数据库多版本实现内幕

osdba

杭州云徙科技有限公司

PostgreSQL

个人简介

唐成 osdba
云徙科技云技术总监

《PostgreSQL修炼之道：从小工到专家》作者；

PostgreSQL中国用户会CPUG 核心成员
(http://www.postgres.cn/community)；

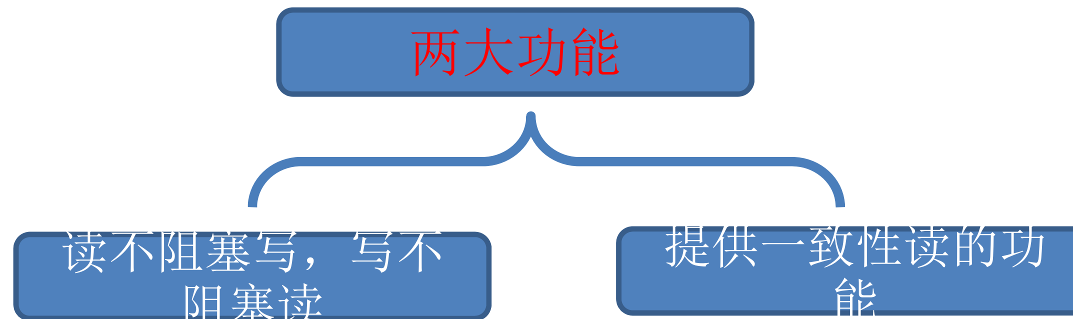
曾任网易杭州研究院开发专家，主导了网易云计算中的云硬盘产品（类似amazon EBS）的设计和开发

1. 为什么需要MVCC
2. innodb的MVCC实现
3. Oracle的MVCC实现
4. PostgreSQL MVCC实现

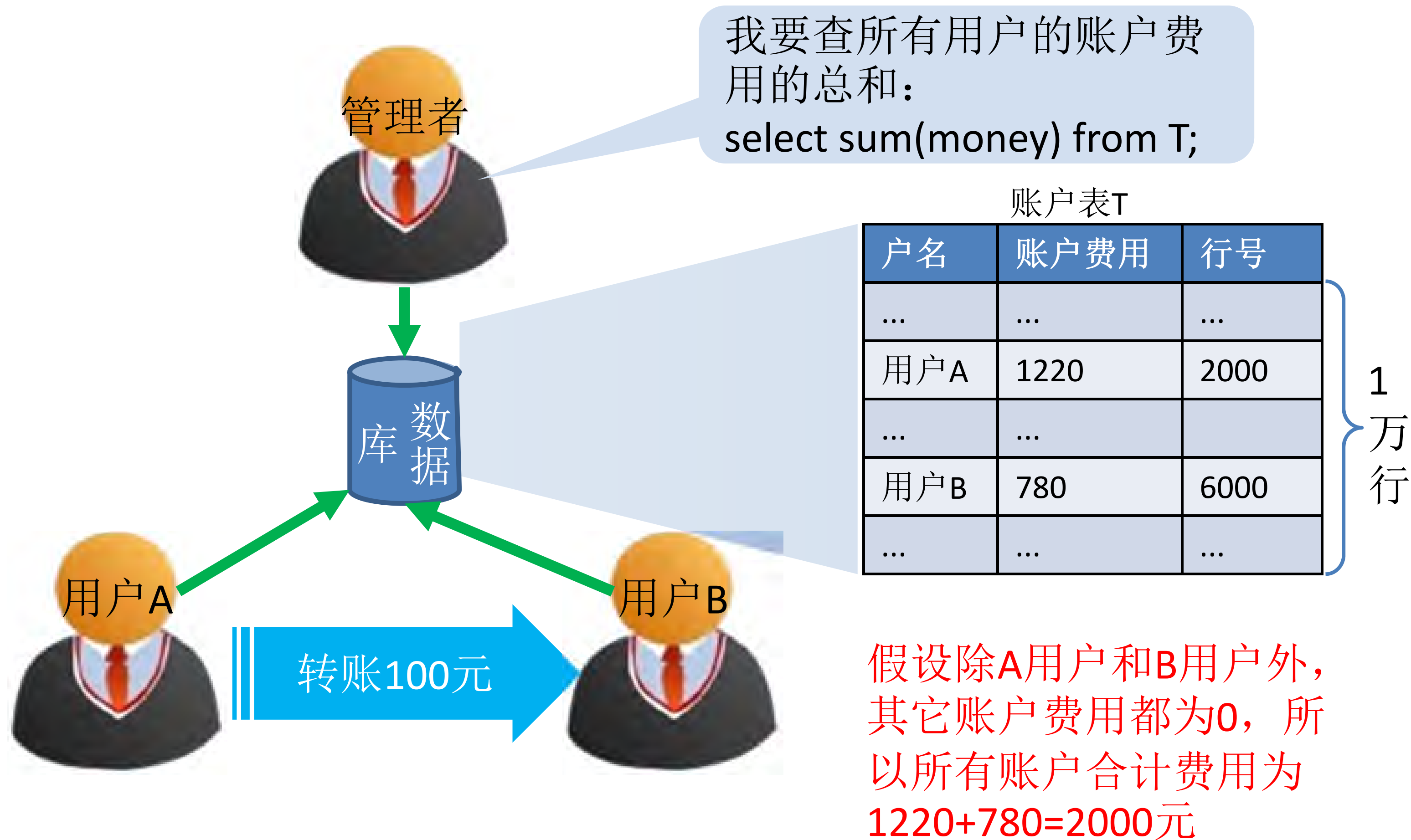
什么是MVCC?

借助wiki上的解释：

Multiversion concurrency control (MCC or MVCC), is a **concurrency control method commonly** used by database management systems to provide **concurrent access to the database** and in programming languages to implement transactional memory.



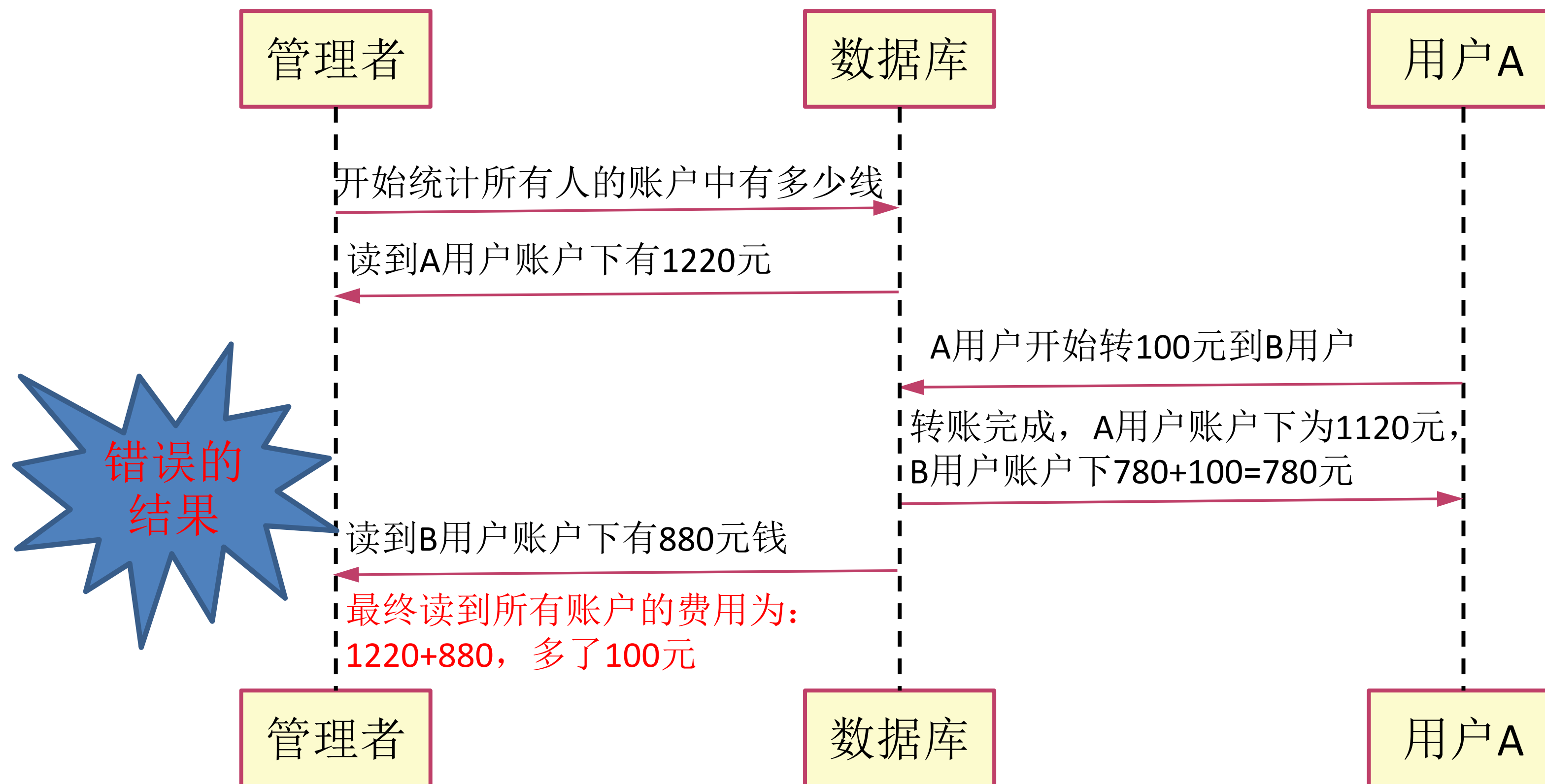
考虑一个场景：转账场景



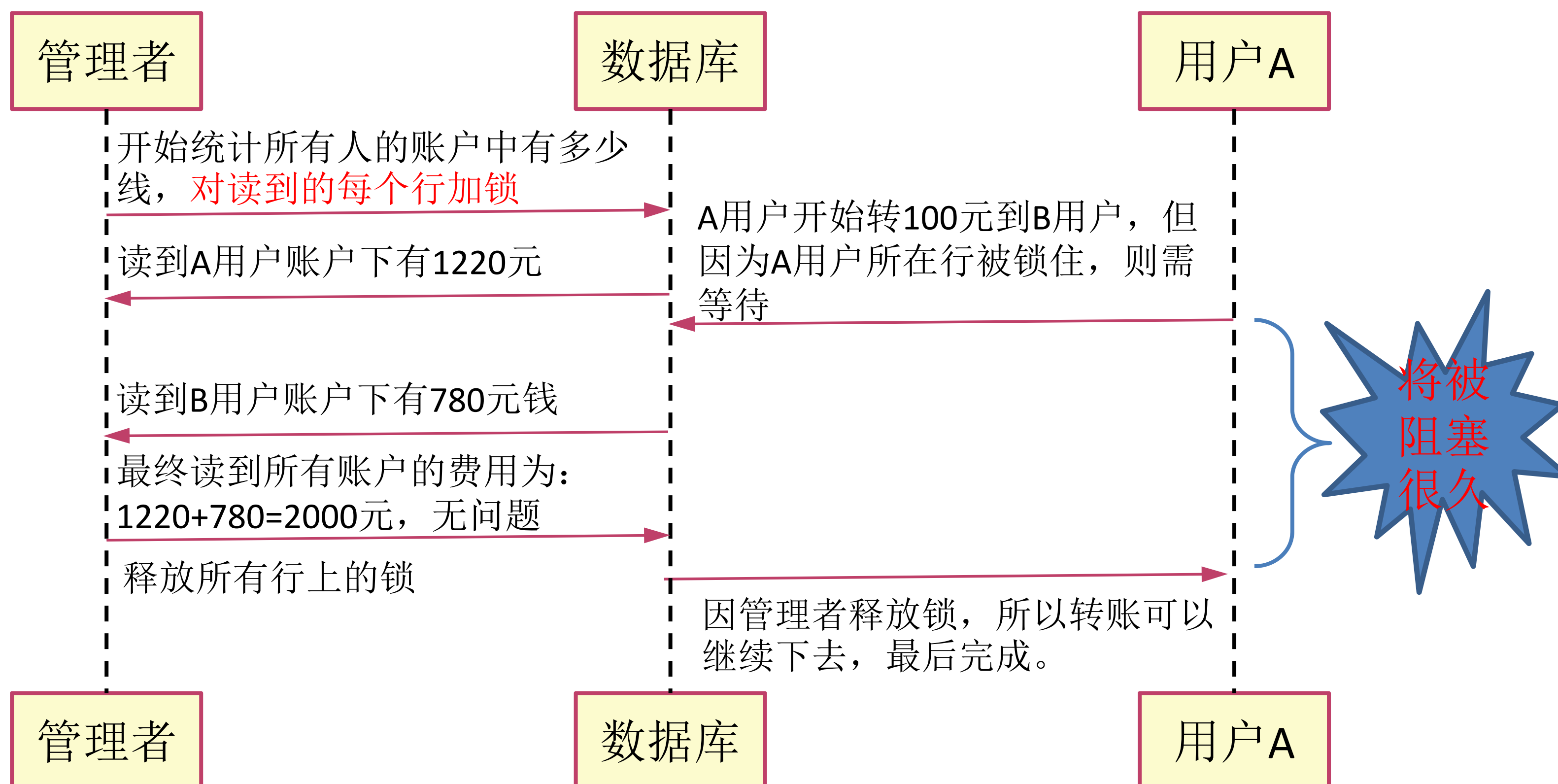
户名	账户费用	行号
…	…	…
用户A	1220	2000
…	…	
用户B	780	6000
…	…	…

假设除A用户和B用户外，其它账户费用都为0，所以所有账户合计费用为1220+780=2000元

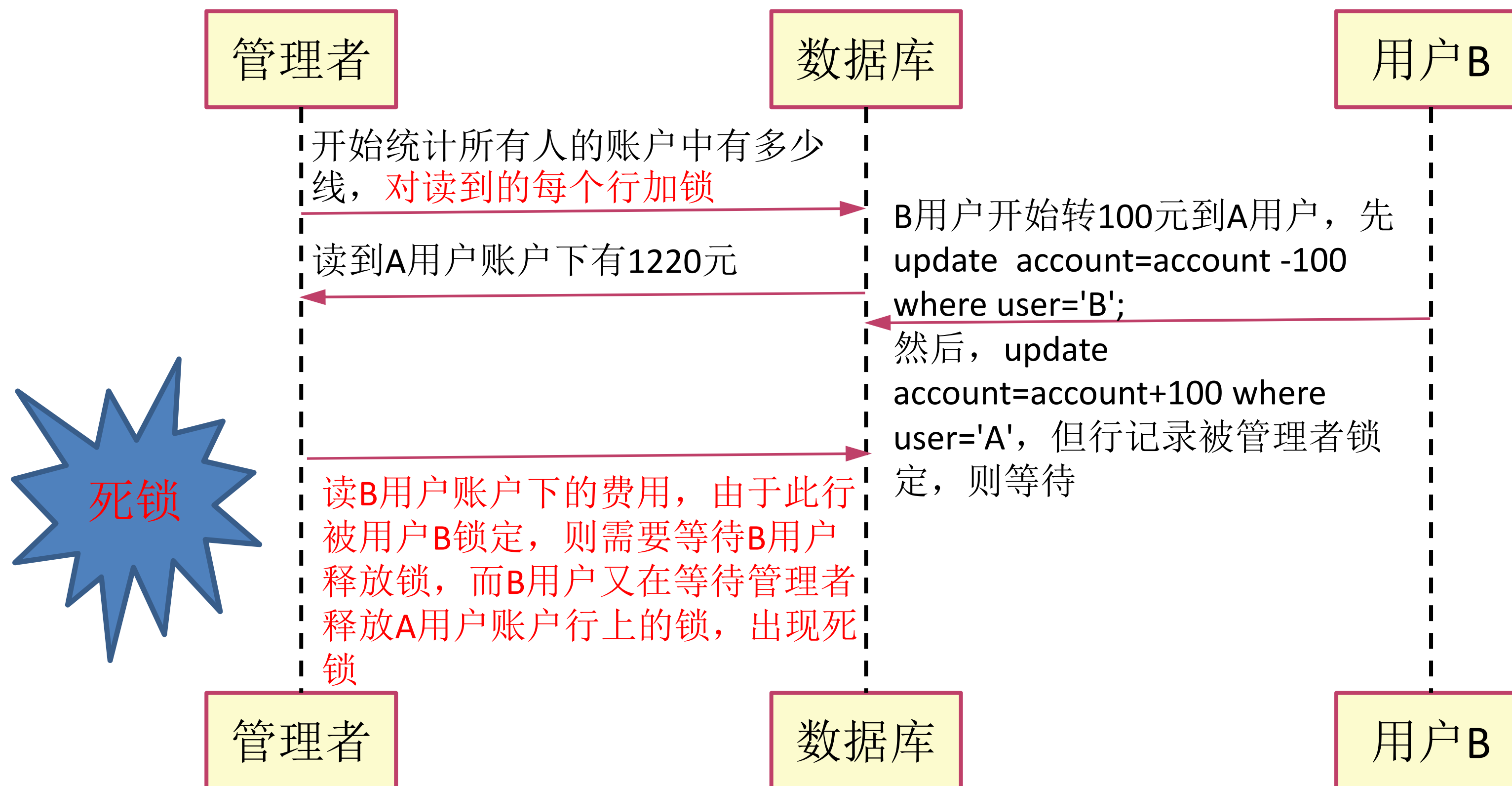
转账与统计总账的一个时序图



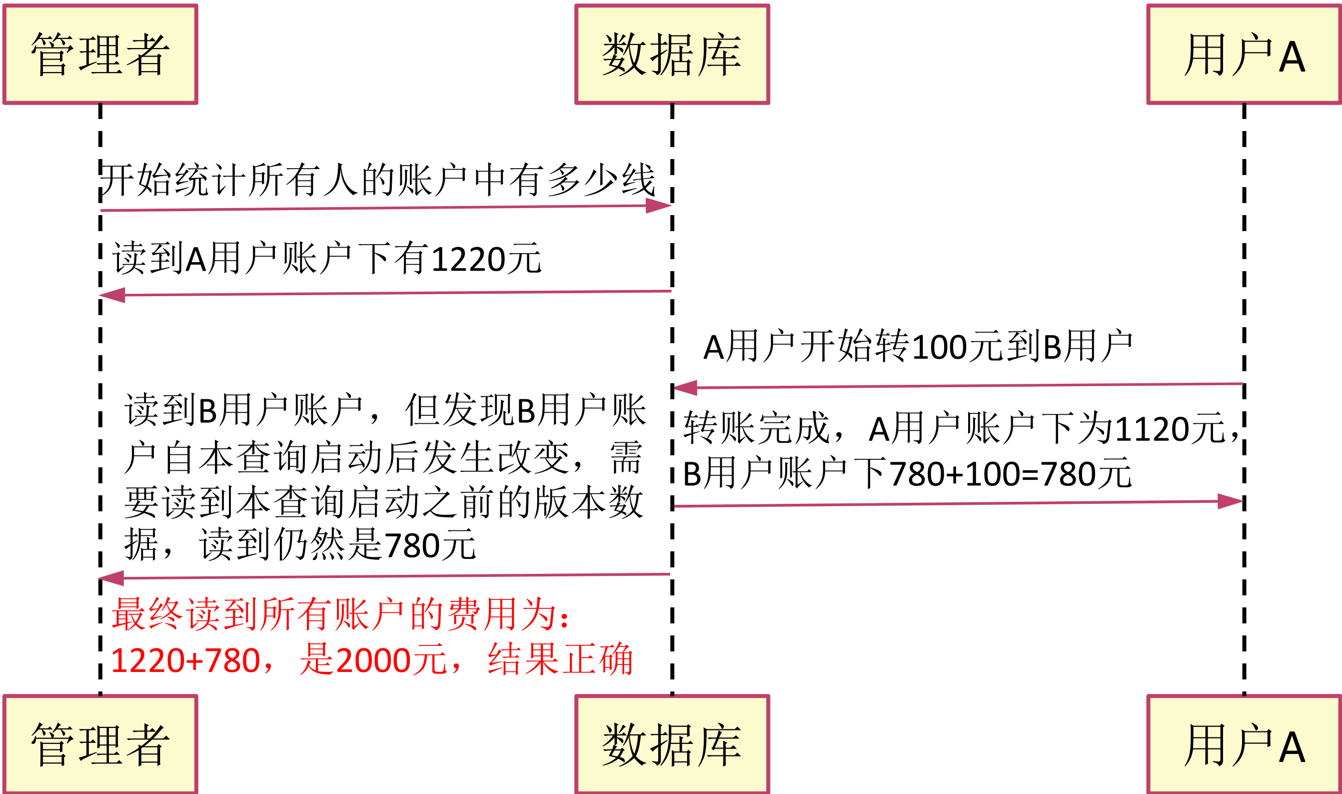
无MVCC数据库解决办法



无MVCC产生的新问题：死锁



MVCC下的转账与统计



MVCC的总结

- 查询和更新、删除、插入操作互相不阻塞
- 当开始一个查询后，读到的数据总是查询开始时那个时间点的快照
  - 在查询开始后，发生的变更（即使已提交），这次查询也是看不到的。
  - 一个事务无论运行多长时间，看到数据都是相同的
  - 不同开始时间的事务中相同的查询，返回的数据也可能不同

1 为什么需要MVCC

2 innodb的MVCC实现

3 Oracle的MVCC实现

4 PostgreSQL MVCC实现

Innodb的事务相关概念

- redo log
  - 在变更数据之前，把变更先记录到一个文件中，称为redo log
- undo log
  - 与redo log相反，undo log是为回滚而用
- rollback segment
  - 在Innodb中，undo log被划分为多个段，具体某行的undo log就保存在某个段中，称为回滚段。

Innodb多版本的实现

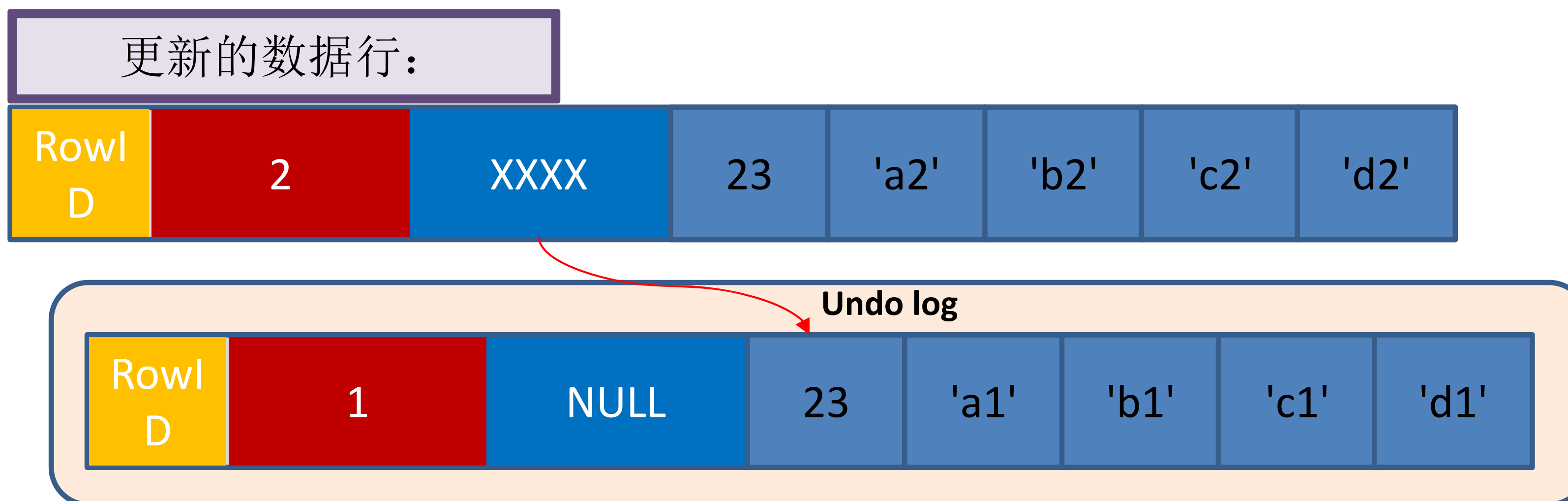
事务1：insert t(id, col1,col2,col3,col4) values(23,'a1','b1','c1','d1');

RowID	事务ID DB_TRX_ID	回滚指针 DB_ROLL_PT	id	col1	col2	col3	col4
RowI D	1	NULL	23	'a1'	'b1'	'c1'	'd1'

行上有三个隐含字段：分别对应该行的rowid、事务号和回滚指针，id、Col1～Col4是表各列的名字，23、’ 'a1'～'d1'是其对应的数据。

Innodb多版本的实现

事务2: update table t set col1='a2', col2='b2', col3='c2', col4='d2' where id=23;

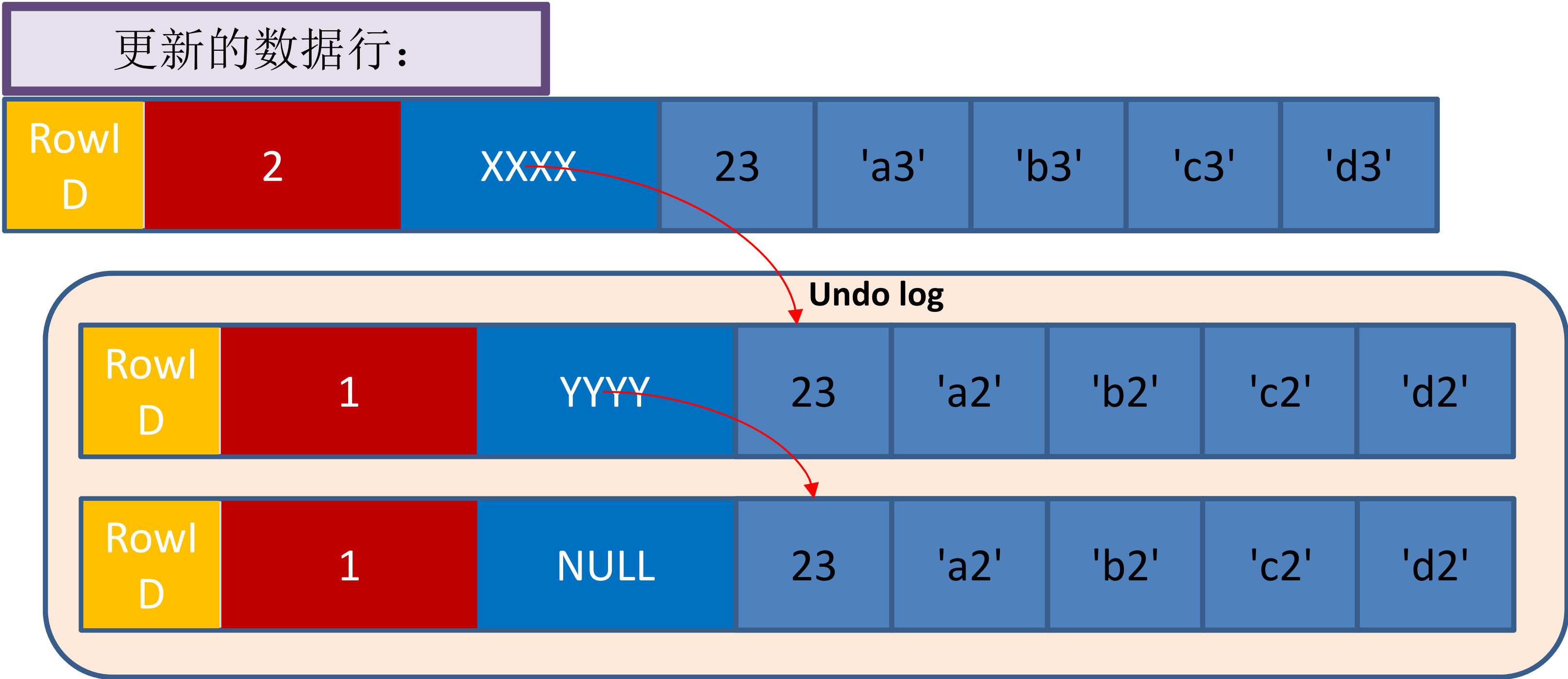


- 用排他锁锁定该行
- 把该行修改前的值Copy到undo log中。
- 修改当前行的值，填写事务编号，使回滚指针指向undo log中的修改前的行
- 记录redo log，包括undo log中的变化

YUNXI 云徙科技

Innodb多版本的实现

事务3: update table t set col1='a3', col2='b3', col3='c3', col4='d3' where id=23;



Innodb多版本的实现

- 多次更新后，回滚指针会把不同版本的记录串在一起。
- 在Innodb中存在purge线程，它会查询那些比现在最老的活动事务还早的undo log，并删除它们，从而保证undo log文件不至于无限增长。

Innodb的事务的提交与回滚

- 提交与回滚
  - 当事务正常提交时，Innodb只需要更改事务状态为COMMIT即可，不需做其他额外的工作
  - Rollback需要根据当前回滚指针从undo log中找出事务修改前的版本，并恢复。
    - 如果事务影响的行非常多，回滚则可能会很慢，根据经验值没提交的事务行数在1000～10000之间，Innodb效率还是非常高的。
    - 回滚时，也会产生redo日志
  - Innodb的COMMIT效率高，Rollback代价大

Innodb的可见性判断

- InnoDB表会有三个隐藏字段
  - 6字节的DB_ROW_ID
  - 6字节的DB_TX_ID
    - InnoDB内部维护了一个递增的tx id counter，其当前值可以通过show engine innodb status获得
  - 7字节的DB_ROLL_PTR(指向回滚段的地址)

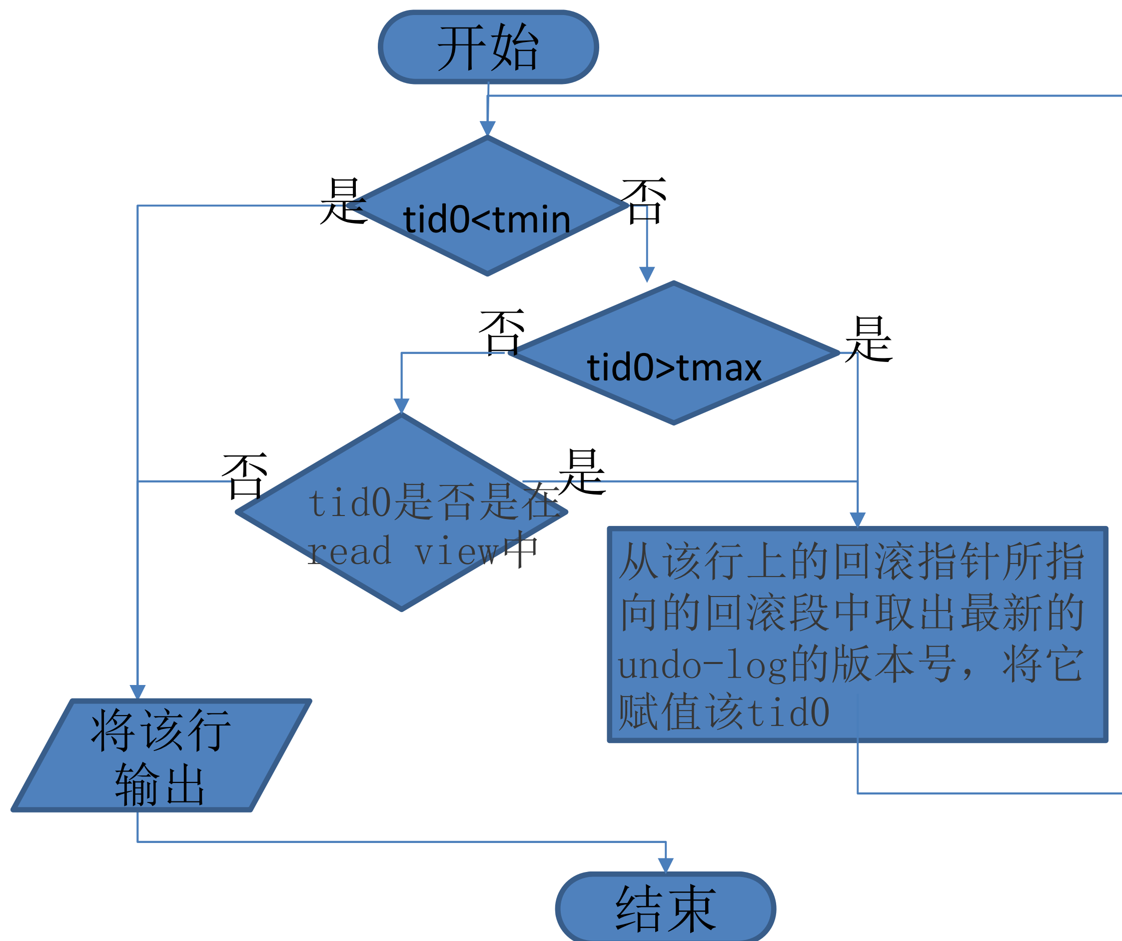
Innodb的可见性判断

- 可见性比较的方法：
  - 并不是用当前事务ID与表中各个数据行上的事务ID去比较的
  - 在每个事务开始的时候，会将当前系统中的所有的活跃事务拷贝到一个列表中 (read view)，，根据read view最早一个事务ID和最晚的一个事务ID来做比较的，这样就能确保在当前事务之前没有提交的所有事务的变更以及后续新启动的事务的变更，在当前事务中都是看不到的。
  - 当然，当前事务自身的变更还是需要看到的。

可见性判断的流程

当开始一个事务时，把当前系统中活动的事务的ID都拷贝到一个列表（read view)中，这个列表中最早的事务ID为tmin，最晚的事务ID为tmax

当读到一行时，该行上当前事务id为tid0，当前行是否可见的判断逻辑见右图



YUNXI 云 徙 科 技

1 为什么需要MVCC

2 innodb的MVCC实现

3 Oracle的MVCC实现

4 PostgreSQL MVCC实现

Oracle的多版本实现

- Oracle也是通过回滚段来实现多版本的
  - 但Oracle的实现更复杂，更精细一些。
- Oracle中也有事务ID，但不是递增的
  - Undo Segment Number + Transaction Table Slot Number + Wrap
- 与innodb不一样的地方：
  - 事务信息并不是记录在每个数据行上的，而是在块头中的ITL槽上，所以相对来说更省空间

ITL解释

- ITL(Interested Transaction List)在Oracle数据块的头部
  - ITL记录在一个数据块中有多条，每一条itl可以看作是一个记录，每条记录常被称为槽位（itl slot)
  - 一个itl slot只可以记录一个事务的信息，如果这个事务已经提交或回滚了，那么这个itl的位置就可以被反复使用。

ITL解释（续）

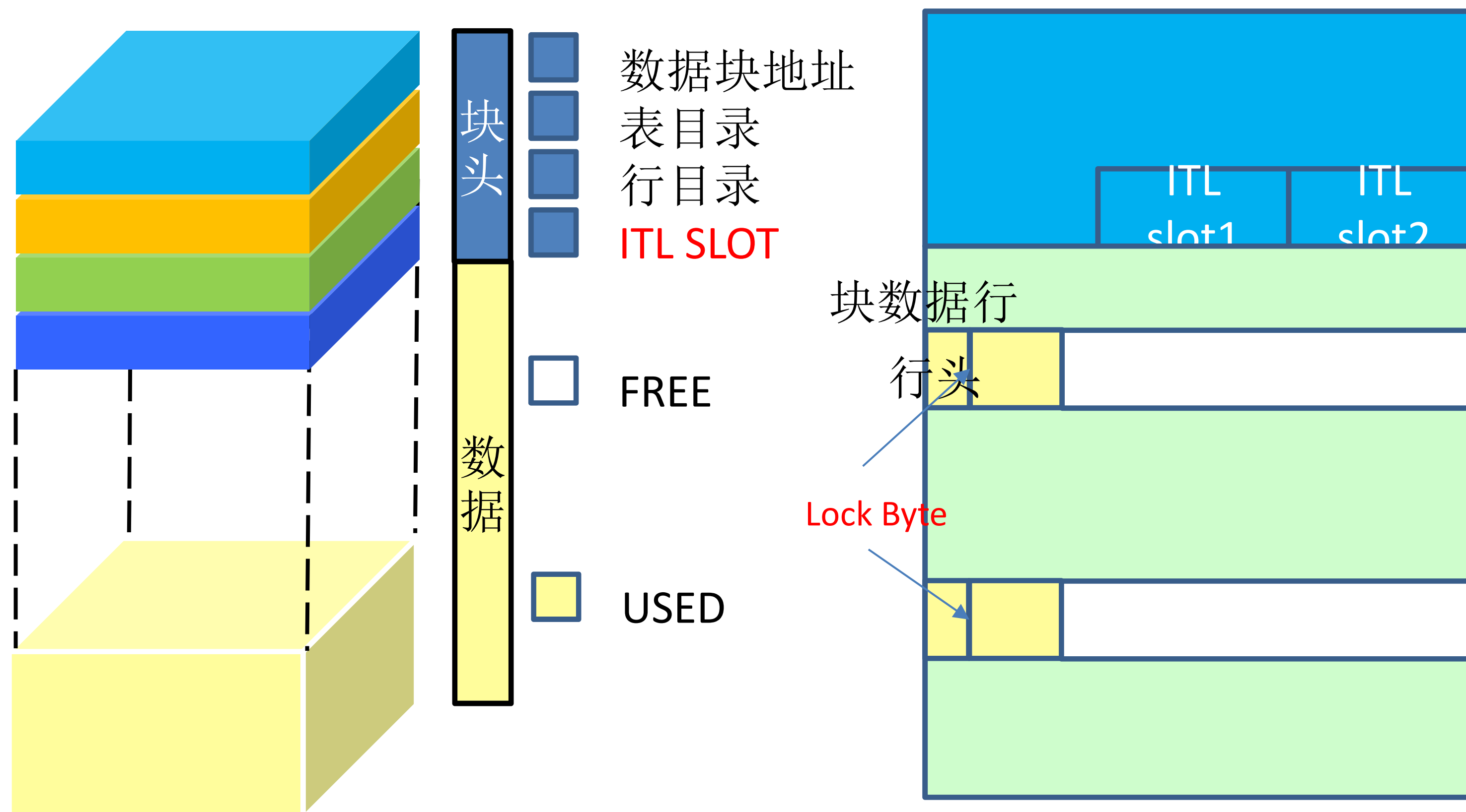
- 每个数据块上itl槽的多少可以动态创建，建表时可以指定：
  - initrans：每个数据块默认ITL槽数目，默认为2
    maxtrans：每个数据块最多的ITL槽数，最大255

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0006.002.0000158e	0x0080104d.00a1.6e	--U-	734	fsc 0x0000.6c9deff0
0x02	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000
…	…	…	…	…	…

ITL内容

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0006.002.0000158e	0x0080104d.00a1.6e	--U-	734	fsc 0x0000.6c9deff0
0x02	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000
…	…	…	…	…	…

Oracle数据块的结构



YUNXI 云徙科技

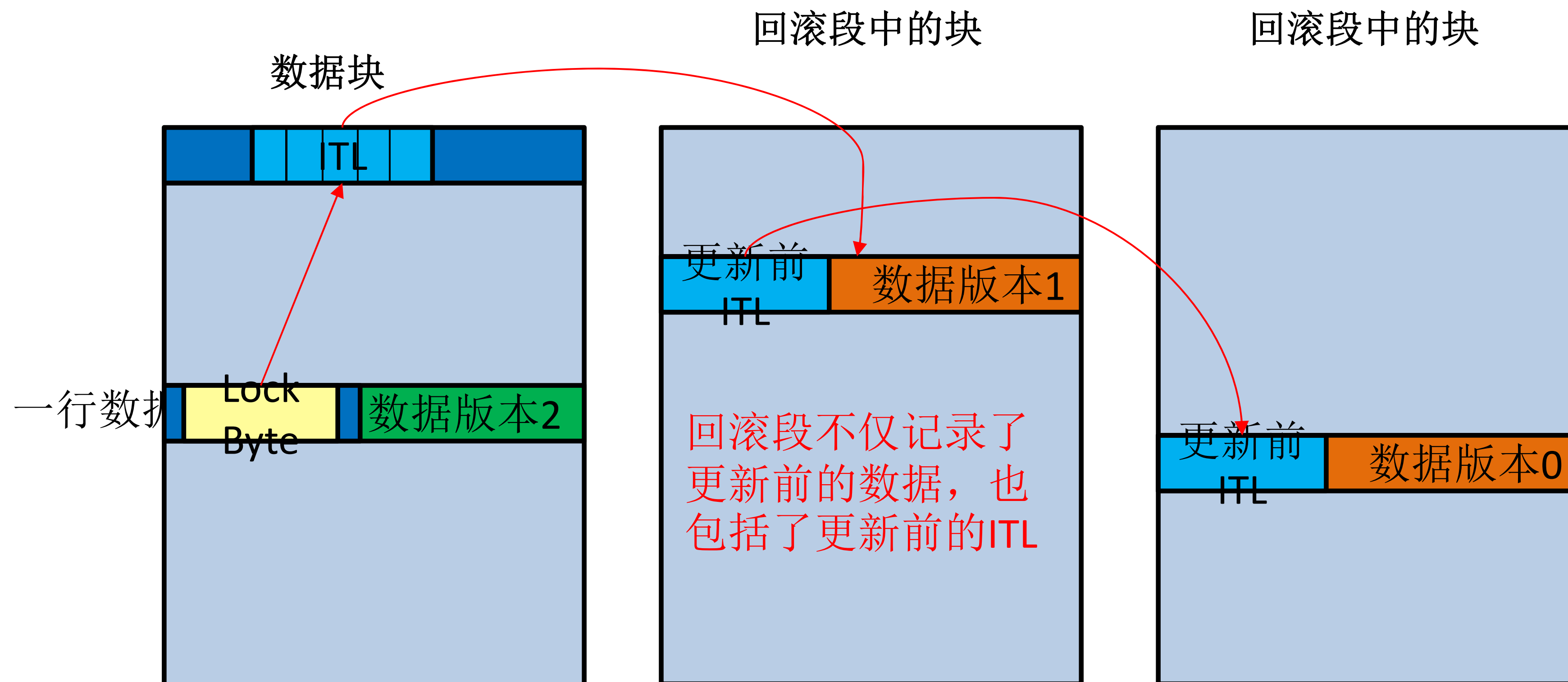
Oracle的多版本实现

- 由于Oracle中的事务ID不是递增的，为了判断事务之间的先后关系，需要一个递增序号，这个序号在Oracle中就叫SCN（当然SCN还有其它用处）

Oracle的多版本实现

- SCN（System Change Number）
  - 是顺序递增的一个数字，在Oracle 中用来标识数据库的每一次改动，及其先后顺序。
  - SCN是由6字节组成，最大值是0xffff.ffffffff
  - 单节点的instance中，SCN值存在SGA区，由system commit number latch保护。任何进程要得到当前的SCN值，都要先得到这个latch。
  - RAC中是通过排队机制(Enqueue)实现SCN在各并行节点之间的顺序增长，这里不再赘述。

旧版本的数据在回滚段中的结构



ITL中的Uba字段指向了回滚段中旧镜像的数据位置
ITL中的SCN用于比较版本的新旧

1 为什么需要MVCC

2 innodb的MVCC实现

3 Oracle的MVCC实现

4 PostgreSQL MVCC实现

PostgreSQL MVCC实现

- 什么？PostgreSQL没有回滚段！！！
  - 是的，没有回滚段。旧数据是放在原有数据文件中的
  - 如果放在原有的数据文件中，旧数据越来越多怎么办？
    - 垃圾回收操作vacuum来做这个事。
    - 有自动垃圾回收autovacuum
  - 更新操作中新行的物理位置发生了变化，非更新列的索引是不是也要更新？
    - 通常不会，HOT技术。如果原有的数据块之间有空间，旧行与新行之间会建一个链接，索引上仍然指向旧的数据行

PostgreSQL MVCC实现

- 更新操作中新行的物理位置发生了变化，非更新列的索引是不是也要更新？
  - 通常不会，HOT技术。如果原有的数据块之间有空间，旧行与新行之间会建一个链接，索引上仍然指向旧的数据行。
- 垃圾回收的代价会不会影响性能？
    - 有很多参数控制这个影响：vacuum_cost_delay，vacuum_cost_limit

PostgreSQL MVCC实现

- 实现方法
  - 每行上有xmin和xmax两个系统字段
  - 当插入一行数据时，将这行上的xmin设置为当前的事务id，而xmax设置为0
  - 当更新一行时，实际上是插入新行，把旧行上的xmax设置为当前事务id，新插入行的xmin设置为当前事务id，新行的xmax设置为0

PostgreSQL MVCC实现

- 实现方法（续）
  - 当删除一行时，把当前行的xmax设置为当前事务id
  - 当读到一行时，到commitlog中查询xmin和xmax对应的事务状态是否是已提交还是回滚了，就能判断出此行对当前行是否是可见。
  - autovacuum进程会把一些不要的旧行清理掉

PostgreSQL事务ID

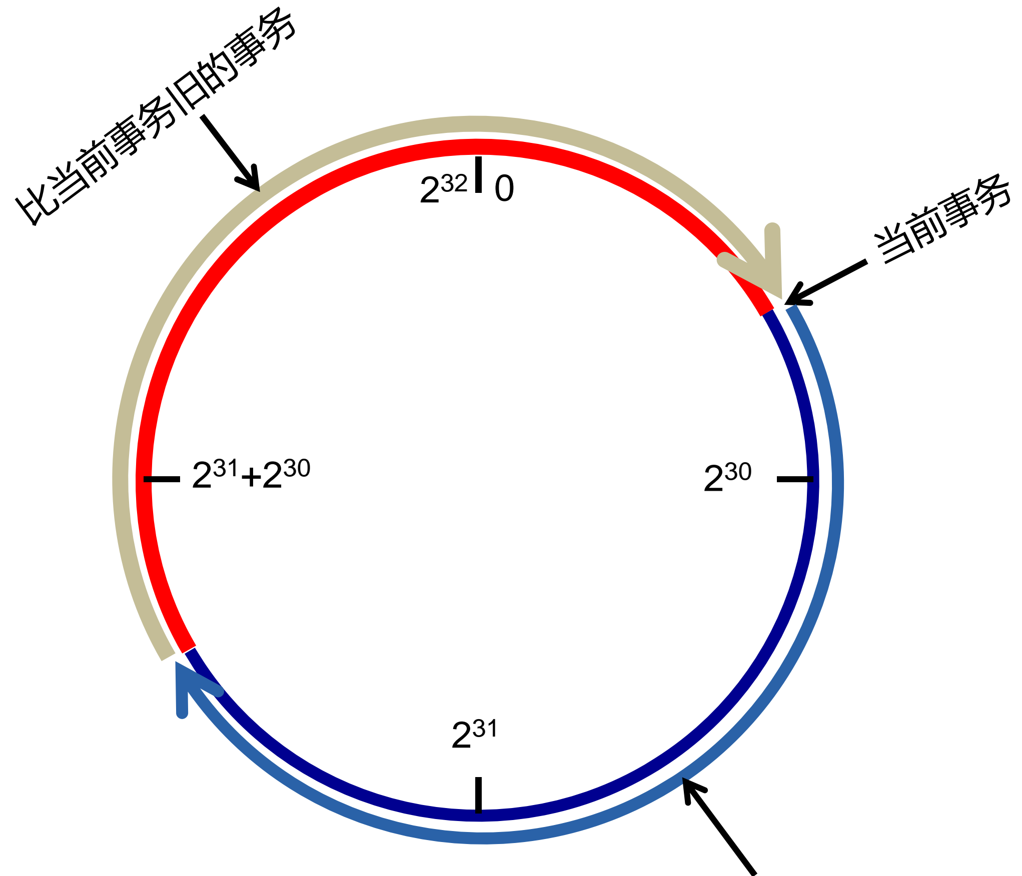
- 与innodb类似，是一个递增的数字，常常被称为xid
  - 但是一个无符号的32bit的数字表示
- 如何知道事务是提交了还是回滚了？
  - 事务的状态记录一个叫commitlog的位图文件中，即pg_clog目录下的文件中
  - 每个事务的状态用两个bit来表示：
    - #define TRANSACTION_STATUS_IN_PROGRESS 0x00
    - #define TRANSACTION_STATUS_COMMITTED 0x01
    - #define TRANSACTION_STATUS_ABORTED 0x02
    - #define TRANSACTION_STATUS_SUB_COMMITTED 0x03

PostgreSQL MVCC实现

- 事务ID到达最大的一个数字后，怎么办？
  - 到大最大后，又重新开始，
  - 此问题被称之为事务ID回卷问题(Transaction ID Wraparound)
- 记录事务的commitlog文件会不会不断的增大？

事务id回卷问题的解决

- 事务id的范围可以认为组成了一个圆
- 事务id从0开始，到达最大之后又变回0.
- 对一个指定的事务ID，当前位置之前有2^{31}个事务比其旧，位置之后有2^{31}个比其新
- 由vacuum保证整个数据库中的所有表中的xmin和xmax中事务ID的值的范围小于2^{31}，太旧的事务ID值会设置为一个特殊的事务ID: FrozenXID
- FrozeXID认为是比所有的事务ID都旧



YUNXI
云徙科技

事务id回卷解决的细节

- 有三个特殊值的事务ID
  - 0: InvalidXID，无效事务ID
  - 1: BootstrapXID，表示系统表初使化时的事务ID，比任务普通的事务ID都旧。
  - 2:FrozenXID，冻结的事务ID，比任务普通的事务ID都旧。
  - 大于2的事务ID都是普通的事务ID。

事务id回卷解决的细节

- commitlog的大小
  - 理论上，数据库中事务ID最多2^{31}个，每个事务占用2bit，所以commitlog最大512M字节
  - autovacuum_freeze_max_age为2亿，

比较事务新旧的方法

- 普通事务的比较方法：(int32) (id1 - id2) < 0
- 表达式算出来值为真，则id1比id2更旧一些，为假则id1比id2新
  - 例子：id1=4294967290，id2=5，id2是当事务回卷后的值，id1-id2=4294967285，而4294967285因大于2^31，转成int32后会变成一个负数，表过式为真，所以id1比id2更旧
- BootstrapXID比所有其它事务都旧，包括FrozenXID
- FrozenXID比普通事务旧

可见性判断

- 与innodb一样，事务开始时，会把当前活跃的事务ID记录到一个列表中，这称之为快照。
- 系统先通过判断t_xmin是否在全局活跃事务列表中、是否在事务快照活跃事务列表中、根据事务提交日志判断事务是提交还是回滚了等来判断t_xmin事务是否在事务开始时已经提交

YUNXI 云徙科技

可见性判断（续）

- 然后用类似的方法判断 t_xmax是否在事务开始时已经提交。如果t_xmin在事务开始时没有提交则不可见;
- 如果t_xmin在事务开始时已经提交而t_xmax没有，则可见;
- 如果t_xmin和t_xmax在事务开始时都已经提交了则不可见。
  - 详细过程见HeapTupleSatisfiesMVCC、TransactionIdDidCommit、XidInMVCCSnapshot等函数)。

Thanks!

Q & A

PostgreSQL