Concurrency Control in Database Management System

Thi Ngoc Thanh Nguyen

Abstract — Nowadays, most application software systems are aimed at multi-user environments and simultaneous access to the same database. Therefore, concurrency controllers are an integral part of any database management system. The concurrency controller plays an important role in controlling transactions which perform simultaneous access to the database without conflict. This paper presents an overview and illustration how to use concurrency control in database management system of Microsoft SQL Server.

Key words — Concurrency Control, Database Management System, Transaction, Conflict Database.

I. INTRODUCTION

Concurrency control in DBMSs is not a new problem. However, in the current era of strong development of e-commerce services, concurrency control in a multi-user environment is even more important to ensure that transactions are performed and ensured data integrity and consistency [1]-[4].

This paper presents the concurrency problems, some of the basic concurrency control techniques, concurrency control in database management system and some of concurrency control solutions.

The remainder of the paper is organized as follows. Section II presents several concurrency problems. In section III, concurrency control techniques in database management system of Microsoft SQL SERVER. Section IV illustrates concurrency problems and solutions. Finally, Section V concludes this paper and figures out the future works.

II. SEVERAL CONCURRENCY PROBLEMS

Some problems can occur when concurrent transactions execute to access and update the same data items. There are 4 problems such as the lost update problem, the temporary update (or dirty read) problem, the unrepeatable read problem and phantom problem [5].

A. The lost Update Problem

This problem occurs when two transactions access the same data item at the same time [4]. In the problem, update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

B. The Temporary Update (or Dirty Read) Problem

This problem occurs when the transaction T executes to update the data item and then the transaction T fails for some reason such as system crash, transaction error, system error, local errors or exception conditions detected by the transaction, disk failure, physical problem etc… Meanwhile, the updated data item is accessed (read) by another transaction T’ before the transaction T is rolled back to its original value. Therefore, the transition T’ access the temporary updated value.

C. The Phantom Problem

The phantom problem occurs when a transaction is reading the data set for statistical purposes, while another transaction is adding the data items into the data table at the same time.

Assume one transaction is calculating a sum function on a number of data items, while other transactions is updating some of these data items. The sum function may wrong because some of data items may updated by other transactions. That for, it led to loss of consistency.

D. The Unrepeatable Read Problem

This problem occurs when two or more read operations of the same transaction read different values of the same variable, even though there is no data update action in the interval between two reads of transaction. The reason for this problem is another transaction updated this data item.

III. CONCURRENCY CONTROL TECHNIQUES IN DATABASE MANAGEMENT SYSTEM OF MICROSOFT SQL SERVER

Nowadays, there are many concurrency control techniques that are used to ensure non interference or isolation property of concurrent executing transactions.

Most of these techniques ensure serializability of transaction schedules such as locking protocols [6], [7], timestamps [6], [7], validation [6], [7]. The purpose of these techniques avoids these concurrency problems when many transactions execute to access the same data items at the same time.

The database management system of Microsoft SQL Server was integrated locking techniques [6], [7] to concurrency control. That help programmers avoid these concurrency problems.

There are different locking protocols such as locking-based protocols (binary locks), read/write locks (shared/exclusive locks), two-phase locking protocols (2PL), strict two-phase locking, etc [6], [7]. Each protocol has advantages and disadvantages.

In practice, the database management system of Microsoft SQL Server often use Read/Write lock and two-phase Read/Write locks for concurrency control.
Each lock has different isolation level. There are four isolation levels. That are READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ and SERIALIZABLE. These isolation levels are used to control transactions [5], [7]. Depending on the problem when the transactions execute concurrently, choose the appropriate isolation level.

A. Serializable Isolation Level

SERIALIZABLE isolation level is the highest isolation level. When transaction T is set to this isolation level, transaction T can avoid most data collisions from other transactions.

This isolation level ensures that transaction T reads only the data items of accepted transactions, no data items being read or written by transaction T can be changed by other transactions until transaction T completes. This isolation level helps transaction T can avoid the phantom problem.

The locking usage of a transaction with SERIALIZABLE isolation level occurs as follows: the transaction is granted locks prior to reading or writing the data items, and locks granted on the data items are not allowed to be released until the transaction completes (two-phase locking protocol [6], [7]).

Thus, the SERIALIZABLE isolation level is a strict two-phase read and write lock protocol [6], [7]. However, it should be noted that using the SERIALIZABLE isolation level can easily cause deadlock problems [4].

B. Repeatable Read Isolation Level

The REPEATABLE READ isolation level is the second isolation level. When transaction T is set to this isolation level, it is guaranteed that transaction T only reads the data items changed by accepted transactions and no the data items being read or written by transaction T may be changed by other transactions until transaction T completes.

However, transaction T may occur the phantom problem. Because while transaction T is executing on the data items that satisfy a certain condition, there may still be another transaction adding a new record to the data, so transaction T can be error by this new record.

The lock usage of a transaction with the REPEATABLE READ isolation level is the same as that of a transaction with the SERIALIZABLE isolation level, except it does not use the full-table key index table. The REPEATABLE READ isolation level locks only the data items, not sets of data items.

C. Read Committed Isolation Level

The READ COMMITTED isolation level is the third isolation level. This isolation level ensures that transaction T only reads the data items changed by accepted transactions, and not the data items written by transaction T can be changed by other transactions until transaction T completes.

Thus, in this isolation level, the data items being read by transaction T can still be modified by another transaction while transaction T is still in execution, so transaction T can occur the lost update problem and the phantom problem.

The lock usage of a transaction with the READ COMMITTED isolation level is as follows: the transaction receives exclusive locks before writing the data items and holds these locks until the transaction terminates. The transaction also receives shared locks before reading the data items, but these shared locks are released immediately.

When transactions establish this isolation level, it is always guaranteed that the transaction does not encounter the dirty read problem. This is guaranteed because all transactions are granted an exclusive lock before writing the data items and hold the exclusive lock until the transaction terminates. For the lost update problem, it will be necessary to combine with an update lock (UpdateLock) [6], [7].

By default, transactions in Microsoft SQL Server are set up with the READ COMMITTED isolation level.

D. Read Uncommitted Isolation Level

The READ UNCOMMITTED isolation level is the lowest isolation level. With this isolation level, transaction T can read the data items changed by an ongoing transaction; So, the data items can continue to be changed while the transaction T is in progress, and the transaction T is also susceptible to the phantom problem.

A transaction with the READ UNCOMMITTED isolation level must not receive a shared lock before reading the data items. Transactions with this isolation level can occur the lost update problem, the dirty read problem, the unrepeatable read problem, and the phantom problem.

IV. CONCURRENCY PROBLEMS AND SOLUTIONS

A. The Lost Update Problem

Assume T1 and T2 are two deposit transactions:

T1: Deposit to account ‘ACC01’, deposit amount of 1,000.
T2: Deposit to account ‘ACC01’, deposit amount of 2,500.

Before performing two transactions T1 and T2, the account balance of ‘ACC01’ is 10,000.

| Time | T1 (Deposit) | T2 (Deposit) |
|------|-------------|-------------|
| t1   | Read (Account ‘ACC01’) | Read (Account ‘ACC01’) |
| t2   | Read (Account balance of ‘ACC01’) | Read (Account ‘ACC01’) |
| t3   | Read (Account ‘ACC01’) | Read (Account balance of ‘ACC01’) |
| t4   | Read (Account balance of ‘ACC01’) | Write (Account balance of ‘ACC01’) |
| t5   | Write (Account balance of ‘ACC01’) | Write (Account balance of ‘ACC01’) |
| t6   | Write (Transaction) | Write (Transaction) |
| t7   | Write(Transaction) | Write(Transaction) |
| t8   | Write(Transaction) | Write(Transaction) |

At time t6, transaction T2 overwrites the data written by transaction T1 at time t5. Therefore, the execution result is not correct (Account Balance=12,500), the correct execution result should be 13,500 (Account Balance = 13,500).

To solve the lost update problem, we set the transaction with READ COMMITTED isolation level and the Update Lock. Specifically, add the following codes into transactions of Microsoft SQL Server.

```sql
set transaction isolation level repeatable read
update Table_of_Account
set Account Balance = Account Balance + @Amount with (uplock)
where IDACCOUNT = ‘ACC01’
```
B. The Unreliable Read Problem

Assume T1 and T2 are Deposit transaction and Withdraw transaction respectively.

T1: Deposit to account ‘ACC01’, deposit amount of 1,000.
T2: Withdraw from account ‘ACC01’, withdrawal amount of 3,500.

Before performing two transactions T1 and T2, the account balance of ‘ACC01’ is 10,000.

At time t4, transaction T2 reads the account balance of 10,000 and at time t7 reads the account balance of 11,000. Thus, the account balance of two readings in transaction T2 are not the same, even though there is no writing action of other transactions between t4 and t7. This is called the unrepeatability read error.

To solve the unrepeatability read problem, we set transaction T2 with REPEATABLE READ isolation level.

C. The Phantom Problem

Assume T1 and T2 are Deposit transaction and Reading Account Balance transaction respectively.

T1: Deposit to account ‘ACC01’, deposit amount of 1,000.
T2: Reading Account Balance of account ‘ACC01’.

Suppose, before executing two transactions T1 and T2, account balance of account ‘ACC01’ is 10,000, the total of deposit amount is 20,000 and the total of withdrawal amount is 10,000.

At time t6, transaction T2 reads the deposit amount of 20,000, and T2 reads the total of withdrawal amount of 10,000 at time t8. Transaction T1 adds 1,000 into the account ‘ACC01’ at time t7, then the account balance is 11,000. However, the deposit amount - The total of withdrawal amount = 10,000 is not equal to the account balance of 11,000. This is phantom error.

To solve the phantom problem, we set transaction T2 with SERIALIZABLE isolation level.

D. The Temporary Update (or Dirty Read) Problem

Assume T1 and T2 are Deposit transaction and Withdraw transaction respectively.

T1: Deposit to account ‘ACC01’, deposit amount of 2,000.
T2: Withdraw from account ‘ACC01’, withdrawal amount of 21,500.

Before performing two transactions T1 and T2, the account balance of ‘ACC01’ is 20,000.

At time t6, transaction T2 reads the account balance of 22,000, and reads the account balance of 22,000 at time t8. Thus, the two readings in transaction T2 are not the same, even though there is no writing action of other transactions between t4 and t7. The cause is transaction T1 adds 2,000 into the account ‘ACC01’ at time t7. This is the unrepeatability read error.

Moreover, transaction T1 is error at time t11 and rollback, so the account balance update was undone. Thus, transaction T2 cannot be withdrawn at time t9 because the account balance of 20,000, but T2 still withdraws 21,500. This problem is very Hazardous. This is the temporary update (or dirty read) error.

To solve the temporary update (or dirty read) problem, we just need to set transaction T2 with READ COMMITTED isolation level. However, since T2 causes both the temporary update (or dirty read) problem and the unrepeatability read problem, we need to set up transaction T2 with the REPEATABLE READ isolation level to be able to solve both problems.

### TABLE II: ACTIONS OF TRANSACTION T1 AND T2

| Time | T1 (Deposit) | T2 (Withdraw) |
|------|--------------|---------------|
| t1   | Read (Account ‘ACC01’) |               |
| t2   | Read (Account balance of ‘ACC01’) |               |
| t3   |                   | Read (Account ‘ACC01’) |
| t4   |                   | Read (Account balance of ‘ACC01’) |
| t5   | Write (Account balance of ‘ACC01’) |               |
| t6   | Write (Transaction) |               |
| t7   |                   | Read (Account balance of ‘ACC01’) |
| t8   | Commit tran |               |
| t9   | Write (Account balance of ‘ACC01’) |               |
| t10  | Write (Transaction) |               |
| t11  | Commit tran |               |

### TABLE III: ACTIONS OF TRANSACTION T1 AND T2

| Time | T1 (Deposit) | T2 (Reading Account Balance) |
|------|--------------|-------------------------------|
| t1   | Begin tran |               |
| t2   | Read (Account ‘ACC01’) |               |
| t3   | Read (Account balance of ‘ACC01’) |               |
| t4   | Begin tran |               |
| t5   | Read (Account ‘ACC01’) |               |
| t6   | Read (Total of deposit amount) |               |
| t7   | Write (Account balance of ‘ACC01’) |               |
| t8   |                   | Read (Total of withdrawal amount) |
| t9   | Write (Transaction) |               |
| t10  |                   | Commit tran |               |
| t11  |                   | Commit tran |               |
| t12  |                   | Commit tran |               |

### TABLE IV: ACTIONS OF TRANSACTION T1 AND T2

| Time | T1 (Deposit) | T2 (Withdraw) |
|------|--------------|---------------|
| t1   | Begin tran |               |
| t2   | Read (Account ‘ACC01’) |               |
| t3   | Read (Account balance of ‘ACC01’) |               |
| t4   |                   |               |
| t5   |                    |               |
| t6   | Write (Account balance of ‘ACC01’) |               |
| t7   | Write (Account balance of ‘ACC01’) |               |
| t8   | Read (Account balance of ‘ACC01’) |               |
| t9   | Write (Account balance of ‘ACC01’) |               |
| t10  | Write (Transaction) |               |
| t11  | Error (Rollback) |               |
| t12  | Write (Transaction) |               |
| t13  | Commit tran |               |
| t14  | Commit tran |               |

DOI: http://dx.doi.org/10.24018/ejece.2021.5.4.352
V. CONCLUSION

This paper presents an overview of how to use concurrency control in database management system to help know how to apply isolation levels when data dispute occurs.

The SERIALIZABLE isolation level is the safest and recommended isolation level for most transactions. However, some transactions can be performed with a lower isolation level, and a lower number of locks required that can contribute to improved system performance such as the READ COMMITTED isolation level and the READ UNCOMMITTED isolation level.

In summary, depending on the requirements for data accuracy, time performance and data dispute errors, we can choose the appropriate isolation level for each situation.

ACKNOWLEDGMENT

The author thanks reviewers for their reading of our manuscript and their insightful comments and suggestions.

REFERENCES

[1] A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts – 3rd Edition, McGraw - Hill, 1999.
[2] C. J. Date, Database System, Pearson Addison Wesley, 2004.
[3] H. G. Molina, J. D. Ullman, J. Widom, Database Systems: The Complete Book, Prentice - Hall, 2002.
[4] R. Elmasri, S. R. Navathe, Fundamentals of Database Systems – 7th Edition, Pearson - Addison Wesley, 2016.
[5] Concurrency Control, Berkeley University, http://inst.eecs.berkeley.edu/~cs186/fa05/lecs/18cc-6up.pdf, January 10, 2018.
[6] Raghu Ramakrishnan, Johannes Gehrke, Database Management Systems – 3rd, International Edition 2003.
[7] SQL Server Recovery Models, http://databases.about.com/od/sqlserver/a/recoverymodels.htm, March 5, 2018.

Thi Ngoc Thanh Nguyen is the lecturer of Information Technology Faculty, Ho Chi Minh City University of Transport, Vietnam. She received B.Sc. and M.Sc. in Computer Science from VNU - University of Science and VNU - University of Engineering and Technology. Her current research interests include Database, Data Mining, Artificial Intelligence and Soft Computing.