COMMUNITY
THE ASF CONFERENCE
CODE

BEGIN TRANSACTION

Apache Cassandra as a Transactional Database

C. Scott Andreas, Apple Inc.

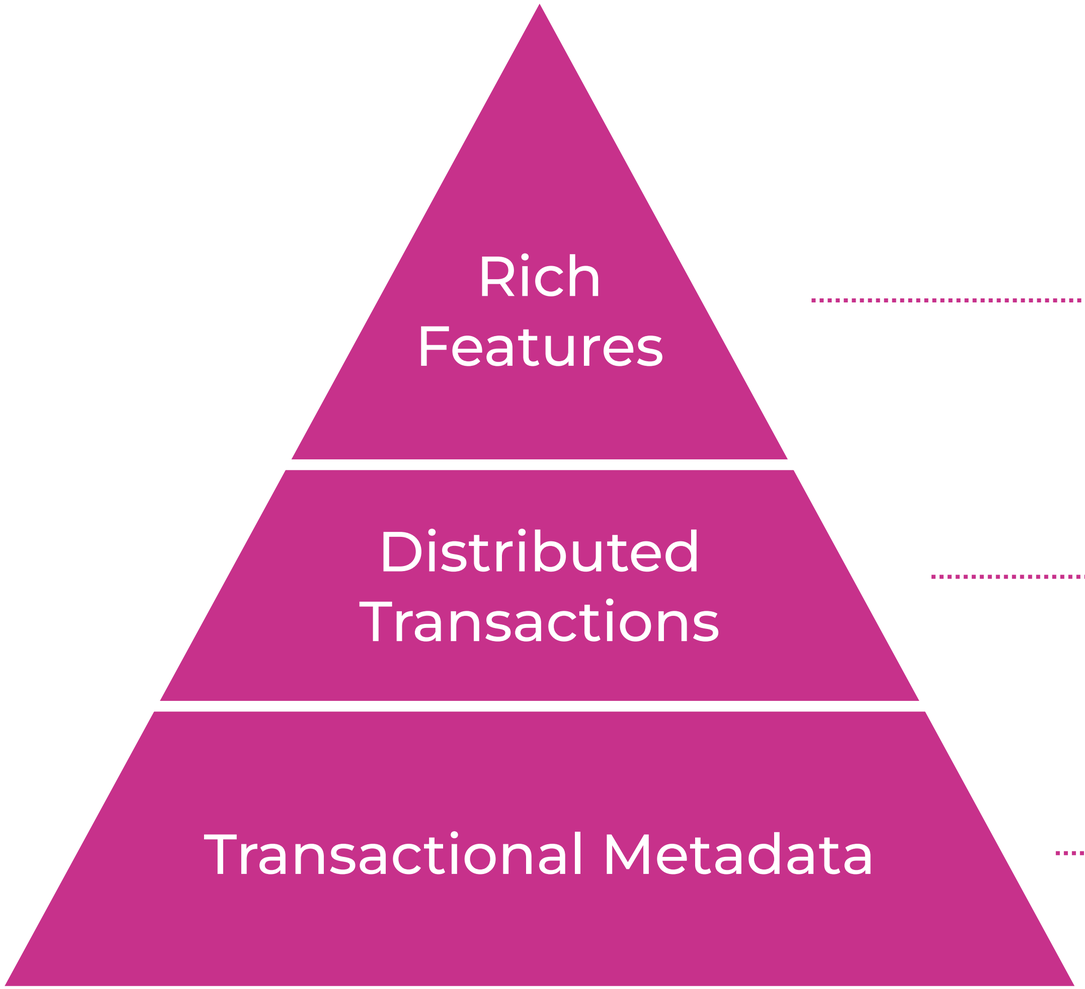
Outline

- Apache Cassandra Today
- Transactional Capabilities Coming in Apache Cassandra
- Demo Application
- Future Directions

Apache Cassandra Today

Capability	Description	Status
Programmatic DDL	Ability to safely use tools like Liquibase to programmatically manage schema changes instead of executing by hand.	✗
Safe Drop / Recreate Table	Ability to recreate tables with same name after dropping.	✗
Transactions Across Partitions / Tables	Ability to transact across different partition keys in a table.	✗
Fast Multi-Region Transactions	Transacting across regions in 1x round trip. (Minimum write transaction: 2x WAN latency; 4x for paxos_v1)	✗
Referential Integrity	Ability to enforce relationships and data integrity constraints across tables via transactional capability.	✗
Feature-Rich Secondary Indexes	Ability to define an index on a column and perform prefix queries over SSTable-attached data structures.	✗

Foundations



SAI: New high-performance index in Cassandra. Materialized views possible via transactions.

State of the art, novel Paxos protocol powering transactions across keys and tables at 1xRTT.

Serializable log of all changes to cluster config: membership, ring ownership, schema, and more.

Demo

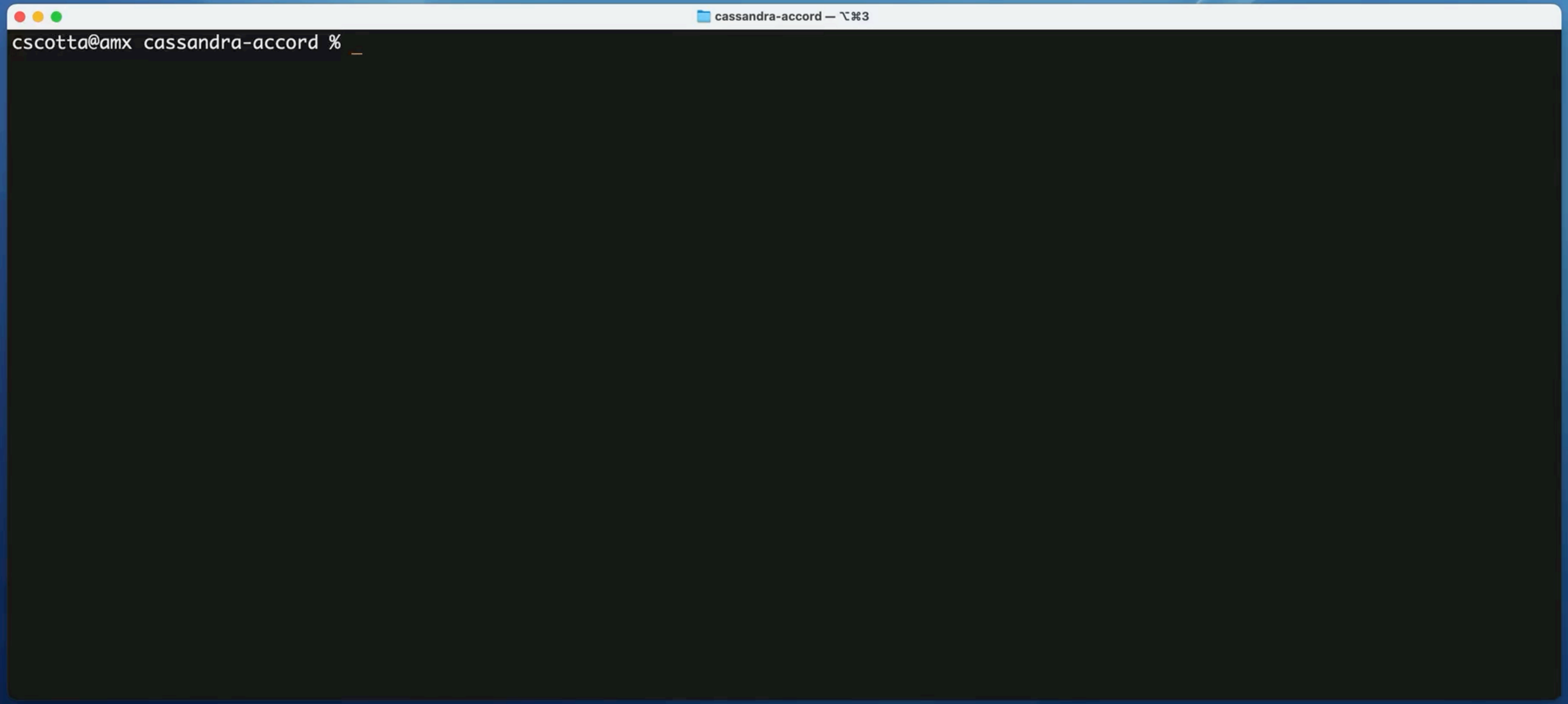
A Toy Filesystem

– Feature: Users should be able to arrange files in folders and search folders by file type.
– Constraints: All files must belong to a user and a folder. All folders must belong to a user.
– Approach: Three tables: Users, Folders, and Files.

schema.txt — Desktop

```
1
2  CREATE TABLE demo.users(user text, first_name text, last_name text,
3      PRIMARY KEY (user)) WITH transactional_mode = 'full';
4
5  CREATE TABLE demo.folders(user text, folder_name text,
6      PRIMARY KEY (user, folder_name)) WITH transactional_mode = 'full';
7
8  CREATE TABLE demo.files(user text, file_name text, folder_name text, file_type text, contents blob,
9      PRIMARY KEY ((user), file_name)) WITH transactional_mode = 'full';
10
```

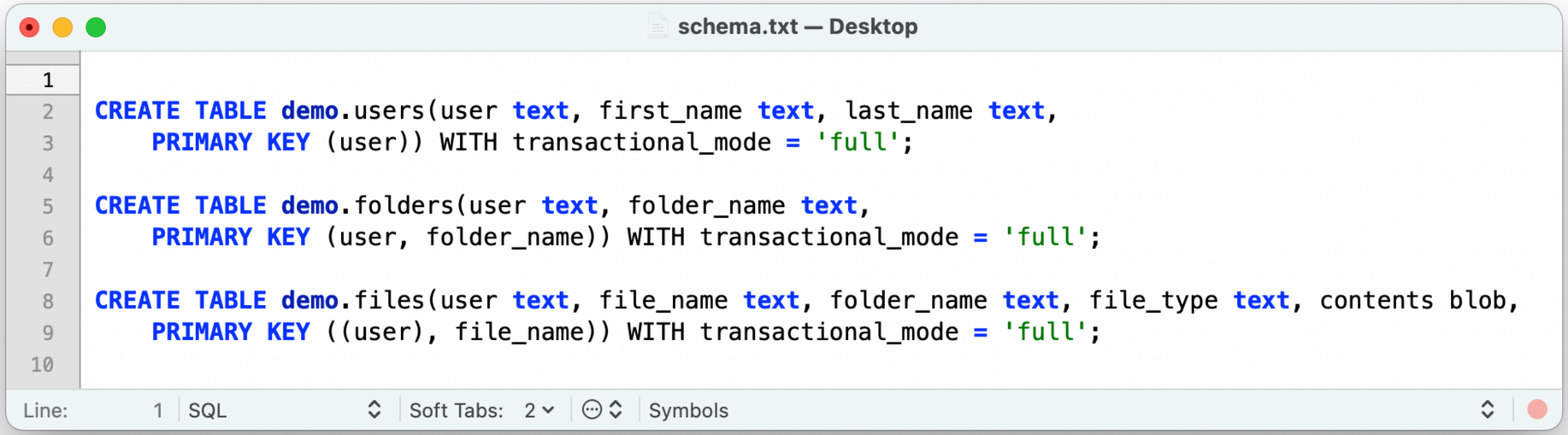
Line: 1 SQL Soft Tabs: 2 Symbols

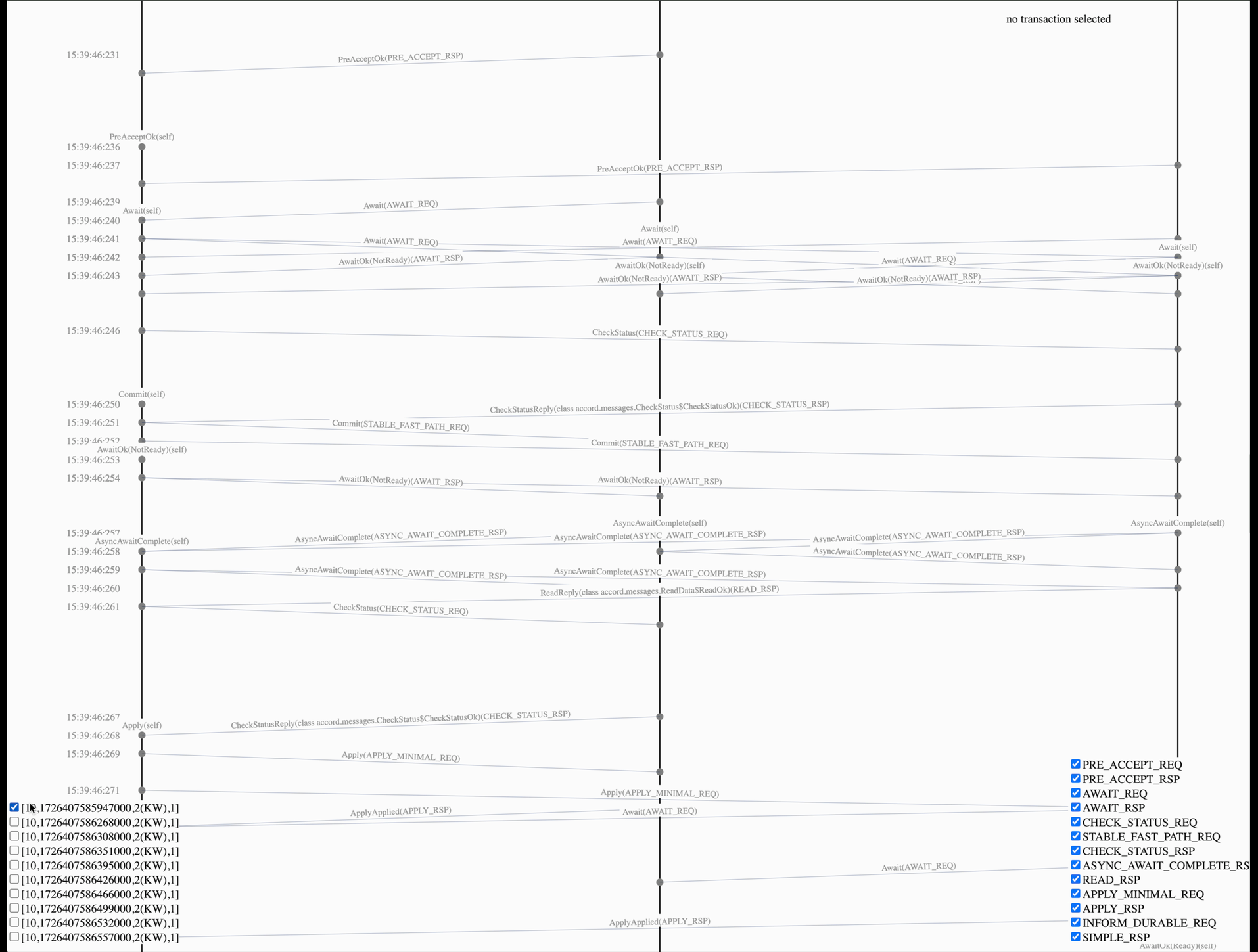

cassandra-accord — ⌥⌘3
cscotta@amx cassandra-accord %

Demo Recap

Distributed Transactions

– Multi-Table Transactions: Atomic modification of records across tables
– Strict Serializable Reads: Strongest isolation level available to tables by default.
– Referential Integrity: Enforcement of relationship of entities across tables.
– Transactional DDL: Safe, rapid modification of tables; ability to drop/recreate.




no transaction selected
15:39:46:231
PreAcceptOk(PRE_ACCEPT_RSP)
PreAcceptOk(self)
15:39:46:236
15:39:46:237
PreAcceptOk(PRE_ACCEPT_RSP)
15:39:46:239
Await(self)
Await(AWAIT_REQ)
15:39:46:240
Await(self)
15:39:46:241
Await(AWAIT_REQ)
Await(AWAIT_REQ)
Await(self)
15:39:46:242
AwaitOk(NotReady)(AWAIT_RSP)
AwaitOk(NotReady)(self)
Await(AWAIT_REQ)
AwaitOk(NotReady)(self)
15:39:46:243
AwaitOk(NotReady)(AWAIT_RSP)
AwaitOk(NotReady)(AWAIT_RSP)
15:39:46:246
CheckStatus(CHECK_STATUS_REQ)
Commit(self)
15:39:46:250
CheckStatusReply(class accord.messages.CheckStatus$CheckStatusOk)(CHECK_STATUS_RSP)
15:39:46:251
Commit(STABLE_FAST_PATH_REQ)
Commit(STABLE_FAST_PATH_REQ)
AwaitOk(NotReady)(self)
15:39:46:253
15:39:46:254
AwaitOk(NotReady)(AWAIT_RSP)
AwaitOk(NotReady)(AWAIT_RSP)
AsyncAwaitComplete(self)
AsyncAwaitComplete(self)
AsyncAwaitComplete(self)
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
15:39:46:258
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
15:39:46:259
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
AsyncAwaitComplete(ASYNC_AWAIT_COMPLETE_RSP)
15:39:46:260
ReadReply(class accord.messages.ReadData$ReadOk)(READ_RSP)
15:39:46:261
CheckStatus(CHECK_STATUS_REQ)
15:39:46:267
Apply(self)
CheckStatusReply(class accord.messages.CheckStatus$CheckStatusOk)(CHECK_STATUS_RSP)
15:39:46:268
15:39:46:269
Apply(APPLY_MINIMAL_REQ)
15:39:46:271
Apply(APPLY_MINIMAL_REQ)
ApplyApplied(APPLY_RSP)
Await(AWAIT_REQ)
Await(AWAIT_REQ)
ApplyApplied(APPLY_RSP)
[10,1726407585947000,2(KW),1]
[10,1726407586268000,2(KW),1]
[10,1726407586308000,2(KW),1]
[10,1726407586351000,2(KW),1]
[10,1726407586395000,2(KW),1]
[10,1726407586426000,2(KW),1]
[10,1726407586466000,2(KW),1]
[10,1726407586499000,2(KW),1]
[10,1726407586532000,2(KW),1]
[10,1726407586557000,2(KW),1]
PRE_ACCEPT_REQ
PRE_ACCEPT_RSP
AWAIT_REQ
AWAIT_RSP
CHECK_STATUS_REQ
STABLE_FAST_PATH_REQ
CHECK_STATUS_RSP
ASYNC_AWAIT_COMPLETE_RS
READ_RSP
APPLY_MINIMAL_REQ
APPLY_RSP
INFORM_DURABLE_REQ
SIMPLE_RSP

Referential Integrity

Semi-Relational Features in Cassandra

Referential integrity enforces relationships between records across tables.

E.g., “All files must be in a valid folder. All folders must belong to an active user.”

Distributed transactions enable enforcement of these relationships in Cassandra.

referential-integrity.sql — go-faster

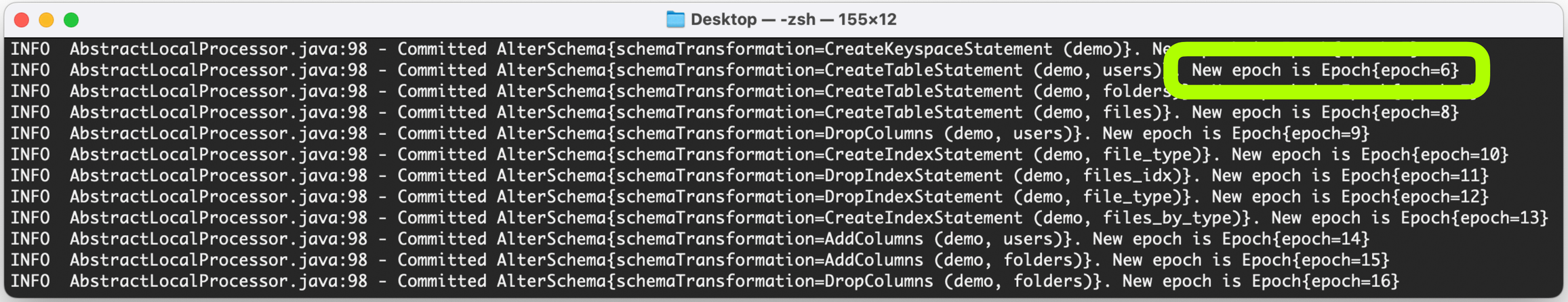
```
1
2   -- Insert some files of different types, asserting valid account and target folder.
3   BEGIN TRANSACTION
4       LET valid_user =      (SELECT user FROM demo.users WHERE user = 'demo@example.com');
5       LET existing_folder = (SELECT folder_name FROM demo.folders WHERE user = 'demo@example.com' AND folder_name='Home');
6
7       IF valid_user IS NOT NULL AND existing_folder IS NOT NULL THEN
8           INSERT INTO demo.files (user, file_name, folder_name, file_type, contents)
9               VALUES ('demo@example.com', 'Test', 'Home', 'txt', textAsBlob('Welcome...'));
10          INSERT INTO demo.files (user, file_name, folder_name, file_type, contents)
11              VALUES ('demo@example.com', 'Photo', 'Home', 'jpg', textAsBlob('...'));
12      END IF
13  COMMIT TRANSACTION;
```

Line: 7:5-7:67 | SQL | Soft Tabs: 2

Transactional Schema

Managing Cluster State in Cassandra

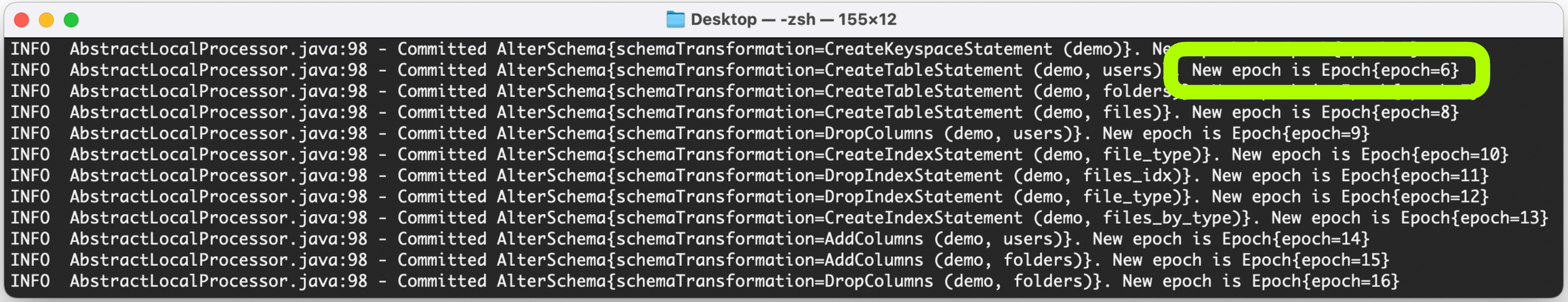
– Epochs ensure all replicas agree on cluster configuration when serving a request.
– All changes to cluster config pass through a serialized log and increment the epoch.
– Epochs enable safe and rapid changes to cluster state via Paxos.



Transactional Schema

Transactional DDL is safer DDL

– Keyspaces and Tables in Cassandra are now versioned by epoch.
– Impossible for schema conflicts to emerge within a cluster.
– Impossible for duplicate table IDs to emerge for same CREATE TABLE statement.
– Safe to drop and recreate tables with same names – C* will recognize the difference.



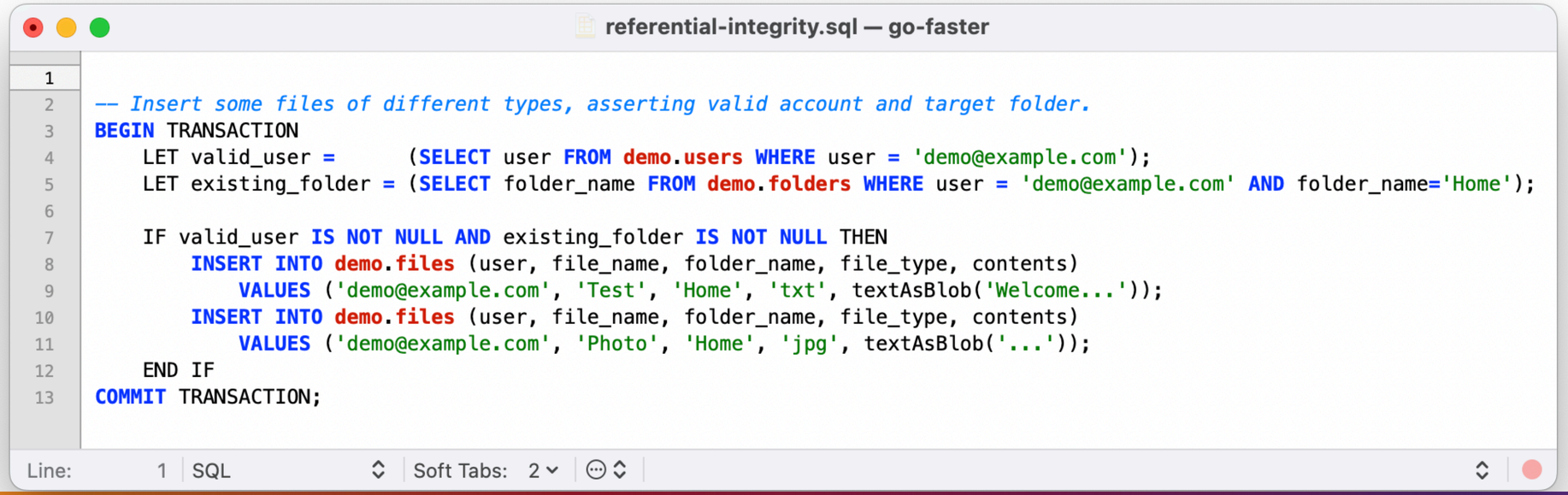
Transacting Across Tables and Partitions

Distributed Transactions

Transactions make data modeling simpler.

Think in terms of your application's data model. No complex schemes for maintaining consistency.

Transactions make building applications on Cassandra safer.



Fast, Multi-Region Transactions

	Scale	Isolation	Multi Cloud	Leaderless	Single Key Round-Trips				Multi Key Round-Trips			
					Local		Remote		Local		Remote	
					Read	Write	Read	Write	Read	Write	Read	Write
CockroachDB	Terabytes	Serializable	✓	✗	1	1	2	2	1	1	2-3	2-3
DynamoDB	Petabytes	Serializable	✗	✗	1	1	2	2	1	1	NA	NA
Spanner	Petabytes	Strict Serializable	✗	✗	0.5	1	0.5	2	0.5	1	0.5	2-3
Cassandra (2013)	Petabytes	Linearizable	✓	✓	2	4	2	4	NA	NA	NA	NA
Cassandra (2022)	Petabytes	Linearizable	✓	✓	1	2	1	2	NA	NA	NA	NA
Cassandra / Accord	Petabytes	Strict Serializable	✓	✓	1	1	1	1	1	1	1	1

Secondary Indexes

Storage-Attached indexes (SAI)

Work best as a partition-restricted index.

Ensures that your queries contact only a single replica set and don’t scatter-gather.

Efficient storage mechanism.

Postings-list design more efficient than any other C* secondary index mechanism.

Feature-rich

AND/OR logic, IN logic, numeric ranges, collections CONTAINS, optional case-sensitivity.

Anticipated in next iteration

Prefix queries (LIKE) and OR queries. Major enhancement to Cassandra UX.

Apache Cassandra Today

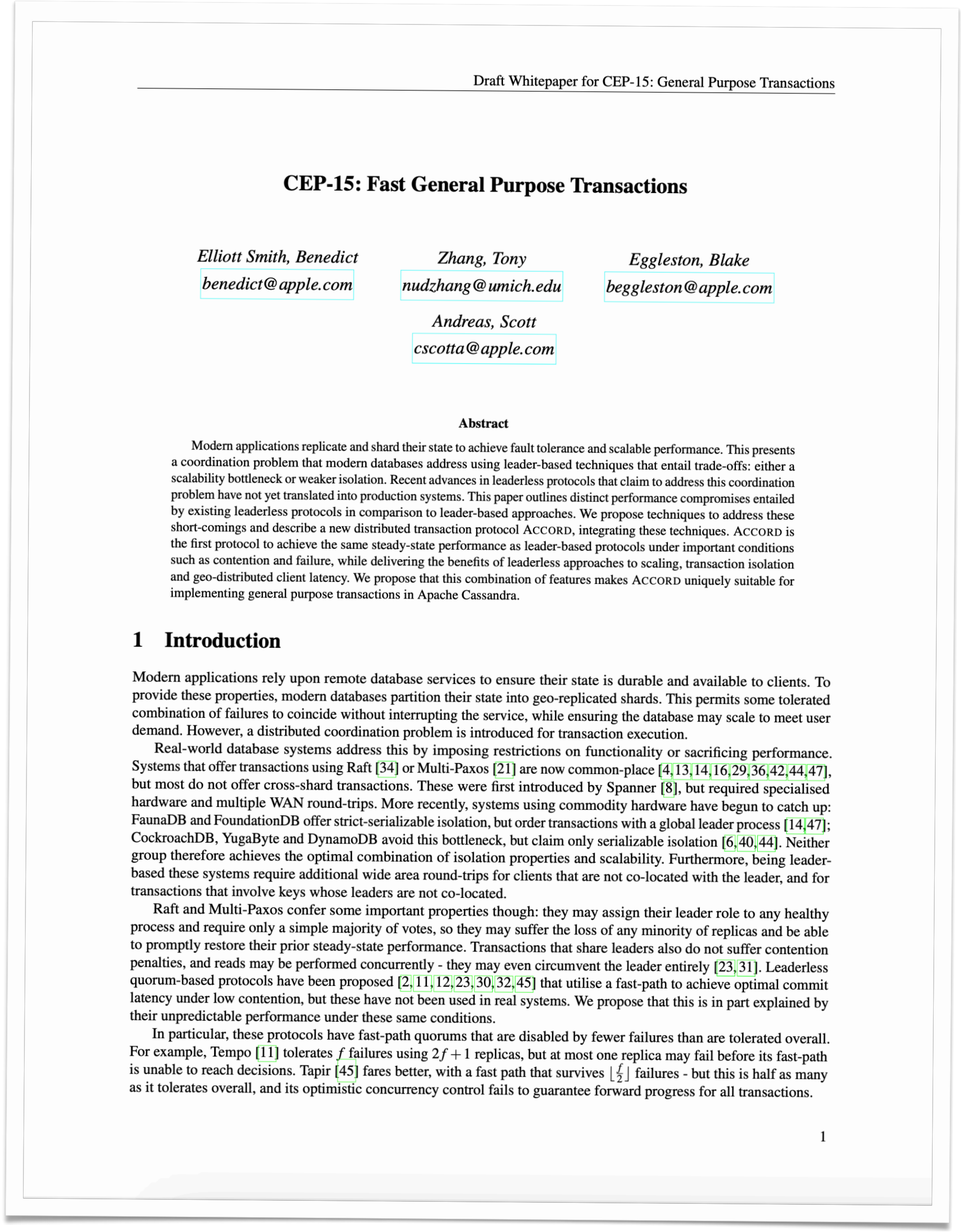
Capability	Description	Status
Programmatic DDL	Ability to safely use tools like Liquibase to programmatically manage schema changes instead of executing by hand.	✗
Safe Drop / Recreate Table	Ability to recreate tables with same name after dropping.	✗
Transactions Across Partitions / Tables	Ability to transact across different partition keys in a table.	✗
Fast Multi-Region Transactions	Transacting across regions in 1x round trip. (Minimum write transaction: 2x WAN latency; 4x for paxos_v1)	✗
Referential Integrity	Ability to enforce relationships and data integrity constraints across tables via transactional capability.	✗
Feature-Rich Secondary Indexes	Ability to define an index on a column and perform prefix queries over SSTable-attached data structures.	✗

Apache Cassandra 5.1+

Capability	Description	Status
Programmatic DDL	Ability to safely use tools like Liquibase to programmatically manage schema changes instead of executing by hand.	✅
Safe Drop / Recreate Table	Ability to recreate tables with same name after dropping.	✅
Transactions Across Partitions / Tables	Ability to transact across different partition keys in a table.	✅
Fast Multi-Region Transactions	Transacting across regions in 1x round trip. (Minimum write transaction: 2x WAN latency; 4x for paxos_v1)	✅
Referential Integrity	Ability to enforce relationships and data integrity constraints across tables via transactional capability.	✅
Feature-Rich Secondary Indexes	Ability to define an index on a column and perform prefix queries over SSTable-attached data structures.	✅

Behind the Scenes

- **Accord:** Paxos-based distributed transaction protocol
- Leaderless transactions can be initiated from any region.
- Transactions execute in 1x round-trip in common case (3x fallback).
- Validated via formal proof, research collaboration, and simulation.



CEP-15: Fast General Purpose Transactions

Elliott Smith, Benedict
benedict@apple.com

Zhang, Tony
nudzhang@umich.edu

Eggleston, Blake
beggleston@apple.com

Andreas, Scott
cscotta@apple.com

Abstract

Modern applications replicate and shard their state to achieve fault tolerance and scalable performance. This presents a coordination problem that modern databases address using leader-based techniques that entail trade-offs: either a scalability bottleneck or weaker isolation. Recent advances in leaderless protocols that claim to address this coordination problem have not yet translated into production systems. This paper outlines distinct performance compromises entailed by existing leaderless protocols in comparison to leader-based approaches. We propose techniques to address these short-comings and describe a new distributed transaction protocol ACCORD, integrating these techniques. ACCORD is the first protocol to achieve the same steady-state performance as leader-based protocols under important conditions such as contention and failure, while delivering the benefits of leaderless approaches to scaling, transaction isolation and geo-distributed client latency. We propose that this combination of features makes ACCORD uniquely suitable for implementing general purpose transactions in Apache Cassandra.

1 Introduction

Modern applications rely upon remote database services to ensure their state is durable and available to clients. To provide these properties, modern databases partition their state into geo-replicated shards. This permits some tolerated combination of failures to coincide without interrupting the service, while ensuring the database may scale to meet user demand. However, a distributed coordination problem is introduced for transaction execution.

Real-world database systems address this by imposing restrictions on functionality or sacrificing performance. Systems that offer transactions using Raft [34] or Multi-Paxos [21] are now common-place [4, 13, 14, 16, 29, 36, 42, 44, 47], but most do not offer cross-shard transactions. These were first introduced by Spanner [8], but required specialised hardware and multiple WAN round-trips. More recently, systems using commodity hardware have begun to catch up: FaunaDB and FoundationDB offer strict-serializable isolation, but order transactions with a global leader process [14, 47]; CockroachDB, YugaByte and DynamoDB avoid this bottleneck, but claim only serializable isolation [6, 40, 44]. Neither group therefore achieves the optimal combination of isolation properties and scalability. Furthermore, being leader-based these systems require additional wide area round-trips for clients that are not co-located with the leader, and for transactions that involve keys whose leaders are not co-located.

Raft and Multi-Paxos confer some important properties though: they may assign their leader role to any healthy process and require only a simple majority of votes, so they may suffer the loss of any minority of replicas and be able to promptly restore their prior steady-state performance. Transactions that share leaders also do not suffer contention penalties, and reads may be performed concurrently - they may even circumvent the leader entirely [23, 31]. Leaderless quorum-based protocols have been proposed [2, 11, 12, 23, 30, 32, 45] that utilise a fast-path to achieve optimal commit latency under low contention, but these have not been used in real systems. We propose that this is in part explained by their unpredictable performance under these same conditions.

In particular, these protocols have fast-path quorums that are disabled by fewer failures than are tolerated overall. For example, Tempo [11] tolerates f failures using $2f + 1$ replicas, but at most one replica may fail before its fast-path is unable to reach decisions. Tapir [45] fares better, with a fast path that survives $\lfloor \frac{f}{2} \rfloor$ failures - but this is half as many as it tolerates overall, and its optimistic concurrency control fails to guarantee forward progress for all transactions.

Transactional Tables

```
CREATE TABLE demo.tbl(col text PRIMARY KEY (col))
WITH TRANSACTIONAL_MODE = ‘xxx’;
```

Mode	Behavior	Vibe
“off”	Distributed transactions via Accord disabled (paxos_v1 and paxos_v2 supported).	
“unsafe"	Permit writes via standard StorageProxy write path. Can result in multiple outcomes computed for transactions depending on data written via non-SERIAL writes.	
“unsafe_writes”	Allows non-serial writes, but still forces blocking read repair via Accord. Safe to perform non-serial reads of Accord data, but unsafe to write data Accord may read.	
“mixed_reads”	Executes writes via Accord. Commits at provided consistency level to enable data to be read via non-serial reads. Safe to read/write data Accord will write.	
“full"	Full serializable semantics for all queries. Consistency levels do not apply.	

Transaction Syntax

Overview

```
Initializes a transaction block          ---    BEGIN TRANSACTION
Binds results of a query to a variable   ---        LET existing_user = (SELECT user FROM demo.users WHERE u

Defines return value (pre-execution)     ---        SELECT user FROM demo.users WHERE user = ‘demo@example.c

Predicate that tests whether to apply    ---        IF existing_user IS NULL THEN
Atomic batch of mutations across tables. ---            INSERT INTO demo.users (user, first_name, last_name)
                                                            VALUES (‘demo@example.com', 'Scott', 'Andreas');
                                                        INSERT INTO demo.folders (user, folder_name)
                                                            VALUES ('demo@example.com', 'Home');
                                                        INSERT INTO demo.files (user, file_name, folder_name
                                                            VALUES (‘demo@example.com', 'README', 'Home', 't

Conclude predicate.                      ---        END IF
Concludes a transaction block.           ---    COMMIT TRANSACTION;
```

Composable with Features

Transactions and Secondary Indexes (SAI)

Transactions and Secondary Indexes are composable with Accord.

– In transactional_mode='full', all reads and writes pass through the transactional subsystem.

ACID transactional guarantees apply to secondary indexes.

– On write path, transactions mutate index and guarantee atomic visibility to transactional reads.

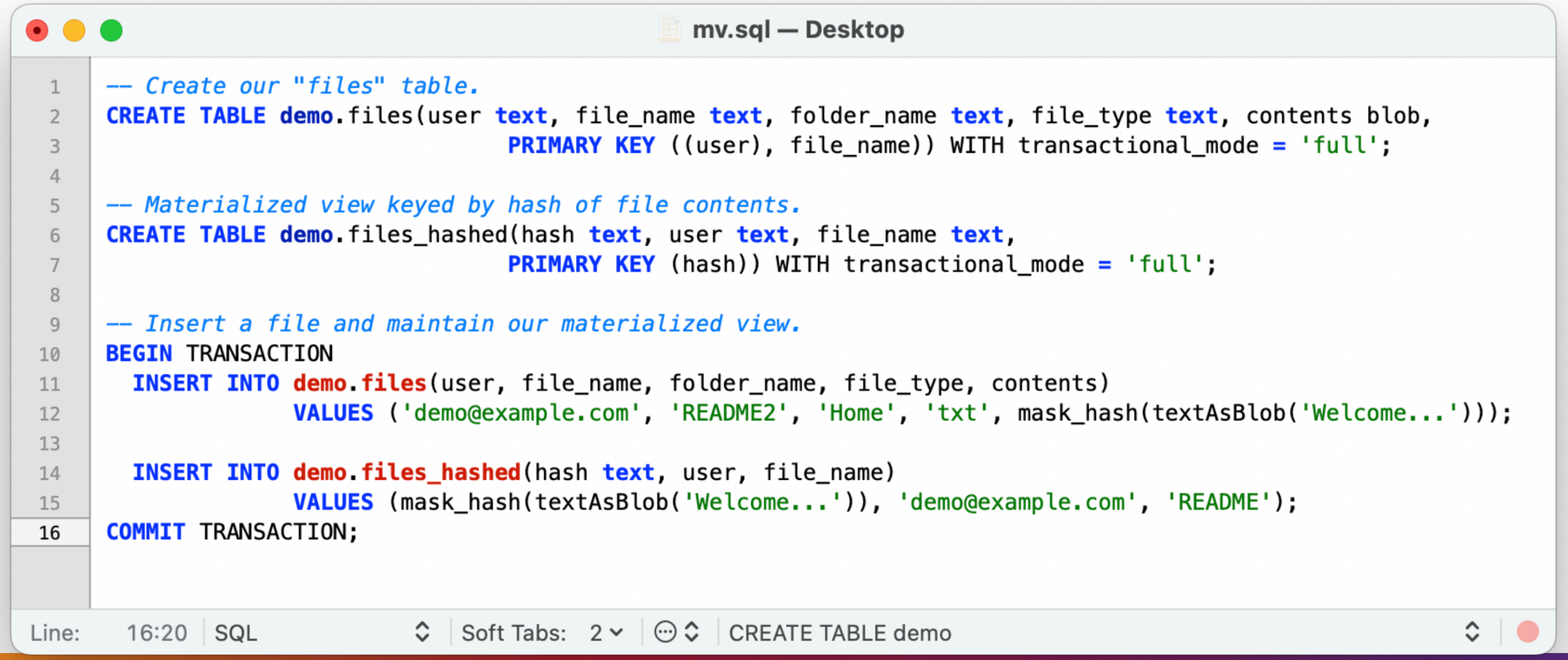
– On read path, transaction protocol ensures execution happens-after all transactions with conflicting dependencies have committed.

Composable with Features

Materialized Views

Distributed Transactions enable query-level construction of materialized views.

Materialized views can be maintained via transactional inserts on the write path.



Future Directions

Where to from here?

Improving ergonomics of whether a transaction was applied.

It’s inconvenient to need to re-select the predicate you’re testing.

Multi-result select statements.

It would be useful to return an array of resultsets from selects in a transaction.

Strict-Serializable Snapshots

Accord may enable strict-serializable snapshots via an exclusive sync point.

Snapshot Isolation

Adding record versioning may enable Cassandra to support proper MVCC.

Foreign Key Constraints

Bringing referential integrity constraints into database schema natively.

Development Status

What's Ahead?

GitHub Branch: cep-15-accord

https://github.com/apache/cassandra/tree/cep-15-accord

Journal: Startup / Replay Complete

Write-ahead log for Accord transactions providing durability across process restarts.

Testing + Validation of Implementation

Advancing from burn tests to full-database simulation.

Performance

Baseline target: "As inexpensive as paxos_v2 to execute, with half the round trips."

Merging to Trunk

Anticipate merging to trunk in 1 - 2 months. Request for review + involvement on mailing list.

COMMUNITY
THE ASF CONFERENCE
CODE

BEGIN TRANSACTION

Apache Cassandra as a Transactional Database

C. Scott Andreas, Apple Inc.