Database Design & Applications (SS ZG 518)

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Lecture Session-15

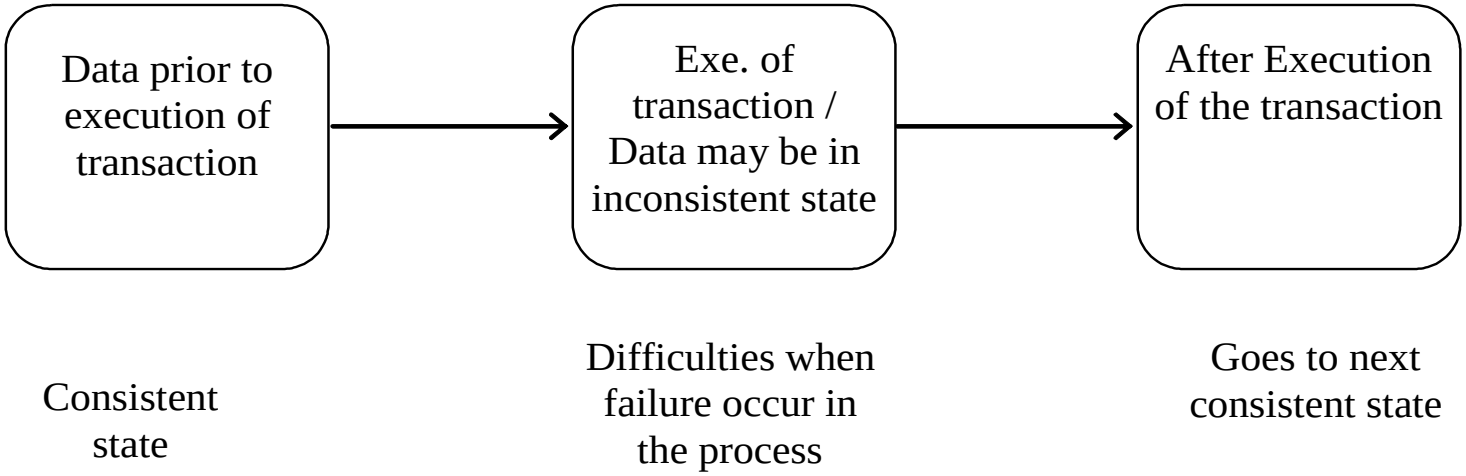
Database Recovery

innovate achieve lead

Contents

- Introduction to Recovery
- Recovery strategies
- Log-based recovery
- Checkpointing
- Shadow paging
- Exercise problems

Introduction to Recovery



As a result of failure of a transaction, the state of the system will not reflect the state of the real world that the database is supposed to capture. We call that state as *inconsistent state*.

When such thing happens we must make sure that the database is restored to its previous consistent which existed before the start of the transaction (which has failed).
This process is known as *recovery process*.

Recovery Techniques

If a transaction *T* performed multiple database modifications, several output operations may be required and a failure may occur after some of these modifications have been made but before all of them are made.

In order to restore to the recent consistent state, we must first write the information describing the modifications to *System log* without modifying the database itself.

This helps us to remove the modification done by a failed transaction.

Now we discuss some important recovery techniques.

Log-based Recovery

Database System Log:

Each *log record* describes a single database write operation and contains the following details.

- Transaction name
- Data item name
- Old value
- New value

Types of log records

$< T_i$ *start*$>$ - indicates transaction T_i started

$<T_i, X_j, V_1, V_2>$ - transaction T_i has performed a write operation on data item X_j and value V_1 before the write and will have value V_2 after the write.

$<T_i$ *commit* $>$ - transaction T_i commits.

With these log records we have the ability to *undo* or *redo* a modification that has already been output to the DB.

I. Deferred Database Modification

This technique ensures atomicity by recording all database modifications(updates) in the log, but deferring (postpone) the actual updates to the database until the transaction commits.

As no data item is written before commit record of the transaction, we need only new value. Hence we perform only *redo* operation.
The *redo* (T_i) operation sets the value of all data items updated by transaction T_i to the new values.

All new values will be found in the log records.

Redoing is needed when we have all modifications on log, and have doubts about successful writing to the DB.

Ex.

Log	Database
< T_1 starts >	
<T_1, A, 900>	
<T_1, C, 800>	
<T_1, commits>	
	A = 900
	C = 800
<T_2, start >	
< T_2, B, 700>	
<T_2 commit >	
	B = 700

On failure, a transaction need to be redone if and only if the log contains both *<start>* and *<commit>* records.
Otherwise we don't have to do anything.

Log	Database
< T_1 starts >	
<T_1, A, 900>	
<T_1, C, 800>	
<T_1, commits>	
	A = 900
	C = 800
<T_2, start >	
< T_2, B, 700>	
<T_2 commit >	
	B = 700
<T_3, start >	
< T_3, C, 200>	
//FAIL//	

II. Immediate Database Modification

In this, database modifications to be output to the database while the transaction is still in the active state.

If such is the case for incomplete transactions, on failure, *undo* operation is needed and for committed transactions *redo* may be required.

System Log	Database
< T_1 starts >	
<T_1, A, 600, 900>	
	A=900
<T_1, C, 300, 800>	
	C=800
<T_1, commits>	
<T_2, start >	
< T_2, B, 400, 700>	
	B = 700
<T_2 commit >	

System Log	Database
< T_1 starts >	
<T_1, A, 600, 900>	
	A=900
<T_1, C, 300, 800>	
	C=800
<T_1, commits>	
<T_2, start >	
< T_2, B, 400, 700>	
	B = 700
<T_2 commit >	
<T_3, start >	
< T_3, C, 100, 200>	
	C=200
//FAIL//	

Checkpointing

In case of failure, the log needs to be searched to determine the transactions that need to be *redone* or *undone*.

But this searching is time consuming and most of the time the algorithm will redo the transactions which actually written their updates to the DB, redoing them is waste of time.

In order to reduce these types of overheads *check pointing* is helpful.

< T_1 start>
< T_1, D, 20>
< T_1 Commit>
[check point]
< T_4 start>
< T_4,B, 12>
< T_4, A, 20>
< T_4 Commit>
< T_2 start>
< T2,B, 15>
< T_3 start>
< T3,A , 35>
< T2,D, 25>

T2 and T3 are ignored because they did not reach their commit point.

T4 is redone as its commit occurred after latest check pointing.

T1 committed before the latest checkpointing hence no action.

Sequence of actions in checkpointing

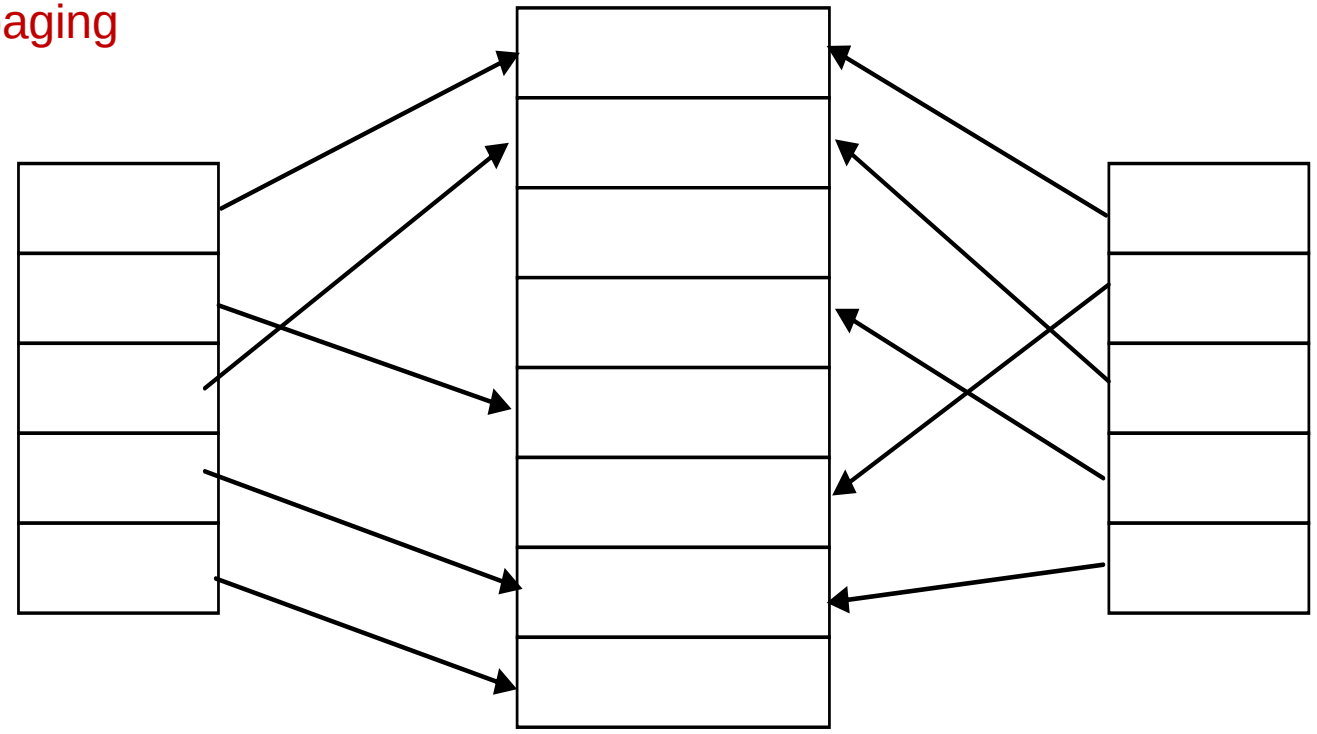
- Output all log records currently in main memory onto stable storage
- Output all modified buffer blocks to the disk.
- Output log record <*check point*> on to stable storage.
- During the recovery process the *redo* / *undo* operations for the transactions will be considered which occur after or just before latest <*check point*> record on log.

Shadow paging

This technique is an alternative to log-based recovery method. We know that the database is partitioned into same fixed length blocks called as pages. These pages need not be in a particular order on disk. A page table is used to find the location of i^{th} block.

In shadow paging technique, two page tables are used. The first is *current page table* and the second is *shadow page table*. When a transaction starts both page tables are identical. The shadow page table is never changed during the execution of the transaction. The current page table may be changed when a transaction performs a write operation. All input and output operations use current page table. When a block page is modified it is written onto different location on the disk and the old block which contains older values exist and can be accessed using the shadow page table.
This is sufficient to recover from the failure.
This technique doesn't require a log and no redo/undo operations are needed.

Shadow paging



Ex 1: Consider the following Log records for the transactions- T1, T2, T3, T4, and T5.

[T1, start]; [T1, A, 30, 40]; [T1, commit]; [T2, start]; [T3, start]; [T2, B, 500, 50]; [T3, D, 20, 10]; [T3, commit]; [checkpoint]; [T4, start]; [T4, P, 12, 87]; [T2, C, 110, 660]; [T4, commit]; [T5, start]; [T5, S, 220, 400]; //***System Crash***//.

For the above sequence of log records inserted in the same order, suggest the recovery actions on system crash for each of the transaction. The recovery strategy used is immediate modification technique.

<start, T1>

< T1, A, 30, 40>

<Commit, T1>

<start, T2>

<start, T3>

<T2, B, 500, 50>

<T3, D, 20, 10>

<Commit, T3>

<Checkpoint>

<Start, T4>

<T4, P, 12, 87>

<T2, C, 110, 660>

<Commit, T4>

<Start, T5>

<T5, S, 220, 400>

====SYSTEM CRASH====

Ex 2: The schedule and the corresponding log records are shown below. We have four transactions- T1, T2, T3 and T4. Suppose that the *immediate update* Recovery technique is adopted with *checkpointing*. Now describe the recovery process after the crash. Specify which transactions need to be rolled-back, redone and undone on crash. Give your reasoning.

<start, T1>
<Read, T1, A>
<write, T1, A, 30, 45>
<Commit, T1>
<start, T2>
<write, T2, D, 20, 79>
<checkpoint>
<start, T3>
<Commit, T2>
<write, T3, C, 10, 90>
<start, T4>
<write, T4, D, 200, 190>
<Commit, T4>
====SYSTEM CRASH====

Ex 3: **Log** **Database**

< T_1 starts >
<T_1, P, 200, 400>
< T_2, starts >
<T_2, C, 500, 300>
< T_3, starts >
<T_2, commits>
< T_1, B, 700, 100>
<T_3, T, 34, 120>
<T_3 commit >
<T_1, D, 12, 67>
//System crash

What is the recovery action – (a) in case of immediate modification and (b) deferred modification techniques

Summary

- ✓ The importance of the recovery mechanism in a DBMS
- ✓ Various recovery strategies
- ✓ Log-based recovery scheme
- ✓ How Deferred and Immediate modification techniques work
- ✓ The concept of Checkpointing in recovery
- ✓ How Shadow paging recovery technique works