CS422 Principles of Database Systems

Failure Recovery

Chengyu Sun
California State University, Los Angeles

ACID Properties of DB Transaction

- Atomicity
- Consistency
- Isolation
- Durability

Failure Recovery

- Ensure atomicity and durability despite system failures

```
             start transaction;
             select balance from accounts where id=1;
             update accounts set balance=balance-100
System crash       where id=1;
  ---------->
             update accounts set balance=balance+100
                   where id=2;
System crash commit;
  ---------->
```

Failure Model

- System crash
  - CPU halts
  - Data in memory is lost
  - *Data on disk is OK*
- Everything else

Logging

- Log
  - A sequence of *log records*
  - Append only

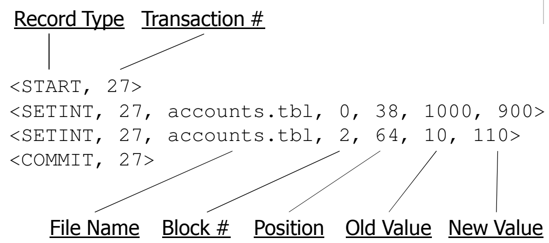
What Do We Log

Transaction → Log

```
start transaction;
select balance
      from accounts
      where id=1;
update accounts
      set balance=balance-100
      where id=1;
update accounts
      set balance=balance+100
      where id=2;
commit;
```

??

Log Records in SimpleDB



```
<START, 27>
<SETINT, 27, accounts.tbl, 0, 38, 1000, 900>
<SETINT, 27, accounts.tbl, 2, 64, 10, 110>
<COMMIT, 27>
```

General Notation for Log Records

- <START, T>
- <UPATE, T, X, v_x, v_x' >
- <COMMIT, T>
- <ABORT, T>

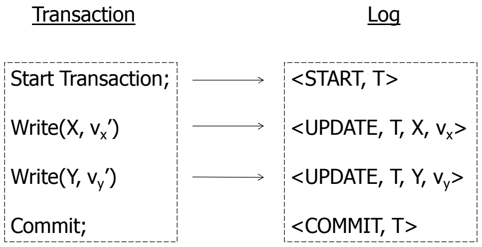
Recover from System Crash

- Remove changes made by uncommitted transactions – Undo
- Reapply changes made by committed transactions – Redo

Recover with Undo Only

- Assumption: all changes made by *committed* transactions have been saved to disk

Example: Create Undo Logging Records



Transaction		Log
Start Transaction;	→	<START, T>
Write(X, v_x')	→	<UPDATE, T, X, v_x>
Write(Y, v_y')	→	<UPDATE, T, Y, v_y>
Commit;		<COMMIT, T>

About Logging

- Undo logging records do not need to store the new values
  - Why??
- The key of logging is to decide when to flush to disk
  - The changes made by the transaction
  - The log records

Example: Flushing for Undo Recovery

- Order the actions, including `Flush(X)` and `Flush(<log>)`, into a sequence that allows Undo Recovery

Transaction	Log
Start Transaction;	<START, T>
Write(X, v_x')	<UPDATE, T, X, v_x>
Write(Y, v_y')	<UPDATE, T, Y, v_y>
Commit;	<COMMIT, T>

Order Flush(X) and Flush(<UPDATE,X>) for Undo

- Consider an incomplete transaction
  - (a) Both X and <UPDATE,X> are written to disk
  - (b) X is written to disk but not <UPDATE,X>
  - (c) <UPDATE,X> is written to disk but not X
  - (d) Neither is written to disk

Write-Ahead Logging

- A modified buffer can be written to disk only *after* all of its update log records have been written to disk

Implement Write-Ahead Logging

- Each log record has a unique id called *log sequence number* (LSN)
- Each buffer page keeps the LSN of the log record corresponding to the latest change
- Before a buffer page is flushed, notify the log manager to flush the log up to the buffer's LSN

Order Flush(<COMMIT,T>) for Undo

- <COMMIT,T> cannot be written to disk before new value of X is written to disk
- Commit statement cannot return before <COMMIT,T> is written to disk

Undo Logging

- Write <UPDATE,T,X,v_x> to disk *before* writing new value of X to disk
- Write <COMMIT,T> *after* writing all new values to disk
- COMMIT returns *after* writing <COMMIT,T> to disk

Undo Recovery

- Scan the log
  - *Forward or backward??*
- <COMMIT,T>: add T to a list of committed transactions
- <UPDATE,T,X,v_x>: if T is not in the lists of committed transactions, restore X's value to v_x

Undo Logging and Recovery Example

- Consider two transactions T_1 and T_2
  - T_1 updates X and Y
  - T_2 updates Z
- Show a possible sequence of undo logging
- Discuss possible crushes and recoveries

About Undo Recovery

- No need to keep the new value
- Scan the log once for recovery
- COMMIT must wait until all changes are flushed
- Idempotent – recovery processes can be run multiple times with the same result

Recover with Redo Only

- Assumption: *none* of the changes made by *uncommitted* transactions have been saved to disk

Example: Flushing for Redo Recovery

- Order the actions, including `Flush(X)` and `Flush(<log>)`, into a sequence that allows Undo Recovery

Transaction	Log

Start Transaction;
Write(X, v_x')
Write(Y, v_y')
Commit;
	<START, T>
	<UPDATE, T, X, v_x'>
	<UPDATE, T, Y, v_y'>
	<COMMIT, T>

Redo Logging

- Write <UPDATE,T,X,v_x'> and <COMMIT,T> to disk *before* writing *any* new value of the transaction to disk
- COMMIT returns *after* writing <COMMIT,T> to disk

Redo Recovery

- Scan the log to create a list of committed transactions
- Scan the log again to replay the updates of the committed transactions
  - *Forward or backward??*

About Redo Recovery

- A transaction must keep all the blocks it needs pinned until the transaction completes – increases buffer contention

Combine Undo and Redo – Undo/Redo Logging

- Write <UPDATE,T,X,v_x,v_x'> to disk *before* writing new value of X to disk
- COMMIT returns *after* writing <COMMIT,T> to disk

Undo/Redo Recovery

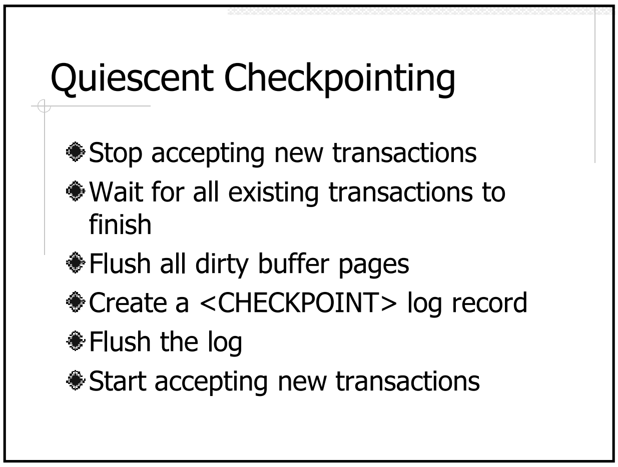
- Stage 1: undo recovery
- Stage 2: redo recovery

Advantages of Undo/Redo

- Vs. Undo??
- Vs. Redo??

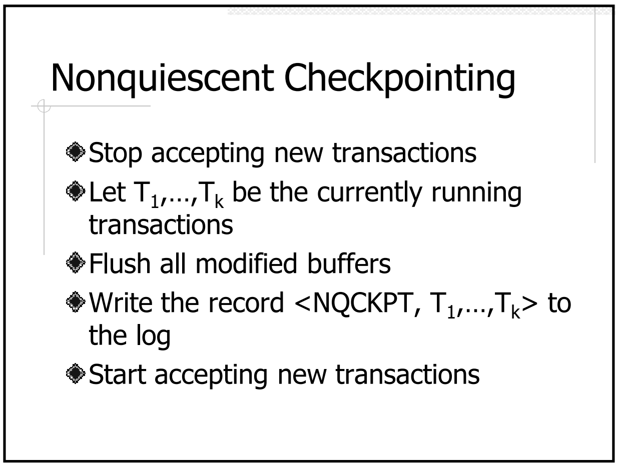
Checkpoint

- Log can get very large
- A recovery algorithm can stop scanning the log if it knows
  - All the remaining records are for completed transactions
  - All the changes made by these transactions have been written to disk



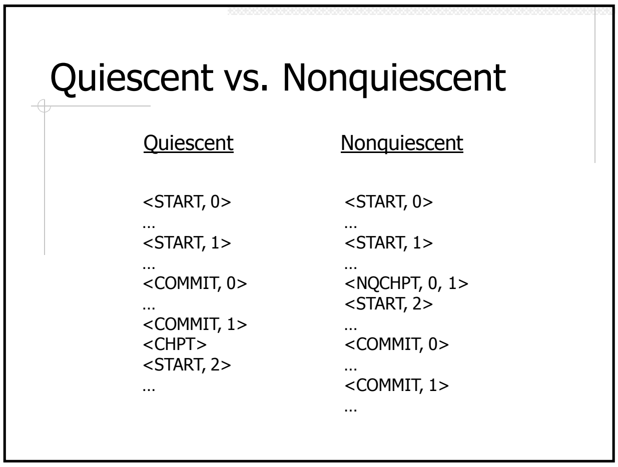
Quiescent Checkpointing

- Stop accepting new transactions
- Wait for all existing transactions to finish
- Flush all dirty buffer pages
- Create a <CHECKPOINT> log record
- Flush the log
- Start accepting new transactions



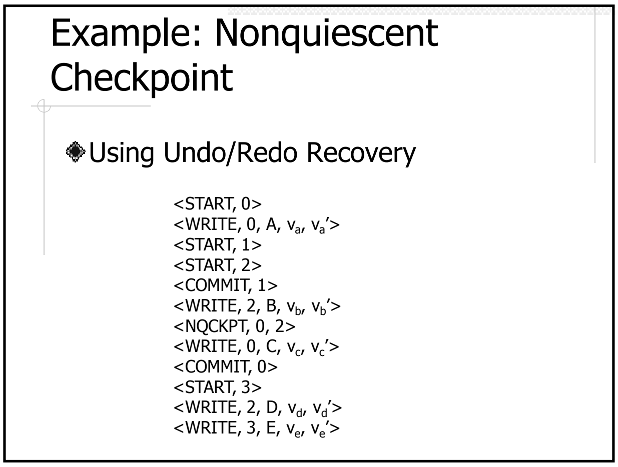
Nonquiescent Checkpointing

- Stop accepting new transactions
- Let $T_1,\ldots,T_k$ be the currently running transactions
- Flush all modified buffers
- Write the record <NQCKPT, $T_1,\ldots,T_k$> to the log
- Start accepting new transactions



Quiescent vs. Nonquiescent

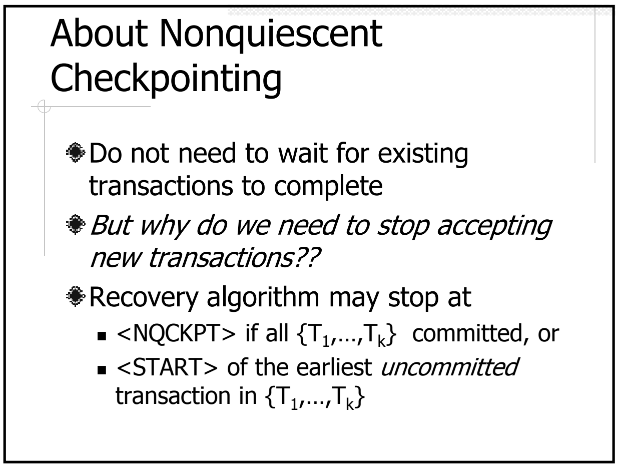
Quiescent	Nonquiescent
<START, 0>	<START, 0>
…	…
<START, 1>	<START, 1>
…	…
<COMMIT, 0>	<NQCHPT, 0, 1>
…	<START, 2>
<COMMIT, 1>	…
<CHPT>	<COMMIT, 0>
<START, 2>	…
…	<COMMIT, 1>
	…



Example: Nonquiescent Checkpoint

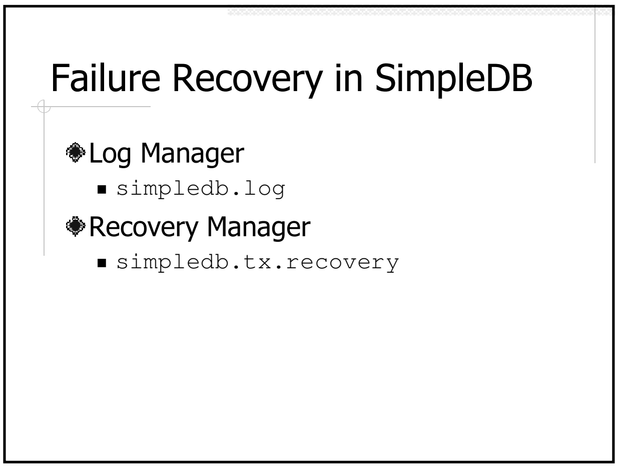
- Using Undo/Redo Recovery

<START, 0>
<WRITE, 0, A, v_a, v_a'>
<START, 1>
<START, 2>
<COMMIT, 1>
<WRITE, 2, B, v_b, v_b'>
<NQCKPT, 0, 2>
<WRITE, 0, C, v_c, v_c'>
<COMMIT, 0>
<START, 3>
<WRITE, 2, D, v_d, v_d'>
<WRITE, 3, E, v_e, v_e'>



About Nonquiescent Checkpointing

- Do not need to wait for existing transactions to complete
- *But why do we need to stop accepting new transactions??*
- Recovery algorithm may stop at
  - <NQCKPT> if all $\{T_1,\ldots,T_k\}$ committed, or
  - <START> of the earliest *uncommitted* transaction in $\{T_1,\ldots,T_k\}$



Failure Recovery in SimpleDB

- Log Manager
  - `simpledb.log`
- Recovery Manager
  - `simpledb.tx.recovery`

SimpleDB Log Manager

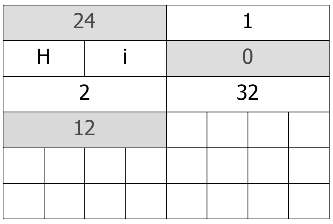
- Log file: `${USER}/${DB}/simpledb.log`
- Grows the log one block at a time
- The last block is kept in memory (i.e. only needs one page)

Append()

- Records are treated as arrays of objects (String or int)
- A new block is created if the current block does not have enough room to hold the new record
- The LSN of a log record is the block number

Locate Records in a Block

Two records: `<1, 'Hi'>`, `<2,32>`



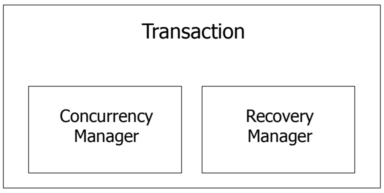
24				1			
H		i		0			
2				32			
12							

LogIterator

- LogIterator iterates through a log *backwards*
- Again, only keeps one block in memory
- `BasicLogRecord` is simply a page and the starting position of a record in the page – it's up to the Recovery Manager to decide how to read the record

SimpleDB Recovery Manager

- Each transaction has its own recovery manager



LogRecord Interface

- Record types
  - Checkpoint (quiescent)
  - Start
  - Commit
  - Rollback
  - SetInt
  - SetString
- Record operations
  - Write to log
  - Get record type
  - Get transaction #
  - Undo
  - [Redo]

Log Record Format

- Array of Integer and String
  - Record type
  - Additional information (optional)
- See the `writeToLog()` method in each log record class

LogRecordIterator

- Built on top of LogIterator
- Convert each BasicLogRecord to an a LogRecord object

Example: LogViewer

- Display the log
  - Up to the last <CHECKPOINT>

Recovery Manager

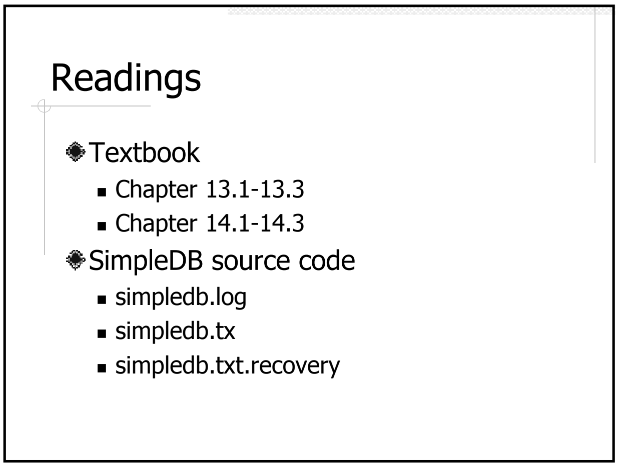
- Each transaction operation (e.g. start, commit, setint, setstring, rollback) creates a log record
- Rollback: undo the changes made by this transaction
- Recovery: perform recovery for the whole database

Undo Recovery in SimpleDB

- Recovery is done inside a transaction
- Iterate through the log backward
  - EOF or <Checkpoint>: stop
  - <Commit> or <Abort>: add transaction number to a list of *finished transactions*
  - Other: if the transaction # is not in the list of finished transactions, call undo()
- Save the changes (i.e. flush buffers)
- Write a <Checkpoint> log record

Example: TestLogWriter

- Write some records in the log for testing purpose



Readings

- Textbook
  - Chapter 13.1-13.3
  - Chapter 14.1-14.3
- SimpleDB source code
  - simpledb.log
  - simpledb.tx
  - simpledb.txt.recovery