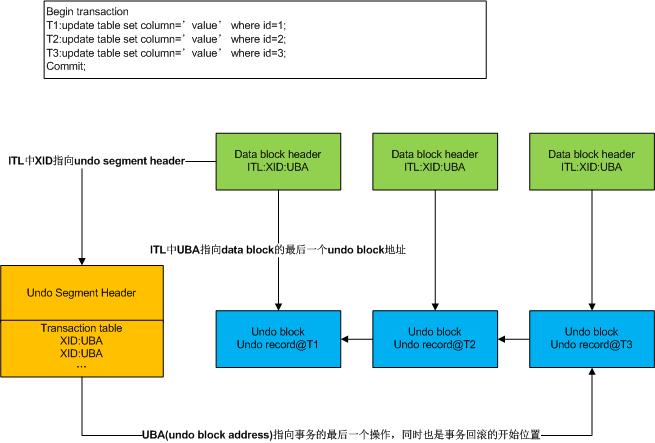
Oracle中undo的作用主要有两个：第一是回滚事务，第二是产生一致性读。同时也衍生出了一些新的 功能，比如Flashback query。传统的undo是通过undo segment来管理的，我们看下面的示例：



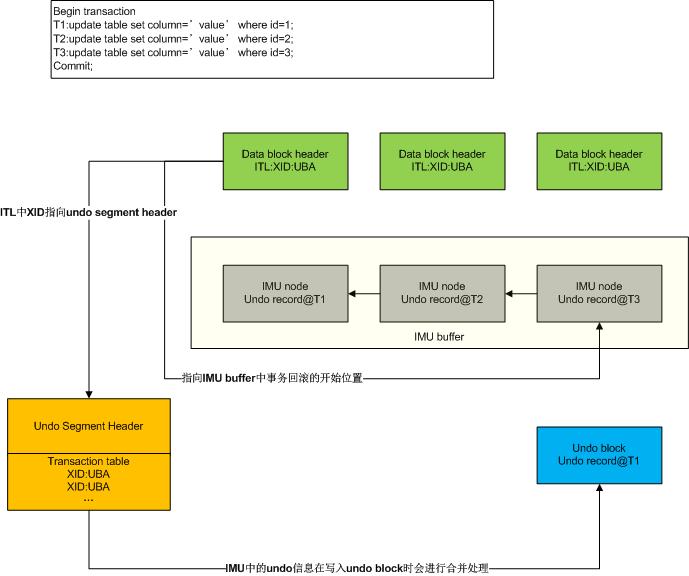
事务开始，必须首先在data block中分配ITL，ITL中记录了事务ID(XID)，XID由三部分内容组成：XIDUSN(回滚段号)，XIDSLOT(回滚段槽号)，XIDSQN(序列号)，在Undo segment header中有一个事务表，记录该回滚段上的事务信息，每个事务都会占据了一个回滚槽，XID对应一个UBA(undo block address)，表示该事务回滚信息的开始位置。 在上面的例子中，事务分别在T1,T2,T3时间执行了三个操作，更新了三个block中的数据，在每个data block中都存在一个ITL，指向undo segment header中的事务表。undo信息分别存放在三个undo block中，undo信息是一个链表结构，而undo segment header中的uba则指向了最后一个undo block，这也是回滚的起始位置。如果事务需要回滚，只需要在undo segment header中的事务表中找到事务回滚的起始位置，然后通过undo链表，就可以依次回滚整个事务。

细心的DBA一定会发现，在每个data block的ITL中也有一个UBA，实际上这个UBA是指向了该block对应的undo信息的起始位置，这个UBA主要的作用是提供一致性读，因为一 致性读需要通过undo信息来构造一个CR block，通过这个UBA就可以直接定位到block的回滚信息的起始位置，而不再需要通过undo segment header中的事务表。

在传统的undo管理模式中，Oracle对于undo和data block是一视同仁的，他们都需要首先读入到data buffer中进行修改，并都会产生redo信息，修改的过程大致是：产生undo的redo，更改undo block，产生data的redo，修改data block。总之redo必须要先于数据被记录下来。当数据库发生crash，可以通过redo日志，恢复data和undo block，然后再通过undo信息去回滚未提交的事务，保证数据的一致性，所以说instance recovery的过程是先前滚，再回滚的过程。

传统的undo管理有弊端，第一是undo信息如果不在data buffer中，必须首先从外部文件中读入；第二是undo的所有变化也必须同时记录redo，在事务提交时被写入到redo log中。Oracle提出了In-Memory UNDO的新特性，将undo信息都存放在内存结构中，缓解传统undo管理中带来的开销。

IMU在shared pool中分配一片内存空间（IMU pool），每个新的事务都会分配一个IMU buffer，它相当于一片事物私有的undo buffer，用来记录undo的信息。Data block中记录了IMU node的起始位置，通过IMU buffer中的信息就可以完成一致读，从而大大提升了效率。

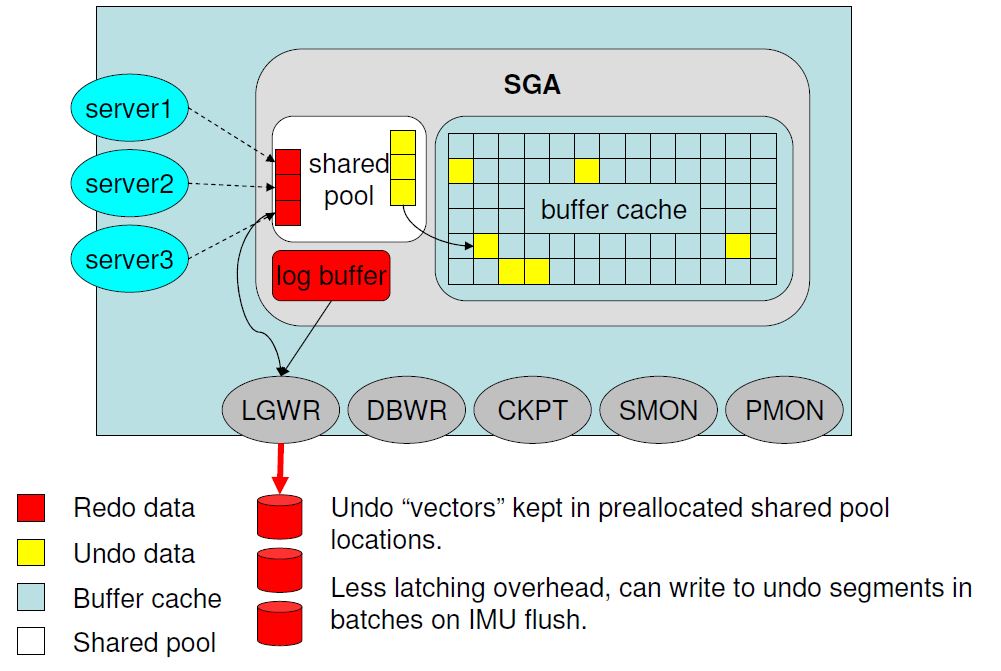


在IMU模式下，undo信息依然会被写入到redo中，理解这点很重要！因为Instance recovery需要undo的信息去回滚未提交的事物，使数据库处于一致状态，如果redo中没有undo变化的信息，那么一旦发生Instance crash，数据库将有可能处于一个不一致的状态。

事务开始依旧会在data block中的分配ITL，并且它依然会指向undo segment header的事物表，但是undo block中的信息并不需要马上写入，这时undo信息是记录在IMU Buffer中的，这时也不会产生undo block的redo信息。在以下两种情况时，undo buffer中的信息会被写入到undo block中：1.IMU buffer空间不足；2.LGWR将redo信息被写入到redo log中时（比如commit），在v$sysstat中可以看到IMU flush和IMU commit，分别表示以上两种情况，如果你发现这两个值不断增加，代表系统开启了IMU特性。

现在我们已经了解到，IMU中的undo信息依旧会被写入到redo log中，只不过在shared pool中分配了一个private undo buffer，一方面可以在内存中完成一致读的操作，另一方面，undo信息只在必要情况下批量写入到redo log中，保证数据库crash后可以恢复到一致状态。另外，Oracle总是会尽可能的保留undo buffer中的信息，以便可以在内存中完成一致读的操作，而且undo信息在写入undo block时，Oracle进行了合并处理，减少了undo block的消耗量和对应的redo产生量。

从Oracle 10g开始，引入了private redo strands的概念，在shared pool中分配了一个private redo buffer的空间，每个事务产生的redo都放在这里（9i是放在PGA里面），每个buffer分配一个redo allocation latch，用来解决9i中redo allocation latch的争用问题。其实IMU和private redo strands这个特性是相关的，IMU相当于private undo buffer，当redo strand或者undo buffer空间不足时，会发生IMU flash事件，将redo信息（包括undo）写入到redo log中。



IMU实现复杂，在很多情况下Oracle会自动禁用IMU特性，比如RAC和Stream环境。