ARIES (& Logging)

April 2-4, 2018
What does it mean for a transaction to be committed?
If commit returns successfully, the transaction…

- … is recorded completely (atomicity)
- … left the database in a stable state (consistency)
- …’s effects are independent of other xacts (isolation)
- … will survive failures (durability)
commit returns successfully = the xact’s effects are visible forever
commit returns successfully = the xact’s effects are visible forever

commit called but doesn’t return = the xact’s effects may be visible
Motivation

T1
T2
T3
T4
T5

Time
Motivation
Motivation

T1  
T2  
T3  
T4  
T5  

Time
Motivation

T1
T2
T3
T4
T5

Time
Motivation

T1

T2

T3

T4

T5

Time
Motivation

T1, T2, T3, T4, T5

Time
Motivation

Time

T1

T2

T3

T4

T5
Motivation

T1
T2
T3
T4
T5

Time

CRASH!
Motivation

Committed Transactions. These should be present when the DB restarts.

T1
T2
T3
T4
T5

Time

Image copyright: Wikimedia Commons
Motivation

Committed Transactions. These should be present when the DB restarts.

Uncommitted Transactions. These should leave no trace.
• How do we guarantee durability under failures?

• How do aborted transactions get rolled back?

• How do we guarantee atomicity under failures?
Problem 1: Providing durability under failures.
Simplified Model
When a write succeeds, the data is completely written
Problems

• A crash occurs part-way through the write.

• A crash occurs before buffered data is written.
Write-Ahead Logging

Before writing to the database, first write what you plan to write to a log file...

\[ W(A:10) \]
Write-Ahead Logging

Once the log is safely on disk you can write the database

Log

\[ W(A:10) \]
Write-Ahead Logging

Log is append-only, so writes are always efficient

Log
W(A:10)
W(C:8)
W(E:9)
Write-Ahead Logging

...allowing random writes to be safely batched

Log
W(A:10)
W(C:8)
W(E:9)
Problem 2: Providing rollback.
Single DB Model

Txn 1
A = 20
B = 14
COMMIT

Txn 2
E = 19
B = 15
ABORT

Image copyright: OpenClipart (rg1024)
Single DB Model

Txn 1
A = 20
B = 14
COMMIT

Txn 2
E = 19
B = 15
ABORT

Image copyright: OpenClipart (rg1024)
Single DB Model

**Txn 1**
- A = 20
- B = 14
- COMMIT

**Txn 2**
- E = 19
- B = 15
- ABORT

Image copyright: OpenClipart (rg1024)
Single DB Model

Txn 1
A = 20
B = 14
COMMIT

Txn 2
E = 19
B = 15
ABORT
Single DB Model

**Txn 1**

A = 20  
B = 14  
COMMIT

**Txn 2**

E = 19  
B = 15  
ABORT

![Database Diagram](image)
Staged DB Model

**Txn 1**
A = 20  
B = 14  
COMMIT

**Txn 2**
E = 19  
B = 15  
ABORT

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>5</td>
<td>18</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>5</td>
<td>18</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Staged DB Model

**Txn 1**
A = 20
B = 14
COMMIT

**Txn 2**
E = 19
B = 15
ABORT
Staged DB Model

**Txn 1**

A = 20
B = 14

COMMIT

**Txn 2**

E = 19
B = 15

ABORT
Is staging always possible?
• Staging takes up more memory.

• Merging after-the-fact can be harder.

• Merging after-the-fact introduces more latency!
Problem 2: Providing rollback.

for the single database model
UNDO Logging

Store both the “old” and the “new” values of the record being replaced

\[
\begin{align*}
\text{Log} \\
W(A:8\rightarrow 10) \\
W(C:5\rightarrow 8) \\
W(E:16\rightarrow 9)
\end{align*}
\]
UNDO Logging

Active Xacts
Xact: 1, Log: 45
Xact: 2, Log: 32

Log
43: W(A: 8 → 10)
44: W(C: 5 → 8)
45: W(E: 16 → 9)
UNDO Logging

Active Xacts
Xact: 1, Log: 45
Xact: 2, Log: 32

Log
43: W(A: 8→10)
44: W(C: 5→8)
45: W(E: 16→9)

Image copyright: OpenClipart (rg1024)
UNDO Logging

Active Xacts

Xact: 1, Log: 45
Xact: 2, Log: 32

Log

43: W(A: 8→10)
44: W(C: 5→8)
45: W(E: 16→9)

Image copyright: OpenClipart (rg1024)
UNDO Logging

Active Xacts
Xact: 1, Log: 45
Xact: 2, Log: 32

Log
43: W(A: 8→10)
44: W(C: 5→8)
45: W(E: 16→9)

Image copyright: OpenClipart (rg1024)
UNDO Logging

Active Xacts
Xact: 45
Xact: 2, Log: 32

Log
43: W(A: 8 → 10)
44: W(C: 5 → 8)
45: W(E: 16 → 9)

Image copyright: OpenClipart (rg1024)
Log Sequence Number
Linked Lists

Transaction Table

Log
Log Sequence Number
Linked Lists

Transaction Table

Log (necessary for crash recovery)
Log Sequence Number
Linked Lists

Transaction Table
XID, LastLSN

Log
(necessary for crash recovery)
Log Sequence Number
Linked Lists

Transaction Table
- XID, LastLSN

Log
- LSN, Prev LSN, Prev Image, ...
- ABORT [XID]

(necessary for crash recovery)
Log Sequence Number
Linked Lists

Transaction Table

XID, LastLSN

LSN, Prev LSN, Prev Image, ...

ABORT [XID]

Log (necessary for crash recovery)
Log Sequence Number
Linked Lists

Log

LSN, Prev LSN, Prev Image, …

Transaction Table
XID, LastLSN

LSN, Prev LSN, Prev Image, …

ABORT [XID]

(necessary for crash recovery)
Problem 3: Providing atomicity.
Goal: Be able to reconstruct all state at the time of the DB’s crash (minus all running xacts)
What state is relevant?
DB State

Active Xacts
Xact:1, Log: 45
Xact:2, Log: 32

Log
43: W(A: 8 → 10)
44: W(C: 5 → 8)
45: W(E: 16 → 9)
**DB State**

**On-Disk**
(or rebuildable)

**In-Memory Only!**

**Active Xacts**

- Xact: 1, Log: 45
- Xact: 2, Log: 32

**On-Disk**

**Log**

- Xact 43: $W(A:8\rightarrow 10)$
- Xact 44: $W(C:5\rightarrow 8)$
- Xact 45: $W(E:16\rightarrow 9)$

Image copyright: OpenClipart (rg1024)
Rebuilding the Xact Table

Log every COMMIT
(replay triggers commit process)

Log every ABORT
(replay triggers abort process)

New message: END
(replay removes Xact from Xact Table)
Rebuilding the Xact Table

Log every COMMIT
(replay triggers commit process)

Log every ABORT
(replay triggers abort process)

New message: END
(replay removes Xact from Xact Table)

What about BEGIN?
(when does an Xact get added to the Table?)
Transaction Commit

- Write **Commit** Record to Log

- All Log records up to the transaction’s LastLSN are flushed.

- Note that Log Flushes are Sequential, Synchronous Writes to Disk

- **Commit()** returns.

- Write **End** record to log.
Simple Transaction Abort (supporting crash recovery)

• Before restoring the old value of a page, write a Compensation Log Record (CLR).
  • Logging continues during UNDO processing.
  • CLR has an extra field: UndoNextLSN
    • Points to the next LSN to undo (the PrevLSN of the record currently being undone)
  • CLRs are never UNDOne.
    • But might be REDOne when repeating history.
    • (Why?)
Rebuilding the Xact Table

**Optimization**: Write the Xact Table to the log periodically. (checkpointing)
ARIES Crash Recovery

- Start from checkpoint stored in master record.

- **Analysis:** Rebuild the Xact Table

- **Redo:** Replay operations from all live Xacts (even uncommitted ones).

- **Undo:** Revert operations from all uncommitted/aborted Xacts.