Week14_Course

Database System
Crash Recovery part1-4

Database System - Nankai

Crash Recovery

- Transactions and Crash Recovery
- Undo logging
- Redo logging
- Undo/Redo logging

Database System - Nankai

Week14_Course

Database System_Crash Recovery part1

Transactions and Crash Recovery

课前预习

Database System - Nankai

Transaction Management Overview

- Major consern so far:
  - How to manipulate database efficiently?
- Now shift focus to:
  - How to ensure correctness of database manipulation?

- Concurrency Control
  - Provide correct and highly available data access in the presence of concurrent access by many users
- Recovery
  - Ensures database is fault tolerant, and not corrupted by software, system or media failure
  - 24x7 access to mission critical data

Database System - Nankai

Transactions

A ***transaction*** is the DBMS's abstract view of a user program

- A sequence of reads and writes of database objects.
- Unit of work that must commit or abort as an atomic unit

- In "ad-hoc" SQL:
  - Default: each statement = one transaction
- In a program:

START TRANSACTION
[SQL statements]
COMMIT or ROLLBACK (=ABORT)

```
DELETE  FROM workson
        WHERE empno= '10102'
SELECT * FROM workson
```

```
START TRANSACTION
    DELETE  FROM workson
            WHERE empno = '10102'
    SELECT * FROM workson
COMMIT [ROLLBACK]
```

Database System - Nankai

Concrete Motivation

Client 1:

```
UPDATE Product
SET Price = Price – 1.99
WHERE pname =  'Gizmo'
```

Client 2:

```
UPDATE Product
SET Price = Price*0.5
WHERE pname=  'Gizmo'
```

Two managers attempt to discount products.
What could go wrong?

Revised Code

Client 1: START TRANSACTION

```
UPDATE Product
SET Price = Price – 1.99
WHERE pname =  'Gizmo'
COMMIT
```

Client 2: START TRANSACTION

```
UPDATE Product
SET Price = Price*0.5
WHERE pname=  'Gizmo'
COMMIT
```

Now it works like a charm

Multiple users: multiple statements

```
Client 1:
    INSERT INTO SmallProduct(name, price)
        SELECT pname, price
          FROM Product
          WHERE price <= 0.99
    DELETE Product
          WHERE price <=0.99

Client 2:
        SELECT count(*)
          FROM Product
        SELECT count(*)
        FROM SmallProduct
```

What’ s wrong ?

START TRANSACTION

COMMIT or ROLLBACK

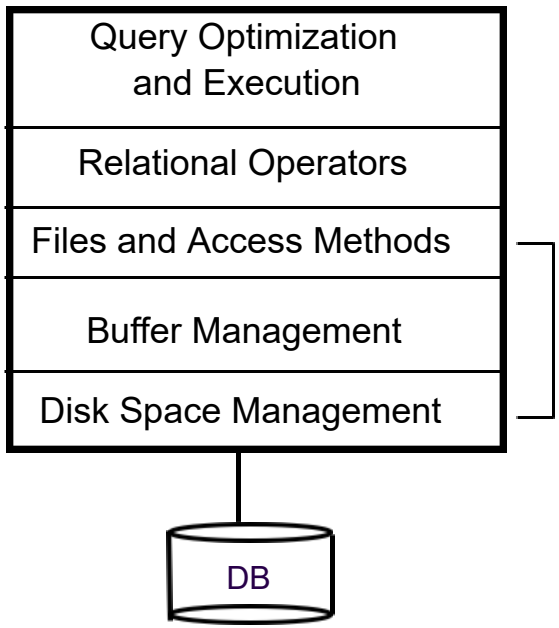
ACID properties of Transaction Executions

- **A** tomicity(原子性): All actions in the transaction happen, or none happen.
- **C** onsistency (一致性): : If each transaction is consistent, and the DB starts consistent, it ends up consistent.
- **I** solation (隔离性): : Execution of one transaction is isolated from that of other transactions.
- **D** urability (持久性): : If a transaction commits, its effects persist.

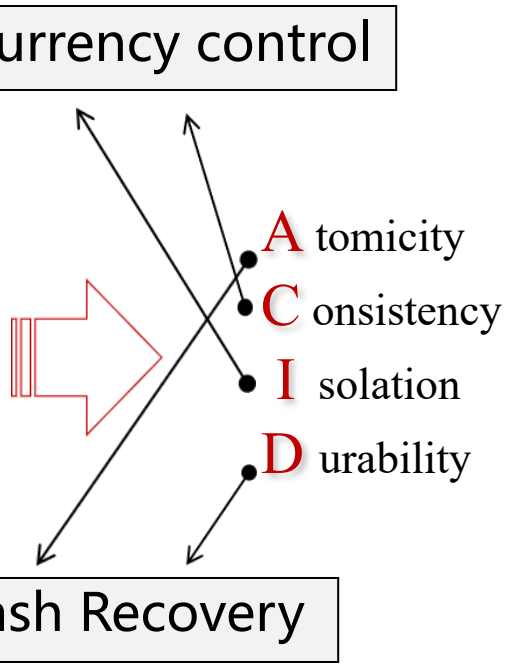
Allows concurrency and recovery!

Database System - Nankai

Structure of a DBMS



These layers must consider concurrency control and recovery (Transaction, Lock, Recovery Managers)



Database System - Nankai

Integrity or correctness of data

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

EMP

Name	Age
White Green Gray	52 3421 1

1—8000 ✓

数据完整性（Data Integrity）：数据的精确性（Accuracy）和可靠性（Reliability），数据语义的正确性（Correctness）

Integrity or consistency constraints

- Predicates that the data must satisfy
- Examples:
  - x is key of relation R
  - x → y holds in R
  - Domain(x) = {Red, Blue, Green}
  - no employee should make more than twice the average salary

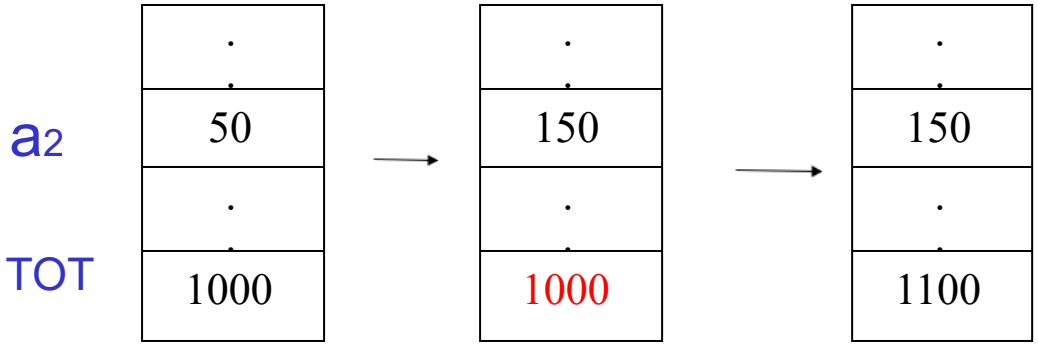
Definition:

- Consistent state: satisfies all constraints
- Consistent DB: DB in a consistent state

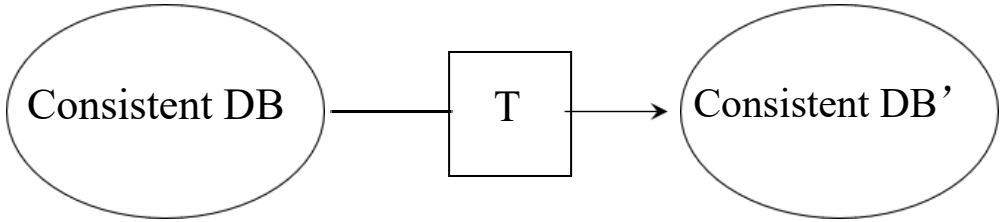
Obs: DB cannot be consistent always!

Example: $a_1 + a_2 + \ldots + a_n = TOT$ (constraint)

Deposit \$100 in a_2: $a_2 \leftarrow a_2 + 100$; $TOT \leftarrow TOT + 100$



Transaction: collection of actions that preserve consistency (≈ process)



SQL: each query or modification statement
Embedded SQL: Sequence of DB operations up to COMMIT or ROLLBACK ("abort")

How can constraints be violated?

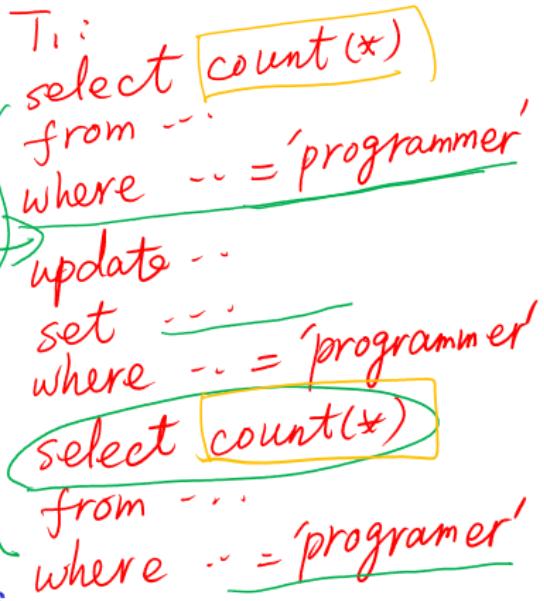
- Transaction bug
- DBMS bug
- Hardware failure

  e.g., disk crash alters balance of account

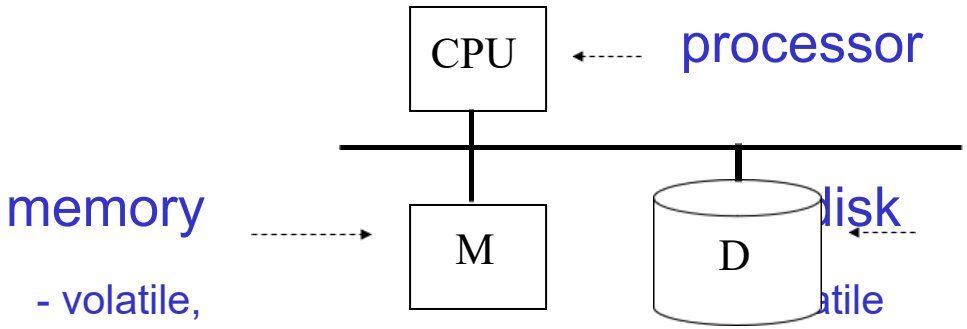
- Simultaneous transactions accessing shared data

  e.g.: T1: give 10% raise to programmers

  T2: change programmers ⟹ systems analysts



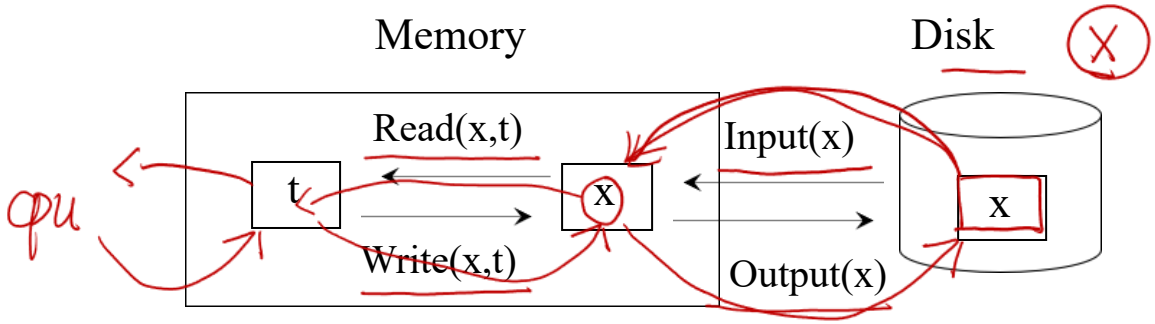
Context of failure model



Database System - Nankai

Interacting Address Spaces:

Storage hierarchy



Input (x): block with x from disk to buffer
Output (x): block with x from buffer to disk

Read (x,t): transaction variable t:= X; (Performs Input(x) if needed)
Write (x,t): X (in buffer) := t

Note on "database elements" X:

- Anything that can have value and be accessed of modified by transactions:
  - a relation (or extent of an object class)
  - a disk block
  - a record
- Easiest to use blocks as database elements

Key problem Unfinished transaction

Example

Constraint: A=B (*)

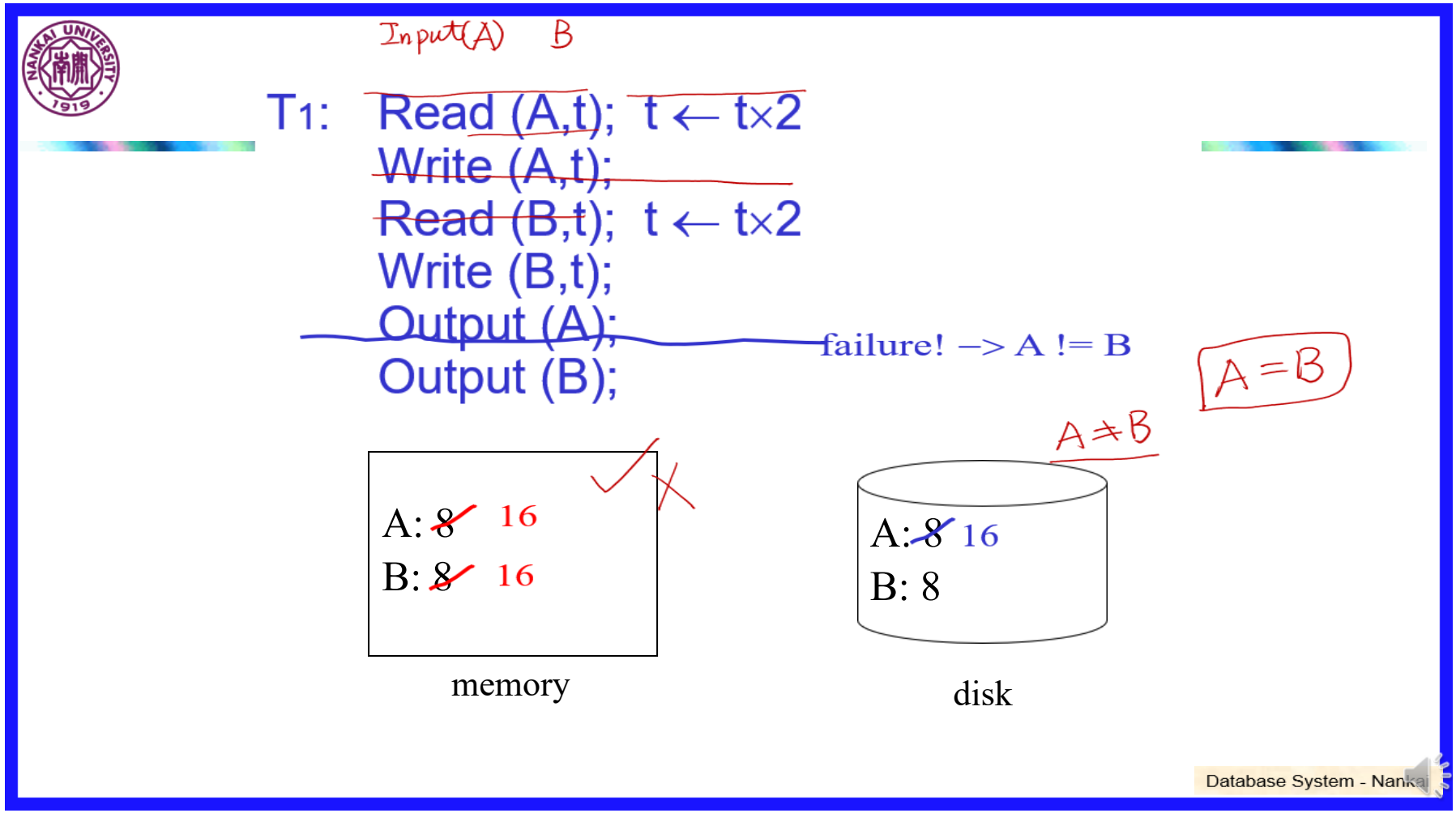
T_1: $A \leftarrow A \times 2$

$B \leftarrow B \times 2$

A=B ?

(*) simplification; a more realistic example: the sum of loan balances = the total debt of a bank

A=B


Input(A) B
T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);
failure! –> A != B
A=B
A≠B
A: 8 16
B: 8 16
memory
A: 8 16
B: 8
disk
Database System - Nankai

Summary – Transaction and CR

- Transaction
  - Atomicity， Consistency， Isolation， Durability
- Crash Recovery
  - 数据完整性（**Data Integrity**）
  - 数据库一致性（**Consistent DB**）

单选题 1分

互动交流一

数据库事务中包括的操作要么都做，要么都不做，这是事务的什么特性?

A 原子性（Atomicity）

B 一致性（Consistency）

C 隔离性（Isolation）

D 持久性（Durability）

提交

Database System - Nankai

单选题 1分

互动交流二

在数据库中，当一个事务提交之后，数据库状态永远的发生了改变，这是事务的什么特性?

A 原子性（Atomicity）

B 一致性（Consistency）

C 隔离性（Isolation）

D 持久性（Durability）

提交

Database System - Nankai

单选题 1分

互动交流三

在数据库中，一个事务的执行不能被其他事务干扰，这是事务的什么特性?

A 原子性（Atomicity）

B 一致性（Consistency）

C 隔离性（Isolation）

D 持久性（Durability）

提交

Database System - Nankai

多选题 1分

互动交流四

数据库事务的四大特性（ACID）包括：

A 绝对性（Absolutely）

B 原子性（Atomicity）

C 并发性（Concurrency）

D 一致性（Consistency）

E 完整性（Integrity）

F 隔离性（Isolation）

G 发展性（Development）

H 持久性（Durability）

提交

se System - Nankai

多选题 1分

互动交流五-不定项选组题

以下哪个命令表示事务的结束?

A Transaction

B Commit

C Rollback

D End Transaction

提交

Database System - Nankai

Week14_Course

Database System_Crash Recovery part2

Undo Logging 课堂讲授

Database System - Nankai

本周课要剖析和解决三个问题

Why database system needs crash recovery technology?

What is the core technique of crash recovery?

课前预习

How to realize crash recovery of database system?

课堂讲授

Database System - Nankai

预习知识回顾

Why database system needs crash recovery technology?

Database system may crash

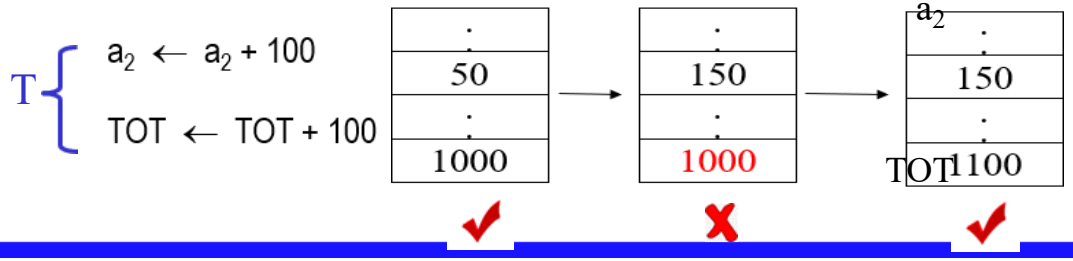
due to software, hardware, computer virus and other failures

The stored data is corrupted

recovery

Resume data to correct state

数据库存储错误的数据、丢失数据、数据不一致



Database System - Nankai

预习知识回顾

What is the core technique of crash recovery?

Resume data to correct state

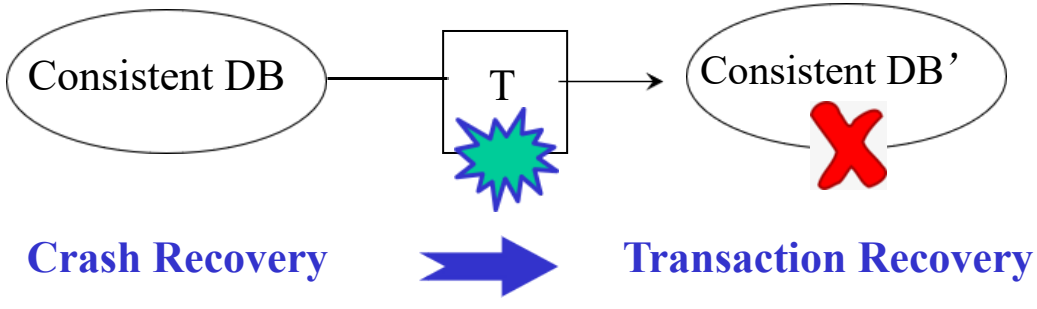
Correct state:

- ✓ 数据的精确性(Accuracy)
- ✓ 数据的可靠性(Reliability)
- ✓ 数据语义的正确性(Correctness)

术语：数据完整性（Data Integrity）

EMP

Name	Age
White	52
Green	3421
Gray	1



Crash Recovery → **Transaction Recovery**

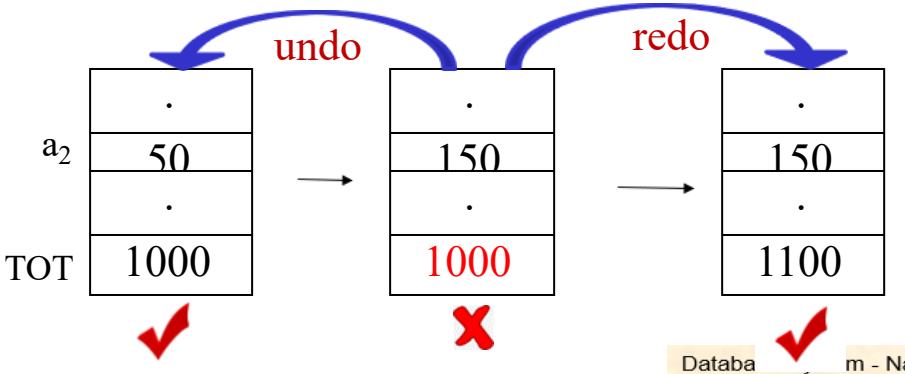
core technique

Database System - Nankai

How to realize crash recovery of database system?

- Undo logging
- Redo logging
- Undo/Redo logging

T { $a_2 \leftarrow a_2 + 100$; $TOT \leftarrow TOT + 100$



Crash Recovery

- Transactions and Crash Recovery
- Undo logging
- Redo logging
- Undo/Redo logging

Database System - Nankai

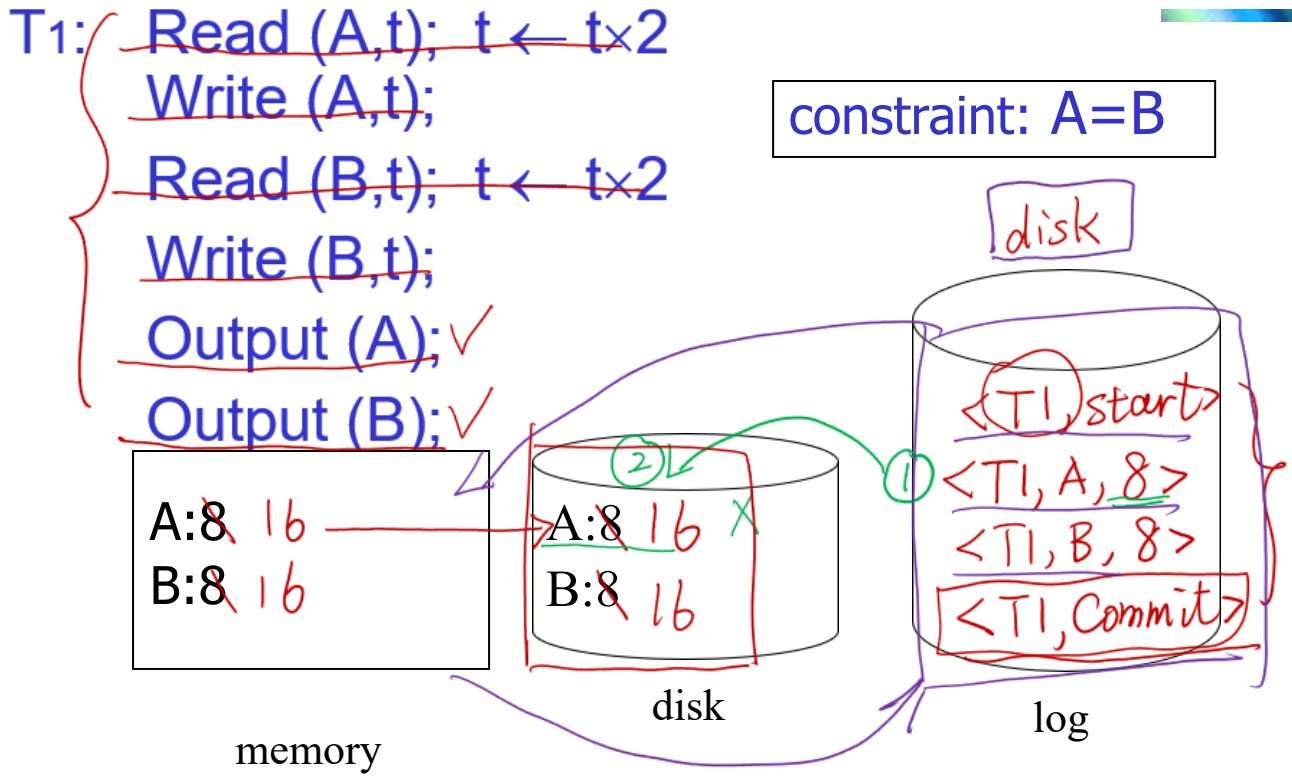
- Need atomicity: either execute all actions of a transaction or none at all

One solution: undo logging (immediate modification)

Log: sequence of log records, recording actions of transactions T_i:

- $<T_i$, start$>$: transaction has begun
- $<T_i$, commit$>$: completed succesfully
- $<T_i$, abort$>$: could not complete succesfully
- $<T_i, X, v>$: T_i has updated element X, whose old value was v

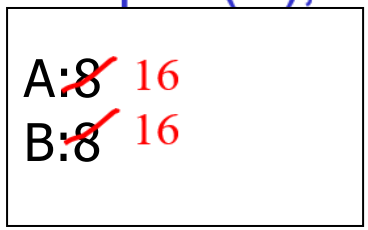
Undo logging (Immediate modification)



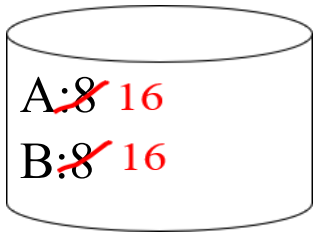
Undo logging (Immediate modification)

T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);

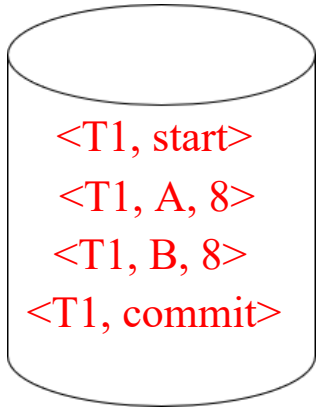
constraint: A=B



memory



disk



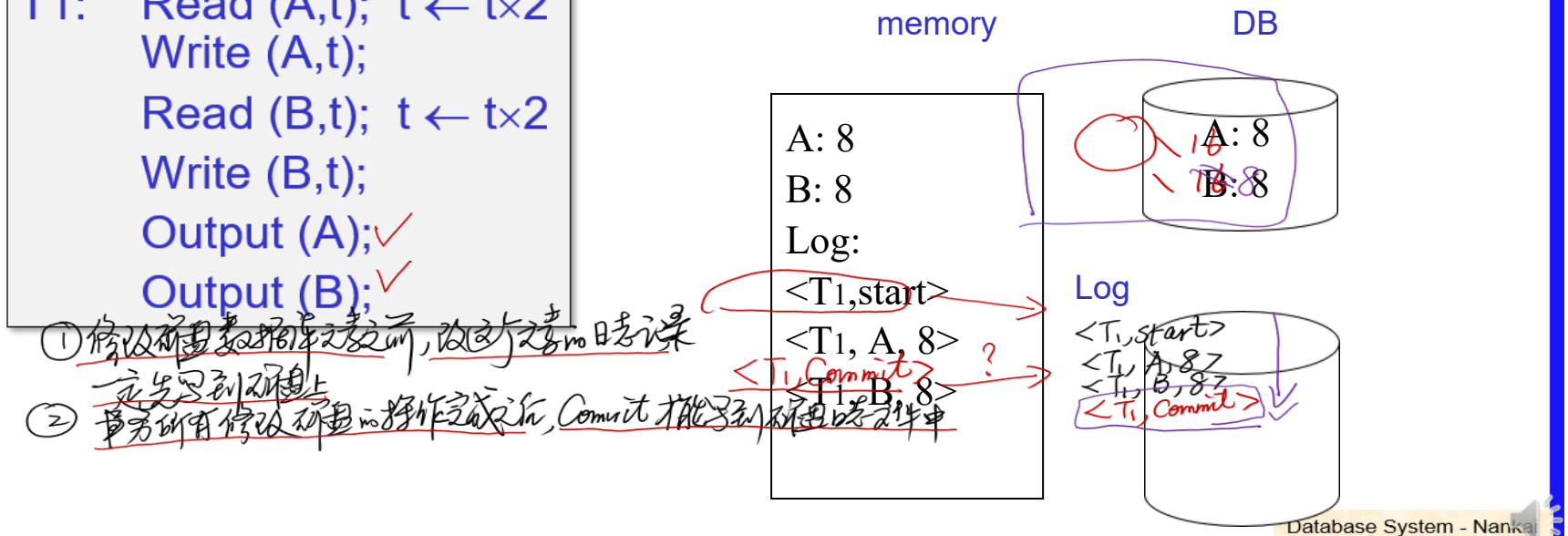
log

Database System - Nankai

One "complication"

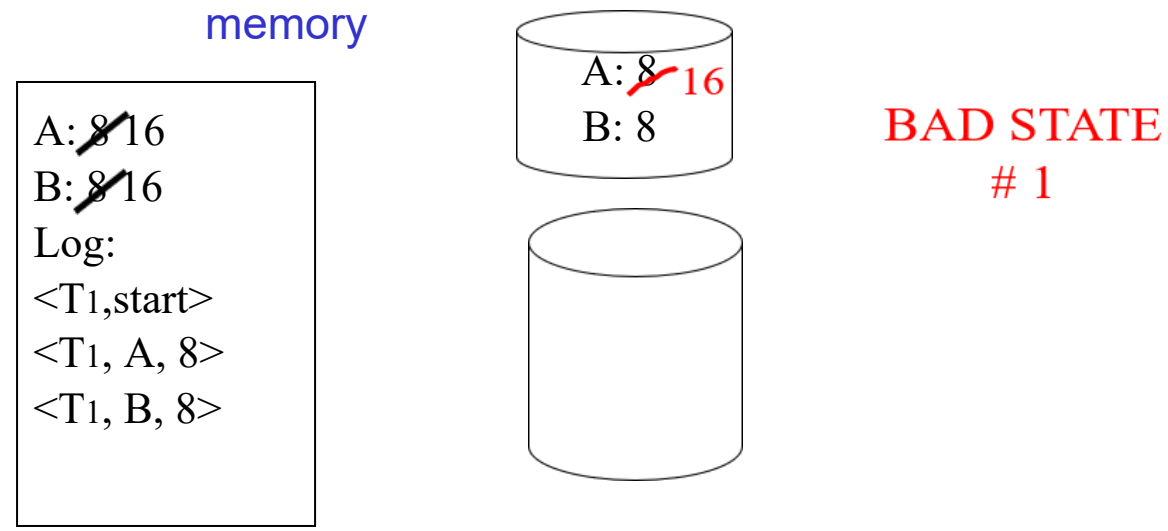
- Log is first written in memory
- Not written to disk on every action

T1: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);✓
Output (B);✓



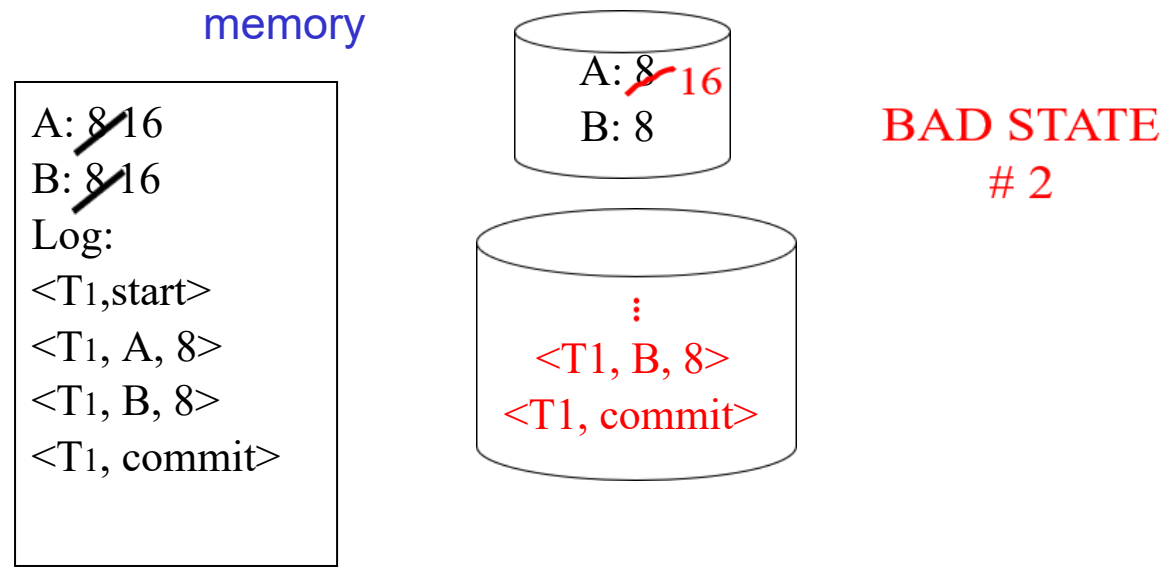
One "complication"

- 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 logging rules

(1) Generate an undo log record for every update action (with the old value)

(2) Before modifying element X on disk, any log records for X must be flushed to disk (write-ahead logging)

(3) All WRITEs of transaction T must be OUTPUT to disk before <T, commit> is flushed to log

Recovery rules: Undo logging

- For every Ti with <Ti, start> in log:
  - If <Ti,commit> or <Ti,abort> in log, do nothing
  - Else
    - For all <Ti, X, v> in log:
      - write (X, v)
      - output (X)
    - Write <Ti, abort> to log

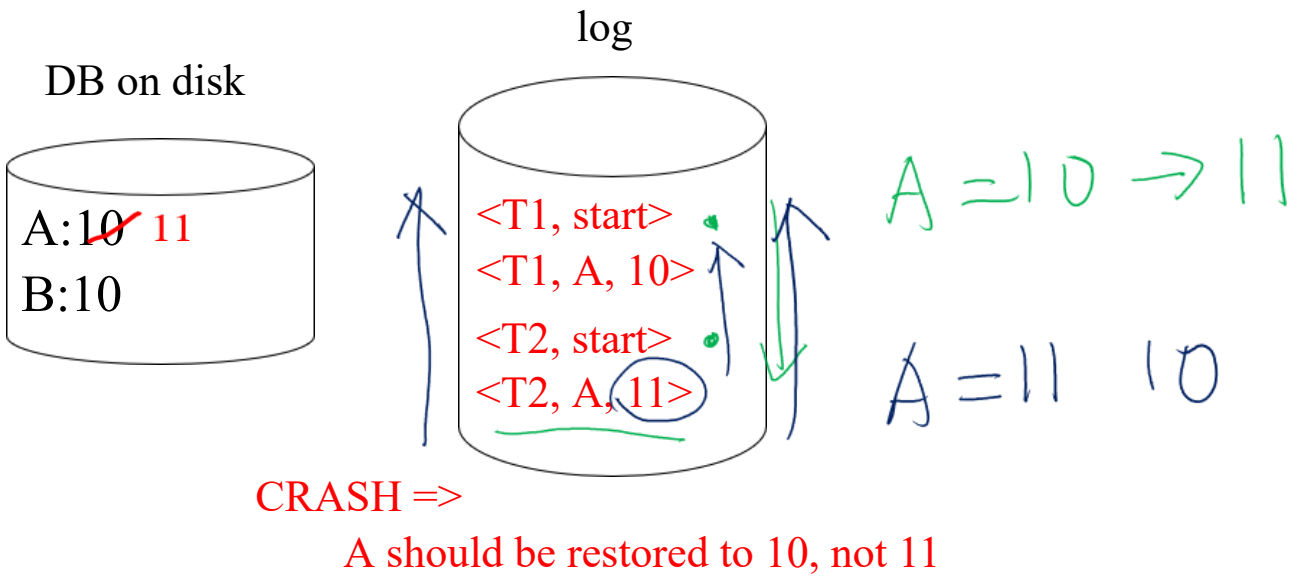
☒IS THIS CORRECT??

Recovery with Undo logging

T_1: A ← A+1; B ← B+1;

T_2: A ← A+1; B ← B+1;

constraint: A=B



CRASH =>

A should be restored to 10, not 11

Database System - Nankai

Undo logging Recovery : (more precisely)

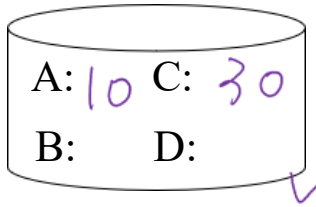
(1) Let S = set of transactions with <Ti, start> in log, but no <Ti, commit> (or <Ti, abort>) record in log

(2) For each <Ti, X, v> in log,
in reverse order (latest → earliest) do:
- if Ti ∈ S then
  - write (X, v)
  - output (X)

(3) For each Ti ∈ S do
- write <Ti, abort> to log

Undo log:

<START T>;
<T,A,10>;
<START U>;
<U,B,20>;
<T,C,30>;
<U,D,40>;
<COMMIT U>;
<T, abort>



S = { T }

Undo logging Recovery : (more precisely)

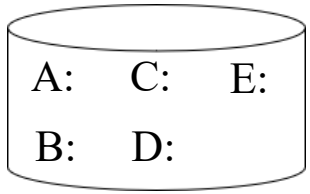
(1) Let S = set of transactions with <Ti, start> in log, but no <Ti, commit> (or <Ti, abort>) record in log

(2) For each <Ti, X, v> in log,
in reverse order (latest → earliest) do:
- if Ti ∈ S then
  - write (X, v)
  - output (X)

(3) For each Ti ∈ S do
- write <Ti, abort> to log

Undo log:

<START T>;
<T,A,10>;
<START U>;
<U,B,20>;
<T,C,30>;
<U,D,40>;
<COMMIT U>;
<T,E,50>;
<COMMIT T>.



S = { }

Summary - undo logging

- Undo日志是如何记录的?
  - 所有修改数据库元素的操作都要按顺序记录日志文件，日志记录记旧值
  - 修改磁盘数据库元素之前，一定要先将修改元素的日志记录输出到磁盘日志文件里
  - 事务修改数据库的所有操作完成后，再将Commit写出到磁盘的日志文件里
- Undo日志是如何恢复数据库的?
  - 查找日志文件中没有结束标记（commit或abort）的事务，从后向前扫描日志，进行回滚
  - 回滚完成后，需要在日志文件添加<T,Abort>记录

填空题 3分

互动交流一

Consider a system that uses pure undo logging. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>; <T1,B,1>; <COMMITT1>; <T2,B,11>; <T2,C,8>;<COMMIT T2>; <START T3>; <T3,A,10>; <START T4>; <T4,A,11>, <T3,C,7>;<T4,B,22>

The last four actions in the schedule were incrementing each object by 1(e.g.<T4,A,11> was updating A to 12).

If finished the system recovery, what are the values on disk for each of the database elements A, B and C?

A: [填空1] B: [填空2] C: [填空3]

提交

Database System - Nankai

单选题 1分

互动交流二

Consider a system that uses pure undo logging. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>;
<T1,B,1>; <COMMITT1>; <T2,B,11>;
<T2,C,8>;<COMMIT T2>; <START T3>;
<T3,A,10>; <START T4>; <T4,A,11>,
<T3,C,7>;<T4,B,22>

The last four actions in the schedule were incrementing each object by 1(e.g.<T4,A,11> was updating A to 12).

What are the all of the possible values on disk for each of the database elements A?

A 5, 10, 11, 12

B 10, 11, 12

C 11, 12

D 12

提交

Database System - Nankai

单选题 1分

互动交流三

Consider a system that uses pure undo logging. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>;
<T1,B,1>; <COMMITT1>; <T2,B,11>;
<T2,C,8>;<COMMIT T2>; <START T3>;
<T3,A,10>; <START T4>; <T4,A,11>,
<T3,C,7>;<T4,B,22>

The last four actions in the schedule were incrementing each object by 1(e.g.<T4,A,11> was updating A to 12).

What are the all of the possible values on disk for each of the database elements B?

A 1, 11, 22, 23

B 11, 22, 23

C 22, 23

D 23

提交

Database System - Nankai

单选题 1分

互动交流四

Consider a system that uses pure undo logging. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>;
<T1,B,1>; <COMMITT1>; <T2,B,11>;
<T2,C,8>;<COMMIT T2>; <START T3>;
<T3,A,10>; <START T4>; <T4,A,11>,
<T3,C,7>;<T4,B,22>

The last four actions in the schedule were incrementing each object by 1(e.g.<T4,A,11> was updating A to 12).

What are the all of the possible values on disk for each of the database elements C?

A 8, 7

B 8

C 7

D 以上都不是

提交

Database System - Nankai

Week14_Course

Database System_Crash Recovery part3

Redo Logging

Database System - Nankai

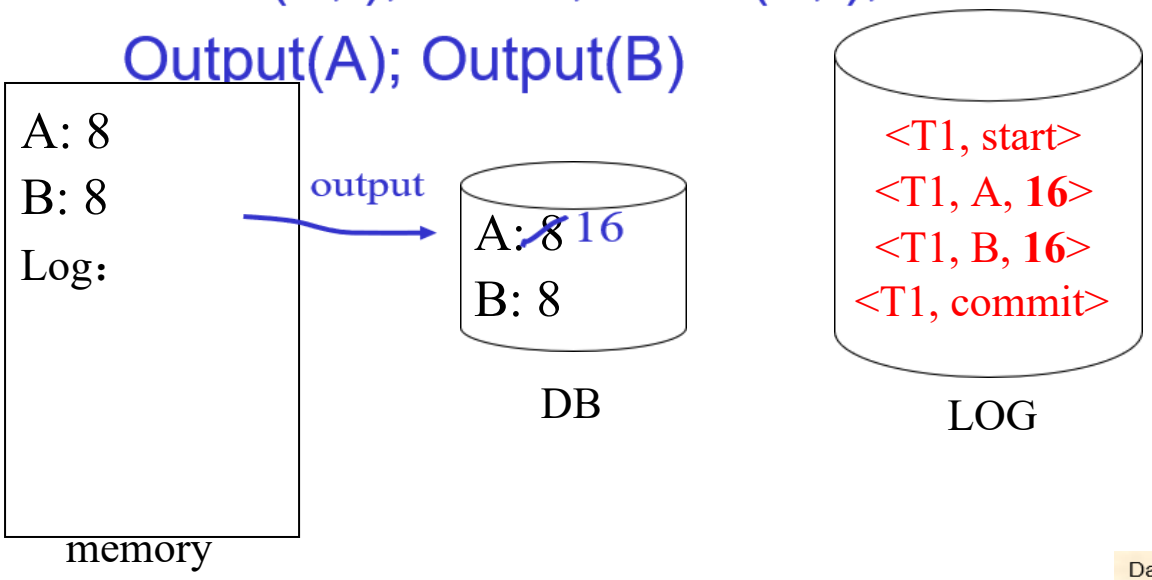
Crash Recovery

- Transactions and Crash Recovery
- Undo logging
- Redo logging
- Undo/Redo logging

Database System - Nankai

Redo logging (deferred modification)

T1: Read(A,t); t← t×2; write (A,t);
Read(B,t); t←t×2; write (B,t);
Output(A); Output(B)



Redo logging rules

(1) For every disk update, generate a redo log record (with new value) <T, X, V>

(2) Before modifying DB element X on disk, all log records for the transaction that (including COMMIT) must be flushed to disk

Recovery rules: Redo logging

- For every Ti with <Ti, commit> in log:
  - For all <Ti, X, v> in log:
    - Write(X, v)
    - Output(X)

This time, the last updates should remain in effect

Recovery with Redo Logging: (more precisely)

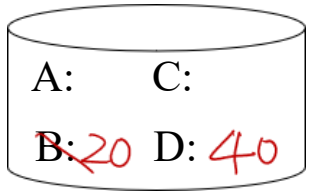
(1) Let S = set of transactions with <Ti, commit> in log

(2) For each <Ti, X, v> in log, in forward order (earliest → latest) do:

- if Ti ∈ S then Write(X, v)
  Output(X)

Redo log:

<START T>;
<T,A,10>;
<START U>;
<U,B,20>;
<T,C,30>;
<U,D,40>;
<COMMIT U>;



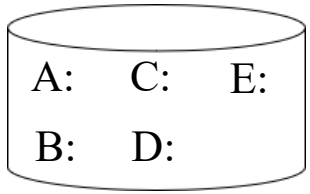
S = { U }

Recovery with Redo Logging: (more precisely)

(1) Let S = set of transactions
with <Ti, commit> in log
(2) For each <Ti, X, v> in log, in
forward order (earliest → latest)
do:
- if Ti ∈ S then { Write(X, v); Output(X) }

Redo log:

<START T>;
<T,A,10>;
<START U>;
<U,B,20>;
<T,C,30>;
<U,D,40>;
<COMMIT U>;
<T,E,50>;
<COMMIT T>.

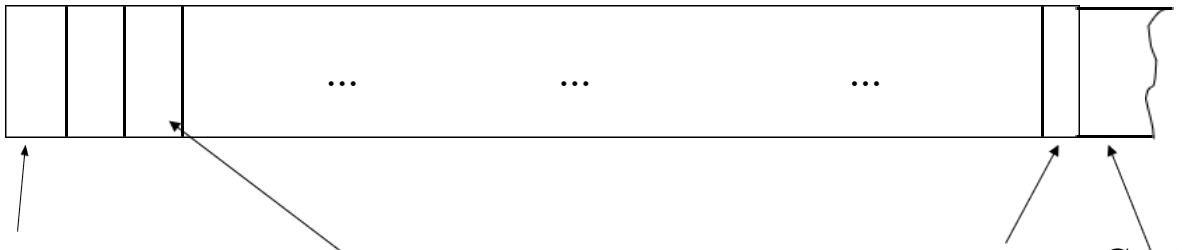


S = { }

Database System - Nankai

Recovery is very, very SLOW !

Redo log:



First Record (1 year ago)

T1 wrote A,B Committed a year ago --> STILL, Need to redo after crash!!

Last Record

Solution: Checkpointing (for redo-logging, simple version)

Periodically:

(1) Stop accepting new transactions

(2) Wait all transactions to finish (commit/abort)

(3) Flush all log records to disk (log)

(4) Flush all buffers to disk (DB)

(5) Write & flush a <CKPT> record on disk (log)

(6) Resume transaction processing

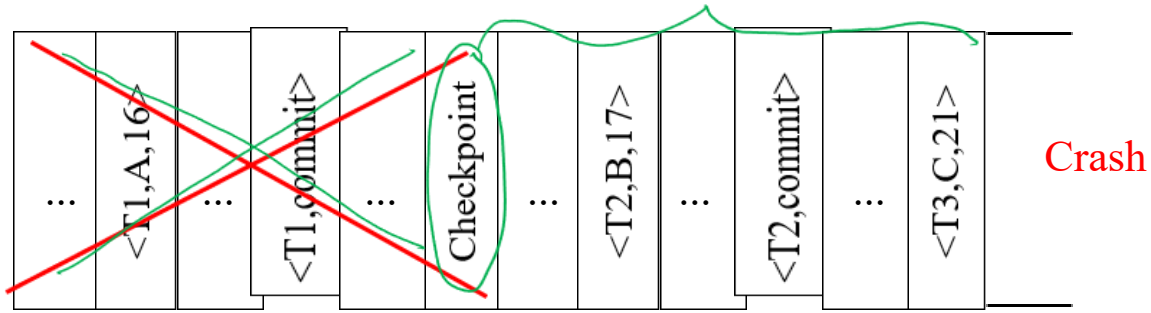
Redo log:

<START T>;
<T,A,10>;
<START U>;
<U,B,20>;
<U,D,40>;
<COMMIT U>;
<T,E,50>;
<CKPT>
<T,C,32>;
<COMMIT T>;

Database System - Nankai

Example: what to do at recovery?

Redo log (on disk):



Result of transactions completed prior to the checkpoint are stored permanently on disk

-> Redo

```
write(B, 17)
output(B)
```

Database System - Nankai

Summary - redo logging

- Redo日志是如何记录的?
  - 所有修改数据库元素的操作都要按顺序记录日志文件，日志记录记新值
  - 先将Commit写出到磁盘的日志文件里之后，再去修改数据库中的元素
- Redo日志是如何恢复数据库的?
  - 查找日志文件中带有结束标记（commit）的事务，从前向后扫描日志，对事务进行重新执行
- 静态检查点机制、带有检查点的恢复策略

填空题 3分

互动交流一

Consider a system that uses pure redo logging. The initial database state was A=0,B=1,C=2. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>; <T1,B,1>; <COMMITT1>; <T2,B,11>; <T2,C,8>;<COMMIT T2>; <CKPT>, <START T3>; <T3,A,10>; <START T4>; <T4,A,11>, <T3,C,7>;<T4,B,22>

If finished the system recovery, what are the values on disk for each of the database elements A, B and C?

A: [填空1] B: [填空2] C: [填空3]

提交

Database System - Nankai

单选题 1分

互动交流二

Consider a system that uses pure redo logging. The initial database state was A=0,B=1,C=2. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>; <T1,B,1>; <COMMITT1>; <T2,B,11>; <T2,C,8>;<COMMIT T2>; <CKPT>, <START T3>; <T3,A,10>; <START T4>; <T4,A,11>, <T3,C,7>;<T4,B,22>

What are the all of the possible values on disk for each of the database elements A?

A 0, 5, 10,11

B 0

C 5

D 0, 5

提交

Database System - Nankai

单选题 1分

互动交流三

Consider a system that uses pure redo logging. The initial database state was A=0,B=1,C=2. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>; <T1,B,1>; <COMMITT1>; <T2,B,11>; <T2,C,8>;<COMMIT T2>; <CKPT>, <START T3>; <T3,A,10>; <START T4>; <T4,A,11>, <T3,C,7>;<T4,B,22>

What are the all of the possible values on disk for each of the database elements B?

A 1, 11, 22

B 1, 11

C 1

D 11

提交

Database System - Nankai

单选题 1分

互动交流四

Consider a system that uses pure redo logging. The initial database state was A=0,B=1,C=2. After a system crash, we find the following log entries on disk:

<START T1>; <T1,A,5>; <START T2>; <T1,B,1>; <COMMITT1>; <T2,B,11>; <T2,C,8>;<COMMIT T2>; <CKPT>, <START T3>; <T3,A,10>; <START T4>; <T4,A,11>, <T3,C,7>;<T4,B,22>

What are the all of the possible values on disk for each of the database elements B?

A 1, 11, 22

B 1, 11

C 1

D 11

提交

Database System - Nankai

Week14_Course

Database System_Crash Recovery part4

Undo/Redo Logging

Database System - Nankai

Crash Recovery

- Transactions and Crash Recovery
- Undo logging
- Redo logging
- Undo/Redo logging

Database System - Nankai

Drawbacks:

- *Undo logging:*
  - System forced to update disk at the end of transactions (may increase disk I/O)
- *Redo logging:*
  - need to keep all modified blocks in memory until commit (may lead to shortage of buffer pool)

Solution: Undo/redo logging!

Update record

⇒ <Ti, X, Old-X-value, New-X-value>

Provides more flexibility to order actions

Database System - Nankai

Rules for undo/redo logging

(1) Log record <Ti, X, old, new> flushed to disk before writing X to disk

(2) Flush the log at commit

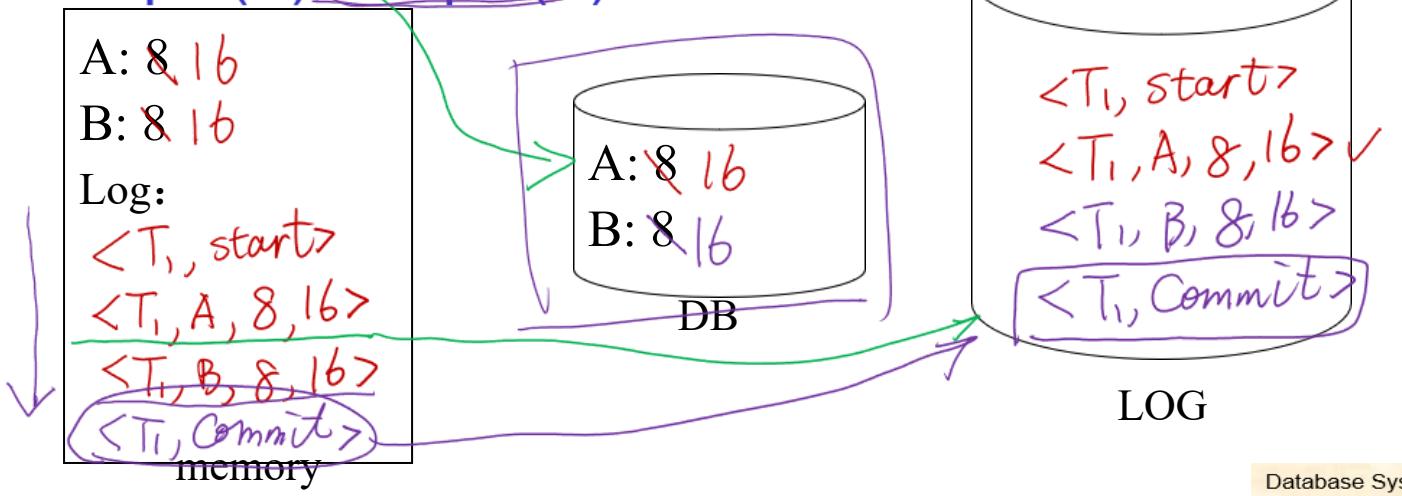
- Element X can be written to disk either before or after the commit of Ti

Example - Undo/Redo logging

T1: Read(A,t); t ← t×2; write (A,t);

Read(B,t); t ← t×2; write (B,t);

Output(A); Output(B)



Undo/Redo Recovery Policy

- To recover from a crash,
  - redo updates by any committed transactions

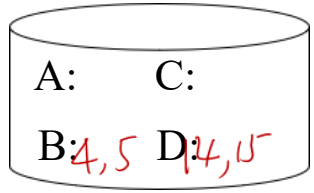
    (in forward-order, earliest first)
  - undo updates by any incomplete transactions

    (in backward-order, latest first)

Example - Undo/Redo Recovery

Log :

<START T1>
<T1, A, 4, 5>
<START T2>
<COMMIT T1>
<T2, B, 9, 10>
<T2, C, 14, 15>
<START T3>
<T3, D, 19, 20>
<COMMIT T3>



DBMS crash 后磁盘上数据元素可能存在的值是多少

A: 4, 5 B:

C: 14, 15 D:

Database System - Nankai

Problem with Simple Checkpointing

- Simple checkpointing stops the system from accepting new transactions
  -> system effectively shuts down
- Solution: "nonquiescent" checkpointing
  - accepts new transactions during a checkpoint

Nonquiescent Checkpointing

- Slightly different for various logging policies
- Rules for undo/redo logging:
  - Write log record (and flush log)

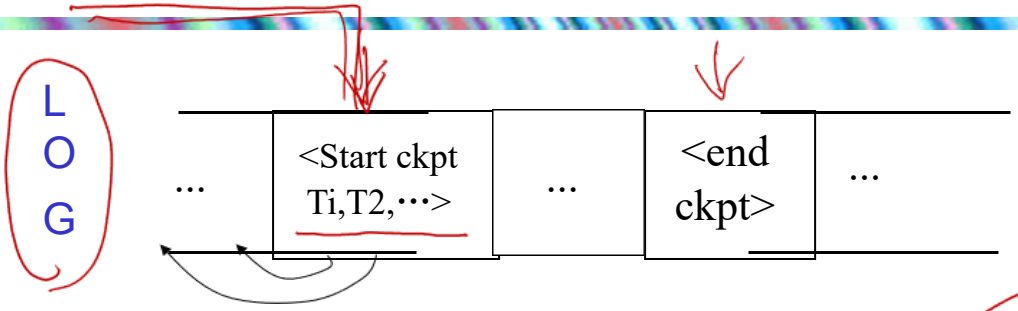
    <START CKPT T1, ···, Tk>,

    where T1, ···, Tk are the active transactions
  - Flush to disk all dirty buffers which contain changed database elements (Corresponding log records first!)

    => Effect of writes prior to the chkpt made permanent
  - Write & flush log record

    <END CKPT>

Non-quiescent checkpoint



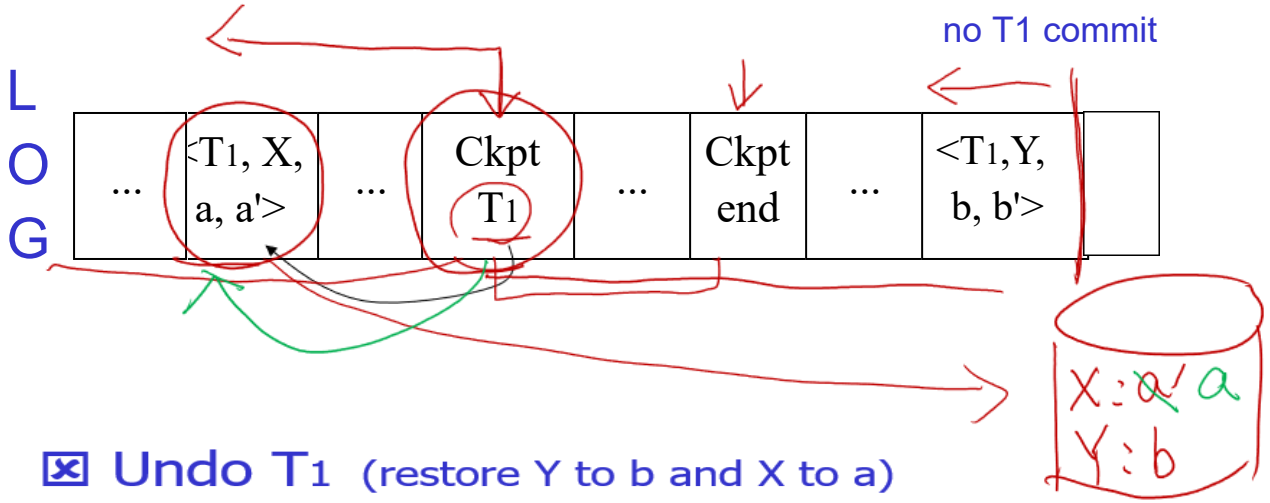
for

undo dirty buffer pool pages flushed

<T, A, 8, 16>

A: 8→16

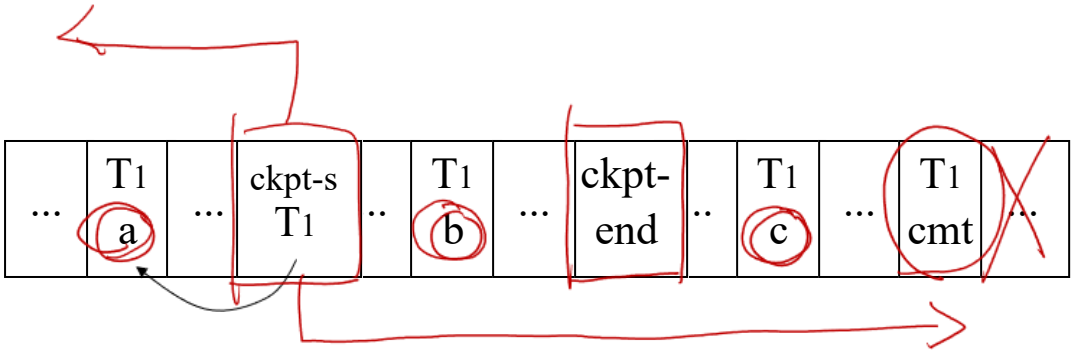
Examples what to do at recovery time?



☒ Undo T1 (restore Y to b and X to a)

Example

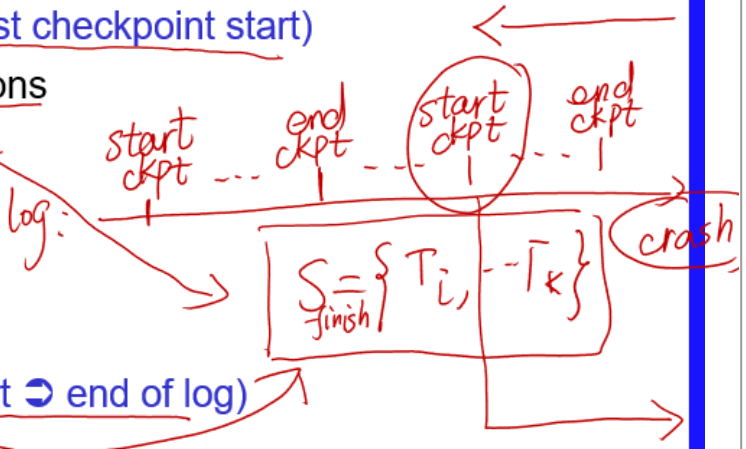
LOG

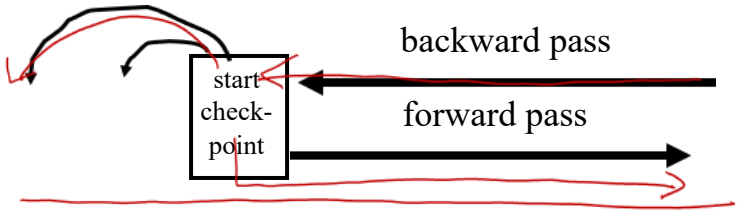


☒ Redo T1: (redo update of elements to b and to c)

Recovery process:

- **Backwards pass** (end of log ➲ latest checkpoint start)
  - construct set S of committed transactions
  - undo actions of transactions *not* in S
- **Undo pending transactions**
  - follow undo chains for transactions in (checkpoint active list) - S
- **Forward pass** (latest checkpoint start ➲ end of log)
  - redo actions of transactions in set S





<START T1>
<T1,A,4,5>
<START T2>
<COMMIT T1>
<T2,B,9,10>
<START CKPT(T2)>
<T2,C,14,15>
<START T3>
<T3,D,19,20>
<END CKPT>
<COMMIT T2>

检查点插入日志的过程:

根据日志恢复数据库的过程

A:5 B:10 C:15 D:19

带有检查点的日志记录过程: ③

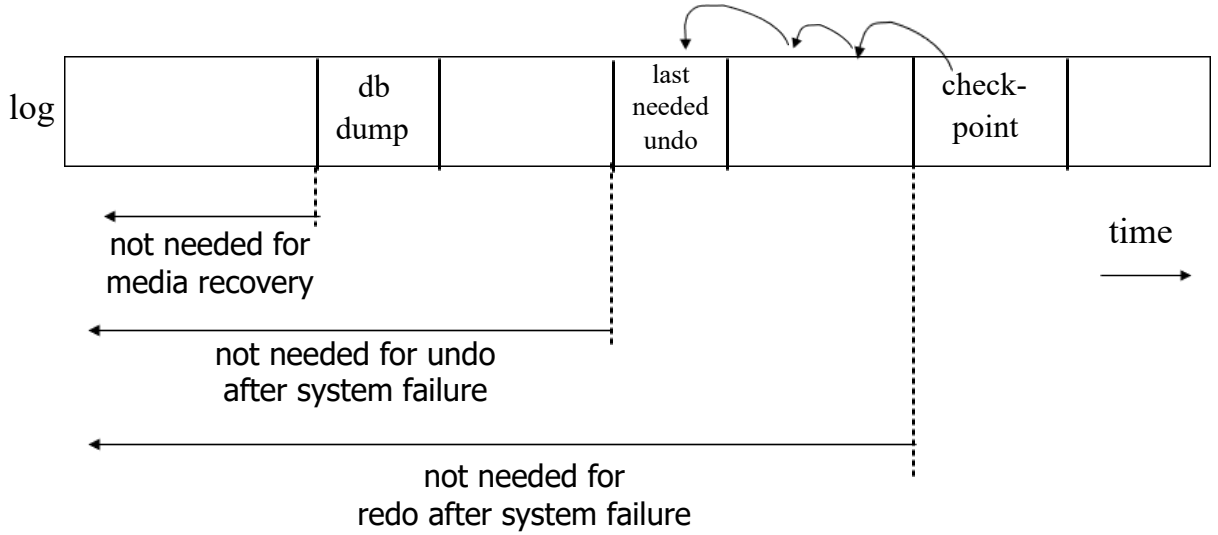
A:5 B:10 C:14,15 D:19,20

2 2

4种

Database System - Nankai

When can we discard the log?



Summary – Undo/Redo logging

- Undo/Redo日志是如何记录的?
  - 事务对数据库元素的修改一定要先记日志后修改数据库，日志要记录事务对数据库元素修改之前的旧值和修改之后的新值
- Undo/Redo日志是如何恢复数据库的?
  - 对没有完成的事务，从后向前扫描日志，用旧值更改数据库；对commit的事务，从前向后扫描日志，用新值更数据库;
- 动态检查点的插入机制和恢复策略

填空题 3分

互动交流一

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

If finished the system recovery, what are the values on disk for each of the database elements A, B and C?

A: [填空1]

B: [填空2]

C: [填空3]

提交

Database System - Nankai

单选题 1分

互动交流二

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

What are the all of the possible values on disk for each of the database elements A?

A 5,0,10,11

B 0,10,11

C 10,11

D 11

提交

Database System - Nankai

单选题 1分

互动交流二

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

What are the all of the possible values on disk for each of the database elements A?

A 5,0,10,11

B 0,10,11

C 10,11

D 11

提交

Database System - Nankai

单选题 1分

互动交流三

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

What are the all of the possible values on disk for each of the database elements B?

A 1,2,3,22

B 2,3,22

C 3,22

D 22

提交

Database System - Nankai

单选题 1分

互动交流四

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

What are the all of the possible values on disk for each of the database elements C?

A 8, 9, 7

B 9, 7

C 7

D 以上都不正确

提交

Database System - Nankai

填空题 2分

互动交流五

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

Which, if any, transactions will need to be redone and undone in the recovery process?

Transactions to Redo: [填空1]

Transaction to Undo: [填空2]

提交

Database System - Nankai

思考题

Consider a system that uses undo-redo logging. After a system crash, we find the following log entries on disk:

<START T1>
<T1,A,5,0>
<START T2>
<T1,B,1,2>
<COMMIT T1>
<T2,B,2,3>
<START CKPT(T2)>
<T2,C,8,9>
<COMMIT T2>
<END CKPT>
<START T3>
<T3,A,0,10>
<START T4>
<T4,A,10,11>
<T3,C,9,7>
<T4,B,3,22>

How would your answers above if <END CKPT> were not present in the log?

Database System - Nankai