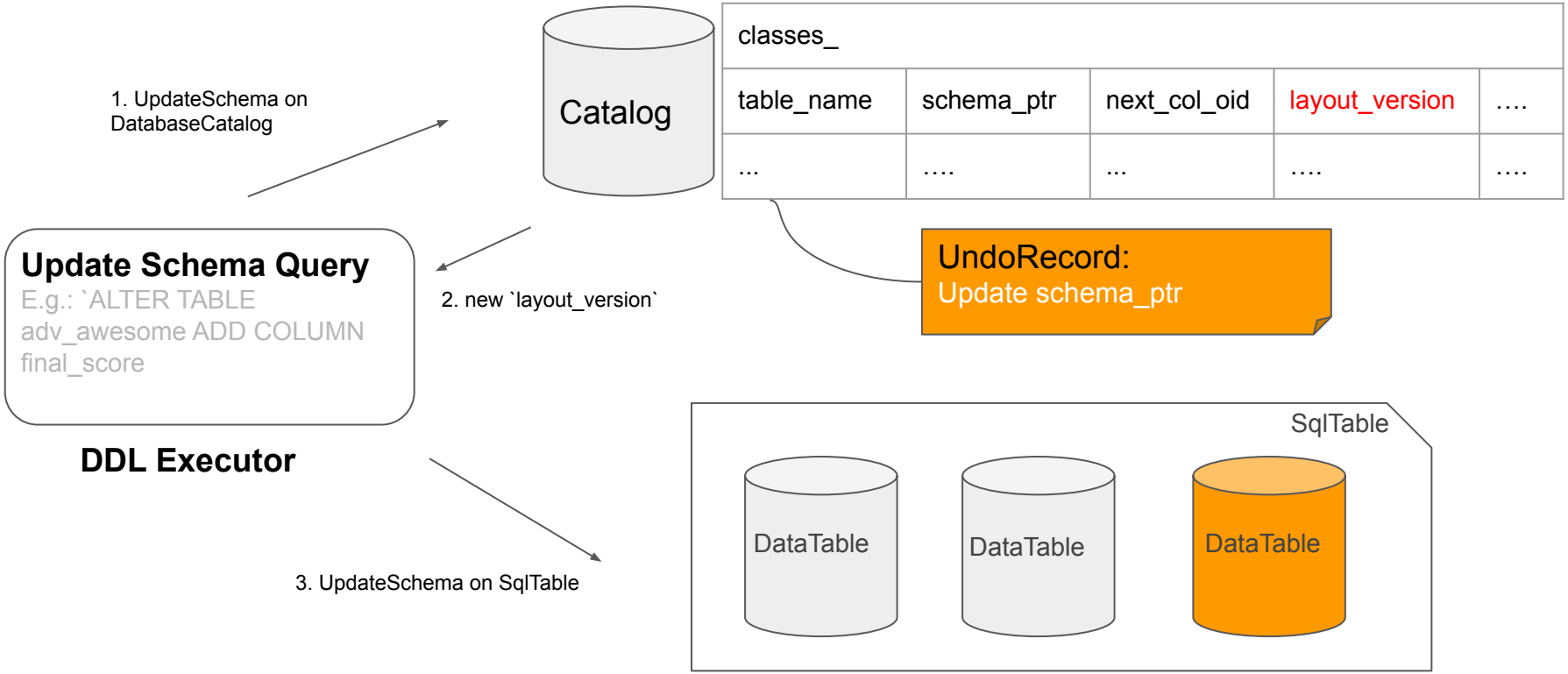
Concurrent Schema Change

Group X:
Ling Zhang
Sheng Xu
Ricky Xu

Solution Overview



Development Goals Review

75% : **support add/drop column of multi-version schemas (can involve multiple columns)**

- Implemented the pipeline from parser to execution (DEMO)

100%: **lazy migration of data to the new schemas**

- We are not doing background migration for now.

125% : **support (almost) arbitrary number of schema versions** / **support updating constraints**

- We adopted an efficient vector approach and set a maximal number of schemas a SqlTable could have
- Could substitute the current bounded data structure with a concurrent unbounded data structure

150% : **compile the transformation from one schema to another**

DEMO!

ALTER TABLE with add and drop column

(change column type in progress…)

Testing Effort

- **Correctness tests:**
  - binder_cpp.test // test binding of the AlterTable nodes
  - ddl_executors_test.cpp // test the AlterTable DDL executors
  - logical_operator_test.cpp // test the AlterTable logical operators
  - operator_transformer_test.cpp // test transforming logical operator to plan node
  - parser_test.cpp // test parsing correctness
  - sql_table_test.cpp // test correctness of different sqltable operations
    - coupled with schema version updates
  - traffic_cop_test.cpp // added Altertable ADD COLUMN

- **Concurrency tests:**
  - sql_table_test.cpp // Concurrent operations with schema change
  - ddl_executors_test.cpp // TODO

TPCC benchmark with/without versions (4 threads)

TPCC (averaged over 5 runs on AWS)	With versions	Without versions
WithoutLogging	3462 ms	3410 ms
WithLogging	3607 ms	3571 ms
WithLoggingAndMetrics	3604 ms	3544 ms
WithMetrics	3815 ms	3766 ms

We plan on adding another benchmark this week that tests throughput of sqltable operations when there are concurrent schema changes

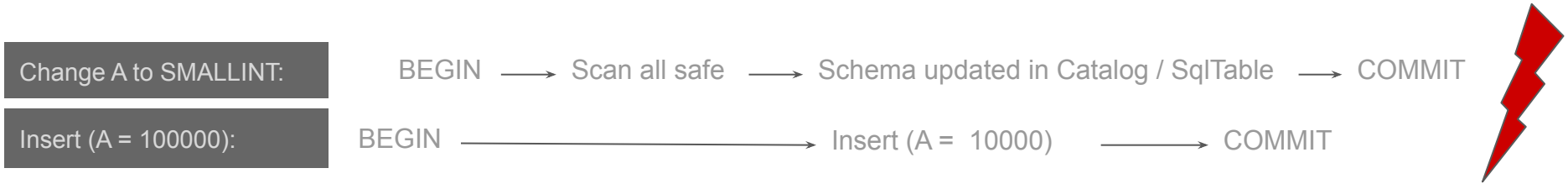
Issues

1. Memory management of pointers
    a. Problem Description:
        i. New schema pointers stored in the UndoRecord when updating
        ii. Memory leak as the UndoRecord in the version chain gets truncated during GC
    b. Possible Solution:
        i. Explicitly “SelectAllVersions” to delete the pointers when tearing down the catalog

            But GC would have truncated the version chains, and the UndoBuffer that stores the UndoRecords is currently owned by TransactionContext, which will be destructed at Commit/Abort → UndoRecord in the version chain in storage layer no longer valid
        ii. Some sort of DeferredAction which allows UndoRecord’ destructor to know how to delete the pointers

Issues

2. Unsafe schema change that requires scanning all tuples and block in-flight txns



Solution:

1. Block concurrent transactions when unsafe schema change?

2. Concurrent transactions Insert/Update to shadow place?

Future Tasks

Before Final Code Drop:

- Fix the memory leak
- Some other safe schema change commands (change default, drop default)
- Concurrent schema update throughput benchmark:
  - Have X threads that select/insert/update/scan Sqltable using the previous layout version
  - Have a schema update thread that concurrently updates the schema version
  - Repeat the above scenario multiple times
- Benchmark on TPCC with 32+ threads

After Code Drop:

- Unsafe schema change commands