**Database Concepts**
- Application: DBM – eCommerce
  - Supply DB (inventory) – decrement
  - Billing - create bill
  - Shipping - create order for factory for ship
- Failure reasons (3)
  - Normally just overhead – it all works
  - Programming style error, e.g. abort changes
  - System crashes, credit card authorized failure, DB crashes

**ACID**
- **Atomicity**
  - "All or nothing"
  - undo aborted
  - redo committed
  - coordinate with distributed
  - A correct transformation of the source
- **Consistency**
  - Aborting any transactions that fail to pass R.M. consistency tests
  - "Once done, it is done"
- **Durability**
  - Forcing all log records of committed transactions to durable memory
  - "A correct transformation of the source"
- **Isolation**
  - Individual transactions do not bump into one another

**DO-UNDO-REDO Protocol**
- **Programming style for resource managers**
- **Structure each operation ("DO") so it can be undone or redone**
  - Any operation should perform
  - The actual operation ("DO")
  - A log record
  - An "UNDO" program
  - A "REDO" program
**DO-UNDO-REDO Protocol**

**DO (Normal Operations):**
- Old State
- Log Record
- New State

**UNDO (Back out changes):**
- New State
- Log Record
- Old State

**REDO (Ensure changes):**
- Old State
- Log Record
- New State

**Log Records:**
- Begin_Tx = "123"
- UNDO/REDO
- Widget | OLD = 22, NEW = 21
- REDO/UNDO
- Quantity | OLD = 21, NEW = 30
- COMMIT
- Tx = "123"

**Example Transactions:**
- **1.** Begin
  - Decrement widget: (22 -> 21)
  - Increment Bob’s Balance: (21 -> 30)
- **2.** Commit

**Database Disk Blocks:**
- Tx - 1
  - Start
- Tx - 2
  - [x]
- Tx - 3
  - [z]
- Tx - 1
  - Commit
- Tx - 2
  - [x]
- Tx - 1
  - [x]

**Buffer Manager:**
- Replaces page - 2

**Log is a critical component:**
- Very asynchronous – lots of I/O must occur concurrently
- Allows regular I/O to be asynchronous or deferred
- Buffer manager

**Logs (continued):**
- Log force at every Tx would be a serial bottleneck
- If lots of Txs are concurrently committing (approximately) a log force can handle lots of Txs
- WADS concept

**Data Flow in a System Restart:**
- Transaction Manager
  1. Find checkpoint
  2. Read log forward
  3. REDO each operation
- At END:
  1. UNDO un-commit transactions

**Buffer Manager:**
- Memory
  - Numerous changes
  - Logs can be the bridge between these
Normal (no failure) execution of an application interacting with a resource manager

```
Begin_work()
Application
Commit_work()
```

Normal functions

```
Lock
Log
Transaction Manager
Write_commit()
```

Commit phase 1?

YES/NO

Commit phase 2?

ACK

```
Work request
Join_work
Log request
```

The Heart of the Issue: Force Log of Prepare & Commit

```
Begin_Tx= " 123 "
```

UNDO/REDO

```
Widget | OLD=22, NEW=21
```

REDO/UNDO

```
Quantity | OLD=21, NEW=30
```

Commit

Tx= " 123 "

```
Answer YES to Phase 1: "prepared to Commit OR Abort"
```

Two Phase Commit

- We have shown two phase for one system, one RM
- Can be extended:
  - Multiple RM’s on a single system
  - Multiple RM’s on distributed systems

Two phase commit: Making computations atomic

- Tx Manager asks RM’s to commit
  - Can be multiple ones
  - Can be on different systems
  - Each Vote YES or NO
    - Vote YES means “Prepared to commit or Abort”
  - Tx Manager proceeds to commit if all YES
    - Forces “End Phase 1” log record
    - This is the “atomic instance”

For example…

- Application wants to commit Tx TRID=555
  1. Application must begin a Tx
  2. RM’s must tell Tx Manager they want to join a transaction.

Two Phase Commit

Restart Processing

- Transaction manager – finds all
  - Begin_work with no commit record => Transactions to abort
  - Begin_work with commit => RM’s must be told with no complete about Tx’s
    - Begin_work + commit + complete => Ignore
- RM Restart
  - Must find work either process an abort (UNDO) or ensure everything is done (REDO)
Why is this complicated?

...between two products

...between distributed systems

Distributed Transactions

Can get complicated...

Distributed Two Phase

- Prepare now has a time-out
- Commit record has names of distributed RMs who need to know about phase 2 commit
- "In doubt" of CPM — voted YES, but does not know. Must poll upstream Tx Mgr
- Presumed abort protocol: optimization, coordinator is queried on "Tx that is not active, prepared, or committed, then assume aborted"
- Assumption that no record found == abort
  - participant sends ACK only when the completion state is durable
  - Coordinator only sends commit records if all commit messages have been acknowledged
Putting it All Together

- Basic transaction & database concepts
- Normal commit processing
- Importance of the LOG
- Two Phase Commit
- Extending for Distributed Processing
  Chapter 10 “Transaction Manager Concepts” pages 529-582
  Note -- 1000 other pages!!!