

# Crash recovery

Hogeschool Rotterdam  
Rotterdam, Netherlands

## Lecture topics

- The log.
- Analysis phase.
- Redo phase.
- Undo phase.

## Transactions vs Crash recovery

- Transaction manager only grants *Consistency* and *Isolation* properties.
- We have not seen the case of an aborting transaction.
- If a transaction aborts we have to undo everything. This grants *Atomicity*.

## Reasons

- We have to grant *Consistency*.
- DBMS malfunctions after `Commit` operations.
- We have to redo everything the transaction committed.

## Cause of malfunctions

- Hardware failure
- Power grid failure
- Flooding
- Nuclear holocaust
- ...
- We must grant that the data is not lost



## Overview

- Algorithm for crash recovery.
- Three phases:
  - **Analysis:** tracks uncommitted data and active transactions during the crash.
  - **Redo:** repeats all the actions to rebuild a valid state of the DB before the crash.
  - **Undo:** cancel all the actions that were not committed at the crash.

## Algorithm principles

- **Write-ahead logging:** keep track of the actions before you do them.
- **Repeating history:** After restarting retrace all the actions before the crash to bring back the DB to a consistent state.
- **Logging undo:** keep track of the undo actions before you do them (crash during restart).

## Overview

- It is a history of the operations on the database.
- Each entry is called *log record*.
- Each record contains a unique id (*Log Sequence Number* - LSN), a type (kind of operation), and extra info.
- *Log tail* partially maintained in main memory (RAM).
- Periodically stored into persistent memory.



## Update log record

- Used when a transaction modifies (i.e. writes) an object.
- Add the record to the log tail.
- The log record contains the transaction id, the object modified, the old value, and the updated value.

Update					
LSN	Type	TransID	Object	OldValue	NewValue

## Commit log record

- Add the record to the log tail
- Force writing the log tail to permanent storage.
- The transaction is considered committed when the log tail is written successfully (handle crashes while writing the commit).
- The log record contains the transaction id that committed.

Commit		
LSN	Type	TransID

## Abort log record

- Add the record to the log tail
- Start undo phase for that transaction (see slides about undo phase).
- The log record contains the transaction id that committed.

Abort		
LSN	Type	TransID

## End log record

- Add the record to the log tail
- Written after extra actions of Commit or Abort are successfully executed.

End		
LSN	Type	TransID

## Compensation log record

- Add the record to the log tail.
- Added after an undo operation is executed.
- It contains the type of the undo operation, and the LSN of the next record to be undone.

CLR			
LSN	Type	UndoType	NextLSN

## Checkpoint

- Snapshot of the DBMS state.
- Used to reduce the amount of work during a restart.
- Insert a *BeginCheckpoint* record in the log.
- Save the infos on active transactions and the dirty objects (i.e. written but uncommitted objects).
- Insert a *EndCheckpoint* record in the log after this phase.
- Inexpensive: the system does not write the state, it writes the info to rebuild the state.

BeginCheckpoint	
LSN	Type

EndCheckpoint			
LSN	Type	TransactionTable	DirtyObjects

## Logging active transactions

- **Transaction id:** the name of the transaction.
- **LastLSN:** the LSN of the most recent log for this transaction.
- **Status:** In progress (P), Committed (C), or Undone (U).

## Logging dirty objects

- **Object id:** The name of the modified object/variable.
- **RecLSN:** LSN of the first record that caused the object to become dirty.
- If possible (only if committed) the DBMS periodically writes to disk the dirty objects.
- When the objects are written to the disk they are removed from the table.



## Example

- Consider the execution of operations on the database below and the initial state below.
- Write the log that must be created for that execution to support crash recovery.

Variable	Value
A	2
B	0

Time	Operation
16:00 PM	T1 writes A + 1
16:01 PM	T2 writes B + 5
16:02 PM	Checkpoint
16:03 PM	Commit T1
16:05 PM	T2 writes A + 3
16:06 PM	T2 writes B - 2
16:07 PM	Commit T2

## Example

- The following is the created log:

0	Update	T1	A	2	3
1	Update	T2	B	0	5
2	BeginCheckpoint				
3	EndCheckpoint	(T1,0,P),(T2,1,P)	(A,0),(B,1)		
4	Commit	T1			
5	End	T1			
6	Update	T2	A	3	6
7	Update	T2	B	0	3
8	Commit	T2			
9	End	T2			

## Assignment

- Consider the execution of operations on the database below and the initial state below.
- Write the log that must be created for that execution to support crash recovery.

Variable	Value
A	2
B	0
C	3

Time	Operation
10:00 AM	T1 writes A - 5
10:02 AM	T2 writes B + 3
10:03 AM	Commit T1
10:05 AM	T2 writes A + 2
10:06 AM	T3 writes C - 4
10:10 AM	T2 writes A + 1
10:12 AM	Checkpoint
10:14 AM	T2 Commit
10:20 AM	T3 writes A + 3
10:21AM	T3 writes C + 2
10:22AM	T3 Commit

## Analysis phase

- We need a point in the log to start from.
- The latest checkpoint is the point where we could have a valid state of the DBMS.
- Start from the latest checkpoint.
- Scan forward the log.
- From simplicity we assume that no record is written between the start and end checkpoint logs (the operation is atomic and never fails).

## Analysis phase

- If we find an end log for  $T$ , we remove  $T$  from the active transactions
- If we find a log record different from an end log, we add transaction  $T$  to the active transactions if not there.
  - Set  $\text{LastLSN}$  for  $T$  to be the current LSN.
  - If the log record is a Commit change the state into  $C$ , otherwise into  $U$ .
- If we find an update log affecting object  $A$ , and  $A$  is not among the dirty objects, we add  $A$  to the dirty object and set  $\text{RecLSN}$  to the current LSN.

## Example

- Given the log below, show the active transaction table, and the dirty object table after analysing each log record.

0	Update	T1	A	2	3
1	Update	T2	B	0	5
2	BeginCheckpoint	(T1,0,P),(T2,1,P) (A,0),(B,1)			
3	EndCheckpoint				
4	Commit				
5	End	T1			
6	Update	T2			
7	Update	T2	D	1	3
Crash, restart					

## Example

- The following tables are the active transaction table and the dirty object table at each step:

Active transactions			
LSN	TransactionId	LastLSN	Status
4	T1	4	C
	T2	4	P
5	T2	5	P
6	T2	6	P
7	T2	7	P

Dirty Objects		
LSN	Object	RecLSN
4 - 9	A	0
	B	1
6 - 7	A	0
	B	1
	C	6
	D	7

## Redo Phase

- Redo all the updates of all the transactions in the active transaction table.
- Find the smallest among all RecLSN of all the objects.
- This phase redoes also all the CLR's (see undo phase).
- In this phase we assume that we maintain a ObjectLSN used after each redo operation on an object.



## Redo Phase

- Each action must be redone unless one of the following rules is satisfied:
  - 1 The affected object is not dirty.
  - 2 The affected object is dirty, but RecLSN is greater than the LSN of the current log record.
  - 3 The ObjectLSN is greater than or equal to the LSN of the log record.

## Redo Phase: Rule 1

- The first rule means that the object has been written to disk.
- It happens when there is a crash after a checkpoint and the object was added in the dirty object table at that checkpoint.
- The page might have been written to disk but we have gone back before the checkpoint.

## Redo Phase: Rule 2

- The first rule means that the object is still in the dirty object table but it was later written to disk.
- It happens when there is a crash after a checkpoint and the object was added in the dirty object table at that checkpoint.
- The page might have been written to disk but we have gone back before the checkpoint.

## Redo Phase: Rule 3

- The third rule requires to access the dirty object table
- It might happen when there is a crash during a redo phase which successfully redid some of the operations.
- This condition alone is sufficient also for rules 1 and 2.
- It is an expensive operations because we have to access to the disk. Better check also rule 1 and rule 2 that do not require this.

## Redo Phase: redoing operations

- The logged operation is re-applied.
- The ObjectLSN is set to the LSN of the log record that was re-applied.

## Example

- Given the log used in the analysis phase and the generated tables, determine the log from which the redo phase start and what operations are affected. Motivate the answer.
- The smallest  $\text{RecLSN} = 6$ .
- The update on C is redone, because  $\text{SLN} \leq \text{RecLSN}$ .
- The update on D is redone, same reason.

## Undo Phase

- Start from the transaction with largest LastLSN.
- For each transaction do the following
  - Write a CLR setting NextLSN to the LSN of the action of the operation on this transaction before this log record.
  - If it does not exist, set it to null.
  - Undo the operation.
- If the action is a CLR:
  - if NextSLN is not null, repeat the undo on that operation.
  - if it is null add an end record for the transaction because it has completely undone.

## Aborting transactions

- Aborting transactions is just like a system crash.
- Consider the entries in the table just for the aborting transaction.
- Apply the undo phase for that transaction.



## Assignment

- Using the log and the tables built in the analysis phase, write an updated log by inserting the appropriate CLR added during the Undo phase.