

Lecture 8

Logs and Recovery

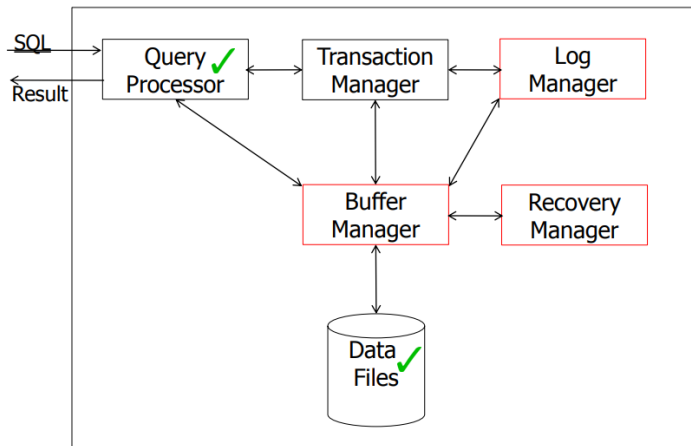
Dr. Caroline Sabty

caroline.sabty@giu-uni.de
Faculty of Informatics and Computer Science
German International University in Cairo

These slides are based on the course of CSEN 604 Data Bases II taught by Dr. Wael Abouelsaadat that are based on the book of Database Systems; the Complete Book

What will we learn in this lecture?

- Logging and recovery features in a DB engine
- Three logging techniques supported by database engines: undo, redo and undo/redo



- Would like data to be “accurate” or “correct” at all times

EMP

<u>Name</u>	<u>Age</u>
White	52
Green	38
Gray	49



<u>Name</u>	<u>Age</u>
White	52
Green	3421
Gray	1



- Predicates data must satisfy
- Examples:
 - x is key of relation R
 - $x \rightarrow y$ holds in R
 - $\text{Domain}(x) = \{\text{Red, Blue, Green}\}$
 - α is valid index for attribute x of R
 - no employee should make more than twice the average salary

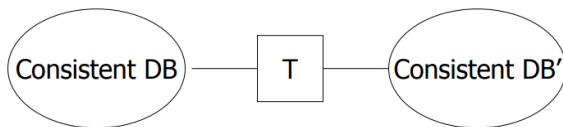
- **Consistent state:**
satisfies all constraints
- **Consistent DB:**
all tables are in consistent state all the time

Constraints may not capture "full correctness"

Example constraints coded in SQL or in client programming language (business logic constraints)

- When salary is updated on a promotion, shouldn't it be $\text{new salary} > \text{old salary}$
- When account record is cancelled, shouldn't it be $\text{balance} = 0$

- An encapsulation mechanism for SQL statements
- List of actions (i.e. SQL) that preserve consistency



If T starts with consistent state +
 T executes in isolation
 $\Rightarrow T$ leaves consistent state

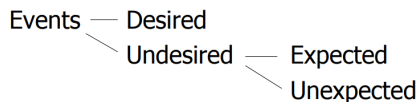
- If we stop running transactions,
DB left consistent
- Each transaction sees a consistent DB

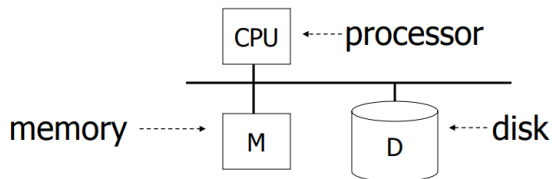
- Transaction bug
- DBMS bug
- Hardware failure
e.g., disk sector failure alters balance of account
- Data sharing logical bug
e.g.: T1: give 10% raise to programmers
T2: change programmers \Rightarrow systems analysts

How can we prevent/fix violations?

- Chapter 17: due to failures only
- Chapter 18: due to data sharing only
- Chapter 19: due to failures and sharing

First order of business: Failure Model





- Desired events: what is intended and coded in SQL
- Undesired expected events:
 - System crash due
 - memory damaged
 - CPU halts, resets
 - Current goes off

- Desired events: what is intended and coded in SQL
- Undesired expected events:
 - System crash due
 - memory damaged
 - CPU halts, resets
 - Current goes off

that's it!!

Undesired Unexpected: Everything else!

Examples:

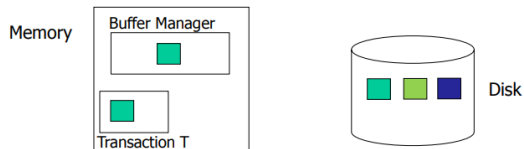
- Disk data is lost/stolen
- Server power supply implodes wiping out universe

Is this model reasonable?

Approach: Add low level checks +
redundancy to the increase
the probability that the server holds

E.g., { Replicate disk storage (stable store)
Memory parity
Server check daemons

Storage hierarchy



Operations between buffer manager and disk:

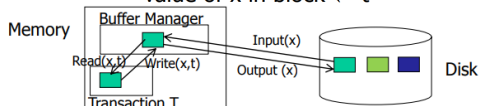
- Input (x): block containing x \rightarrow memory
- Output (x): block containing x \rightarrow disk

Operations between buffer manager and disk and then between transaction and buffer manager:

- Input (x): block containing $x \rightarrow$ memory
- Output (x): block containing $x \rightarrow$ disk
- Read (x,t): do input(x) if necessary
 $t \leftarrow$ value of x in block
- Write (x,t): do output(x) if necessary
 value of x in block $\leftarrow t$

Storage hierarchy

- Input (x): block containing x \rightarrow memory
- Output (x): block containing x \rightarrow disk
- Read (x,t): do input(x) if necessary
t \leftarrow value of x in block
- Write (x,t): do output(x) if necessary
value of x in block \leftarrow t



Example

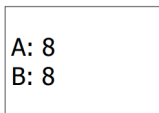
Constraint: $A=B$

$T_1: A \leftarrow A \times 2$

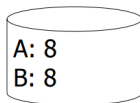
$B \leftarrow B \times 2$

```
start transaction;  
  update TableR set A = A * 2, B = B * 2;  
end transaction;
```


T₁: Read (A,t); $t \leftarrow t \times 2$
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);

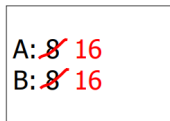


memory

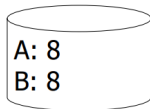


disk

T₁: Read (A,t); $t \leftarrow t \times 2$
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);

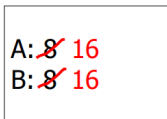


memory

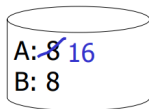


disk

T1: Read (A,t); $t \leftarrow t \times 2$
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
Output (A);
Output (B); failure!

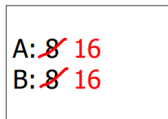


memory

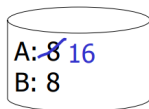


disk

Need atomicity: execute all actions of a transaction or none at all



memory



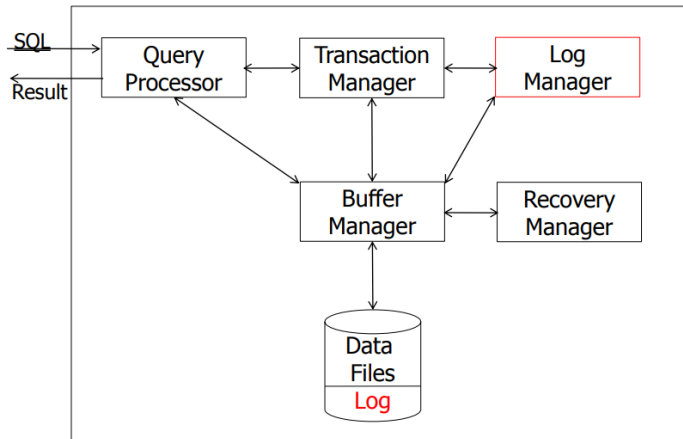
disk

Solution: keep a log to track

Which transaction started?

What did it do? (or what it is going to do?)

Which transaction finished?



Log Commands:

<Start T>

log the start of transaction T

<T, X, value>

log that T (transaction identifier) modified X (database record) affecting value (value)

<COMMIT T>

log the completion of transaction T

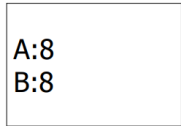
```

<START T1>
<T1,A,5>
<START T2>
<T2,B,10>
<T2,C,15>
<T1,D,20>
<COMMIT T1>
<COMMIT T2>
<START T3>
<T3,E,25>
<T3,F,30>
    
```

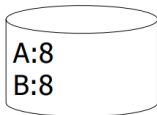
- Undo logging: makes repairs to the database state by undoing the effects of transactions that may not have completed before the crash (brings database to an old consistent state)
- Redo logging: ignores incomplete transactions and repeats the changes made by committed transactions
- Undo/Redo

Undo logging (Immediate modification)

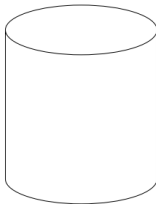
T1: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



memory



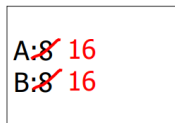
disk



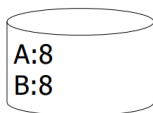
log

Undo logging (Immediate modification)

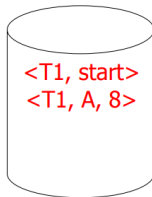
T₁: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



memory



disk



log

Undo logging (Immediate modification)

T1: Read (A,t); $t \leftarrow t \times 2$ A=B

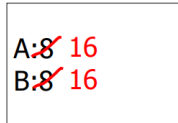
Write (A,t);

Read (B,t); $t \leftarrow t \times 2$

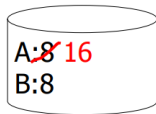
Write (B,t);

Output (A);

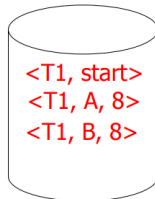
Output (B);



memory



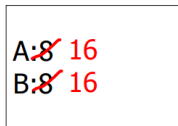
disk



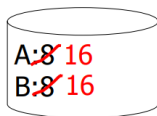
log

Undo logging (Immediate modification)

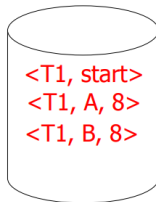
T1: Read (A,t); $t \leftarrow t \times 2$ $A = B$
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



memory



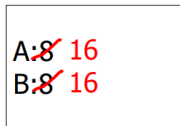
disk



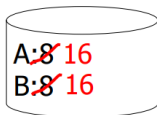
log

Undo logging (Immediate modification)

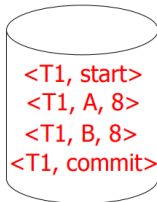
T1: Read (A,t); $t \leftarrow t \times 2$ $A = B$
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



memory



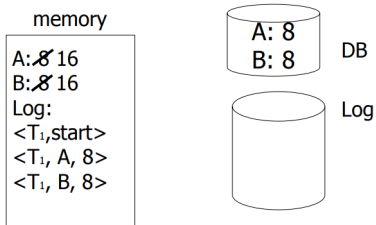
disk



log

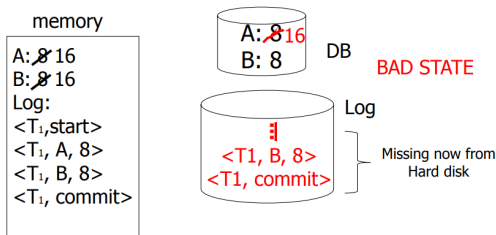
One “complication”

- Log is first written in memory
- Not written to disk on every action



One "complication"

- Log is first written in memory
- Not written to disk on every action



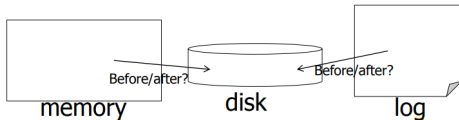
Undo Log Steps:

U1:

If transaction T modifies X, then the log record $\langle T, X, v \rangle$ must be written to disk **before** the new value of X is written to disk

U2:

If a transaction commits, then its COMMIT log record must be written to disk only **after** database record written to disk.



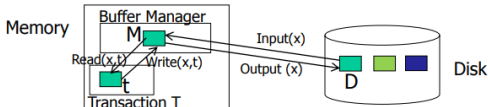
Undo Log Example:

$A := A * 2;$
 $B := B * 2;$

U1: If transaction T modifies X, then the log record $\langle T, X, v \rangle$ must be written to disk **before** the new value of X is written to disk

U2: If a transaction commits, then its COMMIT log record must be written to disk only **after** database record written to disk.

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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	16	8	
10)	OUTPUT (B)	16	16	16	16	16	
11)							<COMMIT T>
12)	FLUSH LOG						



Undo Log: what if a crash happens?

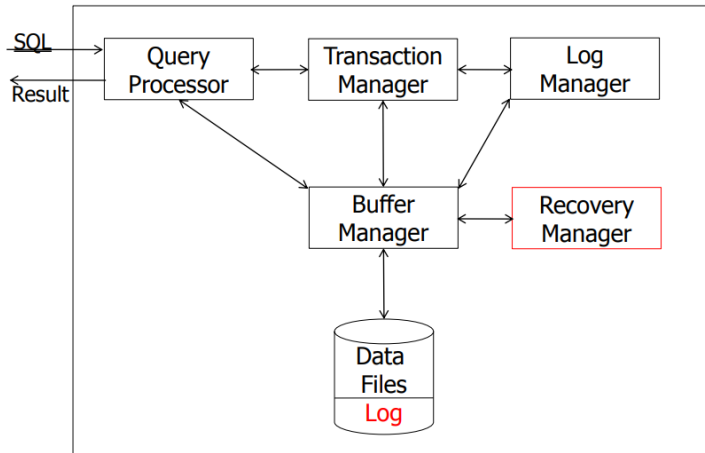
$A := A * 2;$
 $B := B * 2;$

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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	16	8	
10)	OUTPUT (B)	16	16	16	16	16	
11)							<COMMIT T>
12)	FLUSH LOG						

Undo Log: what if a crash happens?

$A := A * 2;$
 $B := B * 2;$

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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	16	8	
10)	OUTPUT (B)	16	16	16	16	16	
11)							<COMMIT T>
12)	FLUSH LOG						



Recovery rules: Undo logging

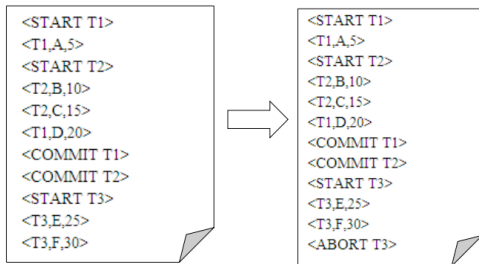
- 1) If T is a transaction whose COMMIT record has been seen, then do nothing. (T is committed and must not be undone)
- 2) Otherwise, T is an incomplete transaction, or an aborted transaction. The recovery manager change the value of X in the database to v (old value)

Undo Log: what if a crash happens?

$A := A * 2;$
 $B := B * 2;$

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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	16	8	
10)	OUTPUT (B)	16	16	16	16	16	
11)							<COMMIT T>
12)	FLUSH LOG						

Undo Log Example



Undo Log: how far to recover?

Real Problem!

Undo log file could contain Mn of records/lines

Need to check all!!

- ① Stop accepting new transactions
- ② Wait until all currently active transactions commit or abort and have written a COMMIT or ABORT record on the log
- ③ Flush the log to disk
- ④ Write a log record <CKPT>, and flush the log again
- ⑤ Resume accepting transactions

Undo Log: how far to recover?

Solution: insert checkpoints in log file

How it works?

- 1) Queue new transactions
- 2) Wait until all running transactions commit
- 3) Flush the log
- 4) Write a log <CKPT>
- 5) Resume accepting transactions

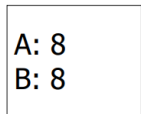
Undo Log with CheckPoint Example

```

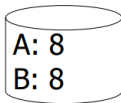
<START T1>
<T1,A,5>
<START T2>
<T2,B,10>
<T2,C,15>
<T1,D,20>
<COMMIT T1>
<COMMIT T2>
<CKPT> ←
<START T3>
<T3,E,25>
<T3,F,30>
    
```

Redo logging (deferred modification)

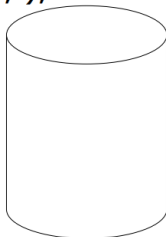
T₁: Read(A,t); $t \leftarrow t \times 2$; write (A,t);
 Read(B,t); $t \leftarrow t \times 2$; write (B,t);
 Output(A); Output(B)



memory



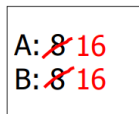
DB



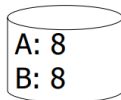
LOG

Redo logging (deferred modification)

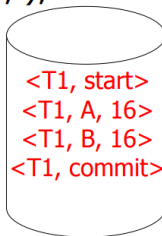
T1: Read(A,t); $t \leftarrow t \times 2$; write (A,t);
 Read(B,t); $t \leftarrow t \times 2$; write (B,t);
 Output(A); Output(B)



memory



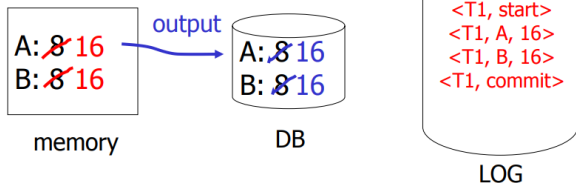
DB



LOG

Redo logging (deferred modification)

T1: Read(A,t); $t \leftarrow t \times 2$; write (A,t);
 Read(B,t); $t \leftarrow t \times 2$; write (B,t);
 Output(A); Output(B)



Redo Log Rule:

R1:

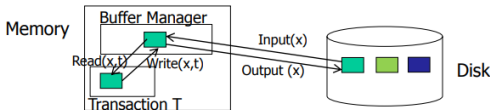
Before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X , including **both** the **update** record $\langle T, X, v \rangle$ and the **<COMMIT T>** record, must appear on disk

Redo Example:

$A := A * 2;$
 $B := B * 2;$

R1: Before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X, including both the update record $\langle T, X, v \rangle$ and the $\langle \text{COMMIT } T \rangle$ record, must appear on disk

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							$\langle \text{START } T \rangle$
2)	READ(A,t)	8	8		8	8	
3)	$t := t * 2$	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	$\langle T, A, 16 \rangle$
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	$\langle T, B, 16 \rangle$
8)							$\langle \text{COMMIT } T \rangle$
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	



Redo Log: what if a crash happens?

A := A * 2;
 B := B * 2;

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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,16>
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,16>
8)							<COMMIT T>
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	

Recovery rules: Redo logging

- (1) Scan log from beginning. For each log record $\langle T, X, v \rangle$ encountered:
 - a. If T is not committed, do nothing
 - b. If T is committed, write value of v for database element X
- (2) For each incomplete transaction T , write an $\langle \text{ABORT } T \rangle$ record to the log

Redo Log: what if a crash happens?

- (1) Scan log from beginning. For each log record $\langle T, X, v \rangle$ encountered:
- If T is not committed, do nothing
 - If T is committed, write value of v for database element X

$A := A * 2;$
 $B := B * 2;$

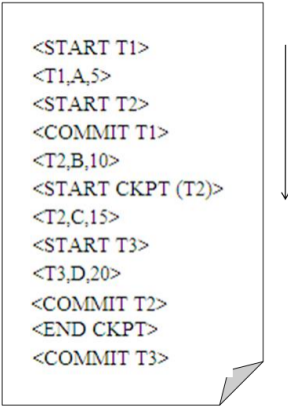
- (2) For each incomplete transaction T , write an $\langle \text{ABORT } T \rangle$ record to the log

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							$\langle \text{START } T \rangle$
2)	READ(A,t)	8	8		8	8	
3)	$t := t * 2$	16	8		8	8	
4)	WRITE(A,t)	16	16		8	8	$\langle T, A, 16 \rangle$
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	$\langle T, B, 16 \rangle$
8)							$\langle \text{COMMIT } T \rangle$
9)	FLUSH LOG						
10)	OUTPUT (A)	16	16	16	16	8	
11)	OUTPUT (B)	16	16	16	16	16	

Redo Log with CheckPoint

- (1) Write a log record <CKPT (T1,...Tk) for all active transactions
- (2) Flush the log
- (3) Write to disk all database elements that were written to memory buffers by transactions that had already committed when <CKPT> was inserted to log
- (4) Write <END CKPT>

Redo Log with CheckPoint Example



```

<START T1>
<T1,A,5>
<START T2>
<COMMIT T1>
<T2,B,10>
<START CKPT (T2)>
<T2,C,15>
<START T3>
<T3,D,20>
<COMMIT T2>
<END CKPT>
<COMMIT T3>
    
```

Key drawbacks:

- *Undo logging*: cannot bring backup DB copies up to date
- *Redo logging*: need to keep all modified blocks in memory until commit

Solution: undo/redo logging!

Update \Rightarrow $\langle T_i, X_{id}, \text{New } X \text{ val}, \text{Old } X \text{ val} \rangle$
page X

- After a crash, the system consults the log to determine which transactions after the last checkpoint need to be redone and which need to be undone
- Transaction T_i needs to be undone if the log contains the record $\langle T_i \text{ start} \rangle$ but does not contain either the record $\langle T_i \text{ commit} \rangle$ or the record $\langle T_i \text{ abort} \rangle$
- Transaction T_i needs to be redone if log contains record $\langle T_i \text{ start} \rangle$ and either the record $\langle T_i \text{ commit} \rangle$ or the record $\langle T_i \text{ abort} \rangle$

Undo/Redo Log Rules:

UR₁:

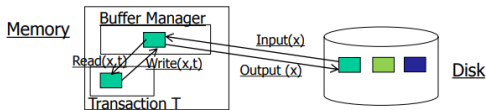
Before modifying any database element X on disk because of changes made by some transaction T, it is necessary that the update record $\langle T, X, \text{old-val}, \text{new-val} \rangle$ appear on disk

Undo/Redo Log Example:

A := A * 2;
B := B * 2;

UR1: Before modifying any database element X on disk because of changes made by some transaction T, it is necessary that the update record $\langle T, X, v, w \rangle$ appear on disk

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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,8,16>
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,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B) FLUSH LOG	16	16	16	16	16	



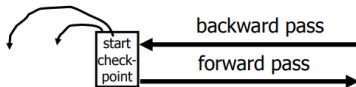
Undo/Redo Log: what if a crash happens?

A := A * 2;
B := B * 2;

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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,8,16>
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,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B)	16	16	16	16	16	
	FLUSH LOG						

Recovery process:

- **Backwards pass** (end of log to latest valid checkpoint start)
 - construct set S of committed transactions
- **Backward Pass**
 - follow undo chains for transactions in (checkpoint active list) - S
- **Forward pass** (latest checkpoint start -> end of log)
 - redo actions of S transactions



Undo/Redo Log: what if a crash happens?

$A := A * 2;$
 $B := B * 2;$

Step	Action	t	M-A	M-B	D-A	D-B	Log
1)							<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,8,16>
5)	READ(B,t)	8	8	8	8	8	
6)	$t := t * 2$	16	16	8	8	8	
7)	WRITE(B,t)	16	16	16	8	8	<T,B,8,16>
8)	FLUSH LOG						
9)	OUTPUT (A)	16	16	16	16	8	
10)							<COMMIT T>
11)	OUTPUT (B)	16	16	16	16	16	
	FLUSH LOG						

- Logging and recovery is an important feature of a database engine which enables it to protect the integrity of the data from undesired expected events.
- Logging involves writing down all changes made by a transaction before actually changing values on disk. The log must be written first to disk.
- There are three logging techniques supported by database engines; undo, redo and undo/redo. Each has its own rules and own recovery policy.

Thank You :)