# 22 养成好的事务习惯

更新时间：2019-09-26 09:37:42

> 衡量一个人的真正品格，是看他在知道没人看见的时候干些什么。
>
> ——孟德斯鸠

有时好的事务习惯也影响着业务访问速度。

我们先来看看一些不好的事务习惯：

## 1 不好的事务习惯

### 1.1 在循环中提交

在大多数情况下，MySQL 都是开启自动提交的，如果遇到循环执行 SQL，则相当于每个循环中都会进行一次提交，实际这算一个不好的事务习惯了。下面我创建一张测试表，并定义两个循环写入数据的存储过程：一个是自动提交，另一个是在循环前开启一个事务，在循环后一次性提交。

```
use muke; /* 使用muke这个database */
drop table if exists t22; /* 如果表t22存在则删除表t22 */
CREATE TABLE `t22` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`a` int(11) DEFAULT NULL,
`b` int(11) NOT NULL,
`c` int(11) DEFAULT NULL,
`d` int(11) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `idx_a` (`a`),
KEY `idx_b` (`b`)
) ENGINE=InnoDB  CHARSET=utf8mb4;

drop procedure if exists insert_t22_1; /* 如果存在存储过程insert_t22_1，则删除 */
delimiter ;;
create procedure insert_t22_1() /* 创建存储过程insert_t22_1 */
begin
declare i int; /* 声明变量i */
set i=1; /* 设置i的初始值为1 */
while(i<=10000)do /* 对满足i<=10000的值进行while循环 */
insert into t22(a,b,c,d) values(i,i,i,i); /* 写入表t22中a、b两个字段，值都为i当前的值 */
set i=i+1; /* 将i加1 */
end while;
end;;
delimiter ; /* 创建批量写入10000条数据到表t22的存储过程insert_t22_1 */

drop procedure if exists insert_t22_2; /* 如果存在存储过程insert_t22_2，则删除 */
delimiter ;;
create procedure insert_t22_2() /* 创建存储过程insert_t22_2 */
begin
declare i int; /* 声明变量i */
set i=1; /* 设置i的初始值为1 */
start transaction;
while(i<=10000)do /* 对满足i<=10000的值进行while循环 */
insert into t22(a,b,c,d) values(i,i,i,i); /* 写入表t22中a、b两个字段，值都为i当前的值 */
set i=i+1; /* 将i加1 */
end while;
commit;
end;;
delimiter ; /* 创建批量写入10000条数据到表t22的存储过程insert_t22_2 */
```

我们来对比两个存储过程的速度：

```
call insert_t22_1(); /* 运行存储过程insert_t22_1 */
```

```
mysql> call insert_t22_1(); /* 运行存储过程insert_t22_1 */
Query OK, 1 row affected (14.76 sec)
```

```
call insert_t22_2(); /* 运行存储过程insert_t22_2 */
```

```
mysql> call insert_t22_2(); /* 运行存储过程insert_t22_2 */
Query OK, 0 rows affected (1.46 sec)
```

明显第二种方式快的多。因为 insert_t22_1 每一次提交都要写一次重做日志，实际写了 10000 次重做日志，而存储过程 insert_t22_2 只写了 1 次重做日志。

因此，在类似这种循环写入的情况，如果循环次数不是太多，建议在循环前开启一个事务，循环结束后统一提交。

**1.2** 不关注同一个事务里语句顺序

比如 A 在超市购买 100 元的商品，付款操作可以简化为：

| 序号 | 操作 |
| --- | --- |
| 1 | A 的账户中扣除 100 |
| 2 | 超市的账户增加 100 |
| 3 | 在超市系统中记录一条日志 |

很多时候我们会按上面的 SQL 步骤放入一个事务里执行，不关注里面语句的顺序。实际可以优化的。

根据两阶段锁，整个事务里面涉及的锁，需要等到事务提交时才会释放。因此我们在**同一个事务中，可以把没锁或者锁范围小的语句放在事务前面执行，而锁定范围大的语句放在后面执行。**

> 这里来回顾一下第 16 节中提到的两阶段锁：锁操作分为两个阶段，加锁阶段和解锁阶段，并且保证加锁阶段和解锁阶段不相交。在执行语句的时候加上锁，但并不是语句执行完就立刻释放锁，而是要等到事务结束时才释放。

因此上面 A 购买商品的例子中，可能很多人同时在超市付款，那么存在锁竞争的最可能是超市账户增加 100 元的操作。

那么付款操作可以这么优化：

| 序号 | 操作 |
| --- | --- |
| 1 | 在超市系统中记录一条日志 |
| 2 | A 的账户中扣除 100 |
| 3 | 超市的账户增加 100 |

把可能存在锁竞争的操作放在最后执行，从而优化整个事务。

因此在写程序时，应该去关注事务里的语句顺序。

#### 1.3 不关注不同事务访问资源的顺序

各位是否还记得，在第 18 节中，讲到了几种产生死锁的原因，其中有两条就跟不同事务访问资源顺序有关，我们来回顾一下：

- 不同线程并发访问同一张表的多行数据，未按顺序访问导致死锁。
- 不同线程并发访问多个表时，未按顺序访问导致死锁。

如果不关注并发访问的不同事务中访问资源的顺序，就会增大出现死锁的概率。

因此，为了降低死锁，我们需要去关注不同事务访问资源的顺序。

#### 1.4 不关注事务隔离级别

在上一节中，我们详细聊到了事务隔离级别，因此也知道，不同事务隔离级别加锁的情况也是不同的。

如果完全不关注自己业务使用的 MySQL 是什么隔离级别，可能会降低程序的并发能力或者导致死锁。

比如业务场景完全能接受幻读，如果要求更高的 QPS，使用 RR 隔离级别显然不是最好的选择，因此可以改为 RC 隔离级别。

而如果业务使用的是 RR 隔离级别，可能由于间隙锁导致死锁（可参考第 18 节 2.3 中的例子），因此也应该在程序编写时关注 RR 隔离级别下是否会有间隙锁。

因此，为了更高的并发和降低死锁概率，在创建事务前，也应该去关注自己业务的数据库是什么事务隔离级别。

#### 1.5 在事务中混合使用存储引擎

在事务中混合使用事务型（比如 InnoDB）和非事务型（比如 MyISAM）表，如果是正常提交，到没什么问题。

但是，如果该事务回滚了，事务型的表可以正常回滚，而非事务型的表的变更就无法回滚了。这种情况就会导致数据不正常，并且事务最终的结果也难以确定。

因此，在事务中混合使用存储引擎也是一个不好的事务习惯。

> 值得一提的是：如果开启 GTID，那么当同一个事务中使用不同存储引擎的表时，会出现如下报错：
>
> ERROR 1785 (HY000): Statement violates GTID consistency: Updates to non-transactional tables can only be done in either autocommitted statements or single-statement transactions, and never in the same statement as updates to transactional tables.
>
> 因此，开启 GTID 的情况，可以避免同一个事务中混合使用存储引擎的情况。

## 2 总结一下好的事务习惯

在本节中，我们列举了几种不好的事务习惯，这里总结一下好的事务习惯：

- 循环写入的情况，如果循环次数不是太多，建议在循环前开启一个事务，循环结束后统一提交。
- 优化事务里的语句顺序，减少锁时间。
- 关注不同事务访问资源的顺序。
- 创建事务之前，关注事务隔离级别。
- 在事务中混合使用存储引擎。

## 3 问题

你认为还有哪些不好的事务习惯？

## 4 参考资料

《MySQL 技术内幕》第 2 版 7.8 不好的事务习惯

}

21 不同事务隔离级别有哪些区别?

23 细聊分布式事务