Announcements

- Wrap-up lecture on Friday
  - Short review + example problems on the board

- Project 4 due this Friday
  - Don’t forget to terminate your jobs!!!

- Course evaluations at the end of this lecture

- Today: Distributed transactions
  - Because you loved transactions so much the first time
### Partitioned data

#### Employee

<table>
<thead>
<tr>
<th>TID</th>
<th>eid</th>
<th>name</th>
<th>city</th>
<th>age</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>53666</td>
<td>Jones</td>
<td>Madras</td>
<td>28</td>
<td>35k</td>
</tr>
<tr>
<td>t2</td>
<td>53688</td>
<td>Smith</td>
<td>Chicago</td>
<td>38</td>
<td>32k</td>
</tr>
<tr>
<td>t3</td>
<td>53650</td>
<td>Smith</td>
<td>Chicago</td>
<td>29</td>
<td>48k</td>
</tr>
<tr>
<td>t4</td>
<td>53831</td>
<td>Madayan</td>
<td>Bombay</td>
<td>41</td>
<td>20k</td>
</tr>
<tr>
<td>t5</td>
<td>53832</td>
<td>Guldu</td>
<td>Bombay</td>
<td>32</td>
<td>20k</td>
</tr>
</tbody>
</table>

**Vertical Fragment**

**Horizontal Fragment**
Distributed Data

UPDATE Employee
SET salary=salary+2k
WHERE age>30
Distributed Catalog

How do we identify a relation?

- Naming issues:
  - local name + birth site = global relation name
  - +replica_id = global replica name

Centralized catalog

- Vulnerable to single-site failure
- Compromizes site autonomy

R* approach:

- Local catalog describing all local relations
- Birth site also keeps track of replicas and fragments
  - Could be cached at other sites
Remember Transactions?

- ACID

- Distributed Concurrency Control
  - How can locks for objects be managed?
  - How can deadlocks be detected?

- Distributed Recovery
  - Atomicity and Durability need to be enforced across sites

- In a distributed setting, a Xact spawns subtransactions
Distributed Lock management

- **Centralized**
  - one site deals with lock and unlock requests

- **Primary Copy**
  - One copy of an object is designated as primary, and requests are handled at that site

- **Fully Distributed**
  - Manage requests locally
Deadlock detection

- Local and global waits-for graphs
  - At site A
  - At site B

- 3 algorithms:
  1. Construct global waits-for graph periodically at a centralized site
  2. Construct waits-for graphs hierarchically
  3. Abort long waiting transactions

- Phantom Deadlocks!
Distributed Recovery

- Either all subtransactions must commit or none of them

- Regular logging + commit protocol

- The transaction manager at the originating site is the coordinator

- The transaction managers at the subtransactions’ sites are the subordinates
2 Phase Commit: Motivation

1. User decides to commit
   coordinator

2. commit
   subordinate 1

3. commit
   subordinate 2
   But I already aborted 😞

4. coordinator crash!
   subordinate 3
2 Phase Commit

- Use 2 phases: a voting phase and a termination phase

- Principle:
  - When a process makes a decision, it votes yes/no or commit/abort
  - A subordinate acknowledges messages (acks)
  - Force-write log record before sending
  - Log records include Xact and coordinator ids
  - Coordinator logs ids of subordinates
2 Phase Commit: Phase 1

1. User decides to commit

2. prepare

3. force-write prepare

4. yes

coordinate

subordinate 1

subordinate 2

subordinate 3
2 Phase Commit: Phase 2

1. Force-write commit
2. commit
3. force-write commit
4. ack
5. Commit Xact and forget it

Xact is now committed
2 Phase Commit: Phase 1 with abort

1. User decides to commit

2. prepare

subordinate 1

3. force-write prepare

4. yes

subordinate 2

4. no

3. force-write abort

4. no

subordinate 3

5. Abort Xact and forget it

2. prepare

4. no
2 Phase Commit: Phase 2

1. Force-write abort
2. abort
3. force-write abort
4. ack
5. Write end then forget Xact

subordinate 1
subordinate 2
subordinate 3

5. Abort Xact and forget it
Restart after failure

- How do we know if we are coordinator or subordinate, and what do we do?

- We see a commit or abort record
  - We are coordinator: send to subordinates until we get an ack

- We see a prepare record
  - We are subordinate: contact coordinator to determine status

- We see no prepare, commit or abort
  - We can unilaterally abort

Any issues?
Refinement: 2PC with presumed abort

Observations:
- Coordinator waits for acks to ‘forget’ Xact
- no information = abort
- A reader does not care for commit/abort outcome

Refinements:
- If abort is decided, remove Xact from Xact table immediately
- If I get an abort msg, no need to ack
- The abort log record of the coordinator does not need the subordinate list
- Abort records don’t need to be force-written
- A reader Xact votes reader instead of yes/no
- Coordinator does not need to communicate further with readers
- If all are readers, no need for the 2\(^{nd}\) phase