MySQL中的事务及其ACID四个特性

# MySQL事务(Transaction)

当多个用户访问同一份数据时，一个用户在更改数据的过程中可能有其他用户同时发起更改请求，为保证数据的更新从一个一致性状态变更为另外一个一致性状态，这时有必要引入事务的概念。

MySQL提供了多种存储引擎支持事务，支持事务的引擎有InnoDB和BDB。InnoDB存储引擎事务主要通过UNDO日志和REDO日志实现，MyISAM和MEMORY存储引擎则不支持事务。

REDO日志与UNDO日志。UNDO日志主要用于**事务异常时的数据回滚**。

# 事务的四个特性ACID

事务的四个特性ACID：原子性(**A**tomicity)、一致性(**C**onsistency)、隔离性(**I**solation)、持久性(**D**urability)。

**atomicity**英 [ˌætəmˈɪsɪti] n．原子数;原子价;原子化合力;原子力

## 原子性(Atomicity):事务中所有的操作视为一个原子单元，即对于事务所进行的数据修改等操作只能是完全提交或者完全回滚。

## 一致性(Consistency)：事务在完成时，必须使所有的数据从一种一致性状态变更为另外一种一致性状态，所有的变更都必须应用于事务的修改，以确保数据的完整性。

## 隔离性(Isolation): 一个事务中的操作语句所做的修改必须与其他事务所做的修改相隔离。

在进行事务查看数据时数据所处的状态，要么是被另一并发事务修改之前的状态，要么是被另一并发事务修改之后的状态，即当前事务不会查看由另一个并发事务正在修改的数据。这种特性通过**锁机制**实现。

## 持久性(Durability)：事务完成之后，所做的修改对数据的影响是永久的，即使系统重启或者出现系统故障数据仍可以恢复。

# MySQL的事务控制语句

MySQL使用**BEGIN**开始事务，使用**COMMIT**结束事务，中间使用ROLLBACK回滚事务。

在默认情况下，MySQL的**单条语句**都是**默认自动提交**的（即autocommit=1），执行单条语句是**无法回滚**的。如果需要对**某些语句**进行控制，使用BEGIN或者START TRANSACTION开启一个事务，在事务内部可以利用**rollback**进行回滚。这样事务结束之后，就进入了默认自动提交的方式。当然也可以修改**autocommit**的值，set autocommit = 0; 这样执行单条语句就不会自动提交了，需要提交时利用**commit**手动提交，利用**rollback**回滚到上次提交之后的位置。

简单示例1：

**BEGIN [work]; #开始事务 或者 start transaction;**

use db1; select \* from user;

delete from user where id = 1; select \* from user;

ROLLBACK; **#回滚事务#**

select \* from user;

**COMMIT [work]; #提交事务（结束事务）**

select \* from user;

示例2：

set autocommit = 0; #取消自动提交

delete from user where id = 1;

**commit; #id=1的数据记录无法回滚**

delete from user where id = 2;

delete from user where id = 3;

delete from user where id = 4;

**rollback**; #回滚到上次提交的位置，即上面三条删除语句都被回滚

commit;

# MySQL的事务隔离级别

SQL标准定义了**4种隔离级别**，指定了**事务中哪些数据改变其他事务可见，哪些数据改变其他事务不可见**。低级别的隔离级别可以支持更高的并发处理，同时占用的系统资源更少。

## SQL的四种隔离级别：

**未提交可读**READ UNCOMMITTED(read uncommited)

**提交可读**READ COMMITTED(read committed)

**可重复读** REPEATABLE READ(repeatable read)

**可串行化** SERIALIZABLE(serializable)

事务隔离级别设置方式：

**SET GLOBAL TRANSACTION ISOLATION LEVEL 隔离级别**

**(set global transaction isolation level隔离级别)**

## 未提交可读（读取未提交内容）-脏读

**脏读（Dirty Read）**：读取了未提交的数据。

示例：在隔离级别为未提交可读级别下，同时开启事务A和B，事务B更新但未提交之前，事务A读取到了更新后的数据，但是事务B回滚，此时事务A就出现了**脏读**的现象。

BEGIN; #开始事务A BEGIN;#开始事务B

select score from user where id =2; #查询结果为100

update user set score =99 where id =2;#更改为99

select score from user where id =2; #查询结果为99

rollback; #回滚

commit; #提交

select score from user where id =2; #查询结果为100

commit;

在事务A中，三次查询结果不一致，因此中间读取到了事务B未提交的数据，出现了**脏读（Dirty Read）**现象。

## 提交可读（读取提交后的内容）-不可重复读

大多数数据库系统的默认隔离级别就是**提交可读**级别，但是MySQL数据库的默认级别是**可重复读**级别。

**提交可读**级别：一个事务从开始到提交前所做的任何改变都是**不可见的**，**事务只能看见其他已经提交过的事务所做的改变**。

示例：在隔离级别为提交可读级别下，先开启事务A，查询数据，然后开启事务B，事务B更新并提交数据后，事务A再次读取数据，此时事务A查询出现了不同的查询结果，这就是不可重复读现象。

BEGIN; #开始事务A BEGIN;#开始事务B

select score from user where id =2; #查询结果为100

update user set score =99 where id =2;#更改为99

select score from user where id =2; #查询结果仍为100，未提交的数据读取不到

commit; #提交

select score from user where id =2; #查询结果为99

commit; #事务A提交

在事务A中，后两次查询结果不一致，因为两次查询之间，事务B更新并提交了数据，出现了**不可重复读**现象。

## 可重读(REPEATABLE READ)

MySQL数据库的默认事务隔离级别是可重读级别，可以确保同一事务的多个实例在并发读取数据时，会看到同样的数据记录。

导致问题：**幻读(Phantom Read)**。

示例：

事务A对表格的所有数据做了修改，同时，事务B想表格中添加了一条新的数据记录。之后，发现事务A还有没有做修改的数据行，这就是**幻读**现象。

BEGIN; #开始事务A BEGIN;#开始事务B

update user set score= 100;#修改所有行score都为60分

**insert into user(score,sex,name) values(88,’女’,’刘媛媛’);**#插入新的数据

commit; #事务A提交 commit; #事务B提交

select \* from user;#发现有人的分数不是100，这就是**幻读**现象。

## 

# InnoDB锁机制