LTeam Milestone 3

Alejandro Torres, Jenny Wang, Ho-Chih Ma, Jamie Wu, Karthik Palanisamy
Team Member Roles

Leadership Roles:
- Team Coordinators: Jenny, Alejandro
- System Architects: Everyone
- Developers: Everyone
- Testers: Everyone

Implementation and Design Areas:
- Transaction Semantics: Jamie, Howard
- Multithreading Concurrency Control: Jamie, Howard
- Future Implementation: Karthik, Jenny, Alejandro
- Performance: Karthik, Jenny, Alejandro
transaction semantics
multithreading concurrency control
future implementation
performance
live demo and q&a
(1) Transaction Semantics
Transaction & Transaction Worker

Transaction:
- A sequence of queries
- Either read operations or write operations

Transaction worker:
- Contains a list of transactions
- Keeps track of the status of transactions

Transaction Worker

Transaction
- R(A)
- W(A)
- R(B)
- ...
- W(X)

Query
- R(X) • Select
- W(X) • Insert
  • Delete
  • Update
Goal: Achieve atomicity by determining whether transactions are valid and committing groups of queries or none with aborts to release any acquired locks to prevent deadlock.

Transaction Worker

Run a transaction

Iteratively check query arguments: are the arguments valid?

Yes

Outcome 3: Commit

Commit to the transaction, execute queries on database

No

Outcome 1: “Hard” Abort

Rollback any acquired locks in the translation

Are all necessary locks acquired?

Yes

No

Outcome 2: “Soft” Abort

Rollback any acquired locks, transaction is moved to end of transaction worker queue to be attempted again later
Concurrent Using Strict 2PL Policy

Motivation
- To avoid race conditions – no threads should have access to resources at the same time
- Preventing anomalies with interleaved execution
  - WR Conflicts
  - RW Conflicts
  - WW Conflicts

Implementation
- Locks are implemented in record level (Physical Page)
- If a column in a record is accessed, all columns will be locked
- Using Python Threading, Lock(), Acquire(), and Release()
Shared and Exclusive Locks

Goal: To achieve isolation in the database while preserving read efficiency

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Reading</th>
<th>Writing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Writing</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Post Milestone 3 Ideas

1] Use a different programming language, such as C
   a] To achieve true parallelism
   b] Better memory management
   c] Overall increase in performance time

2] Establish a special priority algorithm that executes threads in the most efficient order
[5] Performance
Overall Milestone Performance

Inserting 10k records took: 3.875
Updating 10k records took: 19.46875
Selecting 10k records took: 2.3125
Aggregate 10k of 100 record batch took: 0.234375
DELETING 10 RECORDS
Deleting 10k records took: 0.765625
total db time: 26.65625
Live Demo and Q&A
Thank you!