Transaction Semantics
Shared & Exclusive Locks (Granting Conditions)
Lock Management

Concurrency Control
No-Wait 2PL Policy

Forward Looking
Post-M3 Database Program
Transaction Semantics
# Shared & Exclusive Locks

<table>
<thead>
<tr>
<th>Conditions of granting different locks:</th>
<th>Initially Holds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request For</td>
<td>Shared</td>
</tr>
<tr>
<td>Shared</td>
<td>Grant</td>
</tr>
<tr>
<td>Exclusive</td>
<td>No Grant</td>
</tr>
</tbody>
</table>

* One exception if the same thread tried to exchange its shared lock for an exclusive lock, we will grant the request.

**Shared Locks:**
1. Select
2. Sum

**Exclusive Locks:**
1. Insert
2. Delete
3. Update
Lock Management

Using Python Dictionary to keep track of all the locks on different RIDs

<table>
<thead>
<tr>
<th>RID</th>
<th>Lock(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[Exclusive: Worker 10]</td>
</tr>
<tr>
<td>2</td>
<td>[Shared: Worker 4]</td>
</tr>
<tr>
<td>3</td>
<td>[Exclusive: Worker 9]</td>
</tr>
<tr>
<td>4</td>
<td>[Shared: Worker 8, Shared: Worker 2]</td>
</tr>
<tr>
<td>5</td>
<td>[Exclusive: Worker 7]</td>
</tr>
</tbody>
</table>

Abort Transaction

Commit Transactions

Lock Management Map
Concurrency Control
Strict No-Wait 2PL Policy

Scenario 1: The thread was able to obtain the lock successfully

*Note: Rollback is only for insert, update, and delete. For read and sum, the thread will simply abort
Strict No-Wait 2PL Policy

Scenario 2: The thread was not able to obtain the lock and has to abort

---

*Note: Rollback is only for insert, update, and delete. For read and sum, the thread will simply abort*
Roadmap

- Discussed and implemented shared, exclusive locks, and lock map
- Implemented abort and rollback mechanism for failed threads
- Integrated meaningful logging when threads call abort method
- Attempted a retry mechanism for aborted threads using Priority Queue
- Rolled back to former implementation without retry mechanism
Attempted a second version of Milestone 3 as a prototype, which replicates the Consumer-Producer model to resemble the real world use of database with threads.

**Basic Idea:**
- Create a transaction queue that stores all the aborted thread so that we can re-execute them
- Create layers of queues to effectively re-arrange threads based on transactions to enhance performance
Thank you!