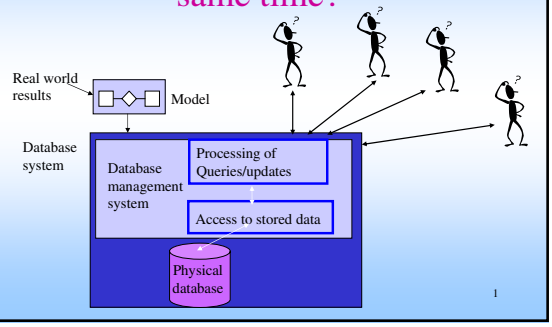
How can several users access and update the information at the same time?



Transactions

- A *transaction* is a logical unit of database processing and consists of one or several operations.
- Database operations in a simplified model:
  - read-item(X)
  - write-item(X)

Properties for transactions

ACID: Atomicity, Consistency preservation, Isolation, Durability

- A: A transaction is an atomic unit: it is either executed completely or not at all
- C: A database that is in a consistent state before the execution of a transaction (i.e. it fulfills the conditions in the schema and other conditions declared for the database), is also in a consistent state after the execution.

Properties for transactions

ACID: Atomicity, Consistency preservation, Isolation, Durability

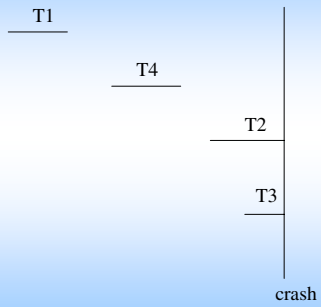
- I: A transaction should act as if it is executed isolated from other transactions.
- D: Changes in the database made by a committed transaction are permanent.

Properties for transactions

How are the ACID properties achieved?

- A: recovery system
- C: programmer + DBMS
- I: concurrency contol
- D: recovery system

start-transaction T1
write-item T1, D, 10, 20
commit T1
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



Recovery
(Atomicity and durability)

Reasons for crash

1. system crash
2. transaction or system error
3. local error or exception has been discovered by a transaction
4. concurrency control
5. disk failure
6. catastrophy

Cause 5-6

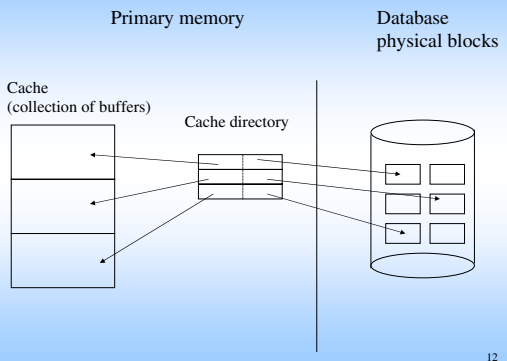
backup of database
backup of system log

→ redo operations för committed transactions

System log

- file with log records
- Saved on disk + periodically on tape
- Types av log records:
  - start-transaction T
  - write-item T, X, oldvalue, newvalue
  - read-item T,X
  - commit T
  - abort T
  - checkpoint

Cause 1-4



Read/Write X

- Check whether the block with element X is in primary memory (buffer)
- If not, get the block with element X.

It is possible that some buffers in the cache need to be replaced.

It is possible that some buffers need to be written to the disk first. (flush the cache buffers)

Read/Write X

- How do we know that a buffer has been changed? ``dirty bit''
- How do we know that a buffer can be written to the disk? ``pin-unpin bit''
- Where is the buffer written on the disk?

``in-place updating'' - ``shadowing''

Checkpoint

- The system writes all buffers that have been changed (dirty bit) and can be written (pin-unpin bit) to the disk
- Advantage: operations belonging to transactions that have committed before a checkpoint do not need to be redone
- How often?: according to time – number of committed transactions

Checkpoint

- How?

1. Temporarily stop all transactions
2. Write all buffers that were changed and can be written to the disk
3. Write``checkpoint'' in the log and write the log to disk
4. Restart execution of the transactions

Fuzzy Checkpointing

- As checkpointing takes time, the execution of transactions is delayed. To reduce the delay we can use fuzzy checkpointing.
- Write checkpoint in the log, but keep the previous checkpoint until the writing to the disk is finished.

Update methods

- Deferred update

- The database is updated physically *after* the transaction has committed.
- before commit the transaction has a local environment
- after commit the log and buffers are written to the disk (note: this does not mean that is is written *immediately* after commit)

Update methods

- Immediate update

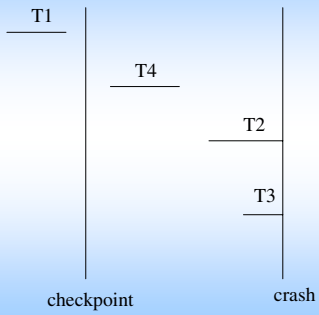
- The database *can* be updated physically before commit (the log is written first, then the database)

Recovery with deferred update

- As the database is physically changed after commit, we never need to take away results from non-committed transactions.
- We need to redo the operations of committed transactions for which the results have not been written to the disk.
- NO-UNDO/REDO

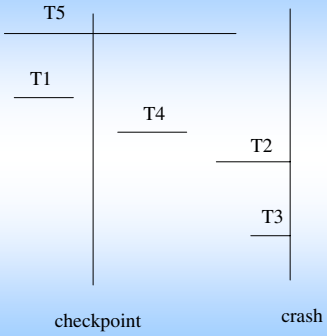
NO-UNDO
REDO: T4

start-transaction T1
write-item T1, D, ~~10~~, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, ~~10~~, 20
write-item T4, A, ~~5~~, 10
commit T4
start-transaction T2
write-item T2, B, ~~20~~, 15
start-transaction T3
write-item T3, A, ~~10~~, 30
write-item T2, D, ~~20~~, 25
CRASH



Start-transaction T5
write-item T5, E, 10, 15

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
commit T5
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



Recovery with deferred update

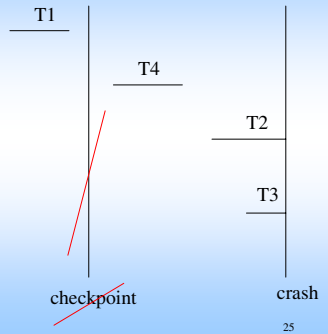
- Algorithm:
- Use two lists: a list with active transactions and a list with committed transactions since the last checkpoint.
- REDO all write operations (write-item) of all committed transactions in the order they appear in the log.

Recovery with immediate update - 1

1. It is required that all updates are written to disk before commit.
- No need to redo committed transactions
- Need to remove results of operations from non-committed transactions
- UNDO/NO-REDO

NO-REDO
UNDO: T2, T3

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



25

Recovery with immediate update - 1

- Algorithm:
- Use two lists: a list with active transactions and a list with committed transactions since the last checkpoint.
- Take away (UNDO) all results of all write operations (write-item) of all active transactions in the reverse order in which they appear in the log.

26

Recovery with immediate update - 2

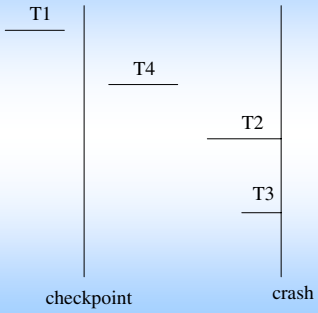
2. No requirement that all updates are written to disk before commit.

- We need to redo the operations of committed transactions for which all results have not been written to the disk.
- Need to remove results of operations from non-committed transactions
- UNDO/REDO

27

UNDO: T2, T3
REDO: T4

start-transaction T1
write-item T1, D, 10, 20
commit T1
checkpoint
start-transaction T4
write-item T4, B, 10, 20
write-item T4, A, 5, 10
commit T4
start-transaction T2
write-item T2, B, 20, 15
start-transaction T3
write-item T3, A, 10, 30
write-item T2, D, 20, 25
CRASH



28

Recovery with immediate update - 2

- Algorithm:
- Use two lists: a list with active transactions and a list with committed transactions since the last checkpoint.
- Take away (UNDO) all results of all write operations (write-item) of all active transactions in the reverse order in which they appear in the log.
- REDO all write operations (write-item) of all committed transactions in the order they appear in the log.

29

Comparison

- Deferred update
  – NO-UNDO/REDO
- Immediate update 1
  – UNDO/NO-REDO

30

Comparison

- Deferred update
  - NO-UNDO/REDO
  - Requires large buffer space (pinned blocks in cache)
- Immediate update 2
  - UNDO/REDO
  - Blocks can be written to disk at any time (flush the cache)

Comparison

- Immediate update 1
  - UNDO/NO-REDO
  - Required to write to the database latest at commit
- Immediate update 2
  - UNDO/REDO
  - Can wait to write to the database (e.g. until checkpoint)

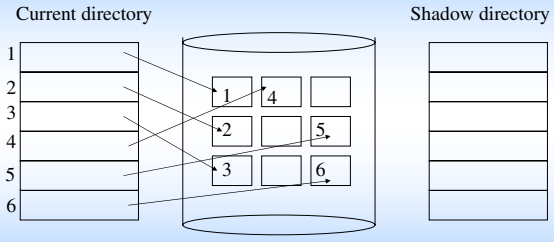
Shadow paging

- Database size is *n* blocks
- current directory: i:th element points to i:th block on disk

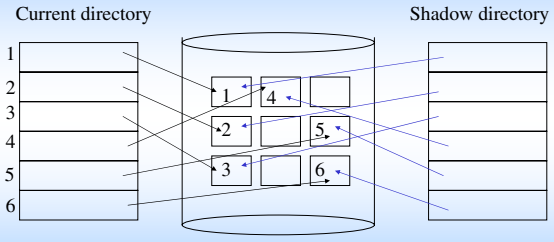
Shadow paging

- When a transaction starts:
  - Copy current directory to a shadow directory
  - Save the shadow directory on disk
  - The shadow directory is not modified during the transaction
  - write-item: a new copy of the block is saved, but the old block is NOT overwritten.
  - commit: take away the shadow directory

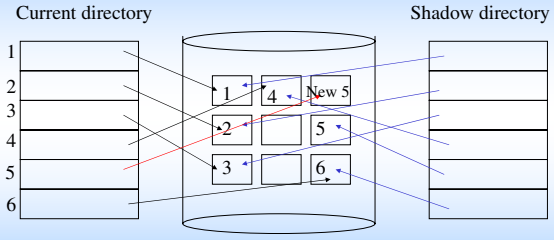
Shadow paging



Shadow paging



Shadow paging



Write-item --> block 5

Shadow paging

- Recovery: use the shadow directory
- Advantages:
  - Single user system does not need logging
  - NO-UNDO/NO-REDO
- Disadvantages:
  - Blocks can be moved on the disk
  - garbage collection