日志记录的格式

1. 开始一个事务：<Start T>
2. 提交事务T：<Commit T>
3. 放弃事务T：<Abort T>
4. 更新记录：<T, X, V>
   - 事务T修改了数据库元素X的值，而X原来的旧值是V

undo日志的记载规则

- U_1：如果事务T修改数据库元素X，则更新日志<T,X,V>必须在X的新值写到磁盘前写到磁盘；
- U_2：如果事务T提交，则日志记录<Commit T>必须在事务T改变的所有DB元素已写到磁盘后再写到磁盘。

undo日志的例子

- 引入几个新的操作：
  - **Flush Log**：将内存中的日志记录全部写入磁盘
  - **Output(A)**：将数据对象A的值写入数据库的磁盘
- 同时引入下述几个符号
  - **D-A**：数据库元素A在计算机磁盘中的值
  - **M-A**：数据库元素A在内存缓冲中的值
  - **t**：表示内存变量

undo日志的例子

- 假设有一个事务T，需要将数据库中A和B两个数据对象上的值分别乘以2，其执行流程和数据库中记载的日志信息如图-1所示。（假设A，B的初始值都是8）

图-1 undo日志的例子

	T	t	M-A	M-B	D-A	D-B	undo日志
1					8	8	<Start T>
2	Read(A,t)						
3	t := t*2						
4	Write(A,t)						<T, A, 8>
5	Read(B,t)						
6	t := t*2						
7	Write(B,t)						<T, B, 8>
8	Flush Log						
9	Output(A)						
10	Output(B)						
11							<Commit T>
12	Flush Log						

执行‘Flash Log’操作的目的是为了能够执行后续的Output操作（规则U_1）

执行‘Output’操作的目的是为了能够提交事务T（规则U_2）

确保日志记录：<Commit T>能够及时被写入到日志的磁盘中

使用undo日志的恢复过程

1. 将所有事务划分为两个集合
   - 已提交事务：有<Start T>和<Commit T>
   - 未提交事务：有<Start T>但没有<Commit T>
2. 从undo日志的尾部开始向后(日志头部)扫描整个日志，对于每一条更新记录<T,X,V> 作如下处理：
   - 如果<Commit T>已被扫描到，则继续扫描下一条日志记录（基于规则U_2）
   - 否则，由恢复管理器将数据库中X的值改为V（基于规则U_1）
3. 在日志的尾部为每个未结束事务写入一条日志记录<Abort T>，并刷新日志（Flush Log）

- 在图-1中，根据系统故障发生的不同时机，对于事务T的恢复处理过程可能会不一样。
- 假设系统故障发生在：
  - 第12步(Flush Log)之后
  - 第11步(Commit T)与第12步(Flush Log)之间
    - 日志记录<Commit T>已写到磁盘
    - 日志记录<Commit T>尚未写到磁盘
  - 第10步(Output(B))与第11步(Commit T)之间
  - 第8步(Flush Log)与第10步(Output(B))之间
  - 第8步(Flush Log)之前

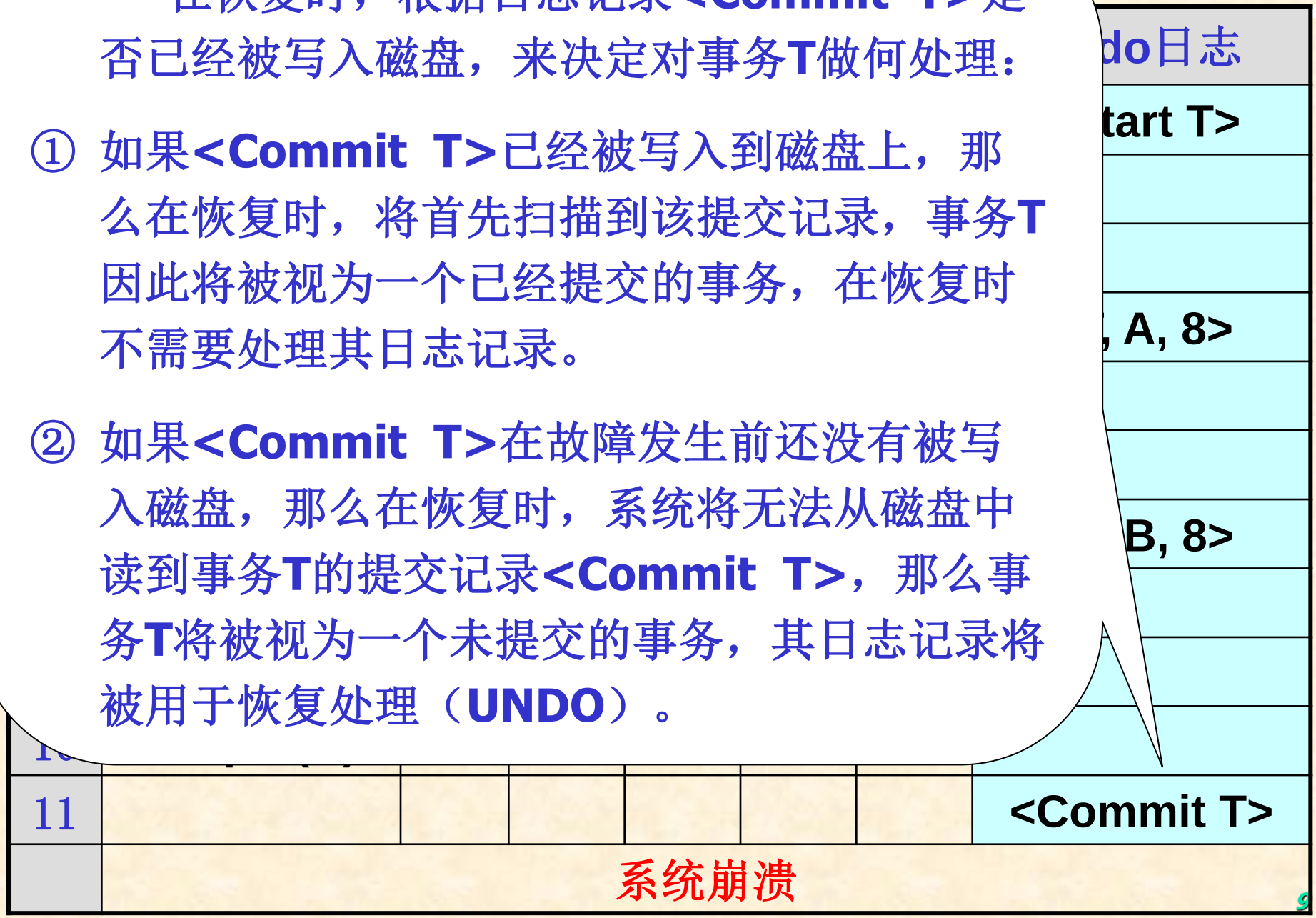
情况1：故障发生在第12步之后

	T	t	M-A	M-B	D-A	D-B	undo日志
1					8	8	<Start T>
2	Read(A,t)	8	8		8	8	
3	t := t*2	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	t := t*2	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush Log						
9	Output(A)		16	16	16	8	
10	Output(B)		16	16	16	16	
11							<Commit T>
12	Flush Log						

在恢复时，根据日志记录<Commit T>是否已经被写入磁盘，来决定对事务T做何处理：

① 如果<Commit T>已经被写入到磁盘上，那么在恢复时，将首先扫描到该提交记录，事务T因此将被视为一个已经提交的事务，在恢复时不需要处理其日志记录。

② 如果<Commit T>在故障发生前还没有被写入磁盘，那么在恢复时，系统将无法从磁盘中读到事务T的提交记录<Commit T>，那么事务T将被视为一个未提交的事务，其日志记录将被用于恢复处理（UNDO）。



虽然事务T已经修改了数据库磁盘上的值，但在第8步执行的‘Flush Log’操作，已经确保与上述修改有关的日志记录被写入了磁盘，因此在恢复时，系统将根据日志中事务T的前像日志对事务T的修改操作做恢复处理

							undo日志
1							<Start T>
2							
3	t := t*2 (UNDO)。	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	t := t*2	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush Log						
9	Output(A)		16	16	16	8	
10	Output(B)		16	16	16	16	
	系统崩溃						

情况4：故障发生在第8步与第10步之间

	T	t	M-A	M-B	D-A	D-B	undo日志
1					8	8	<Start T>
2	Read(A,t)	8	8		8	8	
3	t := t*2	16	8		8	8	
4	Write(A,t)	16	16		8	8	<T, A, 8>
5	Read(B,t)	8	16	8	8	8	
6	t := t*2	16	16	8	8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>
8	Flush Log						
	系统崩溃，不知有没有执行过Output(A)和Output(B)						

恢复原理同前面的情况3.

- 事务T的日志记录是否被写入磁盘是不确定的，但这不影响恢复的结果。因为在第8 步之前，事务T的所有修改操作仅仅发生在内存中，不需要对数据库的磁盘做恢复。
- 即使因为受其它并发事务的影响而导致事务T的某些前像日志已经被写入到磁盘上，那么在恢复时会将其前像值再写回到数据库的磁盘中去，但这并不影响其取值的正确性。

					-A	D-B	undo日志
					8	8	<Start T>?
					8	8	
					8	8	
					8	8	<T, A, 8>?
					8	8	
					8	8	
7	Write(B,t)	16	16	16	8	8	<T, B, 8>?
	在第1步到第8步之间的任何时间点上出现系统崩溃						

- 思考题：如果在恢复过程中再次出现系统崩溃，如何进行系统的故障恢复？

- undo日志的不足
  - 在将事务改变的所有数据写到磁盘前不能提交该事务；
  - 在事务的提交过程中需要执行许多‘写’磁盘操作，从而增加了事务提交的时间开销。
- 在一个事务T被提交后，能否允许将事务T的修改结果暂时保存在内存中，在需要的时候再写入磁盘，以减少磁盘I/O的次数？