Week 12: Transaction Management in SQLite DB Engine

Siyeol Choi
2019315530

1. INTRODUCTION

In this lab experiment, we aim to experiment with SQLite and its Transaction management. We would conduct A) performance evaluation comparison between system transaction and user transaction. Plus, we would B) force the deadlock and resolve it. Then we would C) evaluate another performance of journal mode.

2. METHODS

For A, we would run test query system_trx.sql and user_trx.sql. B would be conducted by opening dual terminals and run multiple queries that would incur deadlock. Then we would be resolving the deadlock by using ROLLBACK command. C would be conducted by evaluating time method for sync 1/10/20 within delete/persist/truncate commands.

Performance Evaluation

3.1 Experimental Setup

[System Setup]

OS Ubuntu 20.04.1 LTS (GNU/Linux 4.19.104-microsoft-standard

cpu cores 4

Vendor: Msft

Product: Virtual Disk

Revision: 1.0

Compliance: SPC-3

User Capacity: 274,877,906,944 bytes [274 GB]

Logical block size: 512 bytes

Physical block size: 4096 bytes

Type Configuration

3.2 Experimental Results

A) Performance evaluation: system transaction vs. user transaction

system_trx.sql : 0m11.764s

user_trx.sql : 0m0.745s

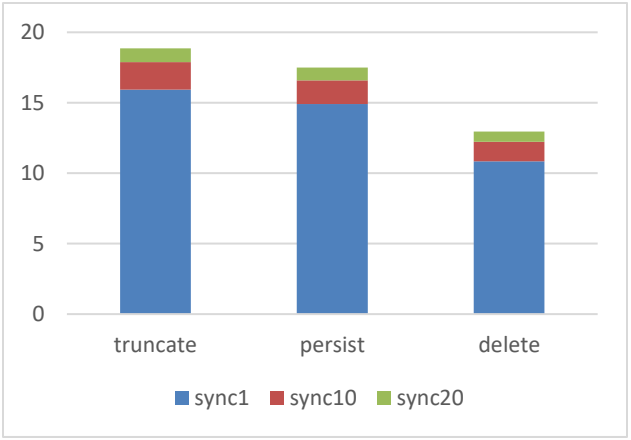
B) Understanding deadlock in SQLite

```
siyeol13@DESKTOP-D7ME2GD:~/SWE3033-F2021/TEST$ sqlite3 deadlock.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> BEGIN;
sqlite> SELECT * FROM TEST;
1
2
sqlite> INSERT INTO TEST VALUES (4);
Error: database is locked
sqlite> ROLLBACK;
sqlite> SELECT * FROM TEST;
1
2
3
sqlite> INSERT INTO TEST VALUES (4);
```

```
siyeol13@DESKTOP-D7ME2GD: ~/SWE3033-F2021/TEST
siyeol13@DESKTOP-D7ME2GD:~/SWE3033-F2021/TEST$ sqlite3 deadlock.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite> BEGIN;
sqlite> INSERT INTO TEST VALUES(3);
sqlite> COMMIT;
Error: database is locked
sqlite> COMMIT;
sqlite> SELECT * FROM TEST;
1
2
3
sqlite> SELECT * FROM TEST;
1
2
3
4
```

C) Performance evaluation: journal mode

	sync1	sync10	sync20
truncate	0m15.935s	0m1.939s	0m0.988s
persist	0m14.911s	0m1.675s	0m0.923s
delete	0m10.852s	0m1.381s	0m0.724s



3.3 Result Analysis

A) Performance evaluation: system transaction vs. user transaction

System Transaction is an autocommit mode and it is expensive since it has journaling overhead and concurrency control overhead.

To resolve such issue, user transaction disabled autocommit mode and SQLite revert to the autocommit mode on completion of the write-transaction.

B) Understanding deadlock in SQLite

Since auto commit is disabled, two insertion queries are in deadlock status without committing manually sequentially. Therefore, we had to revert the latest insertion query with "ROLLBACK;" and then conduct the insertion again. As a result, deadlock is resolved.

C) Performance evaluation: journal mode

Time spent among different journal modes decreases as the number of SQL statements in transaction increases. Among each journal modes, truncate had the longest time spent followed by persist, then delete.

This result is totally consistent with sqlite's document. Explaining why each of the journals are optimized for certain purposes with proposed methods.

DELETE journaling mode is the normal behavior where rollback journal is deleted at the conclusion of each transaction. Indeed, the delete operation causes the transaction to commit

PERSIST journaling mode prevents the rollback journal from being deleted at the end of each transaction. In return, the header of the journal is overwritten with zeros. This will prevent other database connections from rolling the journal back.

TRUNCATE journaling mode commits transactions by truncating the rollback journal to zero-length instead of deleting it. Therefore, truncating a file is much faster than deleting the file since the containing directory does not need to be changed

4. Conclusion

With benchmarking SQLite, we could find out that User Transaction performs better since it disabled autocommit. Also, deadlock can be resolved by rollingback and committing manually sequentially. Lastly, each of the journaling modes are optimized for its own purposes.

5. REFERENCES

[1] SQLite Transaction management
https://github.com/meeeejin/SWE3033-F2021/tree/main/week-12

[2] SQLite Journal Mode
https://www.sqlite.org/pragma.html#pragma_journal_mode