Atitit 事务解决方案 应用层事务

目录

[1.1. Try catch 全部delete即可。。。简单。。存入log 1](#_Toc28644)

[1.2. 使用db定时器，找出不一致对数据，使用多个表 not exits对比，存入零时表。。Then delte 1](#_Toc23183)

[1.3. 回滚的方式是提取undo日志，做逆向操作。 1](#_Toc14748)

## Try catch 全部delete即可。。。简单。。存入log

Undo日志，就是java内存数据变量表。。Type,table,pk

回滚，就是取得回滚的记录id，然后根据type来回滚。。

## 使用db定时器，找出不一致对数据，使用多个表 not exits对比，存入零时表。。Then delte

记录日志临时表状态。。

Update回滚模式

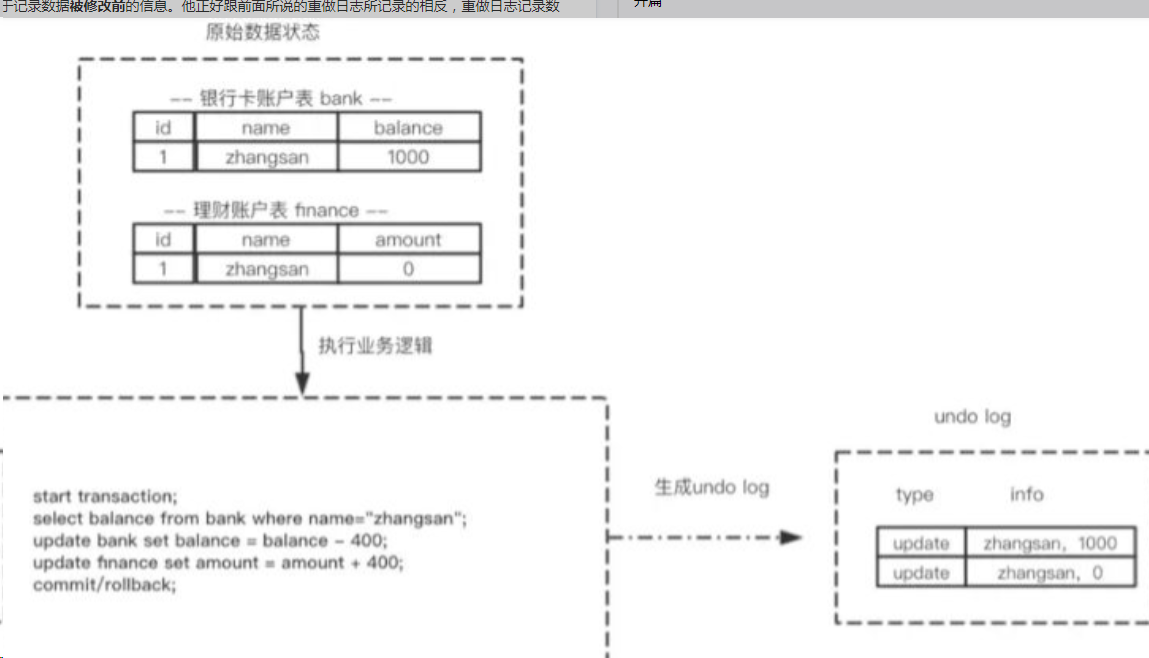
可以吧updt语句放到最后，这样就不用单独回滚，麻烦对。。运维Insert类对回滚很简单对，delete就ok。。

## 回滚的方式是提取undo日志，做逆向操作。

由于InnoDB的undo是单独写在表空间中的，本质上和普通的数据页是一样的。如果在事务回滚时，undo页已经被从内存淘汰，回滚操作（特别是大事务变更回滚）就可能伴随大量的磁盘IO。因此InnoDB的回滚效率非常低。有的数据库管理系统，例如PostgreSQL，通过在数据页上冗余数据产生版本链的方式来实现多版本，因此回滚起来非常方便，只需要设置标记即可，但额外带来的问题就是无效数据清理开销。

活着可以在元表上增加冗余列？不可行因为多线程模式，除非单用户分割。。

增加undo表，记录元表内容。。回滚的时候读取，然后设置此undo记录为状态为完成



Undolog每个表弄一个，这样方便表复制，结构一样。。

Type字段，增加一个全局undo表里面关联起来即可。。。Update,insert ,delete

Pk fld...table fld...

Update,tab1,pk

解到数据的变更都伴随着回滚日志的产生： (1) 产生了被修改前数据(zhangsan,1000) 的回滚日志

(2) 产生了被修改前数据(zhangsan,0) 的回滚日志

根据上面流程可以得出如下结论： ****1.每条数据变更(insert/update/delete)操作都伴随一条undo log的生成,并且回滚日志必须先于数据持久化到磁盘上**** ****2.所谓的回滚就是根据回滚日志做逆向操作，比如delete的逆向操作为insert，insert的逆向操作为delete，update的逆向为update等。****

思考：为什么先写日志后写数据库？ ---稍后做解释

回滚可以使用replace语句覆盖。。

以update操作为例：当事务执行update时，其生成的undo log中会包含被修改行的主键(以便知道修改了哪些行)、修改了哪些列、这些列在修改前后的值等信息，回滚时便可以使用这些信息将数据还原到update之前的状态。

# Update对undo恢复

create table cp2.userUndo select \* from user where id= 1689341

Rollback回滚

update user join cp2.userUndo uu on user.id=uu.id

set user.balance=uu.balance

where user.id=1689341

## 自己实现的事务只需要undo。。不需要redo

如果是自己实现的事务机制，不需要redo，因为数据不是刷入cache，而是直接刷入文件。。

只需要undo，记录之前的数据。。App重启对时候，对比undo与表数据，如果一致，说明没有提交。。。如果不一致，说明提交ok。。。

如果俩条数据，一条一致，一条不一致，那么需要redo不一致那条。。