在执行分布式事务（外部XA）的时候，MySQL Server相当于XA事务资源管理器（RM），与MySQL连接的客户端（如Eclipse）相当于事务管理器。





# XA事务基本语法

1. XA {START|BEGIN} xid [JOIN|RESUME]: 启动事务xid (xid 必须是一个唯一值; MySQL暂不支持[JOIN|RESUME])
2. XA END xid [SUSPEND [FOR MIGRATE]]: 结束事务xid (MySQL暂不支持[SUSPEND [FOR MIGRATE]] 子句)
3. XA PREPARE xid: 准备、预提交事务xid
4. XA COMMIT xid [ONE PHASE]: 提交事务xid
5. XA ROLLBACK xid: 回滚事务xid
6. XA RECOVER: 查看处于PREPARE 阶段的所有事务

参考链接:

<https://dev.mysql.com/doc/refman/8.0/en/xa-statements.html>

<http://www.docjar.com/docs/api/com/mysql/jdbc/jdbc2/optional/MysqlXAConnection.html>

# 关于XID的说明

XID包含gtrid、bqual和formatID，其中：

1. gtrid：全局事务标识符(global transaction identifier)
2. bqual:分支限定符(branch qualifier)，如果没有提供bqual，那么默认值为空字符串
3. formatID：用于标记gtrid和bqual值的格式，无符号整数(unsigned integer)，如果没有提供formatID，其默认值为1

```
/*
 * Transaction branch identification: XID and NULLXID:
 */
#define XIDDATASIZE 128 /* size in bytes */
#define MAXGTRIDSIZ 64 /* maximum size in bytes of gtrid */
#define MAXBQUALSIZ 64 /* maximum size in bytes of bqual */
struct xid_t {
    long formatID; /* format identifier */
    long gtrid_length; /* value 1-64 */
    long bqual_length; /* value 1-64 */
    char data[XIDDATASIZE];
};
/*
 * A value of -1 in formatID means that the XID is null.
 */
typedef struct xid_t XID;
```

参考链接：

<https://dev.mysql.com/doc/refman/8.0/en/xa-statements.html>

# 事务状态转移过程

1. 使用XA START来启动一个XA事务（**ACTIVE**）
2. 对一个ACTIVE状态的 XA事务，可以执行构成事务的SQL语句，然后执行XA END语句。XA END把事务置为**IDLE**状态
3. 对一个IDLE 状态XA事务，可以执行XA PREPARE语句或XA COMMIT...ONE PHASE语句：
  - ① XA PREPARE把事务置为**PREPARED**状态。在此点上的XA RECOVER语句将在其输出中包括事务的xid值，因为XA RECOVER会列出处于PREPARED状态的所有XA事务。
  - ② XA COMMIT...ONE PHASE用于预备和提交事务。xid值将不会被XA RECOVER列出，因为事务已经被提交。
4. 对一个状态为PREPARED的XA事务，可以通过执行XA COMMIT语句来提交并终止事务，或者执行XA ROLLBACK来回滚并终止事务。

# 主要内容

---

1. MySQL事务编程的基本概念
2. 数据库事务编程（内容一、二、三）
3. 分布式数据库并发控制（内容四）

# 内容一：使用表级锁操作得到期望结果

新建 Connection1/Connection2，分别在两个连接中打开文件 conn1.sql/conn2.sql，对conn1/conn2.sql中的表级锁进行操作（包括修改锁的种类和控制范围（连接内/跨连接），选择合适的加锁/解锁位置，**不允许直接修改/删除非封锁操作**），并在conn2.sql中执行对应的procedure，达到期望的结果（结果以先执行conn1.sql中**对应问题的**SQL语句，再执行conn2.sql中对应语句为准）：

1. problem1()输出结果：{1,2,3}
2. problem2()输出结果：{1,3}
3. problem3()输出结果：{1,3,4}
4. problem4()输出结果：{1,2,3}
5. problem5()输出结果：{1,2}
6. problem6()输出结果：{2,3}

```
#problem 1
drop table if exists test1;
create table if not exists test1 (
  test1_id int,
  PRIMARY KEY(test1_id)
) engine=MyISAM;
select connection_id();
insert into test1 values(1);
lock tables test1 read;
insert into test1 values(2);
```

```
#problem 1
drop procedure if exists problem1;
delimiter //
create procedure problem1()
begin
  select connection_id();
  insert into test1 values(3);
  select * from test1;
end
//
call problem1;
```

69 14:02:10 call problem1;

Error Code: 2013 Lost connection to MySQL server during query



## 内容二：使用行级锁操作得到期望结果

新建 Connection3/Connection4，分别在两个连接中打开文件 conn3.sql/conn4.sql，对conn3.sql/conn4.sql中的表级锁进行操作（允许的操作：提交/回滚，修改行锁的类型，删除行锁。**不允许直接修改/删除非封锁操作**），并在conn4.sql中执行对应的procedure，达到期望的结果（结果以先执行conn3.sql中**对应问题的SQL语句**，再执行conn4.sql中对应语句为准）：

1. problem1()输出结果：{1}
2. problem2()输出结果：{(2,3)}
3. problem3()输出结果：{(1,100)}
4. problem4()输出结果：{(3,3,5)}
5. problem5()输出结果：{}(空集)
6. problem6()输出结果：{1}

```
#problem 4
drop table if exists test4;
create table if not exists test4 (
  test4_id int,
  val int,
  col int,
  PRIMARY KEY(test4_id)
) engine=InnoDB;
select connection_id();
insert into test4 values(1,2,3);
insert into test4 values(2,3,4);
insert into test4 values(3,3,5);
insert into test4 values(4,5,6);
select * from test4 where test4_id = 3 lock in share mode;
update test4 set col= 10 where test4_id = 3;
```

```
#problem 4
drop procedure if exists problem4;
delimiter //
create procedure problem4()
begin
  select connection_id();
  select * from test4 where test4_id = 3 lock in share mode;
end
//
delimiter ;
call problem4;
```

# 内容三：数据库事务编程（1）

## 问题背景：

某在线购物网站数据库中的主要关系如下（与第二次上机相同）

- ① 用户（用户ID（整形），用户名（长度不超过30的字符串），用户密码（长度不超过30的字符串，需包含大小写字母和数字），用户性别（男/女，可用0和1两个整数表示），收货地址（长度不超过100的字符串））
- ② 商品（商品ID（整形），商品名（长度不超过30的字符串），商品价格（非负实数），剩余库存（非负整数））
- ③ 订单（订单ID（整形），购物车ID（整形），用户ID（整形），订单编号（长度为16的数字串），订单金额（非负实数），下单时间（DATETIME））
- ④ 订单详情（订单详情ID（整形），订单ID（整形），商品ID（整形），商品数量（非负整数），商品价格（非负实数））
- ⑤ 购物车（购物车ID（整形），用户ID（整形），商品总价（非负实数））
- ⑥ 购物车详情（购物车详情ID（整形），购物车ID（整形），商品ID（整形），商品数量（非负整数），添加时间（DATETIME），商品价格（非负实数））

# 内容三：数据库事务编程（2）

场景一：

用户A：查询所有价格高于100元的商品名

用户B：将所有商品的价格设置九折优惠

场景二：商品ID为1,2,3的剩余库存分别为50,100,200件（初始状态）

用户A：订单中包含商品1三十件，商品2五十件

用户B：订单中包含商品2七十件，商品3八十件

作业要求：

1. 在MySQL默认的事务隔离级别下，使用数据库事务编程接口实现上述两位用户正确的业务逻辑。
2. 分析在该隔离级别下是否存在脏读/幻读/不可重复读的问题，若有，请设计场景并验证

# 内容三：数据库事务编程（3）

## 附加题：

1. 将事务隔离级别修改为read uncommitted，在该隔离级别下，设计脏读/幻读/不可重复读三类场景并验证
2. 考虑不同MySQL引擎（MyISAM和InnoDB）支持的封锁粒度，请分析出现活锁或死锁对应的隔离级别和封锁粒度，在上述在线购物系统中设计对应场景进行验证，并给出解决思路。

# 主要内容

---

1. MySQL事务编程的基本概念
2. 数据库事务编程（内容一、二、三）
3. 分布式数据库并发控制（内容四）



# 内容四：分布式事务程序设计

## 问题场景和作业要求：

1. 在分布式数据库场景下，用户、订单和订单详情三个关系保存在非本地数据库中，其他关系保存在本地数据库中。
2. 远程数据库地址：202.120.38.131
3. 用户名：学号；密码：123456；默认schema名称：db+学号。如db517021910000
4. 通过JDBC的XA事务接口成功连接数据库进行操作
5. 使用Main2.java的两阶段提交（2 phase commit）框架，实现分布式场景下，实现上述两个场景中的业务逻辑。

# 提交内容与截止时间

1. 内容一、内容二：提交修改后的conn1/2/3/4.sql
2. 内容三：业务逻辑实现文件（Main1.java）。
3. 内容四：业务逻辑实现文件（Main2.java）
4. 分析报告（ result.pdf ）：描述内容一、二的具体操作和原因；针对内容三中事务隔离级别的分析；针对附加题，除构造具体场景，还需给出对应解决方法
5. 注1：基于JDBC模拟多用户并发访问例子见Main1.java
6. 注2：XA事务处理架构见Main2.java

- 所有输出文件压缩（姓名+学号.zip）上传至ftp://public.sjtu.edu.cn 中 upload/DS107/exe3目录下（账号：nirver1994，密码:public），上传截止时间：12月28日24点
- 超过截止时间提交按50%的成绩计算