第26章：事务及隔离机制

## 一、事务的定义

事务是指一组**相互依赖的操作单元的集合**，用来保证对数据库的正确修改，保持数据的完整性，如果一个事务的某个单元操作失败，将取消本次事务的全部操作。

事务具备的特征（**ACID**）：

1. **原子性：**每个事务都必须被看作是一个不可分割的单元；
2. **一致性：**不管事务是完全成功完成还是中途失败，当事务使系统处于一致的状态时存在一致性。例如从雇员数据库中删除了一个雇员，则所有和该雇员相关的数据，包括工资记录、职务变动记录也要被删除。

*（在MySQL中，一致性主要由MySQL的日志机制处理，它记录了数据库的所有变化，为事务恢复提供了跟踪记录。如果系统在事务处理中间发生错误，MySQL恢复过程将使用这些日志来发现事务是否已经完全成功地执行，是否需要返回。因而一致性属性保证了数据库从不返回一个未处理完的事务。）*

1. **隔离性**：每个事务在它自己的空间发生，和其他发生在系统中的事务隔离，而且事务的结果只有在它完全被执行时才能看到。隔离性原则**保证某个特定事务在完全完成之前，其结果是看不见的。**

*（当系统支持多个同时存在的用户和连接时，这就尤为重要。如果系统不遵循这个基本规则，就可能导致大量数据的破坏，如每个事务的各自空间的完整性很快地被其他冲突事务所侵犯。获得绝对隔离性的唯一方法是保证在任意时刻只能有一个用户访问数据库。当处理像MySQL这样多用户的RDBMS时，这不是一个实际的解决方法。但是，大多数事务系统使用****页级锁定****或****行级锁定****隔离不同事务之间的变化，这是要以降低性能为代价的。）*

1. **持久性**：即使系统崩溃，一个提交的事务仍然存在。当一个事务完成，数据库的日志已经被更新时，持久性就开始发生作用。

*（大多数RDBMS产品通过保存所有行为的日志来保证数据的持久性，这些行为是指在数据库中以任何方法更改数据。如果系统崩溃或者数据存储介质被破坏，通过使用日志，系统能够恢复在重启前进行的最后一次成功的更新，反映了在崩溃时处于过程的事务的变化。MySQL通过保存一条记录事务过程中系统变化的二进制事务日志文件来实现持久性。如果遇到硬件破坏或者突然的系统关机，在系统重启时，通过使用最后的备份和日志就可以很容易地恢复丢失的数据。默认情况下，InnDB表是100%持久的。MyISAM表提供部分持久性，所有在最后一个FLUSH TABLES命令前进行的变化都能保证被存盘。）*

## 二、事务处理

1. **关闭自动提交功能**

在MySQL中，当一个会话开始时，**系统变量AUTOCOMMIT值为1，即自动提交功能是打开的**，当用户每执行一条SQL语句后，该语句对数据库的修改就立即被提交成为持久性修改保存到磁盘上，一个事务也就结束了。

测试效果：

（1）自动提交功能关闭前：

|  |
| --- |
| update vendors set vend\_name=’New’ where vend\_id=1001;  select \* from vendors; |

（2）自动提交功能关闭后：

|  |
| --- |
| SET @@AUTOCOMMIT=0; #关闭自动提交  update vendors set vend\_name=’New’ where vend\_id=1001;  select \* from vendors; |

**注**：自动提交关闭后，从执行结果中发现对应行的数据已经被更新，但由于没有自动提交，结果数据并没有被持久化处理，会出现其他终端查询时数据并没有被修改。

1. **开始事务**

**（1）隐式开启：**

当一个应用程序的第一条SQL语句或者在COMMIT或ROLLBACK语句（后面介绍）后的第一条SQL执行后，一个新的事务也就开始了。

**（2）显式开启：**

|  |
| --- |
| START TRANSACTION | BEGIN [WORK] # Start Transaction 更常用 |

**应用举例**：从订单表orders中完全删除订单20009。

|  |
| --- |
| START TRANSACTION; # 事务开始  Delete from orderitems Where order\_num = 20009;  Delete from orders Where order\_num = 20009;  COMMIT; #提交并关闭事务 |

**注**：当COMMIT|ROOLBACK语句执行后，隐含了事务的关闭

1. **结束事务**

**COMMIT**语句是提交语句，它使得自从事务开始以来所执行的所有数据修改成为数据库的永久部分，也标志一个事务的结束。

|  |
| --- |
| COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]; |

说明：

1. 可选的AND CHAIN子句会**在当前事务结束时，立刻启动一个新事务**，并且新事务与刚结束的事务有相同的隔离等级。
2. RELEASE子句**在终止了当前事务后，会让服务器断开与当前客户端的连接**。
3. 包含NO关键词可以**抑制**CHAIN或RELEASE完成。

**注意：**

MySQL使用的是平面事务模型，因此嵌套的事务是不允许的。在第一个事务里使用START TRANSACTION命令后，当第二个事务开始时，自动地提交第一个事务。

**所有的DDL语句都不能回滚**，即下面的这些MySQL语句运行时都会隐式地执行一个COMMIT命令：

|  |
| --- |
| * DROP DATABASE / DROP TABLE * CREATE INDEX / DROP INDEX * ALTER TABLE / RENAME TABLE * LOCK TABLES / UNLOCK TABLES * SET @@AUTOCOMMIT={ 1 | 0 } |

1. **撤销事务**

**ROLLBACK**语句是撤销语句，它撤销事务所做的修改，并**结束当前这个事务**。

|  |
| --- |
| ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE] |

**应用举例**：撤销ordertotals表记录清空的操作

|  |
| --- |
| SELECT \* FROM ordertotals; #查询显示所有表记录  START TRANSACTION;  DELETE FROM ordertotals;  SELECT \* FROM ordertotals;  ROLLBACK; # 回滚事务  SELECT \* FROM ordertotals; |

1. **使用保留点回滚**

除了撤销整个事务，用户还可以使用**ROLLBACK TO**语句使事务**回滚到某个点**，在这之前需要使用**SAVEPOINT**语句来设置一个保存点。

|  |
| --- |
| SAVEPOINT identifier #每个保留点都取标识它的唯一名字，以便在回退时，MySQL知道要 回退到何处  ROLLBACK [WORK] TO SAVEPOINT identifier #当事务回滚到某个保存点后，在该保存点之后设置的保存点将被删除 |

**应用举例：**新增订单到订单orders表，再添加详情到orderitems表

|  |
| --- |
| START TRANSACTION;  SAVEPOINT beforeInsertOrder;  Insert into orders values(20009, now(), 10001);  Select \* from orders;  SAVEPOINT beforeInsertOrderItem;  Insert into orderitems values(20009, 1, FB, 1, 10);  Select \* from orderitems;  ROLLBACK to beforeInsertOrderItem;  Select \* from orderitems;  Insert into orderitems values(20009, 1, FC, 1, 2.5);  COMMIT; |

1. **删除保留点**

**RELEASE SAVEPOINT**语句会从当前事务的一组保存点中删除已命名的保存点。不出现提交或回滚。如果保存点不存在，会出现错误。

|  |
| --- |
| RELEASE SAVEPOINT identifier |

## 三、事务隔离机制

基于ANSI/ISO SQL规范，MySQL提供了4种隔离级（**只有支持事务的存储引擎(InnoDB)才可以定义一个隔离级**。）：

1. **序列化（SERIALIZABLE）**：如果隔离级为序列化，用户之间通过一个接一个顺序地执行当前的事务（强制串行），提供了事务之间**最大限度的隔离**
2. **可重复读（REPEATABLE READ）**：在这一级上，事务不会被看成是一个序列。不过，当前在执行事务的变化仍然不能看到，也就是说，如果用户在同一个事务中执行同条SELECT语句数次，结果总是相同的，但对不同事务可能出现**幻读**，（如：A用户执行表T的数据修改，B用户同时往表T中插入了新数据，则在A用户修改结束后发现有一条记录没有做改动，就好像出现幻觉一样）。该级别是**MySQL默认隔离级别。**
3. **提交读（READ COMMITTED）**：在这一级上，不仅处于这一级的事务可以看到其他事务添加的新记录，而且其他事务对现存记录做出的修改一旦被提交，也可以看到。也就是说，这意味着在事务处理期间，如果其他事务修改了相应的表，那么同一个事务的多个SELECT语句可能返回不同的结果（**不可重复读**）。
4. **未提交读（READ UNCOMMITTED）**：处于这个隔离级的事务可以读到其他事务还没有提交的数据，如果这个事务使用其他事务不提交的变化作为计算的基础，然后那些未提交的变化被它们的父事务撤销，这就导致了大量的数据变化。（容易出现**脏读**）

### 1．查看隔离级别

系统变量TX\_ISOLATION中存储了事务的隔离级，可以使用SELECT随时获得当前隔离级的值。

|  |
| --- |
| SELECT @@TX\_ISOLATION;  +---------------------------+  | @@TX\_isolation |  +---------------------------+  | REPEATABLE-READ |  +----------------------------+  1 row in set |

**注：MySQL的默认隔离级别为可重复读**

### 2．修改隔离级别

定义隔离级可以使用SET TRANSACTION语句。

|  |
| --- |
| SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL  SERIALIZABLE  | REPEATABLE READ  | READ COMMITTED  | READ UNCOMMITTED ; |

**注：**

1. GLOBAL表示隔离级别适用于所有SQL用户
2. SESSION表示隔离级别仅适用于当前回话和连接

**3．隔离机制实验区分**

建立两个用户连接，模拟事务A、事务B，对比4种隔离机制。

（1）读未提交——脏读

|  |  |
| --- | --- |
| # 设置隔离机制为 读未提交  SET GLOBAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ; | |
| # 事务A：读取  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  #读到了事务B未提交的数据，事务B执行回滚后，数据无效  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num; | #事务B：修改订单详情表内容后回滚  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  #对数据做修改，但未提交  update orderitems set quantity = 1 where order\_num=20005 and order\_item=1;  # 事务回滚  ROLLBACK; |

（2）读已提交——不可重复读

|  |  |
| --- | --- |
| # 设置隔离机制为 读已提交  SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED ; | |
| # 事务A：读取  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A再执行查询，没有脏读  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务B提交后，A读到B提交的数据，前后两次读取结果不一致，出现不可重复读现象  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num; | #事务B  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  #对数据做修改，但未提交  update orderitems set quantity = 1 where order\_num=20005 and order\_item=1;  # 事务提交  commit; |

（3）可重复读——幻读

|  |  |
| --- | --- |
| # 设置隔离机制为 可重复读  SET GLOBAL TRANSACTION ISOLATION LEVEL REPEATABLE READ; | |
| # 事务A：  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A再执行查询，没有脏读  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A修改order\_item=2的记录数据并查询，可修改  update orderitems set quantity = 10 where order\_num=20005 and order\_item=2;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A在事务B提交后查询，数据没有变更，解决不可重复读  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A提交后，执行查询，出现幻读  Commit;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num; | #事务B  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  #对数据先做修改，但未提交  update orderitems set quantity = 1 where order\_num=20005 and order\_item=1;  # 事务B在事务A修改后，查询，没有脏读  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务B提交  commit; |

（4）序列化

|  |  |
| --- | --- |
| # 设置隔离机制为 序列化  SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE; | |
| # 事务A：  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A执行查询，没有脏读  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A修改order\_item=2的记录数据并查询，不能修改，等待  update orderitems set quantity = 10 where order\_num=20005 and order\_item=2;  # 事务A的修改在事务B提交后才执行，强制串行  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务A提交  Commit; | #事务B  Start transaction;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  #事务B对数据先做修改，但未提交  update orderitems set quantity = 3 where order\_num=20005 and order\_item=1;  select order\_num, sum(item\_price\*quantity) from orderitems group by order\_num;  # 事务B提交  commit; |

## 四、测试使用事务

**测试1：**创建存储过程实现用户订单的新增，在存储过程中创建事务。

要求：（1）当新用户提交订单时，判断用户是否在customers表

（2）如果用户存在，添加该用户的订单记录到orders表中

（3）如果用户不存在，添加一条用户记录到customers表中，再同时添加订单记录到orders表中

（4）添加订单的每项物品到orderitems表中

（3）如果orderitems表新增出现故障，则回退之前所有添加的orders行和orderitems行

（4）否则，提交事务，完成订单新增

**测试2:** 创建存储过程实现销量管理，在存储过程中创建事务 （可选）

要求：（1）设定物品库存数量上限值为20；

（2）实现在保存订单记录时，自动更新/判断库存数量；

（3）当库存数量小于0时，回滚事务；

（4）否则，提交事务，完成订单销售。

## 附样例库关系图：

