

## 第七次作业，第九章 9.3, 9.4 9.5

### 习题 9.3:

9.3 考虑图 9.8 (a) 的执行日志记录。

- (1) 说明恢复管理器在分析阶段完成的工作（指明分析扫描开始与结束点 LSN，并描述该阶段构造的有关表内容）
- (2) 说明 Redo 阶段完成的工作，指明 Redo 扫描的开始点和结束点。
- (3) 说明 Undo 阶段完成的工作，指明 Undo 扫描的开始点和结束点。

#### 【解答】

- (1) 分析阶段从最近的、记录在日志主记录中的 begin\_checkpoint 开始，并向前扫描，直到日志的最后一条记录。在向前扫描期间，它确定以下内容：
  - a) Redo 阶段的开始处理点；
  - b) 崩溃发生时，缓冲池中的脏页表；
  - c) 崩溃发生时，仍然活跃的（未提交）事务。（以便在 undo 阶段撤销这些事

务已完成动作）

在图 9.8(a) 这个例子中，我们假设首条日志之前，脏页表和事务表都是空的。分析阶段将确定最后有效的 begin\_checkpoint 为 LSN00，对应的 end\_checkpoint 为 LSN10。事务表记录项<transID, lastLSN>；脏页表记录项<pageID, recLSN>。分析阶段将扫描直到日志记录 LSN70，在扫描期间将完成以下事情：

扫描 LSN 20：添加<T1, 20, 'U'>到事务表；添加<P5, 20>到脏页表；

扫描 LSN 30：添加<T2, 30, 'U'>到事务表；添加<P3, 30>到脏页表；

扫描 LSN 40：将事务表中 T2 的状态由'U'改为'C'；

扫描 LSN 50：将事务 T2 从事务表中删除；

扫描 LSN 60：添加<T3, 60, 'U'>到事务表；但不修改脏页表中 P3 对应的页！

扫描 LSN 70：将事务表中<T1, 20, 'U'>修改为<T1, 70, 'U'>

---

---

最后，事务表中包含两个表项：<T1, 70, 'U'>

<T2, 30, 'U'>

脏页表中包含两个表项：<P5, 20>

<P3, 30>

- (2) Redo 阶段紧跟着分析阶段，该阶段要重做崩溃发生时对缓冲池中脏页所做的任何修改。本例中，Redo 阶段从脏页表中最小的 recLSN，即 LSN20 开始，  
LSN 20 修改 P5 动作，被重做，  
LSN 30 修改的 P3，需要从磁盘读取该页以检查它的 pageLSN，如果  $\text{pageLSN} \geq 30$ ，则说明在崩溃前该页已被写盘，不需要重做这个修改 P3 动作，否则，如果  $\text{pageLSN} < 30$ ，则说明在崩溃前该页未被写盘，需要重做这个修改 P3 动作；  
LSN 40, 50 没有动作；  
LSN 60 修改 P3，因为 LSN60 > 脏页表中 P3 页对应的 30，要重做这个动作；  
LSN 70 没有动作。
- (3) Undo 阶段紧跟着 Redo 阶段，该阶段负责撤销崩溃发生时仍活跃事务所做的任何修改。本例中，Undo 从事务表中最大（后）的那条记录，即 LSN 70 开始处理。  
开始时，丢失事务集为 {70, 60}。  
处理 LSN 70：根据事务 prevLSN 字段，增加 LSN20 到丢失事务集。  
—— 丢失事务集变为：{ 60, 20 }  
处理 LSN 60：undo 对 P3 的修改，增加 CLR 记录到日志尾。  
—— 丢失事务集变为：{ 20 }  
处理 LSN20：undo 对 P5 的修改，增加 CLR 记录到日志尾。  
—— 丢失事务集变为：{ null }—处理结束。

#### 习题 9.4

9.4 考虑图 9.8 (b) 的执行日志记录。

- (4) 在图中增加 prevLSN 和 undonextLSN 标示。
- (5) 描述回滚事务 T2 后执行的动作。
- (6) 给出 T2 回滚后的日志 (包括所有 prevLSN 和 undonextLSN)。

### 【解答】

(1) 增加了 prevLSN 和 undonextLSN 标志的图表如下:

LSN	prevLSN	undonextLSN (与CLR对应的ULR)
00	--	--
10	00	00
20	--	--
30	--	--
40	30	(非update记录)
50	20	20
60	50	50
70	60	(非update记录)

- (2) 第一步 从存储在 LSN60 中的旧值, 恢复 P3;
- 第二步 从存储在 LSN50 中的旧值, 恢复 P5;
- 第三步 从存储在 LSN20 中的旧值, 恢复 P5;
- (3) T2 回滚后, 将会在原有日志记录基础上, 添加如下的一些日志记录到日志尾:

LSN	prevLSN	transID	Type	pageID	undonextLSN
80	70	T2	CLR	P3	50
90	80	T2	CLR	P5	20
100	90	T2	CLR	P5	-
110	100	T2	END	-	-

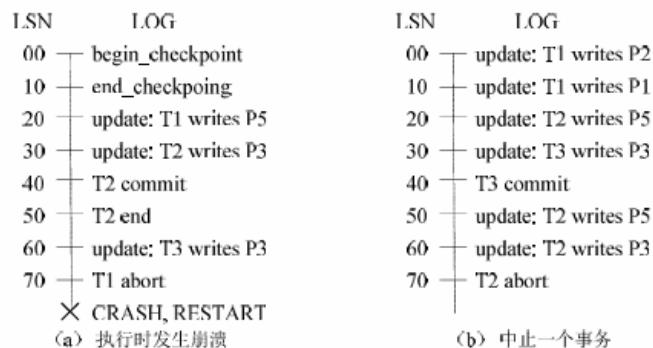


图 9.8 习题 9.3 与习题 9.4 用图

9.5 考虑图 9.9 (a) 的日志片段。另外，在恢复期间当系统写两条日志记录到稳定存储后，再次发生崩溃。之后，在写另外两条日志记录到稳定存储后，又再次发生崩溃。

- (1) 存储在主日志记录中的 LSN 值为多少?
  - (2) 分别描述在三个 ARIES 阶段的工作。
  - (3) 给出最终恢复完成后包含所有非空 prevLSN 和 undonextLSN 的日志记录。
  - (4) 如果系统在恢复期间反复发生崩溃, 那么, 在系统最终成功启动后, 可能写入日志的最大记录数是多少 (提示: 考虑崩溃前的 update 记录数和其它日志记录数)
  - (5) 说明在 begin\_checkpoint 和 end\_checkpoint 记录之间为什么可能有其它日志记录。描述分析阶段如何通过分析这些记录来修改脏页表和事务表内容。
  - (6) 考虑图 9.9 (b), 给出 end\_checkpoint 记录的内容。

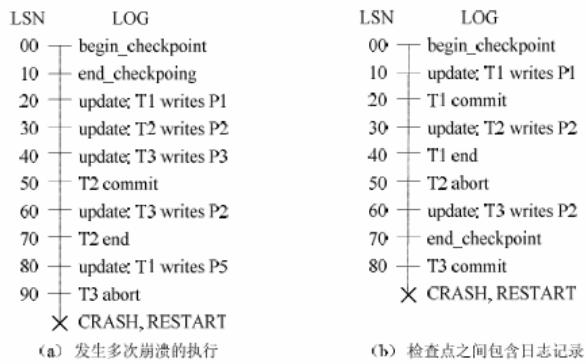


图 9.9 习题 9.5 用图

### 【解答】

- (1) 存储在日志主记录中的 LSN 值为 00。它是最后一个有效 begin\_checkpoint 记录编号。

(2) ARIES 分析阶段工作描述：  
从 LSN00 开始向前扫描，  
遇到LSN 20： 添加(T1, 20, 'U') 到T. T. ; 添加 (P1, 20) 到D. P. T;  
遇到LSN 30： 添加(T2, 30, 'U') 到T. T. ; 添加 (P2, 30) 到D. P. T;  
遇到LSN 40： 添加(T3, 40, 'U') 到T. T. ; 添加 (P3, 40) 到D. P. T;  
遇到LSN 50： 修改T. T. 表中的 (T2, 30, 'U') 为 (T2, 30, 'C');  
遇到LSN 60： 修改T. T. 表中的 (T3, 40, 'U') 为 (T3, 60, 'U');  
遇到LSN 70： 删除T. T. 表中的 (T2, 30, 'C')  
遇到LSN 80： 修改T. T. 表中的 (T1, 20, 'U') 为 (T1, 80, 'U')；添加 (P5, 80) 到D. P. T;  
遇到LSN 90： 无动作；

分析阶段最终获得的事务表(T.T.)	分析阶段最终获得的脏页表(D.P.T.)
(T1, 80, 'U')	(P1, 20)
(T3, 60, 'U')	(P2, 30)
	(P3, 40)
	(P5, 80)

**REDO阶段的工作描述：**从脏页表中最小的记录好LSN20开始向前扫描处理，

遇到LSN 20: 从磁盘检索P1, 比较pageLSN与LSN20:

```
If pageLSN < LSN20 then
    利用LSN20旧值; redo P1;
    End if;
```

遇到LSN 30: 利用LSN30旧值, redo P2;

遇到LSN 40: 利用LSN40旧值, redo P3;

遇到LSN 50: 无动作;

遇到LSN 60: 无动作;

遇到LSN 70: 无动作;

遇到LSN 80: 利用LSN80旧值, redo P5;

遇到LSN 90: 无动作;

**UNDO阶段的工作描述：**从事务表中最大（后）的那条记录，即LSN 80开始处理，

开始时，丢失事务集为{80, 60}。

处理 LSN 80: 根据事务 prevLSN 字段，增加 LSN20 到丢失事务集。

undo 对 P5 的修改，增加 CLR 记录到日志尾。

—— 丢失事务集变为: { 60, 20}

处理 LSN 60: undo 对 P2 的修改，增加 CLR 记录到日志尾。

—— 丢失事务集变为: { 40, 20}

处理 LSN 40: undo 对 P3 的修改，增加 CLR 记录到日志尾。

—— 丢失事务集变为: { 20}

处理 LSN20: undo 对 P1 的修改，增加 CLR 记录到日志尾。

—— 丢失事务集变为: { null}—处理结束。

- (3) 最终恢复完成后包含所有非空 prevLSN 和 undonextLSN 的日志记录如下:

LSN 00	begin checkpoint	
LSN 10	end checkpoint	
LSN 20	update: T1 writes P1	
LSN 30	update: T2 writes P2	
LSN 40	update: T3 writes P3	
LSN 50	T2 commit	prevLSN = 30
LSN 60	update: T3 writes P2	prevLSN = 40
LSN 70	T2 end	prevLSN = 50
LSN 80	update: T1 writes P5	prevLSN = 20
LSN 90	T3 abort	prevLSN = 60
LSN 100	CLR: Undo T1 LSN 80	undonextLSN=20
LSN 110	CLR: Undo T3 LSN 60	undonextLSN=40
LSN 120, 125	CLR: Undo T3 LSN 40	T3 end.
LSN 130, 135	CLR: Undo T1 LSN 20	T1 end.

- (4) 不妨考虑一种比较极端的情形: 日志记录共有  $n$  条, 每条都是不同未提交事务的 update 记录。这意味着: 在恢复期间, 我们必须为每个 update 动作执行 undo 动作, 需要分别添加一条 CLR 记录, 以及在 CLR 记录之后添加一条 END 记录。因此, 在重启完成之后, 我们要写入的日志记录在最多情况下可能会有  $2n$  条。

- (5) ARIES 检查点主要包括 begin\_checkpoint 和 end\_checkpoint 两条记录, begin\_checkpoint 标志检查点构造开始的时间点, 该时间点的缓冲池脏页和未提交活跃事务情况最终要存储到 end\_checkpoint 记录中。而获取脏页表和事务表信息显然需要一定时间, 也即写 end\_checkpoint 前会有一个时间间隔。如果使用的是动态检查点, 则在这个时间间隔内, 还允许继续执行事务, 这又会产生新的日志记录。因此, 在动态检查点情况下, begin\_checkpoint 和 end\_checkpoint 之间可能会有其它日志记录。

begin\_checkpoint 的 LSN 将被写入日志主记录中。分析阶段, 将从这个 LSN 开始向前扫描, 找到对应的 end\_checkpoint 记录, 利用该记录中存储的信息初始化脏页表和事务表。然后, 再次回到这个 LSN, 向前扫描日志, 逐条分析处理遇到的每条日志记录, 以更新脏页表和事务表。这个扫描需要一直进行到日志尾才结束。

- (6) end\_checkpoint 记录中主要包含写入 begin\_checkpoint 那个时间点的脏页表和事务表信息。与 begin\_checkpoint 只有的日志记录内容没有关系, 因此, 我们只能假设脏页表和事务表都是空表。