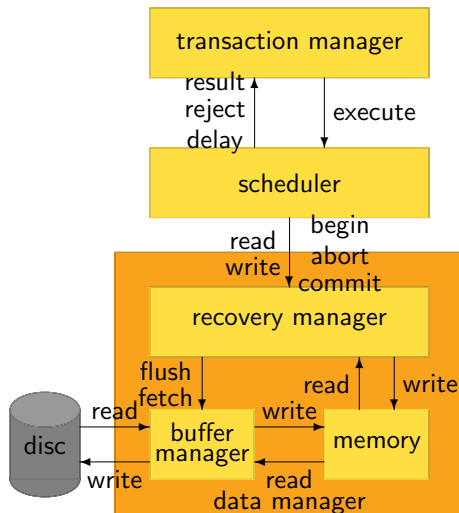


Recovery

P.J. McBrien

Imperial College London

DBMS Architecture



Recovery Manager (RM)

protect the DBMS against failures

- **system failures** loss of volatile storage

- 1 committed transactions written to disc

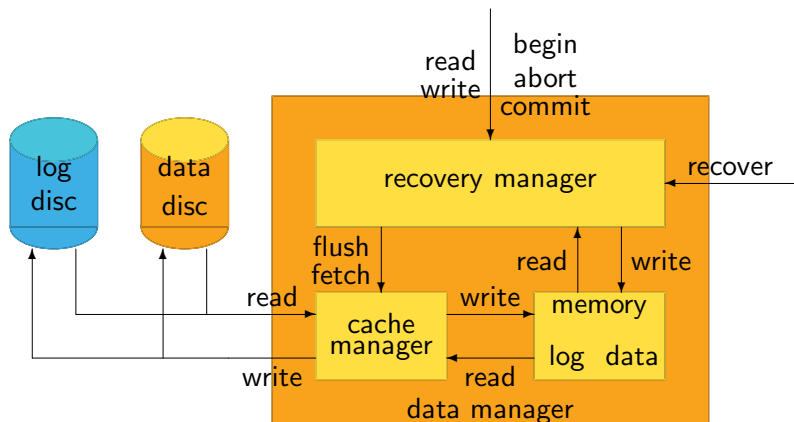
- 2 uncommitted transactions not written to disc

OR

- 3 sufficient information such that (1) and (2) may be met by a
recovery

- **media failures** loss of stable storage

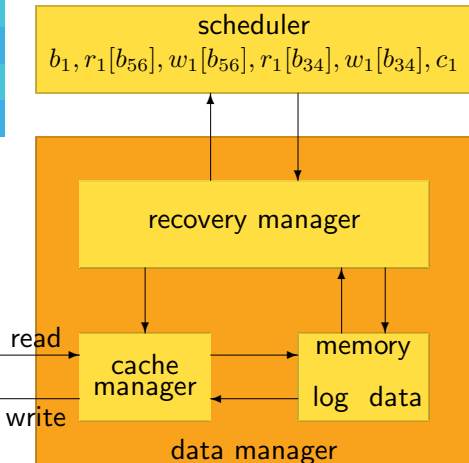
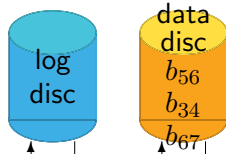
Enhanced Data Manager Architecture



- Need to cache log as well

Need to REDO

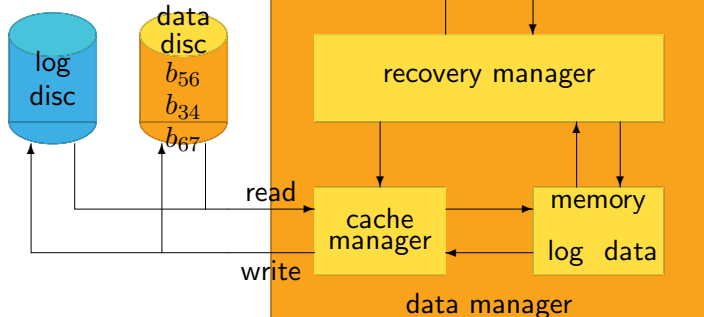
LOG	b_1
REDO	$w_1[b_{56}, \text{cash}=84340.45]$
REDO	$w_1[b_{34}, \text{cash}=18900.67]$
LOG	c_1



- REDO required if committed transactions not in stable storage
- must write all REDO to log before commit of transaction

Need to UNDO

LOG	b_1
UNDO	$w_1[b_{56}, \text{cash}=94340.45]$
UNDO	$w_1[b_{34}, \text{cash}=8900.67]$
LOG	c_1



- UNDO required if non-committed transactions in stable storage
- Must flush UNDO to log before corresponding write to data

Quiz 1: Contents of Data Disc After a Transaction

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

```

BEGIN TRANSACTION T1
UPDATE branch
SET cash=cash-10000.00
WHERE sortcode=56

UPDATE branch
SET cash=cash+10000.00
WHERE sortcode=34
COMMIT TRANSACTION T1
  
```

branch ①		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ③		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ②		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

branch ④		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

What must the contents of the branch table on the data disc be after the transaction commits?

A

④

B

① or ④

C

①, ③ or ④

D

①, ②, ③ or ④

Quiz 2: Contents of Log Disc After a Transaction

Data Disc Before Transaction

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

```

BEGIN TRANSACTION T1
  UPDATE branch
  SET cash=cash-10000.00
  WHERE sortcode=56

  UPDATE branch
  SET cash=cash+10000.00
  WHERE sortcode=34
COMMIT TRANSACTION T1
  
```

Data Disc At Commit Time

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

What must be on the log disc after commit time?

A

REDO r_{56}
 REDO r_{34}
 UNDO r_{56}
 UNDO r_{34}

B

REDO r_{56}
 UNDO r_{34}

C

UNDO r_{34}

D

REDO r_{56}

Before and after images

before image

branch		
<u>sortcode</u>	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

⇓

$w_1[b_{56}]$

⇓

branch		
<u>sortcode</u>	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

after image

- before image allows RM to **undo** $w_1[b_{56}]$
- after image allows RM to **redo** $w_1[b_{56}]$

Database Logs

LOG	b_1
REDO	$w_1[b_{56}, \text{cash}=84340.45]$
REDO	$w_1[b_{34}, \text{cash}=18900.67]$
LOG	c_1



LOG	b_1
UNDO	$w_1[b_{56}, \text{cash}=94340.45]$
REDO	$w_1[b_{56}, \text{cash}=84340.45]$
UNDO	$w_1[b_{34}, \text{cash}=8900.67]$
REDO	$w_1[b_{34}, \text{cash}=18900.67]$
LOG	c_1

LOG	b_1
UNDO	$w_1[b_{56}, \text{cash}=94340.45]$
UNDO	$w_1[b_{34}, \text{cash}=8900.67]$
LOG	c_1



What must a complete REDO/UNDO log contain?

Must contain

- REDO information for each update
- UNDO information for each update
- commit of each transaction

Might contain

- begin of each transaction
 - can be inferred from first REDO/UNDO
 - presence useful to stop search of UNDO records
- abort of each transaction
 - can be inferred from lack of commit
 - presence useful to indicate UNDO already done

Rules for log and data updates

write ahead logging (WAL)

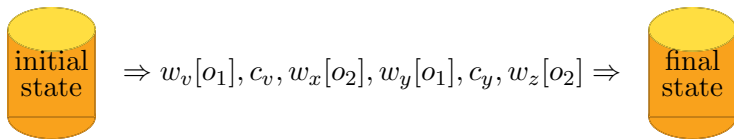
Redo rule

- commit \rightarrow flush log of transaction to disc
- never respond to scheduler before log written

Undo rule:

- flushing uncommitted data \rightarrow flush log of operations

Basic Recovery Procedure

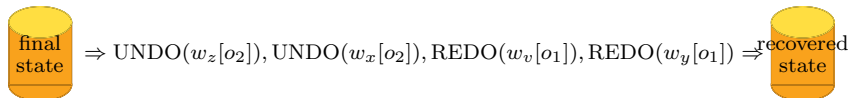


1 UNDO \rightarrow Scan back through the log

- Collect set of committed transactions $C = \{v, y\}$
- Collect set of incomplete transactions $I = \{x, z\}$
- Perform UNDO for any transaction in $I = w_z[o_2], w_x[o_2]$

2 REDO \rightarrow Scan forward through the log

- Perform REDO for any transaction in $C = w_v[o_1], w_y[o_1]$



Example of Recovery

Log

LOG	b_4
LOG	b_1
UNDO	$w_1[b_{56}, \text{cash}=94340.45]$
REDO	$w_1[b_{56}, \text{cash}=84340.45]$
LOG	b_2
UNDO	$w_2[b_{34}, \text{cash}=10900.67]$
REDO	$w_2[b_{34}, \text{cash}=8900.67]$
UNDO	$w_2[b_{67}, \text{cash}=34005.00]$
REDO	$w_2[b_{67}, \text{cash}=36005.25]$
LOG	b_7
LOG	c_2
UNDO	$w_1[b_{34}, \text{cash}=8900.67]$
REDO	$w_1[b_{34}, \text{cash}=18900.67]$
UNDO	$w_7[b_{67}, \text{cash}=36005.25]$
REDO	$w_7[b_{67}, \text{cash}=37005.25]$
LOG	c_7
LOG	c_4

Disc Before Recovery

branch		
<u>sortcode</u>	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

Disc After Recovery

branch		
<u>sortcode</u>	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	37005.25

Omitting the REDO Log

If no REDO records kept

must flush committed transactions to data disc

- 1 $C = \emptyset, D = \emptyset$
- 2 Scan the log backwards from the end.
- 3 commit entry \rightarrow add to C
- 4 undo entry for member of $C \rightarrow$ add object to D *without* making changes to the data.
- 5 perform undo entry for object not of member D

Omitting the Undo Log

If no UNDO records kept

transaction must never write uncommitted data

- add **fix** command between RM and CM to stop CM flushing data
- commit is followed by flush or **unfix** of fixed objects

Omitting UNDO and REDO

atomic commit → out of place updating

Quiz 3: Contents of Disc Before Commit if no UNDO log

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

```
BEGIN TRANSACTION T1
UPDATE branch
SET cash=cash-10000.00
WHERE sortcode=56

UPDATE branch
SET cash=cash+10000.00
WHERE sortcode=34
COMMIT TRANSACTION T1
```

branch ①		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ③		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ②		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

branch ④		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

What must the contents of the branch table on disc be before the transaction commits?

A

①

B

① or ④

C

④

D

①, ②, ③ or ④

Quiz 4: Contents of Disc After Commit if no REDO log

branch		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

```

BEGIN TRANSACTION T1
UPDATE branch
SET cash=cash-10000.00
WHERE sortcode=56

UPDATE branch
SET cash=cash+10000.00
WHERE sortcode=34
COMMIT TRANSACTION T1
  
```

branch ①		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ③		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	8900.67
67	'Strand'	34005.00

branch ②		
sortcode	bname	cash
56	'Wimbledon'	94340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

branch ④		
sortcode	bname	cash
56	'Wimbledon'	84340.45
34	'Goodge St'	18900.67
67	'Strand'	34005.00

What must the contents of the branch table on disc be after the transaction commits?

A

①

B

① or ④

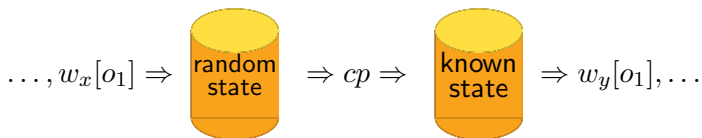
C

④

D

①, ②, ③ or ④

Checkpointing



- Forces the database into some known state
- Recovery limited to only look back to checkpoint (or a ‘bit’ before!)
 - speeds the recovery operation
 - limits the size of log
- The more consistent this known state
 - the easier it is to recover
 - the longer it takes to perform the checkpoint

Commit Consistent Checkpoint

Generating a Commit Consistent Checkpoint

- 1 Stop accepting new transactions
 - 2 Finish existing transactions.
 - 3 Flush all dirty data cache objects to disc.
 - 4 Write a checkpoint to stable log.
- recovery now only needs to scan back to *cp* in log ✓
 - possible long hold-up at checkpoint ✗

Cache Consistent Checkpoint

Generating a Cache Consistent Checkpoint

- 1 Suspend all transactions
- 2 Flush all dirty cache objects to disc
- 3 Write a checkpoint + active transactions to stable log

Recovery from Cache Consistent Checkpoint records

- 1 perform UNDOs of non-committed transactions back to *cp*
 - 2 perform UNDO of non-committed transactions before *cp* if they were active at *cp*
 - 3 perform REDOs of committed transactions after *cp*
- could still have delay whilst flushing cached objects

Worksheet: Cache Consistent Checkpoint

LOG	b_7
UNDO	$w_7[b_{67}, \text{cash}=34005.25]$
REDO	$w_7[b_{67}, \text{cash}=37005.25]$
LOG	b_2
UNDO	$w_2[b_{34}, \text{cash}=10900.67]$
REDO	$w_2[b_{34}, \text{cash}=8900.67]$
LOG	b_6
UNDO	$w_6[a_{101}, \text{rate}=5.25]$
REDO	$w_6[a_{101}, \text{rate}=6.00]$
LOG	b_1
UNDO	$w_1[b_{56}, \text{cash}=94340.45]$
REDO	$w_1[b_{56}, \text{cash}=84340.45]$
LOG	a_7
LOG	$cp\{1, 2, 6\}$
	\vdots

	\vdots
UNDO	$w_6[a_{119}, \text{rate}=5.50]$
REDO	$w_6[a_{119}, \text{rate}=6.00]$
LOG	c_6
UNDO	$w_2[b_{67}, \text{cash}=34005.00]$
REDO	$w_2[b_{67}, \text{cash}=36005.25]$
LOG	b_8
LOG	c_2
UNDO	$w_1[b_{34}, \text{cash}=8900.67]$
REDO	$w_1[b_{34}, \text{cash}=18900.67]$
LOG	b_9
UNDO	$w_9[b_{67}, \text{cash}=36005.00]$
REDO	$w_9[b_{67}, \text{cash}=20000.00]$
LOG	c_9

Fuzzy Checkpointing

Generating a Fuzzy Checkpoint

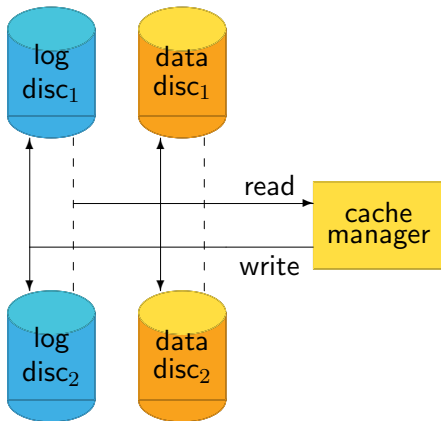
- 1 Suspend all transactions
- 2 Flush any dirty cache objects to disc not flushed in previous *cp*
- 3 Write a checkpoint + active transactions to stable log

Recovery from Fuzzy Checkpoint records

Recovery works like cache consistent checkpoint, but working with penultimate *cp*

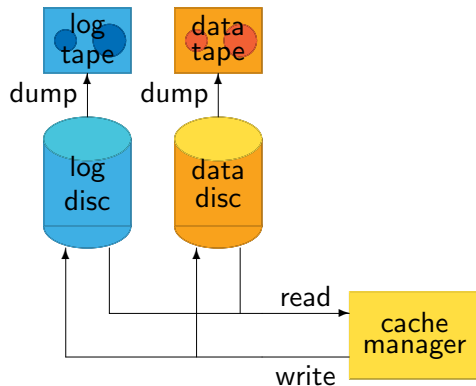
- 1 perform UNDOs of non-committed transactions back to penultimate *cp*
- 2 perform UNDO of non-committed transactions before penultimate *cp* if they were active at *cp*
- 3 perform REDOs of committed transactions after penultimate *cp*

Media Failures: Mirroring (RAID-1)



- Keep more than one active copy of data and log
- Writes sent to both
- Read from either

Media Failures: Dumping

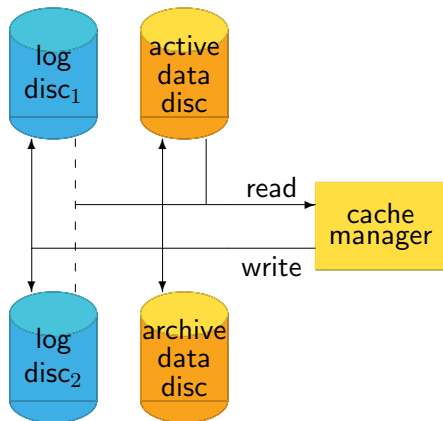


- 'tape' might also be a external file server, removable HD, *etc.*
- To use normal OS backup procedure
 - DBMS must not be still running
 - raw partition must not be used

Checkpoints and Dumps

- Dump must do a checkpoint
- Restore involves:
 - 1 copy tape to disc
 - 2 undo transactions active at the archive time
 - 3 redo transactions that committed after the archive
- commit consistent checkpoint obvious choice

Media Failures: Archive Database



- mirror log, but only have one active database
- periodically archive updates onto archive database
- failure of active database disc involves restore of archive database using logs

THE END

- Content of the course is what has been presented in the lectures
- Revise by reviewing worksheets and courseworks
- 2011 exam papers onwards set to current syllabus
- Older exam questions mostly apply, but there is more emphasis on RA and SQL, less on concurrency.