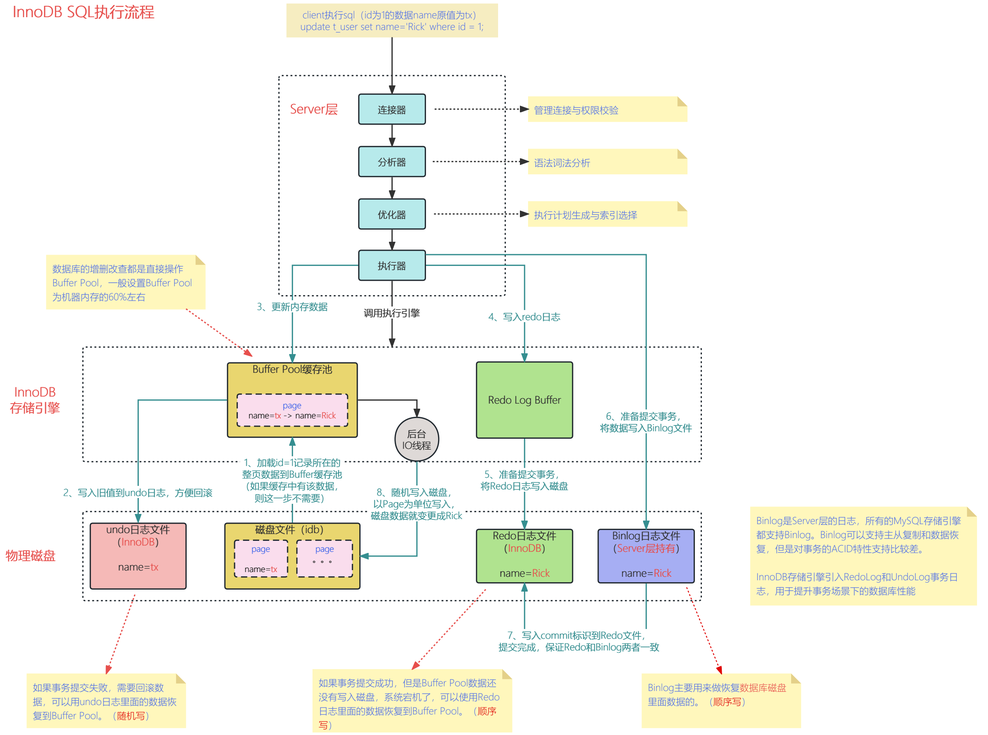
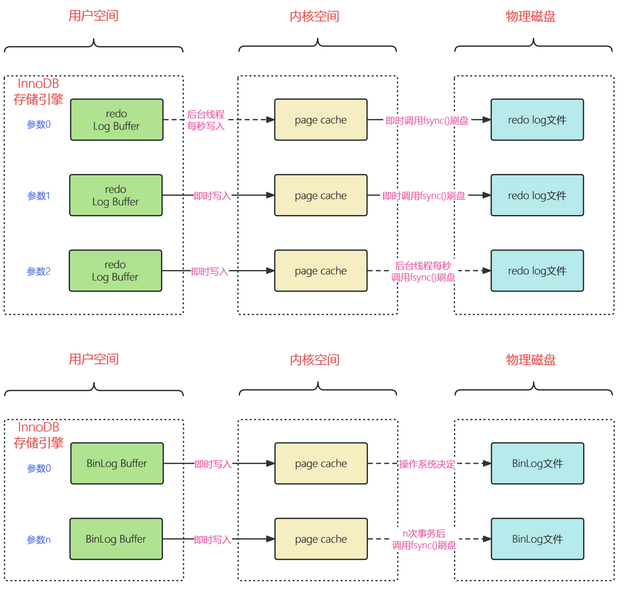
InnoDB SQL执行流程



日志写磁盘流程

RedoLog 和 BinLog 并不是直接实时写入磁盘，中间还有一个页缓存，用来提升写入读取速度（页缓存是在内核空间，属于特殊的内存空间）



InnoDB刷盘策略

数据（脏页）刷盘
- 系统内存不足时，需要将一部分数据页淘汰掉，如果淘汰的是脏页，需要先将脏页同步到磁盘；
- MySQL 认为空闲的时间，这种情况没有性能问题；
- MySQL 正常关闭之前，会把所有的脏页刷入到磁盘，这种情况也没有性能问题。
- redo log 日志满了的情况下，会主动触发脏页刷新到磁盘

Redo Log日志
InnoDB引擎有一个后台线程，每隔1 秒，就会把 redo log buffer 中的内容写到文件系统缓存（page cache），然后调用 fsync 刷盘。也就是说，一个没有提交事务 redo log 记录，也可能会刷盘。

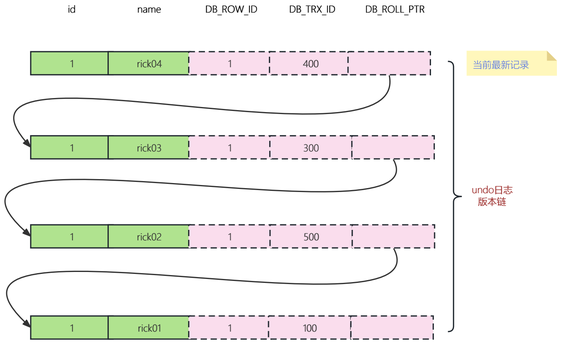
commit刷盘策略由innodb_flush_log_at_trx_commit来控制。
0：提交事务时，不进行刷盘操作。提交成功，数据库或者服务器宕机会导致丢失1秒钟的数据
1：提交事务时，即时写入page cache，并调用fsync()刷盘（默认）。提交成功代表数据已经写入磁盘，数据库或者服务器宕机不会导致数据丢失
2：提交事务时，将操作写入page cache。提交成功，代表数据已经到page cache，此时数据库宕机不会丢失数据，服务器宕机会导致数据丢失

Binlog 日志
刷盘策略由sync_binlog来控制
0：提交事务时，即时写入page cache，由操作系统控制什么时候刷盘（默认）。
n：提交事务时，即时写入page cache，在n次事务后调用fsync()刷盘。
提交成功代表数据已经到page cache，此时数据库宕机不会丢失数据，服务器宕机会丢失数据

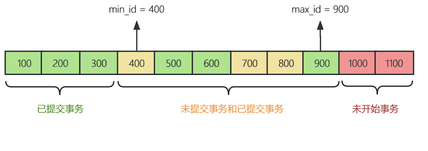
Redo日志是顺序写，循环写，文件大小固定，写满就重头开始写，覆盖原来的内容，断电后可以用来恢复事务数据，但是不可作为整库数据恢复（因为数据被覆盖过）
小贴士：每条 redo 记录由"表空间号+数据页号+偏移量+修改数据长度+具体修改的数据"组成

BinLog是顺序写，达到限制后，会切换到下一个文件继续写，断电后可作为整库恢复
binlog记载的是update/delete/insert这样的SQL语句

undo日志版本链



read-view包含如下信息
视图数组：[400，700，800]
min_id：400
max_id：900



RR级别下：事务开启后，首次查询会生成一致性视图read-view，该视图在事务结束之前都不会变化
RC级别下：事务开启后，每次查询都会重新生成新的read-view

read-view视图由查询时所有未提交事务trx_id数组（数组里最小的id为min_id）和已创建的最大事务id（max_id）组成，事务里的任何sql查询结果需要从对应版本链里的最新数据开始逐条跟read-view做比对从而得到最终的快照结果

版本链比对规则：
1、trx_id<min_id，表示这个trx_id已经提交了，这个数据是可见的
2、trx_id>max_id，表示这个trx_id还没有生成（未来生成），这个数据是不可见的
3、min_id <= trx_id <= max_id，如果trx_id在视图数组中，表示是未提交的事务，则是不可见的；否则是已经提交的事务，是可见的

注：
1、begin/start transaction 命令并不是一个事务的起点，执行第一个增删改操作时，才会向mysql申请事务id，mysql内部是严格按照事务的启动顺序来分配事务id
2、事务id生成是全局顺序的，但undo日志版本链里面的事务ID并不是顺序的，左图只是方便查看，才将事务id做成顺序的
3、删除操作可以认为是特殊update，会将版本链上最新的数据复制一份，然后将trx_id修改成删除操作的trx_id，同时在该条记录的头信息（record header）里的（deleted_flag）标记位写上true，来表示当前记录已经被删除，在查询时按照上面的规则查到对应的记录如果delete_flag标记位为true，意味着记录已被删除，则不返回数据
4、事务开启与观察的表的版本链没有关系，比如你观察的是account表的版本链，但是事务开启是因为user表做了更新操作（后面才做account表的更新操作）