Everything is a Transaction

Unifying Logical Concurrency Control and Physical Data Structure Maintenance in Database Management Systems – CIDR 2021
Motivation

Index

Data Table

Version Deltas

Arrays
Linked Lists
Hash Tables
Trees
Motivation

Transaction
BEGIN;
DELETE FROM table
WHERE key = a;
COMMIT;

Transaction
BEGIN;
SELECT FROM table
WHERE key = a;
COMMIT;

Arrays
Linked Lists
Hash Tables
Trees
Motivation

Transaction
BEGIN;
DELETE FROM table
WHERE key = a;
COMMIT;

Transaction
BEGIN;
SELECT FROM table
WHERE key = a;
COMMIT;

Must **defer** the removal of physical data until no active txn able to see it.

Arrays
Linked Lists
Hash Tables
Trees
Deferred Action Framework (DAF)

Asynchronous execution framework for internal maintenance actions in a multi-versioned DBMS.

**Key Idea:** DAF executes actions as transactions with the same visibility mechanisms.

- Single API call: `DEFER(action)`
- Each action is tagged with txn's commit timestamp.
- Invoke an action when there are no transactions with a start timestamp smaller than tagged timestamp.
Deferred Action Framework (DAF)

Transaction
BEGIN;
DELETE FROM table
WHERE key = a;
COMMIT;

Action Queue
OLDEST \( \omega = t0 \)
Deferred Action Framework (DAF)

Transaction

BEGIN; \( t = t_1 \)
DELETE FROM table
WHERE key = a;
COMMIT;

Action Queue

OLDEST \( t = t_0 \)
Deferred Action Framework (DAF)

Transaction
BEGIN; ⌛ = t1
DELETE FROM table
WHERE key = a;
COMMIT;

Action Queue
OLDEST ⌛ = t1
Deferred Action Framework (DAF)

Transaction
BEGIN; ֺ= t1
DELETE FROM table
WHERE key = a;
COMMIT;

DEFER(Unlink Version Chain)
DEFER(Delete Index Key=a)

Generated Actions

Action Queue
OLDEST ֺ= t1
Deferred Action Framework (DAF)

Transaction
BEGIN; t1 = t1
DELETE FROM table
WHERE key = a;
COMMIT;

Transaction
BEGIN; t2 = t2
SELECT FROM table
WHERE key = a;
COMMIT;

DEFER(Unlink Version Chain)
DEFER(Delete Index Key=a)
Generated Actions

Action Queue
OLDEST t2 = t2
Deferred Action Framework (DAF)

Transaction
BEGIN; \( t_1 \)
DELETE FROM table
WHERE key = a;
COMMIT;

DEFER( Unlink Version Chain )
DEFER( Delete Index Key=a )

Generated Actions

Transaction
BEGIN; \( t_2 \)
SELECT FROM table
WHERE key = a;
COMMIT;

Action Queue
OLDEST \( t_2 \)
Deferred Action Framework (DAF)

Transaction
BEGIN; \( t_1 \)
DELETE FROM table
WHERE key = a;
COMMIT; \( t_3 \)

Transaction
BEGIN; \( t_2 \)
SELECT FROM table
WHERE key = a;
COMMIT;

Action Queue
OLDEST \( t_3 \)

Generated Actions
DEFER( Unlink Version Chain )
DEFER( Delete Index Key=a )
Deferred Action Framework (DAF)

Transaction
BEGIN; \(\mathbb{C}\) = t1
DELETE FROM table
WHERE key = a;
COMMIT; \(\mathbb{C}\) = t3

Transaction
BEGIN; \(\mathbb{C}\) = t2
SELECT FROM table
WHERE key = a;
COMMIT;

Transaction
BEGIN; \(\mathbb{C}\) = t3
UNLINK VERSION CHAIN Delete Index Key=a
COMMIT;

Action Queue
\textbf{OLDEST} \(\mathbb{C}\) = t3

Unlink Version Chain
\(\mathbb{C}\) = t3
Delete Index Key=a
\(\mathbb{C}\) = t3
Deferred Action Framework (DAF)

Transaction
BEGIN; ⌛ = t1
DELETE FROM table
WHERE key = a;
COMMIT; ⌛ = t3

Transaction
BEGIN; ⌛ = t2
SELECT FROM table
WHERE key = a;
COMMIT; ⌛ = t4

Action Queue
OLDEST ⌛ = t4
Unlink Version Chain ⌛ = t3
Delete Index Key=a ⌛ = t3
Deferred Action Framework (DAF)

Transaction
BEGIN; ⌛ = t1
DELETE FROM table
WHERE key = a;
COMMIT; ⌛ = t3

Transaction
BEGIN; ⌛ = t2
SELECT FROM table
WHERE key = a;
COMMIT; ⌛ = t4

Action Queue
OLDEST ⌛ = t4
Unlink Version Chain ⌛ = t3
Delete Index Key=a ⌛ = t3

Data Table
A
B
C
RECORD
Version Deltas
A₃ → A₂
B₅ → B₄ → B₃
C₈ → C₇
Deferred Action Framework (DAF)

Transaction
BEGIN; \( \circ = t1 \)
DELETE FROM table
WHERE key = a;
COMMIT; \( \circ = t3 \)

Transaction
BEGIN; \( \circ = t2 \)
SELECT FROM table
WHERE key = a;
COMMIT; \( \circ = t4 \)

Action Queue
OLDEST \( \circ = t4 \)
Unlink Version Chain \( \circ = t3 \)
Delete Index Key=a \( \circ = t3 \)

Index

Data Table

Version Deltas

Index

Data Table

Version Deltas
Deferred Action Framework (DAF)

Transaction
BEGIN; ⌛ = t1
DELETE FROM table
WHERE key = a;
COMMIT; ⌛ = t3

Transaction
BEGIN; ⌛ = t2
SELECT FROM table
WHERE key = a;
COMMIT; ⌛ = t4

Action Queue
OLDEST ⌛ = t4

Delete Index Key=a ⌛ = t3
Multi-Deferrals

Explicit ordering of concurrent actions.

▶ Example: action A drops a table; action B deletes a tuple in the table.

▶ Solution: Chained deferral of actions.

\[
\text{DEFER(DEFER(… DEFER(action)…))}
\]

Unwrap and reinsert multi-deferred action to queue with a later timestamp.

▶ Separate its execution with other concurrent actions.
Multi-Deferrals

Transaction
BEGIN; 🕒 = t1
DROP TABLE table;
COMMIT;

Action Queue
OLDEST 🕒 = t1
Multi-Deferrals

Transaction
BEGIN;   = t1
DROP TABLE table;
COMMIT;

DEFER(DEFER(Delete Table + Index))

Action Queue
OLDEST = t1
Multi-Deferrals

Transaction
BEGIN; \( t_1 \)
DROP TABLE table;
COMMIT;

Transaction
BEGIN; \( t_2 \)
DELETE FROM table
WHERE key = a;
COMMIT;

DEFER(Delete Table + Index)

Action Queue
OLDEST \( t_2 \)
Multi-Deferrals

Transaction
BEGIN; ⏳ = t1
DROP TABLE table;
COMMIT;

Transaction
BEGIN; ⏳ = t2
DELETE FROM table
WHERE key = a;
COMMIT;

Action Queue
OLDEST ⏳ = t2
Multi-Deferrals

Transaction

BEGIN;  \( t_1 \)
DROP TABLE table;
COMMIT;  \( t_3 \)

Transaction

BEGIN;  \( t_2 \)
DELETE FROM table
WHERE key = a;
COMMIT;

Action Queue

OLDEST  \( t_3 \)

DEFER(Delete Table + Index )  \( t_3 \)

DEFER(Unlink Version Chain )

DEFER(Delete Index Key=a )
Multi-Deferrals

Transaction
BEGIN; \( = t1 \)
DROP TABLE table;
COMMIT; \( = t3 \)

Transaction
BEGIN; \( = t2 \)
DELETE FROM table
WHERE key = a;
COMMIT; \( = t4 \)

Action Queue
OLDEST \( = t4 \)

\textbf{DEFER (Delete Table + Index)} \( = t3 \)
Unlink Version Chain \( = t4 \)
Delete Index Key=a \( = t4 \)
Multi-Deferrals

Transaction
BEGIN; = t1
DROP TABLE table;
COMMIT; = t3

Transaction
BEGIN; = t2
DELETE FROM table
WHERE key = a;
COMMIT; = t4

Action Queue
OLDEST = t5
DEFER (Delete Table + Index ) = t2
Unlink Version Chain = t4
Delete Index Key=a = t4
Multi-Deferrals

Transaction
BEGIN; \( \circ = t1 \)
DROP TABLE table;
COMMIT; \( \circ = t3 \)

Transaction
BEGIN; \( \circ = t2 \)
DELETE FROM table
WHERE key = a;
COMMIT; \( \circ = t4 \)

Action Queue
OLDEST \( \circ = t5 \)

Unlink Version Chain \( \circ = t4 \)
Delete Index Key=a \( \circ = t4 \)
Delete Table + Index \( \circ = t5 \)
Multi-Deferrals

Transaction
BEGIN;  
DROP TABLE table;
COMMIT;  

Transaction
BEGIN;  
DELETE FROM table
WHERE key = a;
COMMIT;  

Action Queue
OLDEST  

Delete Index Key=a  

Delete Table + Index  

At this time the table is not visible to any active txn and is safe to drop.
Applications

Index Cleaning

```
BEGIN;
    UPDATE table SET key=111
    WHERE key=222;
    COMMIT;
```

Delete Index Key=222;

```
\begin{tabular}{|c|c|c|}
\hline
BEGIN-TS & END-TS & KEY \\
\hline
A_1 & 1 & 222 \\
A_2 & 10 & 111 \\
\hline
\end{tabular}
```

Cache Invalidation

```
BEGIN;
    ALTER TABLE table
    DROP COLUMN b;
    COMMIT;
```

Remove Column table.b;

Invalidate Plan Cache;

```
PREPARE stmt1 AS
    SELECT a,b FROM table;
```

Data Transformation

```
BEGIN;
   DELETE Index Key=222;
```

Delete Row Store Block;

```
\begin{tabular}{|c|c|c|c|c|c|}
\hline
Row Store & A & B & C & D \\
\hline
\end{tabular}
```

```
\begin{tabular}{|c|c|c|c|}
\hline
Column Store & A & B & C & D \\
\hline
\end{tabular}
```

Access Monitor
Preliminary Results

Integrated DAF with NoisePage DBMS.

Measure MVCC GC scalability using TPC-C:
▶ One warehouse per worker thread.
▶ Stored Procedure API.
▶ Write-Ahead Logging Enabled.

Compare three system configurations:
▶ Single GC thread
▶ Dedicated DAF threads
▶ Cooperative DAF threads
MVCC GC Scalability (TPC-C)

Throughput (txn/sec) vs. # Threads

- Single-GC
- 2 DAF
- 4 DAF
- 8 DAF
- Coop-DAF
Added notice that the project is dead.

`agavio` committed on Mar 17, 2019

Showing 1 changed file with 9 additions and 0 deletions.

```
+ # UPDATE 2019-03-17
+ The Peloton project is "dead". We have abandoned this repository and moved on to build a new OMS. There are several engineering techniques and designs that we learned from this first system on how to support autonomous operations that we are doing a much better job at implementing in the second system.
+ We will no accept pull requests for this repository. We will also not respond to questions or problems that you may have with running this software.
```
NoisePage Project

In-Memory HTAP DBMS
Postgres compatible (wire, SQL, catalog)
Apache Arrow compatible columnar storage
HyPer-style MVCC (snapshot isolation)
Hybrid Vectorization + Pipeline Query Codegen
JIT Query Compilation (DSL→OpCodes→LLVM)
Integrated self-driving components
NoisePage Project

In-Memory HTAP DBMS
Postgres compatible (wire, SQL, catalog)
Apache Arrow compatible columnar storage
HyPer-style MVCC (snapshot isolation)
Hybrid Vectorization + Pipeline Query Codegen
JIT Query Compilation (DSL→OpCodes→LLVM)
Integrated self-driving components
Summary

DAF shows how to leverage logical data concurrency protocols for physical data structures.

Unifying the notion of visibility makes it easier to integrate new data structures and extend transactions to support maintenance operations.
Acknowledgements
END
https://noise.page