事务概述

2019年3月5日 9:47

1. 事务概述

事务是指逻辑上的一组操作，这组操作要么一起成功，要么一起失败。

```
例如：A——B转帐，对应于如下两条sql语句
update account set money=money-100 where name= ‘a’；
update account set money=money+100 where name= ‘b’；
```

数据库默认有事务的能力，默认情况下一条语句一个事务。

也可以手动的控制事务，实现多条sql语句在一个事务中一起成功或一起失败。

2. 手动管理事务 - sql方式

sql控制事务

start transaction;	开启事务，则这条语句之后的若干条sql都会处在一个事务中
commit;	提交事务，此命令会完成这个事务，使这个事务中的所有的sql语句同时产生效果
rollback;	回滚事务，此命令会取消这个事务，取消这个事务中的所有的sql语句产生的效果

案例：通过sql手动控制事务实现转账

```
例如：A——B转帐，对应于如下两条sql语句
start transaction;
update account set money=money-100 where name= ‘a’；
update account set money=money+100 where name= ‘b’；
commit; 或 rollback;
```

3. 手动管理事务 - JDBC方式

jdbc控制事务

conn.setAutoCommit(false)	jdbc操作数据库时，默认开启了自动提交，即每条sql执行后都立即自动进行提交操作，所以默认情况下，jdbc操作数据库一条语句一个事务。可以手动将自动提交关闭，则在此conn对象上执行的sql将不会自动提交事务，在需要时可以手动进行提交，从而实现手动管理事务。
conn.commit()	手动提交事务
conn.rollback()	手动回滚事务
Savepoint sp = conn.setSavepoint()	设置保存点
conn.rollback(sp)	回滚到保存点。 注意，回滚到保存点后，保存点之前的事务仍未提交，如果需要事务生效，仍需手动提交事务。

案例：通过jdbc手动控制事务实现转账

```
package cn.tedu.trans;

import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Demo01 {
	public static void main(String[] args) {
		Connection conn = null;
		PreparedStatement ps = null;
		try {
			Class.forName("com.mysql.jdbc.Driver");
			conn = DriverManager.getConnection("jdbc:mysql:///day16","root","root");

			//--关闭自动提交，手动管理事务
			conn.setAutoCommit(false);

			//--a扣100
			ps = conn.prepareStatement("update account set money = money - ? where name = ?");
			ps.setDouble(1, 100.0);
			ps.setString(2, "a");
			ps.executeUpdate();

			int i = 1/0;

			//--b加100
			ps = conn.prepareStatement("update account set money = money + ? where name = ?");
			ps.setDouble(1, 100.0);
			ps.setString(2, "b");
			ps.executeUpdate();

			//--提交事务
			conn.commit();

			} catch (Exception e) {
			//--回滚事务
			if(conn!=null){
			try {
				conn.rollback();
			} catch (SQLException e1) {
				e1.printStackTrace();
				throw new RuntimeException(e1);
			}
			}
			e.printStackTrace();
			throw new RuntimeException(e);
			} finally {
			if(ps!=null){
			try {
				ps.close();
			} catch (SQLException e) {
```

```
                e.printStackTrace();
            } finally {
                ps = null;
            }
            }
            if(conn!=null){
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                conn = null;
            }
            }
        }
    }
}
```

案例：通过jdbc手动控制事务实现转账 - 使用保存点

```
package cn.tedu.trans;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Savepoint;

public class Demo02 {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement ps = null;
        Savepoint sp = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql:///day16","root","root");

            //--关闭自动提交，手动管理事务
            conn.setAutoCommit(false);

            //--a扣100
            ps = conn.prepareStatement("update account set money = money - ? where name = ?");
            ps.setDouble(1, 100.0);
            ps.setString(2, "a");
            ps.executeUpdate();
            //--b加100
            ps = conn.prepareStatement("update account set money = money + ? where name = ?");
            ps.setDouble(1, 100.0);
            ps.setString(2, "b");
            ps.executeUpdate();
```

```
            //--设置保存点
            sp = conn.setSavepoint();

            //--a扣100
            ps = conn.prepareStatement("update account set money = money - ? where name = ?");
            ps.setDouble(1, 100.0);
            ps.setString(2, "a");
            ps.executeUpdate();

            int i = 1/0;

            //--b加100
            ps = conn.prepareStatement("update account set money = money + ? where name = ?");
            ps.setDouble(1, 100.0);
            ps.setString(2, "b");
            ps.executeUpdate();

            //--提交事务
            conn.commit();
        } catch (Exception e) {
            //--回滚事务
            if(conn!=null){
                try {
                    if(sp!=null){
                        //--经过了保存点，则回滚到保存点
                        conn.rollback(sp);
                        //--提交事务
                        conn.commit();
                    }else{
                        //--未经过保存点，则全部回滚
                        conn.rollback();
                    }
                } catch (SQLException e1) {
                    e1.printStackTrace();
                    throw new RuntimeException(e1);
                }
            }
            e.printStackTrace();
            throw new RuntimeException(e);
        } finally {
            if(ps!=null){
                try {
                    ps.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                } finally {
                    ps = null;
                }
```

```
                }
                if(conn!=null){
                        try {
                                conn.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        } finally {
                                conn = null;
                        }
                }
        }
    }
}
```

事务的四大特性

2019年3月5日 14:08

1. 事务的四大特性 (ACID)

a. 原子性(Atomicity)

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

b. 一致性(Consistency)

事务前后数据的完整性必须保持一致。

c. 隔离性(Isolation)

事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离。

d. 持久性(Durability)

持久性是指一个事务一旦被提交，它对数据库中数据的改变就真实的发生了，接下来无论做任何操作哪怕是数据库故障也无法再撤销这个事务。

2. 隔离性

数据库的其他三大特性数据库可以帮我们保证，而隔离性我们需要再讨论。

如果我们是数据库的设计者，该如何考虑设计数据库保证数据库的隔离性呢？

我们知道数据库的隔离性问题本质上就是多线程并发安全性问题。

可以用锁来解决多线成并发安全问题,但是如果用了锁,必然会造成程序的性能大大的下降.

对于数据库这种高并发要求的程序来说这是不可接受的.

所以我们需要对隔离性问题进行进一步的分析

经过分析：

1) 如果两个线程并发查询,必然没有问题,不需要隔离
2) 如果两个线程并发修改,必然产生多线程并发安全问题,必须隔离开
3) 如果一个线程修改 一个线程查询 可能会产生 脏读 不可重复读 虚读/幻读问题 而这些问题有些场景下是问题 有些场景下不是问题。而想要防止的问题越多，对数据库性能的影响就越大，所以到底要将数据库设置到何种状态来防止哪类问题不应该在数据库中写死，而应该提供相应的选项让数据库的使用者根据不同的场景灵活的选择 -- 本质上是对 数据的可靠性 和 数据库的效率 之间的选择。

所以最终数据库设计者在设计数据库时，对于并发的查询没有做隔离，对于并发的修

改做了严格的隔离，对于并发的读和写提供了相应的选项允许数据库的使用者选择，来根据需求防止不同的问题，这些选项就称之为数据库的隔离级别。

3. 在一个线程修改一个线程查询的情况下可能产生的问题

a. 脏读：一个事务读取到另一个事务未提交的数据，造成数据混乱产生的问题

```
小a向小b买一双鞋
----------------------
    a 1000
    b 1000
----------------------
a:
    start transaction;
    update account set money = money - 100 where name  = 'a';
    update account set money = money + 100 where name  = 'b';

    ---------------------------------------
        b:
            start transaction;
            select * from account;
            commit;
            ----------------------
                a 900
                b 1100
            ----------------------
    ---------------------------------------

    rollback;

    ---------------------------------------
        b:
            start transaction;
            select * from account;
            commit;
            ----------------------
                a 1000
                b 1000
            ----------------------
    ---------------------------------------
```

b. 不可重复读：一个事务可以读取到另一个事务已经提交的数据,造成同一个事务中对同一数据先后读取不一致造成问题。

```
小b统计银行账户信息，小a取款，造成统计结果出错
---------------
a 1000 1000 1000
---------------
b:
    start transaction;
```

```
select 活期 from account where name = 'a';   ---  活期存款：1000
select 定期 from account where name = 'a';   ---  定期存款：1000
select 固定 from account where name = 'a';   ---  固定资产：1000
------------
a:
    start transaction;
    update account set 活期 = 活期 - 100 where name = 'a';
    commit;
    ---------------
        a 900 1000 1000
    ---------------
------------
select 活期 + 定期 + 固定 from account where name = 'a'; --- 总资产 2900
commit;
```

c. 虚读/幻读：一个事务读取到另一个事务已经提交的数据。只在读写整表数据时产生。虚读/幻读并不是每次都会产生，有可能会发生，也有可能不发生。

```
小c统计银行总体账户信息，小d新建账户，造成统计结果出错
-----------------
a 1000
b 2000
-----------------
c :
    start transaction;
    select sum(money) from account; --- 总存款：3000
    select count(1) from account; --- 总账户数：2
        --------------------
        d:
            start transaction;
            insert into account values ('d',3000);
            commit;
            -----------------
                a 1000
                b 2000
                d 3000
            -----------------
        --------------------
    select avg(money) from account; --- 平均每账户存款 2000
    commit;
```

4. 数据库的隔离级别

隔离级别是基于客户端来讨论的，不同的客户端在和服务器交互式可以有不同的隔离级别，客户端处在什么隔离级别就具有什么隔离级别的问题。

a. mysql数据库的隔离级别一共有四种

read uncommitted	不做任何隔离。可能产生脏读 不可重复读 虚读/幻读问题.

读未提交	性能最好。
read committed 读已提交	一个事务可以读取到另一个事务已经提交的数据。可以防止脏读，但可能存在不可重复读 虚读/幻读 问题。 性能较好。
repeatable read 可重复读取	在查询整表数据时，一个事务可以读取到另一个事务已经提交的数据。可以防止脏读 不可重复读问题，但可能存在虚读/幻读问题。 mysql默认采用此隔离级别。 性能一般。
serializable 序列化	通过锁进行严格隔离，对同一个数据的访问要串行化进行。可以防止脏读 不可重复读 虚读/幻读 问题。但数据库处于串行化状态，效率极其低下。 性能最差。

b. 选择隔离级别的原则

选择不同的隔离级别，就可以防止在并发读写的过程中的不同的隔离性问题，隔离级别设置的越严格，防止的问题就越多但性能就越低，隔离级别设置的越宽松，性能就越好但可能产生的隔离性问题就越多。

数据库使用者应该根据自己的需求选择一个合理的隔离级别 -- 选择一个能够防止想要防止的问题的情况下性能尽量好的隔离级别。

从可靠性角度：

serializable > repeatable read -> read committed -> read uncommitted

从性能角度：

read uncommitted -> read committed -> repeatable read ->serializable

在真正的开发中 脏读问题太严重，所以read uncommitted很少用。serializable性能太差，也很少用。所以只需根据是否需要防止不可重复读，在read committed 和 repeatable read之间选择一个即可。在实际开发中 repeatable read用的跟多一些。

c. 查询当前客户端隔离级别的命令

```
select @@tx_isolation;
```

d. 修改隔离的命令

```
set [session/global] transaction isolation level 隔离级别名称;
可以通过选择[session]来指定修改的是当前客户端的隔离级别，mysql服务器默认的隔离级别不变
可以通过选择[global]来指定修改的是mysql服务器默认的隔离级别，当前客户端隔离级别不变，默认不写就是[global]
```

5. 调整数据库隔离级别，演示脏读 不可重复读 虚读/幻读

略 - 自己做实验

6. 数据库中的锁机制

a. 共享锁、排他锁

数据库也是用锁来保证数据隔离的，但是为了数据库的锁设计的更加精细。体现在数据库中的锁分为 共享锁 和 排他锁。

共享锁和共享锁可以共存，共享锁和排他锁不能共存。
排他锁和任何锁都不能共存。

在非Serializable隔离级别下，查询不加锁。
在Serializable隔离级别下，查询加共享锁。
任意隔离级别下增删改加排他锁。

正是利用了这种锁机制，数据库保证了并发的读不隔离，并发的写一定隔离，并发的读写在某一方或多方为Serializable的级别时，实现串行化，保证完全可靠。

**死锁：

多个客户端都是Serializable的级别下，先查询再修改，可能会进入互相等待状态，其实就是发生了死锁，mysql会检测到死锁，自动退出一方以打断死锁。

b. 行级锁、表级锁：

数据库的锁根据锁定的粒度可以分为行级锁和表级锁。行级锁锁一行。表级锁锁整表。数据库自动根据操作的数据决定加哪种粒度的锁。

作业

2019年3月5日 17:03

复习事务相关概念，整理事务相关知识点

练习事务相关案例

事务隔离级
别练习笔记